1000 **File Formats**

Historie & Budoucnost

Budoucnost

Formáty, Soubory, Tabulky: Samozřejmě další a další popisy...
Přílohy: Rád bych sem zařadil další freeware zdrojové texty. Pokud máte nějaký, který by zde neměl chybět, pošlete mi ho e-mailem (studna@czechia.cz).
Uvítám vaši pomoc při dalším rozšiřování této encyklopedie.

Verze 3.1

Formáty: VOC, T602
Rozšíření: URL, SCR
Zdrojáky: PLAYFLI.ASM

Verze 3.0 (1. 3. 1998)

Formáty: PAL
Soubory: LOGO.SYS, LOGOW.SYS, LOGOS.SYS, AUTORUN.INF (rozšířeno), WINHELP.INI, WININIT.INI
Standardy: Domény 1. řádu, Kódové stránky, Scan kódy kláves, Virtuální kódy kláves
Nová úprava prostředí.

Verze 2.26 (29. 11. 1997)

Formáty: ANI, CDA, CNT, CRD, ICO, REG
Rozšíření: HTML
Soubory: AUTORUN.INF, CDPLAYER.INI
Standardy: Řídící kódy ASCII

Verze 1.20 (2. 9. 1997)

Formáty: CAB, EXE (DOS), EXE (Windows), MS Office, RES, URL
Oprava: FLI
Rubriky: Zdroje, Help!, Historie
Rozšíření témat: Bin, Windows

Verze 1.0 (2. 7. 1997)

Formáty: BMP, CHR, CLP, COM, FLC, FLI, Group File Formats, HTML, MOD, PCX, RAR, SCR, WAV, XM

Autoři & Kontakty

Homepage
http://www.krovina.cz/studna/

Redakce
Hynek Sládeček, Pražská 14, 261 01 Příbram II
studna@czechia.cz

Vlastimil Janda
Lužická 1332, 396 01 Humpolec

Pavel Jisl
Boženy Němcové 788, 252 28 Černošice
jislp@feld.cvut.cz
http://cs.felk.cvut.cz/~jislp

Marek Jurák
uri@psg.sk

Vít Kovalčík
vkovalcik@iname.com
http://www.geocities.com/SiliconValley/Hills/1335/

Bronislav Křístek
Lubina 20, 742 21 Kopřivnice

Petr Nekvinda
Otradov 74, 539 43 Krouna

Petr Říha
594 44 Radostín nad Oslavou 222

Tibor Szabó
Křižná 679, 757 01 Valašské Meziříčí
TSZA7574@menza.mff.cuni.cz
http://port.kolej.mff.cuni.cz/~tsza7574/index.htm

Copyright

Verze 3.1

1000 **File Formats**

1000 **File Formats**

1000 **File Formats**

1000 **File Formats**

1000 **File Formats**

1000 **File Formats**

Autodesk Animator FLI

Hynek Sládeček

Hlavní princip je velice jednoduchý - v souboru nejsou uloženy všechny obrázky kompletní (jako na filmu), ale na dalším obrázku jsou popsány změny, které upravují obrázek minulý.

Pokud tedy v levém dolním rohu myška zahýbe ušima, jsou v souboru uloženy jen informace o překreslování jejího ucha.

Tento postup je velice praktický, co se týče velikosti i rychlosti přehrávání. (Jen se zpětným přehráváním je to horší).

Soubor FLI obsahuje nejprve hlavičku dlouhou 128 bajtů.

Hlavička FLI souboru

offset	velikost	jméno	význam
00h	4 byty	size	délka souboru
04h	2 byty	magic	identifikátor - AF11h
06h	2 byty	frames	počet snímků v souboru (maximálně 4000)
08h	2 byty	width	šířka obrazovky (320)
0Ah	2 byty	height	výška obrazovky (200)
0Ch	2 byty	depth	barevná hloubka pixelu - počet bitů (8)
0Eh	2 byty	flags	0
10h	2 byty	speed	prodleva mezi snímky (v 1/70 sekundy)
12h	2 byty	next	0
14h	4 byty	frit	0
18h	102 bytů	expand	volno - pro budoucí verze

Jediný povolený formát je 320x200.

Následují jednotlivé snímky. Každý snímek obsahuje hlavičku.

Hlavička snímku

offset	velikost	jméno	význam
00h	4 byty	size	počet bajtů ve snímku
04h	2 byty	magic	identifikátor - F1FAh
06h	2 byty	chunks	počet akcí
08h	8 byty	expand	volno - pro budoucí verze

Chunk znamená anglicky poleno. Položka chunks určuje počet akcí, kterými se vykreslí daný snímek. Akcí existuje několik druhů. Následují hned po hlavičce snímku.

Nejprve přichází akce měnící barevnou paletu, potom akce měnící pixely.

Každé poleno má svou hlavičku.

Hlavička akce

offset	velikost	jméno	význam
00h	4 byty	size	počet bajtů akce
04h	2 byty	type	typ akce

Akce může být jedna z těchto:

číslo	jméno	význam
11	FLI_COLOR	změna palety
12	FLI_LC	komprimovaná řádka

13	FLI_BLACK	na celé obrazovce nastavena barva na 0
15	FLI_BRUN	komprimovaná řádka (bez vynechávání)
16	FLI_COPY	nekomprimovaná data pro celou obrazovku (první snímek)

FLI_COLOR

První word (16-bitový integer) určuje počet balíčků (packet). Hlavička akce je přímo následována datovými packety.

Packet:

První bajt packetu určuje kolik barev se má přeskočit (nechat nezměněných), další bajt kolik barev má být změněno (pokud zde bude 0, bude chápáno jako 256). Následují 3 bajty pro každou barvu (hodnoty RGB - červená, zelená, modrá složka - každá od 0 do 63).

FLI_LC

První word určuje počet řádek (počítáno seshora), které budou ponechány beze změny. Další dva bajty určují počet řádek, které budo změněny.

Následují data pro každou řádku. První bajt řádky je (opět) počet datových packetů. Pokud je řádka nezměněná, tento bajt má hodnotu 0.

Packet:

Při ukládání dat o řádce se využívá komprimace (každá řádka se komprimuje zvlášť). Způsob komprimace je patrný z formátu packetu:

offset	velikost	jméno
0	1	skip_count
1	1	size_count
2	x	data

skip_count určuje počet pixelů, které se mají přeskočit (pokud má být přeskočeno více než 255 pixelů, musí se použít 2 packety)

size_count **-** pokud je větší než nula: size_count určuje počet bajtů, které mají být zkopírovány na obrazovku (určuje velikost položky data)

- pokud je menší než nula: -size_count určuje, kolikrát má být na obrazovku zkopírován následující bajt (v položce data - ta má v tomto případě velikost 1)

FLI_LC je nejčastěji používaná akce.

FLI_BLACK

Celá obrazovka je smazána (barva 0). S touto akcí nejsou vázána žádná další data.

FLI_BRUN

Velice podobný akci FLI_LC - packet ale neumožňuje přeskočit žádné pixely.

Packet:

Formát packetu tedy je:

offset	velikost	jméno
0	1	size_count
1	x	data

size_count **-** pokud je větší než nula: size_count určuje, kolikrát má být na obrazovku zkopírován následující bajt (v položce data - ta má v tomto případě velikost 1)

- pokud je menší než nula: -size_count určuje počet bajtů, které mají být zkopírovány na

obrazovku (určuje velikost položky data)
Pozor, tady je to obráceně!

FLI_COPY

Následuje *width*height* bajtů, které budou zkopírovány na obrazovku.

Viz také FLC.

Autodesk Animator FLC

Hynek Sládeček

Hlavní princip je velice jednoduchý - v souboru nejsou uloženy všechny obrázky kompletní (jako na filmu), ale na dalším obrázku jsou popsány změny, které upravují obrázek minulý.

Pokud tedy v levém dolním rohu myška zahýbe ušima, jsou v souboru uloženy jen informace o překreslování jejího ucha.

Tento postup je velice praktický, co se týče velikosti i rychlosti přehrávání. (Jen se zpětným přehráváním je to horší).

Soubor FLC obsahuje nejprve hlavičku dlouhou 128 bajtů.

Hlavička FLC souboru

offset	velikost	jméno	význam
00h	dword	size	Délka souboru
04h	word	magic	Identifikátor - AF12h
06h	word	frames	Počet snímků v souboru
08h	word	width	Šířka obrazovky
0Ah	word	height	Výška obrazovky
0Ch	word	depth	Barevná hloubka pixelu - počet bitů (8)
0Eh	word	flags	bit 0 - ring frame (přechodový obrázek mezi posledním a prvním snímkem) existuje bit 1 - hlavička není updatována bity 2-15 - rezervováno
10h	dword	speed	Prodleva mezi snímky (v milisekundách)
14h	word	reserved	Rezervováno
16h	dword	created	Datum a čas vytvoření souboru
1Ah	dword	creator	Sériové číslo Animatoru Pro použitého pro vytvoření souboru (pokud byla použita knihovna FlicLib, pak se rovná "FLIB", což je 0464c4942h
1Eh	dword	updated	Datum a čas poslední modifikace
22h	dword	updater	Sériové číslo programu použitého pro poslední modifikaci
26h	word	aspectx	Horizontální konstanta poměru stran obrazovky
28h	word	aspecty	Vertikální konstanta poměru stran obrazovky (320x200 má poměr stran 6:5)
2Ah	38 bytů	reserved	Rezervováno
50h	dword	oframe1	Offset (od začátku souboru) prvního snímku.
54h	dword	oframe2	Offset (od začátku souboru) druhého snímku.
58h	40 bytů	reserved	Rezervováno

Povolené formáty snímku u FLC jsou 320x200, 640x480, 800x600, 1280x1024.

Za hlavičkou může následovat prefix block, který je využíván programem Animator Pro. Obsahuje interní informace. Prefix block má identifikátorem F100h (místo F1FAh, viz níže).

Následují jednotlivé snímky. Každý snímek obsahuje hlavičku.

Za posledním snímkem je ještě jeden snímek (ring frame),který zachycuje rozdíly mezi posledním a prvním obrázkem (viz Hlavička souboru).

Hlavička snímku

offset	velikost	jméno	význam
00h	dword	size	počet bajtů ve snímku
04h	word	type	identifikátor - F1FAh

06h	word	chunks	počet akcí
08h	8 bytů	expand	volno - pro budoucí verze

Chunk znamená anglicky poleno. Položka chunks určuje počet akcí, kterými se vykreslí daný snímek. Akcí existuje několik druhů. Následují hned po hlavičce snímku.
Nejprve přichází akce měnící barevnou paletu, potom akce měnící pixely.

Každé poleno má svou hlavičku.

Hlavička akce

offset	velikost	jméno	význam
00h	dword	size	počet bajtů akce
04h	word	type	typ akce

Akce může být jedna z těchto:

číslo	jméno	význam
4	FLI_256_COLOR	změna palety
7	FLI_DELTA	několik komprimovaných řádek
11	FLI_COLOR	změna palety
12	FLI_LC	komprimovaná řádka
13	FLI_BLACK	na celé obrazovce nastavena barva na 0
15	FLI_BRUN	komprimovaná řádka (bez vynechávání)
16	FLI_COPY	nekomprimovaná data pro celou obrazovku (první snímek)
18	FLI_MINI	thumbnail první obrazovky

FLI_256_COLOR

První word určuje počet packetů. Následují jednotlivé packety.

Packet:

První bajt určuje, kolik barev se má přeskočit. Další bajt určuje, kolik barev se má změnit (0 znamená 256). Následují 3 bajty pro každou barvu (hodnoty RGB - červená, zelená, modrá složka - každá od 0 do 255).

FLI_DELTA

První word určuje, kolik kopmrimovaných řádek následuje. Následují data pro jednolivé řádky, počínaje první řádkou.

První word určuje počet packetů v řádce. Pokud je hodnota záporná, - (počet packetů) určuje, kolik řádek se má přeskočit (a nikoli počet packetů).

Packet:

offset	velikost	jméno
00h	byte	skip_count
01h	byte	size_count
02h	x	data

skip_count určuje počet pixelů, které se mají přeskočit (pokud má být přeskočeno více než 255 pixelů, musí se použít 2 packety)
size_count **-** pokud je větší než nula: size_count určuje počet dvojic bajtů, které mají být zkopírovány na obrazovku (určuje velikost položky data)
- pokud je menší než nula: -size_count určuje, kolikrát má být na obrazovku zkopírován

následující dvojice bajtů (v položce data - ta má v tomto případě velikost 2)

FLI_COLOR

První word (16-bitový integer) určuje počet balíčků (packet). Hlavička akce je přímo následována datovými packety.

Packet:

První bajt packetu určuje kolik barev se má přeskočit (nechat nezměněných), další bajt kolik barev má být změněno (pokud zde bude 0, bude chápáno jako 256). Následují 3 bajty pro každou barvu (hodnoty RGB - červená, zelená, modrá složka - každá od 0 do 63).

FLI_LC

První word určuje počet řádek (počítáno seshora), které budou ponechány beze změny. Další dva bajty určují počet řádek, které budo změněny.

Následují data pro každou řádku. První bajt řádky je (opět) počet datových packetů. Pokud je řádka nezměněná, tento bajt má hodnotu 0.

Packet:

Při ukládání dat o řádce se využívá komprimace (každá řádka se komprimuje zvlášť). Způsob komprimace je patrný z formátu packetu:

offset	velikost	jméno
00h	byte	skip_count
01h	byte	size_count
02h	x	data

skip_count určuje počet pixelů, které se mají přeskočit (pokud má být přeskočeno více než 255 pixelů, musí se použít 2 packety)

size_count **-** pokud je větší než nula: size_count určuje počet bajtů, které mají být zkopírovány na obrazovku (určuje velikost položky data)

- pokud je menší než nula: -size_count určuje, kolikrát má být na obrazovku zkopírován následující bajt (v položce data - ta má v tomto případě velikost 1)

FLI_LC je nejčastěji používaná akce.

FLI_BLACK

Celá obrazovka je smazána (barva 0). S touto akcí nejsou vázána žádná další data.

FLI_BRUN

Velice podobný akci FLI_LC - packet ale neumožňuje přeskočit žádné pixely.

Packet:

Formát packetu tedy je:

offset	velikost	jméno
00h	byte	size_count
01h	x	data

size_count **-** pokud je větší než nula: size_count určuje, kolikrát má být na obrazovku zkopírován následující bajt (v položce data - ta má v tomto případě velikost 1)

- pokud je menší než nula: -size_count určuje počet bajtů, které mají být zkopírovány na obrazovku (určuje velikost položky data)

Pozor, tady je to obráceně!

FLI_COPY

Následuje *width*height* bajtů, které budou zkopírovány na obrazovku.

FLI_MINI

Definuje thumbnail prvního snímku. Bývá umístěn jako první chunk v prvním snímku. Je příliš komplikovaný na to, jaká je jeho využitelnost, proto ho nebudu popisovat. Pokud přesto máte zájem o jeho popis, napište si o něj na adresu redakce.

Viz také FLI.

PCX

Pavel Jisl

Grafický formát PCX byl kdysi nejpoužívanější, teď ho nahradili JPEG, GIF a (bohužel) BMP. Hlavička je dlouhá 128b, potom následují pakovaná data a nakonec paleta.

```
tPCXHeader = record
      Zsoft             :byte  zde je vždy $A0
      Version           :byte  verze popsány níže
      Encoding          :byte  vždy 1 - komprimace metodou RLE
      BitsPerPixel      :byte  počet bitů na pixel
      Xmin,                    souřadnice obrazu
      Ymin,
      Xmax,
      Ymax,
      Hres,                    horizontální rozlišení
      Vres              :word  vertikální rozlišení
      ColorMap          :tCMap       mapa barev
      Reserved          :byte
      Nplanes           :byte  počet rovin
      BytesPerLine,                  délka řádku
      PaletteInfo,                   informace o paletě
      HscreenSize,                   velikost obrazovky
      VscreenSize       :word
      Filler            :array[1..54] of byte
end;

tRGBRec = record               Popis fyzické barvy
      R,G,B:byte;
end;

tCMap = packed array[0..15] of tRGBRec;   Mapa barev
```

Version

0 PC Paintbrush 2.5
dovoluje uchovávat obrazy odpovídající schopnostem CGA bez mapy barev

2 PC Paintbrush 2.8
uchovává obrazy s až 16ti barvami s mapou barev pro max. 16 barev

3 totéž jako verze 2, ale bez uchovávání mapy barev

4 PC Paintbrush pro Windows
shodná s verzí 5

5 PC Paintbrush 3.0+
přidána sekundární paleta pro 256barev a schopnost uchovávat obrazy ve 24bitové RGB grafice

Setkáváme se většinou s verzemi 3 a 5.

Standartní hodnoty Nplanes a BitsPerPixel

Typ obrazu	Nplanes	BitsPerPixel
monochromatická grafika	1	1
CGA grafika, 4 barvy	1	2
EGA grafika, 16 barev	4	1
EGA grafika, 8 barev	3	1
VGA grafika, 256 barev	1	8
24 bitová grafika	3	8
barevný scanner, 64 barev	1	8

RLE komprimace

Metoda RLE je často používaná, zde je tato implementace:

Po načtení bajtu X se zjistí, zda dva nejvyšší bity mají hodnotu jedna. Pokud ano, 6 nižších bitů bajtu X má význam počtu opakování a bajt následující za bajtem X obsahuje hodnotu, která se má opakovat. Nejsou-li dva nejvyšší bity jedničkové, X obsahuje přímo nekomprimovanou hodnotu. Toto vše se opakuje, dokud je počet zpracovaných pixelů menší než `BytesPerLine*NPlanes`.

WAVE

Pavel Jisl

Soubory WAV používají strukturu RIFF, která byla navrhnuta pro multimediální použití. RIFF soubor se skládá z několika "chunků".

RIFF chunk je hlavní hlavička souboru:

rID	(4 bajty)	obsahuje znaky "RIFF"
rLen	(4 bajty)	délka následujícího chunku
rData	(rLen)	DATA chunk

rData chunk obsahuje definici WAVE formátu a je rozdělen na další chunks:

wID	(4 bajty)	obsahuje znaky "WAVE"
Format Chunk	(14 bajtů)	obsahuje chunk specifikující formát dat v Data chunku
WAVE data chunk	(? bajtů)	obsahuje zvuková data - WAV

Format Chunk

fId	(4 bajty)	obsahuje znaky "fmt"
fLen	(4 bajty)	délka dat v chunku
wFormatTag	(2 bajty)	specifikace wave formátu: např.: 1 - 8mi bitový samplovaný nekompresovaný zvuk
nChannels	(2 bajty)	počet kanálů (1= mono, 2= stereo)
nSamplePerSec	(2 bajty)	frekvence přehrávání
nAvgBytesPerSec	(2 bajty)	přenosová rychlost bajtů za sekundu = nChannels * nSamplesPerSec * (nBitsPerSample / 8)
nBlockAlign	(2 bajty)	índikuje zarovnání dat v Data chunku = nChannels * (nBitsPerSample / 8)
FormatSpecific	(2 bajty)	formát specifické datové oblasti

Poznámka: nBitsPerSample je pravděpodobně 8, nepodařilo se mi to zjistit.

Wave Data Chunk

dId	(4 bajty)	obsahuje znaky "data"
dLen	(4 bajty)	délka dat v oblasti dData
dData	(dLen)	wave data

dData: V mono 8mi bitovém souboru reprezentuje každý bajt jeden sampl. Ve stereo 8mi bitovém souboru jsou dva bajty uloženy pro každý samp, první bajt je hodnota levého kanálu a druhý bajt hodnota druhého kanálu.

Pokud chceme udělat přehrávač, tak jediné co nás může zajímat je nChannels pro počet kanálů, nSamplePerSec pro rychlost přehrávání a dLen pro délku dat, ostatní položky jsou sice taky důležité, ale nás nemusí zajímat.

Windows Bitmap (BMP)

Petr Nekvinda

Tento formát díky využití ve Windows získal obrovského rozšíření a využití. Jeho klady jsou dány jeho jednoduchostí. Na druhé straně zápory spočívají v nesnadné(neefektivní) komprimaci. Není tedy vzácností vytvářet 24-bitové BMP o velikosti přes 500 kB. Ale nyní již k vlastnímu formátu.

Existující verze:
Microsoft Windows Bitmap verze 1.x a 2.x - pod OS Microsoftu
Bitmap verze 1.x a 2.x - pod OS OS/2
Microsoft Windows Bitmap verze 3.x
Bitmap verze 2.x - pod OS OS/2

Nebudu zde popisovat rozdíly mezi jednotlivými verzemi jelikož se jedná o ryze vývojové prvky, nýbrž se zaměřím na popis v současné době nejvíce používané BMP verze 3.x na platformě Microsoftu.

Verze 3.x má následující strukturu:
1. Bitmapová hlavička
2. Informační hlavička
3. Paleta (je volitelná, např.u 24-bit.forem ztrácí význam)
4. Bitmapová data

1. Bitmapová hlavička (14 bytů)

```
BMP_header = record
        ImageFileType:      Word;       typ vždy BM v hex. 4D42
        FileSize:           Longint;    velikost souboru v bytech
        Reserved1:          Word;       rezervováno vždy 0
        Reserved2:          Word;       rezervováno vždy 0
        ImageDataOffset:    Longint;    počátek obrazových dat v bytech (offset od začátku
souboru)
 end;
```

2. Informační hlavička (40 bytů)

```
BMP_info = record
        HeaderSize:         Longint;    velikost hlavičky vždy 40
        ImageWidth:         Longint;    šířka obrazu v pixlech
        ImageHeight:        Longint;    výška obrazu v pixlech
        NumberOfImagePlanes:Word;       počet rovin vždy 1
        BitsPerPixel:       Word;       bity na pixel př. 1-bit, 4-bit, 8-bit, 24-bit
        CompressionMethod:  Longint;    metoda komprese 0= ne, 1= 8-bit.RLE, 2= 4-bit.RLE
        SizeOfBitmap:       Longint;    velikost bitmapy v bytech
        HorzResolution:     Longint;    horizont.rozlišení v pixlech na metr
        VertResolution:     Longint;    vertikál.rozlišení v pixlech na metr
        NumColorsUsed:      Longint;    počet barev v paletě (0=maximum z možných)
        NumSignificantColors: Longint;  počet významných barev palety (0= všechny významné)
end;
```

Pozor HeaderSize jsem viděl v jedné knize jako typ WORD avšak neměl jsem tu čest někdy vidět formát v této velikosti - u všech mnou testovaných BMP byl vždy typu Longint (4 byty).

3. Paleta

Paletu používají pouze 1-bit, 4-bit a 8-bitové obrázky.Každá složka palety se skládá ze 4 bytů.

```
RGB = record
        Red         : Byte;     červená složka
        Green       : Byte;     zelená složka
        Blue        : Byte;     modrá složka
        Reserv      : Byte;     rezervováno vždy 0
end;
```

Celková velikost palety je *počet RGB * 4* tedy u 8-bit = 256 (8 bitů) * 4 = 1024 Bytů.

Poznámka: Chceme-li efektivně pracovat s BMP zejména v Dosu nesmíme zapomenout na to, že naplníme-li paletové registry počítače paletou obrázku překreslí se veškerý grafický výstup touto paletou - zde to lze však velice trefně obejít, neboť pokud si necháte vylistovat paletu u BMP, tak zjistíte, že nikdy není 100% využita a na konci zbývá několik(někdy až 20 i více bytů u 256-barevné) nevyužitých. Není nic snazšího než-li vytvořit bytový zapisovač a na konec nevyužité palety připsat své odstíny(z praxe vím, že stačí jen několik základních).

Nastavení položky obrazové palety:

```
Procedure SetRGB(Index, R, G, B: Byte); assembler;
asm
  MOV DX, $03C8
  MOV AL, [Index]
  OUT DX, AL
  INC DX
  MOV AL, [R]
  OUT DX, AL
  MOV AL, [G]
  OUT DX, AL
  MOV AL, [B]
  OUT DX, AL
end;
```

4. Bitmapová data

a) nekomprimovaná

1-bit: 1 Byt = 8 pixelů
4-bit: 1 Byt = 2 pixely
8-bit: 1 Byt = 1 pixel
24-bit: 3 Byty = 1 pixel

Jelikož 1, 4 a 8-bit obrázky používají paletu odpovídají bitmapová data indexu v paletě, tj.např. přečtený byt 64 = barva na 64 pozici v paletě přičemž v BMP souboru jsou data zapisována od levého dolního rohu.

24-bitové jsou uloženy v datech přímo, tj.např.přečtené byty 10 30 60 odpovídají barvě Red= 10, Green= 30, Blue= 60 - z tohoto důvodu mají nekomprimované 24-bit obrázky největší velikost.

5. Kódování bitmapových dat

Většina BMP dat není kódována jelikož jsou poté rychleji čtena a ukládána avšak kompresi také podporují a to efektivně pouze 4-bit a 8-bitové verze(z dalších snad 24-bit v šedé nebo 1-bit s dlouhými řetězci stejné barvy) :

a) kódovaný proud

2 byty (v prvním je počet pixelů a v druhém pixelová hodnota)

b) citovaný proud

5 a více bytů (první je 0 (počátek), druhý značí počet bytů v proudu, proud musí vždy končit na sudé bytové hranici

Kódovací i citovaný proud musí končit vždy na konci skanovací řádky

c) escape sekvence

d) delta escape sekvence

Jelikož problematika kódování je velice složitá a textově rozsáhlá, nerozepisuji poslední dva druhy podle významu, nýbrž je uvádím jako možné.

HyperText Markup Language (HTML)

Pavel Jisl

Úvod

HTML se používá k tvorbě webovských stránek na internetu. Jedná se vlastně o několik speciálních značek, které se vpisují do textového souboru a ovlivňují vzhled stránky. Je důležité si uvědomit, že konečný vzhled stránky tvoří WWW Browser (dále jen klient).
Vlastní příkazy HTML jsou značky, které mají obecný tvar <příkaz> a </příkaz>.Tvar s lomítkem zakončuje blok, začatý tvarem bez lomítka. Existují ale také příkazy, které nemají ukončovací tvar.

Každý HTML dokument má tuto základní strukturu:

```
<HTML>
<HEAD>          Začátek hlavičky dokumentu
<TITLE> Titulek, který se objeví v titulku okna klienta </TITLE>
</HEAD>         Konec hlavičky dokumentu

<BODY>          Začátek vlastního těla dokumentu
... vlastní obsah stránky ...
</BODY>         Konec těla dokumentu

</HTML>
```

Většina klientů zvládá dokumenty bez <HTML> a </HTML> , ale je slušností je uvádět.

Sekce <HEAD>

Sekce <HEAD> se uvádí hned na začátku dokumentu. Obsahuje několik tagů, které se využívají pro zjednodušení a zpříjemnění práce s HTML dokumentem. V sekci <HEAD> můžeme použít následující tagy:

- **<TITLE> text </TITLE>** - definuje název dokumentu, který se objeví v titulku okna klienta, **povinný příkaz, který je vyžadován specifikací HTML 3.2**
- **<ISINDEX>** - pro jednodušší vyhledávání slov
- **<BASE>** - definuje základní URL adresu pro vyhodnocení relativní URL adresy, má tento atribut: **href="URL adresa"**
- **<SCRIPT>** - rezervováno pro užití se skriptovacími jazyky (např. JavaScript), viz. neviditelné tagy
- **<STYLE>** - rezervováno pro užití se styly
- **<META>** - užito pro META informace, využitelné pro vyhledáče (search engines), možné atributy:

  NAME="klíčové-slovo" - určuje, co zde bude za informace, za klíčové slovo se může doplnit:
  keywords - což určuje, že atribut CONTENT bude obsahovat klíčová slova pro vyhledávání
  description - atribut CONTENT bude obsahovat stručný popis stránky
  CONTENT="text" - samotný text

  Jak to vlastně použít. Například pro tuto stránku by mohl tag <META> obsahovat toto (doporučuje se používat angličtinu):

<META NAME="keywords" CONTENT="html specification czech software contact paulsoft">

a pro popis by mohl obsahovat zase toto:

<META NAME="description" CONTENT="HTML-specification in czech language, part of SoftwareContact's 1000 File Formats encyclopaedia, Copyleft by PaulSoft (pigus@hotmail.com)">

- **<LINK>** - užito pro definování spojení s dalšími dokumenty

Sekce <BODY>

V sekci <BODY> lze specifikovat některé parametry pro celý dokument. Jedná se například o barvy pozadí, textu, odkazů nebo o obrázky na pozadí stránky. K tomu se používají tyto parametry:

BGCOLOR="barva" - specifikuje barvu pozadí (jak se barvy zapisují, viz. níže)
TEXT="barva" - specifikuje barvu textu
LINK="barva" - specifikuje barvu nenavštíveného odkazu
VLINK="barva" - specifikuje barvu navštíveného odkazu
ALINK="barva" - specifikuje barvu odkazu v momentu, když na něj uživatel kliká
BACKGROUND="zdroj" - specifikuje URL obrázku, který bude na pozadí dokumentu (zápis viz. odkazy nebo obrázky), doporučuje se délka do 5 Kb, aby načítání netrvalo moc dlouho

Například:

<BODY bgcolor="black" text="#CC11FF" background="pozadi.jpg">

Zápis barev

Barvy se zapisují jako hexadecimální hodnota (např. COLOR="#C012FF"} nebo jako jméno jedné z šestnácti předdefinovaných barev:

Předdefinované barvy

barva a hexa-zápis	česky
Black="#000000"	(černá)
Green="#008000"	(zelená)
Silver="#C0C0C0"	(stříbrná)
Lime="#00FF00"	(světle zelená)
Gray="#808080"	(šedivá)
Olive="#808000"	(olivová hněď)
White="#FFFFFF"	(bílá)
Yellow="#FFFF00"	(žlutá)
Maroon="#800000"	(tmavě červená)
Navy="#000080"	(tmavě modrá)
Red="#FF0000"	(červená)
Blue="#0000FF"	(modrá)
Purple="#800080"	(fialová)
Teal="#008080"	(taková zelená)
Fuchsia="#FF00FF"	(světle fialová)
Aqua="#00FFFF"	(modrá jako voda)

Zápis barev hexadecimálně se provádí zápisem **#RRGGBB**, kde **RR** je hodnota pro červenou složku, **GG**

pro zelenou složku a **BB** pro modrou složku.

Úprava textu

- **<Hx> text </Hx>** - tyto párové značky určují velikost textu, většinou se používají na nadpisy. Číslo x je z intervalu 1 až 6, kdy 1 je největší velikost písma.
- **<FONT> text </FONT>** - určuje použitý font a jeho parametry. Atributy:
  SIZE=n - velikost písma, n je celé číslo od 1 do 7, standardně 3, parametrem může být také např. číslo +2, což znamená, že se použije velikost o dvě větší než standardní COLOR="barva" - určuje barvu písmen použitého fontu
  <BASEFONT> - slouží k nastavení základní velikosti fontu, která je užita v celém dokumentu. Ukončovací forma se nepoužívá. Má jeden parametr:
  SIZE=n - kde n je v mezi 1 a 7
- **<P>** - začátek nového odstavce
  ALIGN= - zarovnání odstavce k levé (**"LEFT"**), pravé (**"RIGHT"**) straně nebo na střed (**"CENTER"**)
- **
** - začátek nového řádku
- **<HR>** - vykreslí horizontální řádku přes celou šíři okna klienta. Atributy:
  WIDTH=xx - šířka v pixelech/procentech (%)
  ALIGN=... -zarovnání k levé (**"LEFT"**) a pravé (**"RIGHT"**) straně
  SIZE=n -tloušťka čáry v pixelech NOSHADE - při použití - plné čáry, jinak stínované, plastické
- **<BLOCKQUOTE> text </BLOCKQUOTE>** - text se zvýrazní a zvětší se okraje, tyto bloky se dají vnořovat.
- **<PRE> text </PRE>** - text se zobrazí přesně tak, jak byl napsán (konce řádek, mezery)

Dále máme značky, které nám říkají, jakým typem písma se má text napsat. Tyto značky můžeme rozdělit na fyzické a logické. Fyzické přesně říkají klientovi, jak má text zobrazit, logické mu oznamují, jak by měl text zobrazit. Doporučuje se používat spíše logické značky.

Fyzické značky:

- **<B> text </B>** - tučné písmo
- **<I> text </I>** - kurzívou
- **<U> text </U>** - podtržení, nedoporučuje se používat, protože by se pletlo z odkazy
- **<TT> text </TT>** - neproporciálním písmem

Logické značky:

- **<EM> text </EM>** - text se zvýrazní (doporučeno kurzívou)
- **<STRONG> text </STRONG>** - text se silně zvýrazní (doporuč. tučně)
- **<CODE> text </CODE>** - zdrojový kód (psací stroj - neproporciálně?)
- **<CITE> text </CITE>** - citace (doporuč. kurzívou)
- **<KBD> text </KBD>** - vstup z klávesnice
- **<ADDRESS> text </ADDRESS>** - adresa
- **<XMP> výpis </MXP>** - vypíše přesný obsah textu (i s příkazy HTML)

- **<CENTER> text </CENTER>** - specialita HTML 3, vše, co je mezi těmito značkami se vycentruje (texty, obrázky, odkazy, tabulky, formuláře, ...)
- **<BLINK> text </BLINK>** - způsobuje blikání textu

Do textu se dají dále vkládat některé symboly:

©	copyrightová značka	**©** nebo ©
®	symbol reg. ochr. známky	**®** nebo ®
&	ampersand	**&** nebo &
>	větší než	**>** nebo >
<	menší než	**<** nebo <
"	dvojitá uvozovka	**"** nebo "
	nezalamovatelná mezera	** ** nebo

Seznamy a výčty

I. Seznam definic, který je vhodný pro slovníky používá tyto značky:
- **<DL> seznam </DL>** - omezovače seznamu definic
- **<DT>** - uvádí definici (neukončuje se značkou </DT> !)
- **<DD>** - výklad definice (opět bez </DD> !)

II. Neuspořádaný seznam začíná grafickým znakem (např. černý puntík). Charakteristickým rysem je zvětšený levý okraj.
- **<UL> seznam </UL>** - omezovače seznamu. HTML 3 zavádí u značky <UL> atribut **TYPE**, který určuje, čím bude seznam uvozen. Nabývá hodnot **"DISC"** (černý puntík, implicitní), **"CIRCLE"** (prázdné kolečko) a **"SQUARE"** (čtvereček)
- **<LI>** - položka seznamu

III. Uspořádaný seznam je to číslovaný seznam, hodící se např. na pracovní postupy.
- **<OL> seznam </OL>** - omezovač číslovaného seznamu. HTML 3 zavádí atribut TYPE, který určuje číslování položek. Může mít tyto hodnoty:
  - **1** - arabské číslice - implicitní
  - **A** - velká písmena angl. abecedy
  - **a** - malá písmena angl. abecedy
  - **I** - velké římské číslice
  - **i** - malé římské číslice
- **<LI>** - položka seznamu

Existují ještě další seznamy (např. **MENU** a **DIR**), ale jsou velmi málo používané.

Odkazy a skoky

<A HREF="cílové URL"> odkaz </A>
Toto je schéma odkazu (**A**nchor). HREF určuje, kam bude odkaz směřovat. Může být absolutní (celá cesta i s typem přístupové metody a adresou serveru) nebo relativní (bez specifikace serveru, je relativní buď k aktuálnímu dokumentu nebo k aktuálnímu serveru). Odkazem může být text nebo obrázek. Cílovým URL mohou být tyto typy:

<A HREF="http://..."> - link na dokument na WWW serveru
<A HREF="ftp://..."> - link na ftpčko, většinou anonymní
<A HREF="gopher://..."> - link na gopher server
<A HREF="mailto:..."> - aktivuje klientův e-mailový dialog, pro zaslání zprávy tomu, jehož adresa následuje za mailto: (např. pro e-mail na mě by tam bylo: < HREF="mailto:pigus@hotmail.com">
<A HREF="news:..."> - link na newsgroup
<A HREF="newsrc:..."> - link na určitý newsrc soubor
<A HREF="nntp://..."> - může být použito pro specifikaci jiného news serveru (???)
<A HREF="telnet://..."> - inicializuje telnet (externí aplikaci)
<A HREF="wais://..."> - link pro spojení se specifikovaným WAIS indexovým serverem

Dále můžeme použít tohoto odkazu ke skoku na určitou pozici v HTML dokumentu. To se provede jednoduše tak, že na místo kam chceme skočit vložíme.

<A NAME="jmeno"></A>

a na místo, odkud chceme skákat, vložíme standardní odkaz:

<A HREF="#jmeno"> text nebo obrázek </A>

Obrázky a "mapy s klikou"

Dají se vkládat obrázky GIF a JPG, doporučuje se délka do 20 kB, aby se zrychlilo načítání. Ikonky pro odkazy mají délku kolem 2 - 5 kB.

<IMG atributy>

Nepárová značka, která má mnoho atributů:

- **SCR="zdroj"** - určuje, kde se obrázek nalézá
- **ALT="text"** - alternativní text, pokud se obrázek nedá zobrazit
- **HEIGH=výška**
- **WIDTH=šířka** - obě hodnoty jsouv pixelech a určují plochu (zrychlí se zobrazování, protože klient rezervuje pro obrázek plochu, ale stejně to nikdo nepoužívá)
- **ALIGN=**
  "TOP" - zarovná text k hornímu okraji obrazovky
  "MIDDLE" - zarovná text na střed obrázku (svislý směr)
  "BOTTOM" - zarovná text k dolnímu okraji obrázku
  "LEFT" - zarovná obrázek k levé straně, text obtéká zprava
  "RIGHT" - zarovná obrázek k pravé straně, text obtéká zleva

U obrázku můžeme dále použít atribut usemap="jméno", který oznamuje, že tento obrázek bude použit jako mapa, ve které jsou aktivní plochy, fungující jako tlačítka ke skoku do jiných dokumentů. Parametrem atributu usemap je jméno MAPy, která určuje aktivní oblasti.

Mapa se označuje párovými značkami **<MAP>** a **</MAP>**, která má pouze jeden atribut:

- **NAME="jméno"** - jméno mapy, určuje, ke kterému obrázku mapa patří

Jednotlivé aktivní oblasti mapy se označují tagem **<AREA>**, který má tyto atributy:

- **SHAPE="tvar"** - označuje tvar aktivní plochy, možné tvary:
  RECT - čtverec nebo obdélník
  CIRCLE - kolečko
  POLY - polynom
- **COORDS="souřadnice"** - souřadnice použitého tvaru, vymezující aktivní oblast, možné použití souřadnic je:
  shape=rect coords="x1,y1,x2,y2", kde [x1,y1] jsou souřadnice levého horního rohu a [x2,y2] pravého dolního rohu **shape=circle coords="x,y,r"**, kde [x,y] jsou souřadnice středu a r je poloměr
  shape=poly coords="x1,y1,x2,y2,..."
  souřadnice se počítají od levého horního rohu obrázku a jsou k němu absolutní, mohou se ale použít i souřadnice zadané procenty.

- **HREF="cílové URL"** - určuje, kam se bude skákat NOHREF - pokud oblast nemá žádný skok ALT="text" - alternativní text, který se zobrazuje při najetí na region v informačním pruhu, nebo je pro klienty, kteří neumí obrázky zobrazovat, velmi se doporučuje používat !!!

Jak se tedy tento aktivně-klikačně-odskakovací obrázek vytvoří, zde je příklad:

```
<img src="navig.jpg" border="0" usemap="#mapa1">
<map name="mapa1">
  <area href=index.htm alt="skok na index" shape=rect coords="0,0,100,30">
  <area href=autor.htm alt="nìco o autoru" shape=circle coords="150,40,30">
  <area href=nabidka.htm alt="nabídka produktù" shape=poly
coords="200,0,250,100,400,20,450,120">
</map>
```

Pro jednodušší nadefinování oblastí byly vytvořeny utilitky, které jsou většinou šířeny jako shareware nebo freeware. Jsou to například:

MapEdit - http://www.boutell.com/mapedit
Map This! - http://www.ecaetcohio-state.edu/tc

Neviditelné tagy

V HTML dokumentech se dále mohou vyskytovat tagy, které nejsou v dokumentu viditelné, ale mají (menší nebo větší) vliv na jeho chování. Jsou to například tyto tagy:

<!-- text --> - tento tag se používá jako poznámka a nemá vliv na zobrazení dokumentu. Dále se používá ke skrytí nových tagů pro starší prohlížeče, které nemají např. zabudovanou normu HTML 3.2

A nyní něco k nynějšímu trendu, k JAVĚ:

<APPLET> - tento tag slouží k připojení Javovského appletu k HTML dokumentu a je podporován všemi klienty, kteří podporují Javu. Tento tag je párový a má tyto atributy:

- **CODEBASE="zdrojové URL"** - volitelný parametr, určující, kde se applet nachází. Pokud není tento atribut použit, použije se URL dokumentu.
- **CODE="jméno"** - tento vyžadovaný atribut určuje jméno souboru, který obsahuje zkompilované podtřídy aplety. Je vždy relativní k zdrojovému URL a nemůže být absolutní.
- **ALT="text"** - specifikuje alternativní text pro klienty, kteří mají zabudovaný tag APPLET, ale nemohou spouštět Javovské applety.
- **NAME="jméno"** - určuje jméno appletu, které slouží pro komunikaci mezi dalšími applety zahrnutých do stránky.
- **WIDTH=xxx** - určuje šířku zobrazovacího okna appletu
- **HEIGHT=xxx** - určuje výšku zobrazovacího okna appletu
- **ALIGN="jak"** - vyžadovaný atribut, který určuje zarovnání appletu (možné hodnoty: LEFT, RIGHT, TOP, TEXTTOP, MIDDLE, ABSMIDDLE, BASELINE, BOTTOM, ABSBOTTOM)
- **VSPACE=xxx** - určuje počet pixelů nad a pod appletem
- **HSPACE=xxx** - určuje počet pixelů na každé straně appletu

K tagu <APPLET> ještě patří tag **<PARAM>**, který slouží k předávání parametrů appletu. Používá se s tímto formátem:

<PARAM NAME=appletParameter **VALUE=**value**>**

Applet potom čte tyto parametry příkazem getParameter().

A ještě jak se to všechno používá:

```
<APPLET CODE="vypis.class" WIDTH=500 HEIGHT=500>
// Java applet nìco vypíše
</APPLET>
```

nebo s využitím tagu PARAM:

```
<APPLET CODE="zahraj.class" WIDTH=15 HEIGHT=15>
<PARAM NAME=zvuk VALUE="zdravicko.au">
// Java applet zahraje zvuk se jménem zdravicko.au
</APPLET>
```

A dále tu je něco jednoduššího, JavaScript:

<SCRIPT> - tento tag se používá při zápisu zdrojového textu JavaScriptu přímo do dokumentu HTML. Musí se použít v sekci <HEAD>. Tento tag může mít atribut LANGUAGE="jméno", kde jméno je např. JavaScript pro JavaScript (pravděpodobně je to pro příští rozšiřování možností, že by bylo třeba více scriptových jazyků). Dále se doporučuje zdrojový kód JavaScriptu skrýt mezi poznámkový tag (<!-- poznámka -->, aby nemátl starší klienty.

Takto nějak vypadá velmi jednoduchý HTML dokument s JavaScriptem:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!-- zaèátek skrývání
document.write("Èau lidi!")
konec skrývání -->
</SCRIPT>
</HEAD>
<BODY>
Vše v pohodì ...
</BODY>
</HTML>
```

Ještě k JavaScriptu: Netscape Navigator verze 2.0+ rozšířil možnosti formulářů o práci s JavaScripty, takže příkazy formulářů mohou mít tento formát:

<TAG eventHandler="JavaScript Code**">**

například: v JavaScriptu máme napsanou funkci compute a chceme, aby Navigátor spustil tuto funkci po stisku čudlíku. Takto by vypadala definice čudlíku:

```
<INPUT TYPE="button" VALUE="Calculate" onClick="compute(this.form)">
```

Ale o tomto podrobněji někdy jindy a na jiném místě.

Rámce

Myšlenka rámů spočívá v rozdělení obrazovky klienta na několik částí tak, aby v každém vzniklém okně mohl být jiný dokument nebo po vybrání odkazu v jednom okně by se v jiném zobrazil výsledek.

Tvorba rámů se dělí do dvou částí:

1. V první části se definuje hlavní stránka s jednotlivými rámy. Jde o HTML dokument, pouze kromě příkazu <BODY> se použije **<FRAMESET>** a **</FRAMESET>**. Uvniř jsou povoleny pouze definice rámů, obyčejný text se nesmí vyskytnout. Atributy u příkazu <FRAMESET> definují rozdělení dokumentu - **ROWS** udává počet řádků a **COLS** počet sloupců.

Například:

```
<FRAMESET ROWS="velikost 1. okna,velikost 2. okna,....">
<FRAMESET COLS="velikost 1. sloupce, velikost 2.sloupce,...">
```

Velikost mohou být zadány buď v procentech (50%) nebo pixelech. Pokud se použije *, udává to zbytek velikosti okna, pokuď dvě *, zbývající část okna se rozdělí na půlku.

Okna jsou zde definována pomocí příkazu **<FRAME>**, který má další parametry:

- **NAME** - nepovinný atribut, udává jméno pro odkaz z jiné části dokumentu
- **SCROLLING** - nepovinný atribut, říká, zda bude mít rám rolovací lišty, povolené hodnoty jsou: YES, NO a AUTO (implicitní)
- **SRC** - cesta k zobrazovanému dokumentu

2. V druhé části se definují jednotlivé dokumenty, na které se odkazujeme v první části. Jsou to standardní dokumenty bez <TITLE>, který je nastaven v hlavní stránce.

Je slušné stránku obsahující rámy doplnit textem pro "nerámové" klienty. K tomu se používají značky **<NOFRAMES>** a **</NOFRAMES>**. Vše co je v těchto příkazech, zobrazuje "nerámový" klient na místě, kde jsou rámy.

Z důvodů lepšího pochopení jsem zde vytvořil několik HTML dokumentů, které užití rámců ukazují:

- Tento první dokument se jmenuje hlavni.html a na něj se také odkazujeme browserem (je v něm hlavní definice rámců a užití souborů):

```
<HTML>
<HEAD> <!-- Standarní hlavièka s titulkem -->
<TITLE>PaulSoft's FRAME Page Example</TITLE>
</HEAD>
<FRAMESET COLS="30%,70%"> <!-- místo BODY se užije FRAMESET, zde je obrazovka
rozdìlena
                          // svisle na dvì èásti, jedna zabírá 30% okna a
druhá 70% okna klienta -->
   <FRAME SRC="index.html" NAME="frame1">    <!-- v prvním oknì bude dokument
INDEX.HTML -->
   <FRAMESET ROWS="20%,80%">                 <!-- druhou èást dìlíme vodorovnì
opìt na dvì èásti -->
     <FRAME SRC="nadpis.html" NAME="frame2"> <!-- V první èásti bude dokument
NADPIS.HTML, který se nemìní -->
     <FRAME SRC="text1.html" NAME="frame3">  <!-- V druhé èásti dokument
TEXT1.HTML -->
   </FRAMESET>
</FRAMESET>
<NOFRAMES> <!-- Pro klienty, kteøí nejsou FRAME-Able  -->
Sorry, ale tento dokument vyžaduje browser, který podporuje rámce (FRAMES),
zkuste napø. Netscape Navigator 2.0+.
</NOFRAMES>
</HTML>
```

- Zde je definice dokumentu index.html, který nahrává do okna se jménem frame3 další dokumenty:

```
<HTML>
<BODY BGCOLOR="#C0C0C0">
<H1 ALIGN="center">INDEX</H1>
<P>
<A HREF="text1.html" TARGET=frame3>První text</A><BR> <!-- do okna frame3 se
nahraje TEXT1.HTML -->
<A HREF="text2.html" TARGET=frame3>Druhý text</A><BR> <!-- do okna frame3 se
nahraje TEXT2.HTML -->
<A HREF="text3.html" TARGET=frame3>Tøetí text</A><BR> <!-- do okna frame3 se
nahraje TEXT3.HTML -->
</BODY>
</HTML>
```

- Dokument nadpis.html vypadá takto:

```
<HTML>
<BODY BGCOLOR="#808080" TEXT="#C0C0C0">
<H1 ALIGN=center>PaulSoft's FRAME Page</H1>
</BODY>
</HTML>
```

- text1.html:

```
<HTML>
<BODY BGCOLOR="#FFFF00">
Toto je první zobrazitelný text.
</BODY>
</HTML>
```

- text2.html:

```
<HTML>
<BODY BGCOLOR="#FFFF00">
Toto je první zobrazitelný text.
</BODY>
</HTML>
```

- text3.html:

```
<HTML>
<BODY BGCOLOR="#008000">
A toto je tøetí zobrazitelný text.
</BODY>
</HTML>
```

Tabulky

Dříve bylo možno uspořádat text do tabulek jen pomocí příkazu <PRE>, a to pouze neproporciálně písmem Courier. Bylo to složité, zdlouhavé a ne moc pěkné. Nyní se používá příkaz **<TABLE>** a **</TABLE>**.

Kostra tabulky:

```
<TABLE>
<TR>
<TD> 1. øádek, 1. políèko </TD>
<TD> 1. øádek, 2. políèko </TD>
<TD> 1. øádek, 3. políèko </TD>
</TR>
<TD> 1. øádek, 1. políèko </TD>
<TD> 1. øádek, 2. políèko </TD>
<TD> 1. øádek, 3. políèko </TD>
</TR>
...
</TABLE>
```

- **<TABLE>** - Začátek tabulky, atributy:
  BORDER=n - určuje šířku rámečku, čím je n větší, tím je rámeček tlustší
  WIDTH=xx - tabulka bude vždy široká xx pixelů nebo xx %.
  ALIGN="jak" - zarovnání tabulky vůči oknu klienta
  CELLSPACING=xx - definuje mezery mezi políčky v tabulce
  CELLPADDING=xx - definuje mezeru v políčku

- **<TR> text </TR>** - začátek nového řádku, lze použít atribut:
  HALIGN=x - zarovnání v horizontálním směru
  VALIGN=x - zarovnání ve vertikálním směru
- **<TD> text </TD>** - nové políčko, lze použít tyto atributy:
  HALIGN= - zarovnání v horizontálním směru, známé parametry LEFT, RIGHT a CENTER.
  VALIGN= - zarovnání v vertikálním směru, může nabývat hodnot TOP a BOTTOM.
  ROWSPAN=n - kde n znamená, přes kolik řádek bude políčko vysoké
  COLSPAN=n - kde n znamená, přes kolik sloupců bude políčko dlouhé
  NOWRAP - zakazuje dělení slov v políčku
  WIDTH=n - určuje šířku políčka
  HEIGHT=n - určuje výšku políčka
- **<TH> text </TH>** - políčko s nadpisem, lze použít stejné atributy jako u <TD>
- **<CAPTION> titulek </CAPTION>** - jméno tabulky, píše se hned za <TITLE>, může se použít atribut ALIGN s parametry TOP nebo BOTTOM.

Příklad tabulky:

```
<TABLE WIDTH="75%" BORDER=3>
<CAPTION>Ukázková tabulka</CAPTION>
<TR>
<TH ROWSPAN=2>Hodina</TH>
<TH COLSPAN=2>Rozvrh</TH>
</TR>
<TR>
<TH>1.A</TH>
<TH>1.B</TH>
</TR>
<TR>
<TD ALIGN="CENTER">1.</TD>
<TD>Èeština</TD>
<TD>Matematika</TD>
</TR>
<TR>
```

```
<TD ALIGN="CENTER">2.</TD>
<TD>Matematika</TD>
<TD>Hudební výchova</TD>
</TR>
<TR>
<TD ALIGN="CENTER">3.</TD>
<TD>Hudební výchova</TD>
<TD>Ěeština</TD>
</TR>
</TABLE>
```

Formuláře

Formuláře se používají pro komunikaci uživatele se serverem, kdy uživatel vyplňuje do polí své požadavky, údaje apod. a server podle toho reaguje. Klasické využití je jako dotazník na informace o uživateli. Formulář se definuje pomocí dvojice tagů **<FORM>** a **</FORM>**, přičemž může mít tyto atributy:

- **ACTION="akce"** - specifikuje URL, které přijímá poslaný formulář. Může to být HTTP server nebo Mail-To URL. Mail-To URL umožňuje poslat vyplněný formulář přes e-mail, např. takto: action="mailto:pigus@hotmail.com".
- **METHOD="metoda"** - specifikuje, jakou HTTP-metodou bude formulář HTTP serveru poslán, možné hodnoty jsou:
  GET - standard
  POST
- **ENCTYPE="typ"** - určuje mechanismus kódování obsahu formuláře, standardně je to řetězec "application/x-www-form-urlencoded"

Pokud máme nadefinovanou hlavičku formuláře, můžeme začít do něj vkládat jednotlivé komponenty. K tomu slouží tyto tagy:

<INPUT> - používá se pro pole jako input-lajny, pole na vkládání hesel, checkboxy, radiobuttons, posílací a resetovací tlačítko, skrytá pole nebo pole na uploadování souborů. Možné atributy jsou:

- **TYPE="typ"** - určuje typ pole, možné hodnoty jsou:

text - standardní, používá se ke vkládání jednoho řádku textu, počet viditelných znaků může být určen atributem size, celkový počet znaků může být omezen atributem maxlength, atribut name je použit jako jméno pole a value atribut může být použit k hodnotě, která se bude standardně objevovat.
password - ke vkládání hesla, vkládané znaky se interpretují jako "*", opět se zde mohou použít atributy size a maxlength jako u pole text
checkbox - používá se k určení Booleanovské hodnoty (ano/ne) nebo jako atribut, který může nabývat více hodnot. Různé checkboxy jsou reprezentovány stejným jménem v atributu name a různými hodnotami v atributu value. Každý zaškrtnutý checkbox generuje rozdělené jméno nebo hodnotu v posílaných datech. Atribut checked může být použit k určení startovací hodnoty checkboxu a to k zaškrtnutí
radio - používá se k určení jedné hodnoty z několika variant. Každý radiobutton ve skupině musí mít stejný atribut name a vyžaduje se atribut value, ve kterém se určuje posílaná hodnota a pouze jeden radiobutton ze skupiny ji generuje. Jeden radiobutton ze skupiny by měl být při inicializaci označen atributem checked
submit - slouží k odeslání obsahu formuláře k serveru. Text použitý na čudlíku se definuje atributem value, pokud je použit atribut name, jeho jméno bude vloženo do posílaných dat.
image - používá se jako posílací tlačítko (type="submit"), ale místo textu se použije obrázek. Zdrojové URL obrázku se definuje atributem scr a zarovnání atributem align, což je stejné jako u obrázků. Atributy name a value mají stejné vlastnosti jako u textového čudlíku submit.

reset - používá se k vymazání formuláře a nastavení inicializačních hodnot. Jméno se nastavuje atributem value.

file - slouží k připojení souboru do obsahu formuláře. Většinou je renderován klientem jako textové pole s čudlíkem, sloužícím k vyhledávání souborů. hidden - toto pole není zobrazováno a využívá se pro servery jako informace o obsahu formuláře.

- **NAME="jméno"** - definuje jméno, které bude použito k identifikování obsahu pole po odeslání k serveru
- **VALUE="hodnota"** - užívá se k inicializování hodnoty pole nebo jako text k posílacímu nebo resetovacímu tlačítku
- **CHECKED** - nastavuje u checkboxů nebo radiobuttonů označený stav
- **SIZE="délka"** - nastavuje viditelnou délku textových polí (ve znacích)
- **MAXLENGTH="délka"** - nastavuje maximální počet vložitelných znaků do textových polí
- **SRC="zdrojové URL"** - specifikuje URL obrázku o grafického posílacího tlačítka
- **ALIGN="jak"** - určuje zarovnání obrázku, stejné jako u obrázků

<SELECT> - slouží k definování menu, kde se vybírá buď jedna položka z více položek nebo více položek z více položek. Používá se v párové formě <SELECT> a </SELECT> a mezi ně se vkládá tag **<OPTION>**, kterým se definují jednotlivé položky. Tag **<SELECT>** může mít tyto atributy:

- **NAME="jméno"** - specifikuje jméno, které bude použito k identifikování volby z menu při odeslání formuláře k serveru. Každá označená volba bude vložena do obsahu odesílaného formuláře.
- **SIZE="délka"** - nastavuje počet viditelných znaků pro menu pro výběr více položek z více položek.
- **MULTIPLE** - nastavuje menu pro výběr více položek z více položek, pokud se nepoužije, užije se menu pro výběr jedné položky z více položek

<OPTION> - tento tag se používá k definování jednotlivých položek menu a může mít tyto atributy:

- **SELECTED** - položka bude při startu označena, u menu pro víběr jedné položky z více položek se smí použít pouze jednou!!
- **VALUE="hodnota"** - specifikuje hodnotu, která bude použita při odesílání obsahu formuláře, pokud se za tagem <OPTION> neuvede text, který se bude zobrazovat v menu, bude se zobrazovat tato hodnota.

<TEXTAREA> - užívá se ke vkládání několika řádek textu. Používá se v párové formě a mezi počátečním a ukončovacím tagem může být text, který se bude objevovat při inicializaci. Možné atributy jsou:

- **NAME="jméno"** - definuje jméno, které bude identifikovat toto pole při odeslání obsahu formuláře
- **ROWS="řádek"** - definuje počet viditelných řádek
- **COLS="sloupců"** - definuje počet viditelných sloupců

Změny a rozšíření

verze 1.00
- První uvolněná verze

verze 1.10
- Doplněny skoky na části textu
- Doplněna sekce o odkazech, upravena sekce o tabulkách
- Sekce Obrázky doplněna o "mapy s klikou"
- Sekce Rámce doplněna o příklady.
- Zrušena sekce Specialitky
- Nová sekce o <HEAD> a <BODY>
- Nová sekce o barvách, formulářích a neviditelných tazích

Své připomínky, názory nebo doporučení můžete poslat na tuto adresu **jislp@feld.cvut.cz**. Předem děkuji za všechny připomínky a doporučení k vylepšení.

Amiga MOD

Tibor Szabó

MOD soubory jsou zvukové soubory obsahující notový zápis (podobně jako .mid), ke kterému jsou přidány vzorky těchto nástrojů. Tento typ soborů vznikl na počítačích Amiga a ve standartní verzi (Protracker) umožňuje realizovat čtyřkanálovou stereo hudbu (1. a 4. kanál vlevo , 2. a 3. vpravo). Starší verze mohla obsahovat pouze 15 samplů, novější už 31.

Struktura souboru:

offset	délka	popis
00h	14h	Název skladby

Dále následuje popis hlaviček jednotlivých samplů (1. -31.)

14h	16h	Název samplu
2Ah	2	Délka samplu (Uloženo jako Word, ale pozor wordy tu nejsou uloženy podle konvence Intelu, ale významnější bajt napřed.) - (pro získání skutečné délky vynásob 2).
2Ch	1	4 nižší bity udávají finetune pro sampl. 0=0, 1=1, ..., 7=7, 8=-8, 9=-7, ..., 15=-1
2Dh	1	Hlasitost samplu (0 - 64)
2Eh	2	Offset začátku loopu v samplu. (pro získání skuteč. offsetu nutno násobit 2)
30h	2	Délka loopu. (* 2)
32h-3B5h		Stejné struktury následují pro dalších 30 samplů.
3B6h	1	Délka skladby (1-128) - počet patternů
3B7h	1	Nějaký bajt, o kterém nic nevím. Asi se nepoužívá.
3B8h	80h	Popis pořadí patternů, každý bajt obsahuje číslo patternu (0-63)
438h	4h	Řetězec M.K. (Toto obsahují soubory s 31 samply)

Dále následuje popis jednotlivých patternů. Kolik jich bude se zjistí z nejvyššího čísla v pořadí patternů offset 3B8h) zvětšeného o 1.

43Ch	400h	Popis 1. patternu.
83Ch	400h	Popis 2. patternu.
	...	

Struktura patternu :

0.řádek		Kanál1	Kanál2	Kanál3	Kanál4	
		4 bajty	4 bajty	4 bajty	4 bajty	= 16 bajtů na řádek.
	offset	0	4	8	0Ch	
1.řádek		Kanál1	Kanál2	Kanál3	Kanál4	
	offset	10h	14h	18h	1Ch	
... 63. řádek						

Jednotlivé kanály jsou uloženy takto :

Bajt 1	Bajt 2	Bajt 3	Bajt 4
SSSSRRRR	PPPPpppp	ssssEEEE	eeeeffff

SSSSssss - Číslo samplu.
RRRRPPPPpppp - 12 bit - perioda
EEEEeeeeffff - Efekty.

Přehrávání

Frekvence hraného samplu se zjistí takto f=7093789.2/(2*perioda).
Rychlost přehrávání (default 6) udává na kolikáté vyvolání hrací rutiny se řádek přehraje.

Přehrávací rutina se tuším vyvolává 50x za sekundu (PAL 50 Hz ?). Pokud je nutno vypisovat noty, pak je nutno najít příslušnou hodnotu periody v periodtable a podle ní notu určit.

```
PERIODTABLE pro FINETUNE 0
unsigned int periodtable[36]=
//C   C#  D   D#  E   F   F#  G   G#  A   A#  H
{856,808,762,720,678,640,604,570,538,508,480,453,  //Oktáva 1
 428,404,381,360,339,320,302,285,269,254,240,226,  // Oktáva 2
 214,202,190,180,170,160,151,143,135,127,120,113}; // Oktáva 3
//PERIODTABLE pro FINETUNE +1
{850,802,757,715,674,637,601,567,535,505,477,450,
 425,401,379,357,337,318,300,284,268,253,239,225,
 213,201,189,179,169,159,150,142,134,126,119,113};
//PERIODTABLE pro FINETUNE +2
{844,796,752,709,670,632,597,563,532,502,474,447,
 422,398,376,355,335,316,298,282,266,251,237,224,
 211,199,188,177,167,158,149,141,133,125,118,112};
//PERIODTABLE pro FINETUNE +3
{838,791,746,704,665,628,592,559,528,498,470,444,
 419,395,373,352,332,314,296,280,264,249,235,222,
 209,198,187,176,166,157,148,140,132,125,118,111};
//PERIODTABLE pro FINETUNE +4
(832,785,741,699,660,623,588,555,524,495,467,441,
 416,392,370,350,330,312,294,278,262,247,233,220,
 208,196,185,175,165,156,147,139,131,124,117,110};
//PERIODTABLE pro FINETUNE +5
{826,779,736,694,655,619,584,551,520,491,463,437,
 413,390,368,347,328,309,292,276,260,245,232,219,
 206,195,184,174,164,155,146,138,130,123,116,109};
//PERIODTABLE pro FINETUNE +6
{820,774,730,689,651,614,580,547,516,487,460,434,
 410,387,365,345,325,307,290,274,258,244,230,217,
 205,193,183,172,163,154,145,137,129,122,115,109};
//PERIODTABLE pro FINETUNE +7
{814,768,725,684,646,610,575,543,513,484,457,431,
 407,384,363,342,323,305,288,272,256,242,228,216,
 204,192,181,171,161,152,144,136,128,121,114,108};
//PERIODTABLE pro FINETUNE -8
{907,856,808,762,720,678,640,604,570,538,504,480,
 453,428,404,381,360,339,320,302,285,269,254,240,
 226,214,202,190,180,170,160,151,143,135,127,120};
//PERIODTABLE pro FINETUNE -7
{900,850,802,757,715,675,636,601,567,535,505,477,
 450,425,401,379,357,337,318,300,284,268,253,238,
 225,212,200,189,179,169,159,150,142,134,126,119};
//PERIODTABLE pro FINETUNE -6
{894,844,796,752,709,670,632,597,563,532,502,474,
 447,422,398,376,355,335,316,298,282,266,251,237,
 223,211,199,188,177,167,158,149,141,133,125,118};
//PERIODTABLE pro FINETUNE -5
{887,838,791,746,704,665,628,592,559,528,498,470,
 444,419,395,373,352,332,314,296,280,264,249,235,
 222,209,198,187,176,166,157,148,140,132,125,118};
//PERIODTABLE pro FINETUNE -4
{881,832,785,741,699,660,623,588,555,524,494,467,
 441,416,392,370,350,330,312,294,278,262,247,233,
 220,208,196,185,175,165,156,147,139,131,123,117};
```

```
//PERIODTABLE pro FINETUNE -3
{875,826,779,736,694,655,619,584,551,520,491,463,
 437,413,390,368,347,338,309,292,276,260,245,232,
 219,206,195,184,174,164,155,146,138,130,123,116};
//PERIODTABLE pro FINETUNE -2
{868,820,774,730,689,651,614,580,547,516,487,460,
 434,410,387,365,345,325,307,290,274,258,244,230,
 217,205,193,183,172,163,154,145,137,129,122,115};
//PERIODTABLE pro FINETUNE -1
{862,814,768,725,684,646,610,575,543,513,484,457,
 431,407,384,363,342,323,305,288,272,256,242,228,
 216,203,192,181,171,161,152,144,136,128,121,114};
```

Efekty :

EEEE =
0 - Arpeggio
1 - Porta up
2 - Porta down
3 - Tone portamento
4 - Vibrato
5 - Tone porta + volume slide
6 - Vibrato + volume slide
7 - Tremolo
8 - ?
9 - Sample offset
Ah - Volume slide
Bh - Position jump
Ch - Set volume
Dh - Pattern break
Eh - rozšířené efekty
Fh - Set speed

Rozšířené efekty:

E0 - Set filter
E1 - Fine slide up
E2 - Fine slide down
E3 - Glissando
E4 - Set vibrato waveform
E5 - Set finetune
E6 - Set jump to loop
E7 - Set tremolo wavef.
E8 - ?
E9 - Retrigger note
EA - Fine volume slide up
EB - Fine volume slide down
EC - Cut note
ED - NoteDelay
EE - PatternDelay
EF - InvertLoop

XM Module

Tibor Szabó

Autor formátu : Triton
Vlastnosti : Hudební soubor (rozšířený Amiga module formát)
32 kanálů
128 nástrojů (16 samplů / nástroj)
max délka samplu 4 Gb
...

Struktura souboru:

Offset	**Velikost**	**Popis**
0	17	Podpis souboru - string `Extended module: `
17	20	Název modulu
37	1	Bajt - 0x1A - využití neznám
38	20	Název programu, ve kterém byl soubor vytvořen
58	2	Verze souboru - word uložen konvencí Intelu - tedy LoByteHiByte. př. 0x0401 je verze 0x0104
60	4	Dvojslovo délky hlavičky
64	2	Počet patternů (v tabulce pořadí patternů)
66	2	Restart pozice
68	2	Počet kanálů = 2n, kde n je 1..16
70	2	Počet patternů v souboru
72	2	Počet nástrojů
74	2	bit 0 znamená : 0 - Amigácká freq. tabulka 1- Lineární freq. tabulka
76	2	Tempo
78	2	BMP
80	256	Pole bajtů - tabulka pořadí patternů

Struktura patternů:

0	4	Dvojslovo délky hlavičky patternu
4	1	0 - ?
5	2	Počet řádků v patternu - 1..256 (Proč to není bajt ?)
7	2	Délka těla patternu

Vlastní tělo patternu:

0	1	Nota - 0-71 , vyšší bity využity ke kompresi
1	1	Nástroj 0-128
2	1	Volume
3	1	Typ efektu
4	1	Parametr efektu

Vyšší bity noty jsou využity ke kompresi takto:
nastavení bitu 0: následuje nota
1: následuje nástroj
2: následuje volume
3: následuje efekt

Struktura nástroje:

0	4	Délka nástroje
4	22	Jméno nástroje
26	1	0 - ?

27	2	Počet samplů v nástroji

Struktura jednotlivých samplů:

0	4	Délka hlavičky samplu
4	96	Sample number pro všechny noty
100	48	Hodnoty pro volume envelope
148	48	Hodnoty pro panning envelope
196	1	Počet hodnot volume
197	1	Počet panning hodnot
198	1	Volume sustain point
199	1	Volume loop start point
200	1	Volume loop end point
201	1	Panning sustain point
202	1	Panning loop start point
203	1	Panning loop end point
204	1	Volume type - bit 0-On,1-Sustain,2-Loop
205	1	Panning type -bit 0-On,1-Sustain,2-Loop
206	1	Vibrato type
207	1	Vibrato sweep
208	1	Vibrato depth
209	1	Vibrato rate
210	2	Volume fadeout
212	2	?

Struktura hlavičky samplu:

0	4	Délka samplu
4	4	Start loopu v samplu
8	4	Délka loopu
12	1	Volume
13	1	Signed finetune -16..15
14	1	Typ loopu - Bity 0 a 1 - 0-bez loopu, 1 - forward loop 2- ping pong loop 4 - sample data jsou 16 bitové
15	1	Panning (0-255)
16	1	Relativní nota
17	1	Reservováno
18	22	Název samplu

Sample data:
Vlastní sample je uložen na základě přírůstku hodnot, pro převod na klasický signed sample:

```
add:=0;
for i:=1 to delkasamplu do
begin
  newsample:=sampl[i]+add;
  sampl[i]:=newsample;
  add:=newsample;
end;
```

Výpočet hlasitosti podle Tritonů:

FinalVol=(FadeOutVol/65536)*(EnvelopeVol/64)*(GlobalVol/64)*(Vol/64)*Scal
FinalPan=Pan**+**(EnvelopePan-32)*(128**-**Abs(Pan-128))/32;

Výpočet frekvence:

Lineární frekvence:

Period = 7680 - Nota*64 - FineTune/2;

Frequency = 16760^((4608 - Period) / 768);

Amigácká frekvence:
Period = (PeriodTab[(Nota MOD 12)*8 + FineTune/16]*(1-Frac(FineTune/16)) +
PeriodTab[(Nota MOD 12)*8 + FineTune/16]*(Frac(FineTune/16)))
*16/2^(Note DIV 12);
Frequency = 8363*1712/Period;

PeriodTab = (907,900,894,887,881,875,868,862,856,850,844,838,832,826,820,814,
808,802,796,791,785,779,774,768,762,757,752,746,741,736,730,725,
720,715,709,704,699,694,689,684,678,675,670,665,660,655,651,646,
640,636,632,628,623,619,614,610,604,601,597,592,588,584,580,575,
570,567,563,559,555,551,547,543,538,535,532,528,524,520,516,513,
508,505,502,498,494,491,487,484,480,477,474,470,467,463,460,457);

Efekty:

0	Appregio
1	Porta up
2	Porta down
3	Tone porta
4	Vibrato
5	Tone porta+Volume slide
6	Vibrato+Volume slide
7	Tremolo
8	Set panning
9	Sample offset
A	Volume slide
B	Position jump
C	Set volume
D	Pattern break
E1	Fine porta up
E2	Fine porta down
E3	Set gliss control
E4	Set vibrato control
E5	Set finetune
E6	Set loop begin/loop
E7	Set tremolo control
E9	Retrig note
EA	Fine volume slide up
EB	Fine volume slide down
EC	Note cut
ED	Note delay
EE	Pattern delay
F	Set tempo/BPM
G	Set global volume
H	Global volume slide
K	Key off
L	Set envelope position
P	Panning slide
R	Multi retrig note
T	Tremor
X1	Extra fine porta up
X2	Extra fine porta down

Efekty ve volume column:

Hodnota	Popis

0	Nic
0x10-0x50	Nastav hlasitost honota-=0x10
0x60-0x6f	Volume slide down
0x70-0x7f	Volume slide up
0x80-0x8f	Fine volume slide down
0x90-0x9f	Fine volume slide up
0xa0-0xaf	Set vibrato speed
0xb0-0xbf	Vibrato
0xc0-0xcf	Set panning
0xd0-0xdf	Panning slide left
0xe0-0xef	Panning slide right
0xf0-0xff	Tone porta

Závěr:
XM formát je pravděpodobně nejvyspělejší, ale také nejsložitější hudební formát současnosti. jeho implementace vyžaduje značné zkušenosti. Proto doporučuji nejprve důkladné prostudování základních MOD formátu(jednodušší struktura, řada společných efektů). Dobrým studijním materiálem může být také domovský tracker tohoto formátu Fast Tracker 2.0 a samozřejmě také nějaký XM modul. Pozor - ve struktuře XM modulů různých verzí mohou být rozdíly (tento popis je pro verzi 0x0104). Programátoři, kteří udělají přehrávač zvládající veškeré vymoženosti tohoto formátu (volume+panning envelopes, ping pong loop, ...) jsou na takové úrovni, že už naprogramují cokoli.

Group file formats

Pavel Jisl

Toto je popis několika skupinových formátů, používaných ve hrách (jsou to hry jako třeba Dune][, Duke Nukem 3D nebo Quake). Všechny informace jsem zjišťoval sám, proto je možné, že jsem někde udělal chybu.

DUNE II "PAK" Fileformat

Tento skupinový souborový formát je velmi jednoduchý, hlavičku jednoho souboru můžeme zapsat takto (nejedná se o přesný zápis, neboť délka položky "name" se mění a jméno je ukončeno hodnotou 0h):

```
type
   dune_head = record
     position : longint;
     name : asciiz; {délka se mìní !!}
   end;
```

Mezi konce hlaviček a začátek dat jsou ještě vloženy 4 bajty o hodnotě 0h, s kterými musíme při načítání počítat.

Pro jednodušší pochopení jsem napsal v pascalu prográmek, který soubory z tohoto pak-u vybalí.

Soubor dune.pas

```
{vybalovac souboru z ".PAK" DUNE ][}
{freeware by paulsoft/no!future}

program dune_pak;

type
   dune_head = record
     position : longint;
     name : string[12];
   end;

var hlavicky:array[1..255] of dune_head; {hlavicky}
    p:pointer;
    zacdat:longint; {od teto pozice jiz jsou data}
    f1,f2:file;
    i:integer;
    pocet:byte; {pocet souboru v pak-u}
    ch:char;
    delkapaku:longint; {celkova delka pak-u}
    delka:longint;
    dummy:longint;
    x:byte;

begin
  assign(f1,paramstr(1)); reset(f1,1); {otevreni pak-u}
  blockread(f1,zacdat,sizeof(zacdat),i); {nacte pozici prvnich dat}
  seek(f1,0); pocet:=1;
  writeln('vybaluji soubory z ".pak" souboru ',paramstr(1),' :');
```

```
    while filepos(f1)<zacdat-4 do begin {nacitani dat do hlavicky}
      blockread(f1,hlavicky[pocet].position,4,i); {pozice dat v pak-u}
      repeat {nacitani jmena}
        blockread(f1,ch,1,i); if ch<>#0 then
        hlavicky[pocet].name:=hlavicky[pocet].name+ch;
      until ch=#0;
      writeln(hlavicky[pocet].name); inc(pocet); {zvysi pocet}
    end;

    dec(pocet); {protoze se to zvetsi o jedno}
    writeln('vybaluji ',pocet,' souboru');
    delkapaku:=filesize(f1); {celkova delka pak-u}
    seek(f1,zacdat); {seekne na zacatek dat}

    for x:=1 to pocet do begin {vlastni vybalovani}
      assign(f2,hlavicky[x].name); rewrite(f2,1); {vytvori soubor}
      {pocitame delku vybalovanych souboru, odcitame vetsi pozici od mensi,
      jenom pro posledni soubor odcitame od konce pak-u}
      if x+1>pocet then delka:=delkapaku-hlavicky[x].position else
      delka:=hlavicky[x+1].position-hlavicky[x].position;
      if delka<=32768 then begin {pro delku menci nez 32kb}
        getmem(p,delka); blockread(f1,p^,delka,i);
        blockwrite(f2,p^,delka,i); freemem(p,delka);
      end else begin {pro delku vetsi nez 32kb}
        getmem(p,32768); dummy:=delka;
        while dummy>=32768 do begin
          dummy:=dummy-32768; blockread(f1,p^,32768,i);
          blockwrite(f2,p^,32768);
        end; freemem(p,32768); getmem(p,dummy);
        blockread(f1,p^,dummy,i); blockwrite(f2,p^,dummy,i);
        freemem(p,dummy);
      end;
      close(f2); {uzavre vybaleny soubor}
    end;

    close(f1); {uzavre pak}
  end.
```

DUKE NUKEM 3D "GRP" Fileformat

Formát ".GRP" z Duke Nukema 3D je velmi jednoduchý (jednodušší než formát pak-u z duny][:-). Hlavičky v tomto formátu jsou 16ti bytové a jednoduše zapsány vypadají takto:

```
type
  grp_head = record
    name:array[1..12] of char;  {jedná se o asciiz øetìzec}
    length:longint; {délka souboru uloženého v grp-souboru}
  end;
```

Jedinou vyjímkou je ihned první položka, která neobsahuje jméno souboru a jeho délku, ale obsahuje pod položkou "name" jméno autora enginu Duka (Kena Silvermana), tudíž tam je"KenSilverman" a místo délky je zde uložen počet souborů v grp-fajlu.

Stejný formát grp-souboru používají i hry (nevím jak plné verze, testoval jsem jenom sharewarové verze) Shadow Warrior, Redneck Rampage a Blood, jelikožjsou postaveny na stejném enginu od Kena Silvermana.

Pro grp-soubor jsem vybalovač nedělal, dá se jednoduše sehnat s plnou verzí Duke Nukema a

myslím, že formát je tak pochopitelný, že si ho v případě nouze může kdokoliv napsat.

QUAKE "PAK" Fileformat

Kvejk od id-čka má také vcelku jednoduchý formát. Soubor "pak" má na začátku tuto hlavičku:

```
type
  quake_head = record
    id : array[1..4]of char;  {zde je je text "PACK"}
    position : longint;  {pozice jednotlivých hlavièek souborù}
    length : longint;  {délka hlavièek}
  end;
```

Součet "position+length" by měl dát celkovou délku "pak" souboru.

Od pozice "position" již jsou jednotlivé souborové hlavičky s tímto formátem:

```
type
  quake_file_head = record
    name : array[1..56] of char;  {jedná se o asciiz øetìzec}
    position : longint;
    length : longint;
  end;
```

Délka souborové hlavičky je 64bytes, proto můžeme vypočítat počet souborů v pak-u, a to jednoduše takto: quake.length div sizeof(quake_one_file)

Jména souborů jsou uložena i s cestou a zbytek jména do 56bytes je vyplněno hodnotou 0h. Myslím, že vytvořit vybalovač opět nebude problém, protože formát je skoro shodný s formátem v Duke Nukemu 3D, akorát že hlavičky souborů jsou uloženy na konci pak-u.

RAR Archive

Pavel Jisl

Toto je popis formátu (asi) nejznámějšího a (určitě) nejlepšího pakovacího programu RAR. Tento text vychází z verze 2.50, ale platí i pro starší verze od verze 1.50.

Archív je sestaven z bloků o různé délce. Jejich pořadí může být různé, ale první blok musí být **marker block** následovaný **archive header block**.

Každý blok začíná těmito položkami:

```
HEAD_CRC          2 bytes     celkové CRC bloku nebo èásti bloku
HEAD_TYPE         1 byte      typ bloku
HEAD_FLAGS        2 bytes     znaèky bloku
HEAD_SIZE         2 bytes     délka bloku
ADD_SIZE          4 bytes     volitelné - délka pøidaného bloku
```

Položka `ADD_SIZE` je přítomna pouze, když `(HEAD_FLAGS&0x8000)!=0`

Celková délka bloku je `HEAD_SIZE`, když `(HEAD_FLAGS&0x8000)==0`, nebo, když je přítomna položka `ADD_SIZE`, `HEAD_SIZE`+`ADD_SIZE`.

Bity v `HEAD_FLAGS` mají v každém bloku stejný význam:

0x4000 - pokud je nastaven, starší verze RARu bude ignorovat blok a zruší ho, když bude updateovat archív.
- pokud není nastaven, blok bude kopírován při updateování archívu do nového archívu

0x8000 - pokud je nastaven, je přítomna položka `ADD_SIZE` a celková délka bloku je `HEAD_SIZE`+`ADD_SIZE`

Přednastavené typy bloků jsou:

```
HEAD_TYPE=0x72          marker block
HEAD_TYPE=0x73          hlavièka archívu
HEAD_TYPE=0x74          hlavièka souboru
HEAD_TYPE=0x75          hlavièka poznámek
HEAD_TYPE=0x76          speciální informace
HEAD_TYPE=0x77          podblock
HEAD_TYPE=0x78          obnovovací blok
```

Poznámkový blok je nyní použit pouze v jiných blocích a nemůže existovat samostatně.

S archívem se pracuje tímto postupem:

1. načte se a vyzkouší se marker block
2. načte se hlavička archívu
3. načte se nebo se přeskočí `HEAD_SIZE-sizeof(MAIN_HEAD)` bytes
4. pokud jsme na konci archívu, práce se ukončí, jinak se načte 7bytes do polí `HEAD_CRC`, `HEAD_TYPE`, `HEAD_FLAGS`, `HEAD_SIZE`
5. vyzkouší se položka `HEAD_TYPE`, pokud je `HEAD_TYPE`==0x74, pak se načte souborová hlavička, načte se nebo přeskočí `HEAD_SIZE-sizeof(FILE_HEAD)` bytes a načte se nebo přeskočí `FILE_SIZE` bytes, pokud je `HEAD_TYPE`!=0x74, načte se příslušný `HEAD_TYPE` blok: načte se `HEAD_SIZE`-7 bytes a pokud je `(HEAD_FLAGS&0x8000)`, tak se načte `ADD_SIZE` bytes, v případě, že je nutné blok přeskočit, přeskočíme `HEAD_SIZE` - 7 bytes a pokud je `(HEAD_FLAGS&0x8000)`, tak se přeskočí `ADD_SIZE` bytes

6. skok na bod 4

Formáty bloků

Marker block (MARK_HEAD)

HEAD_CRC 2 bytes Vždy 0x6152
HEAD_TYPE 1 byte Typ hlavičky: 0x72
HEAD_FLAGS 2 bytes Vždy 0x1a21
HEAD_SIZE 2 bytes Délka bloku = 0x0007

Marker block se nyní skládá z této sekvence bytes: 0x52 0x61 0x72 0x21 0x1a 0x07 0x00

Archivní hlavička (MAIN_HEAD)

HEAD_CRC 2 bytes CRC polí od HEAD_CRC do RESERVED2
HEAD_TYPE 1 bytes Typ hlavičky: 0x73
HEAD_FLAGS 2 bytes Bitové vlajky:
0x01 - atribut volume (archive volume)
0x02 - obsahuje archivní poznámky
0x04 - atribut zámku archívu
0x08 - archiv vytvořen metodou SOLID
0x10 - nepoužito
0x20 - obsahuje Authenticity information

další bity jsou reservovány pro vlastní použití
HEAD_SIZE 2 bytes celková délka archivní hlavičky s archivními poznámkami
RESERVED1 2 bytes rezervováno
RESERVED2 4 bytes rezervováno

Blok poznámek je přítomen, pokud (HEAD_FLAGS & 0x02)!=0

Souborová hlavička (soubor v archívu)

HEAD_CRC 2 bytes CRC polí od HEAD_CRC do FILENAME
HEAD_TYPE 1 byte Typ hlavičky: 0x74
HEAD_FLAGS 2 bytes Bitové vlajky:
0x01 - soubor pokračuje z předchozího volume
0x02 - soubor pokračuje v dalším volume
0x04 - soubor je zaheslován
0x08 - soubor obsahuje poznámky (comments)
0x10 - je použita informace z předchozího souboru (solid vlajka)
(pro RAR 2.5 a pozdější)

bity 7 6 5 (pro RAR 2.5 a pozdější)

0 0 0 - délka slovníku (dictionary) 64 kb
0 0 1 - délka slovníku (dictionary) 128 kb
0 1 0 - délka slovníku (dictionary) 256 kb
0 1 1 - délka slovníku (dictionary) 512 kb
1 0 0 - délka slovníku (dictionary) 1024 kb
1 0 1 - rezervováno

1 1 0 - rezervováno
1 1 1 - soubor je adresář

`(HEAD_FLAGS & 0x8000)==1`, protože celková délka bloku je `HEAD_SIZE+PACK_SIZE`

`HEAD_SIZE`	2 bytes	Celková délka souborové hlavičky zahrnující jméno a poznámky
`PACK_SIZE`	4 bytes	Délka zabaleného souboru
`UNP_SIZE`	4 bytes	Délka rozbaleného souboru
`HOST_OS`	1 byte	Operační systém použitý pro archovování

0 - MS DOS
1 - OS/2
2 - Win32
3 - Unix

`FILE_CRC`	4 bytes	CRC souboru
`FTIME`	4 bytes	Datum a čas souboru ve standardním formátu MS DOSu
`UNP_VER`	1 bytes	Verze RARu potřebná k rozbalení
`METHOD`	1 bytes	Metoda komprese
`NAME_SIZE`	2 bytes	Délka jména
`ATTR`	4 bytes	Atributy souboru
`FILENAME`		Jméno souboru - řetězec dlouhý `NAME_SIZE` bytes

Blok poznámek (Comment block)

`HEAD_CRC`	2 bytes	CRC polí od `HEAD_CRC` do `COMM_CRC`
`HEAD_TYPE`	1 byte	Typ hlavičky: `0x75`
`HEAD_FLAGS`	2 bytes	Bitové vlajky
`HEAD_SIZE`	2 bytes	Délka poznámkové hlavičky a délka poznámky
`UNP_SIZE`	2 bytes	Délka nezabalené poznámky
`UNP_VER`	1 byte	Verze RARu, potřebná k vybalení poznámky
`METHOD`	1 byte	Metoda komprese
`COMM_CRC`	2 bytes	CRC poznámky
`COMMENT`		Text poznámky

Blok speciálních informací

HEAD_CRC	2 bytes	CRC bloku
HEAD_TYPE	1 byte	Typ hlavičky: `0x76`
HEAD_FLAGS	2 bytes	Bitové vlajky
HEAD_SIZE	2 bytes	Celková délka bloku
INFO		Další data

Podblok

Za objektem v archívu (blok nebo hlavička) může následovat **podblok**. Druh tohoto podbloku závisí na typu hlavního objektu. Podblok může být vymazán nebo přesunut do nové verze archívu, když je updateován.

`HEAD_CRC`	2 bytes	CRC bloku
`HEAD_TYPE`	1 byte	Typ hlavičky: `0x77`
`HEAD_FLAGS`	2 bytes	Bitové vlajky

`(HEAD_FLAGS & 0X8000)==1`, protože celková délka bloku je `HEAD_SIZE+DATA_SIZE`

`HEAD_SIZE`	2 bytes	Celková délka bloku
`DATA_SIZE`	4 bytes	Celková délka dat

SUB_TYPE	2 bytes	Typ podbloku
RESERVED	1 byte	Musí být vždy 0
other fields		Další položky, závisející na typu podbloku

Rozšířené atributy podbloku v OS/2

HEAD_CRC	2 bytes	CRC bloku
HEAD_TYPE	1 byte	Typ hlavičky: 0x77
HEAD_FLAGS	2 bytes	Bitové vlajky

(HEAD_FLAGS & 0X8000)==1, protože celková délka bloku je HEAD_SIZE+DATA_SIZE

HEAD_SIZE	2 bytes	Celková délka bloku
DATA_SIZE	4 bytes	Celková délka dat (délka zabalených rozšížených atributů)
SUB_TYPE	2 bytes	0x100
RESERVED	1 byte	Musí být vždy 0
UNP_SIZE	4 bytes	Délka nezkompresovaných rozšířených atributů
UNP_VER	1 byte	Verze RARu, potřebná k vybalení rozšířených atributů
METHOD	1 byte	Metoda komprese
EA_CRC	4 bytes	CRC rozšířených atributů

Poznámky k práci s archívy

Pro práci se SFX archívy je potřeba přeskočit SFX rutinu vyhledáním **marker blocku** v archívu. Toto se dá udělat vyhledáním sekvence bytes 0x52 0x61 0x72 0x21 0x1a 0x07 0x00, která se v SFX archívu nevyskytuje.

CRC je vypočítáváno použitím standardního polynomiálu (?) 0xEDB88320. V případě, že je délka CRC menší než 4 bytes, jsou užity pouze byty s nižšího řádu.

Typy kompresních metod:

```
0x30  bez komprese (storing)
0x31  nejrychlejší komprese (fastest)
0x32  rychlá komprese (fast)
0x33  normální komprese (normal)
0x34  lepší komprese (good)
0x35  nejlepší komprese (best)
```

Číslo verze RARu k vybalování je kódována jako 10 * Major + Minor

Literatura:

[technote.doc] - Technická dokumentace k RARu verze 2.50

Windows Clipboard (CLP)

Hynek Sládeček

Hlavička

FileIdentifier	word	Identifikátor, hodnota 50C3h
FormatCount	word	Počet bloků uložených v souboru

Blok

FormatID word Určuje typ bloku, jedna z následujících konstant:

CF_TEXT	1	text, každý řádek je ukončen kombinací CR-LF, konec textu znakem číslo 0
CF_BITMAP	2	bitová mapa
CF_METAFILEPICT	3	obrázek metafile
CF_SYLK	4	SYLK (Microsoft Symbolic Link)
CF_DIF	5	DIF (Data Interchange Format)
CF_TIFF	6	TIFF (Tag Image File Format)
CF_OEMTEXT	7	text ve formátu OEM, každý řádek je ukončen kombinací CR-LF, konec textu znakem číslo 0
CF_DIB	8	DIB (Device Independent Bitmap)
CF_PALETTE	9	barevná paleta
CF_PENDATA	10	data pro pero
CF_RIFF	11	RIFF (Resource Interchange File Format)
CF_WAVE	12	WAVE (pouze pro RIFF WAVE)
CF_OWNERDISPLAY	0x0080	Soukromá data, která musí zobrazit vlastník schránky.
CF_DSPTEXT	0x0081	Soukromá data zobrazovaná jako text.
CF_DSPBITMAP	0x0082	Soukromá data zobrazovaná jako bitová mapa.
CF_DSPMETAFILEPICT	0x0083	Soukromá data zobrazovaná jako obrázek metafile.

LenData	dword	velikost dat
OffData	dword	offset (od začátku souboru) začátku dat
Name	79 bytů	jméno soukromého formátu

Data

Za posledním blokem následují data v příslušném kódu. Pokud je kód bitová mapa nebo metafile, obrazová data následují přímo po hlavičce bitmapy nebo struktuře MATAFILEPICT.

Popisy formátů dat viz:
BMP
WAV

další: MS Windows Software Development Kit 3.1

Windows Screensaver (SCR)

Hynek Sládeček

Formát SCR je zcela totožný s formátem EXE (Windows).

Při tvorbě screensaveru je třeba brát v úvahu:

- Windows program spouští s parametrem /S pro spuštění screensaveru
- Windows program spouští bez parametru nebo s parametrem /C pro konfiguraci screensaveru
- Aby mohl být screensaver rozpoznán Control Panelem, jeho module description string, popis programu, musí začínat řetězcem "SCRNSAVE:"
- Při stisknutí klávesy nebo pohybu myši je třeba program ukončit.

Pro Windows 95/NT:

Parametr	Funkce
/a xxxx	nastavení hesla saveru, xxxx je handler standardního okna windows pro změnu hesla, (pouze win95, u NT se o heslo stará systém)
/p xxxx	preview, xxxx je handler okna, kde má být saver zobrazen (viz. monitor v control panel)

Díky patří Danielu Benetkovi <danben@usa.net>.

Borland BGI Stroked Font (CHR)

Vlastimil Janda

Na začátku souboru je 128 bajtů dlouhá hlavička :

```
offset  -  délka  -  popis
  0     -     4   -  v každém souboru  stejné : "PK", 8, 8
  4     -    86   -  Podpis firmy. (BGI Stroked font V1.1,atd..)
 90     -     2   -  Délka hlavièky (128 Bajtù)
 92     -     4   -  Ètyøpísmenný název fontu.
 96     -     2   -  Velikost souboru s fonty.
 98     -     2   -  Verze (11=1.1)
100     -     2   -  Minimální èísla verzí (?)
102     -    26   -  Výplò do 128-mi
```

Na offsetu 80h (128d) jsou data o souboru :

```
128    -     1      -   Vždycky '+' - oznaèení souboru s èárovými fonty.
129    -     2      -   Poèet znakù v souboru (x)
131    -     1      -   Nedefinováno.
132    -     1      -   ASCII hodnota prvního znaku v souboru.
133    -     2      -   Offset k definicím znakù. Pozor! K tomuto èíslo
                        musíte pøièíst èíslo 134, abyste dostali offset
                        od zaèátku souboru.
135    -     1      -   Scan pøíznak (normálnì nula).
136    -     1      -   Výška písmene É v tomto fontu - nejvyšší evropské
                        písmeno
137    -     1      -   Výška báze(?). Vìtšinou nula.
138    -     1      -   Vzdálenost mezi pomyslnou linkou a "dnem" písmene
                        'q'. Vìtšinou nula nebo záporná hodnota
                        (max +/- 64).
139    -     5      -   Nedefinováno.
144    -    2x      -   Tabulka offsetù k individuálním znakùm (relativnì
                        k zaèátku definicí znakù - 144 + 3.x). Jeden offset
                        = 2 bajty.
144+2x -     x      -   Tabulka šíøek znakù. Každý znak má jinou šíøku.
                        1 bajt = 1 znak
144+3x -     y      -   Zaèátek definic znakù - Adresu libovolného znaku
relativnì
k zaèátku souboru vypoèítáme takto : (144+3.x)+[144+2.(AZ-[132])]. Kde x je
poèet znakù,AZ je ASCII hodnota našeho znaku a hranaté závorky znamenají
obsah adresy uvnitø (jako v Assembleru).
```

Asi bych to měl trochu objasnit. Takže 144+3x je adresa prvního znaku. K té musíme přičíst offset našeho znaku. Tabulka offsetů je na adrese 144. Adresu offsetu našeho znaku získáme tak, že k adrese 144 přičteme dvojnásobek pořadového čísla znaku (jeden znak = 2 bajty). Pořadové číslo je rozdíl ASCII hodnoty našeho znaku a ASCII hodnoty prvního znaku v souboru. Tak je to.

Pokud tomu nerozumíte, tak se tím nezatěžujte, ale vězte, že tento vzorec nemusí platit pro soubory, kde nejsou definovány všechny znaky.

Ale to ještě není všechno. Ještě vám ukážu strukturu jednoho znaku. Každý znak se skládá z proměnného počtu operací potřebných k jeho vykreslení. Každá operace je dlouhá 2 bajty. Přitom první bajt je X-ová souřadnice a druhý bajt Y-ová souřadnice. Nejvyšší (osmé) bity těchto dvou bajtů definují operaci, která se má provést. Takže každá operace vypadá takto :

1. Bajt

bit 7 - operační kód 1
bity 0 až 6 - X-ová souřadnice

2. Bajt

bit 7 - operační kód 2
bity 0 až 6 - Y-ová souřadnice

Jsou celkem tři druhy operací :

- Kresli z aktuální pozice do X,Y - op. kód 1 má hodnotu 1, op.kód 2 taktéž
- Přesuň pozici do X,Y - op.kód 1 má hodnotu 1, op.kód 2 má 0
- Konec definice znaku - op. kód 1 má hodnotu 0, op.kód 2 taky tak

Executable File (COM)

Hynek Sládeček

COM je formát spustitelného souboru z dob CP/M. Z toho také vyplývá jeho hlavní omezení - nesmí být delší než 64 kB (kód i data) = 1 segment.

Jeho struktura je velice jednoduchá - data jsou přímo operační kódy instrukcí. COM nemá žádnou hlavičku.

Program COM je nahrán do segmentu až od offsetu 100h, proto začátek COM programu v assembleru musí vypadat nějak takto:

```
code segment
org 100h    ;nastaví pozici poèítadla v aktuálním segmentu na 100h
...
```

Soubor COM je snadno nakazitelný virem - stačí přidat kód viru na konec souboru, na začátek souboru vložit instrukci skoku na začátek viru a na konci těla viru provést skok na druhou instrukci (původní začátek).

Viz také: EXE (DOS)

DOS Executable File (EXE) Header

Hynek Sládeček

Hlavička souboru

Offset	délka	popis
00h	word	signatura "MZ"
02h	word	počet bytů v poslední stránce souboru
04h	word	délka souboru ve stránkách (stránka=512 bytů)
06h	word	počet položek v relokační tabulce (**n**)
08h	word	velikost hlavičky v paragrafech (paragraf=16 bytů)
0Ah	word	minimální velikost paměti (v paragrafech) potřebné ke spuštění programu
0Ch	word	maximální velikost paměti (v paragrafech)
0Eh	word	počáteční hodnota SS (v paragrafech)
10h	word	počáteční hodnota SP
12h	word	kontrolní součet (negativní součet všech slov)
14h	word	vstupní bod programu (hodnota IP)
16h	word	počáteční hodnota CS (v paragrafech)
18h	word	offset relokační tabulky (**o**) (pro Windows NE je to 40h)
1Ah	word	číslo překryvného modulu (0 pro hlavní program)

DOS nahrává EXE soubor do paměti podle údajů uvedených v hlavičce, nikoli podle skutečné délky souboru.

Další prostor (mezi hlavičkou a relokační tabulkou) může obsahovat různá data, v závislosti na typu programu:

New Executable (viz Windows EXE)

Offset	Délka	Popis
001Ch	4 byty	????
0020h	1 word	behaviour bits ??
0022h	26 bytů	reserved (0)
003Ch	1 dword	offset hlavičky NE (od začátku souboru)

Programy generované Borlandským TLINK

Offset	Délka	Popis
001Ch	2 byte	?? (zřejmě vždy 01h 00h)
001Eh	1 byte	signatura 0FBh
001Fh	1 byte	verze TLINK (hlavní ve čtyřech vyšších bitech)
0020h	2 byte	??

Starý samorozbalovací archív ARJ

Offset	Délka	Popis
001Ch	4 znaky	signatura "RJSX" (starší verze - nová signatura je "aRJsf" v prvních 1000 bytech programu)

Program komprimovaný LZEXE

Offset	Délka	Popis
001Ch	2 znaky	signatura "LZ"
001Eh	2 znaky	číslo verze ("09" - LZExe 0.90, "91" - LZExe 0.91)

Program komprimovaný PKLITE

Offset	Délka	Popis
001Ch	1 byte	minoritní číslo verze
001Dh	1 byte	bity 0-3 hlavní číslo verze
		bit 4 -extra komprese
		bit 5 - více segmentový soubor
001Eh	6 znaků	signatura "PKLITE"

Samorozbalovací archív LHarc 1.x

Offset	Délka	Popis
001Ch	4 byty	nepoužito ???
0020h	3 byty	skok na začátek rozbalovacího kódu
0023h	2 byty	???
0025h	12 znaků	signatura "LHarc's SFX "

Samorozbalovací archív LHarc 2.x

Offset	Délka	Popis
001Ch	8 bytů	???
0024h	10 znaků	signatura "LHa's SFX " (pro verzi 2.10), "LHA's SFX " (pro verzi 2.13)

Samorozbalovací archív LH

Offset	Délka	Popis
001Ch	8 bytů	???
0024h	8 bytů	signatura "LH's SFX"

Program komprimovaný TopSpeed C 3.0 Crunch

Offset	Délka	Popis
001Ch	1 dword	signatura 018A0001h
0020h	1 word	signatura 1565h

Samorozbalovací archív PKARC 3.5

Offset	Délka	Popis
001Ch	1 dword	signatura 00020001h
0020h	1 word	signatura 0700h

Samorozbalovací archív LARC

Offset	Délka	Popis
001Ch	4 byty	???
0020h	11 bytů	signatura "SFX by LARC"

Relokační tabulka
tabulka začíná na offsetu o a obsahuje n 32-bitových položek ve formátu offset, segment

Položka relokační tabulky

Offset	Délka	Popis
0000h	1 word	Offset
0002h	1 word	Segment

Viz také: COM
EXE (Windows)

Windows New Executable File (NE-EXE) Header

Hynek Sládeček

Windowsovský EXE soubor se skládá ze dvou částí: krátkého DOSovského programu (včetně hlavičky), který většinou vytiskne informace o tom, že program je určen pro Windows: "This program requires Microsoft Windows" - může to být samozřejmě jakákoli jiná akce, např. spuštění Windows (se jménem programu jako parametrem). WinZip Self-Extractor generuje soubory, které obsahují rozbalovací programy zároveň pro DOS i Windows.

Informace o tom, kde začíná hlavička NE, jsou uvedeny v hlavičce pro DOS.

Hlavička NE

Offset	Délka	Popis
00h	word	signatura "NE"
02h	byte	Verze linkeru
03h	byte	Revizní číslo linkeru
04h	word	Offset vstupní tabulky (od začátku hlavičky NE)
06h	word	Délka vstupní tabulky v bytech
08h	dword	Rezervováno
0Ch	byte	Program flags: bit 0: SINGLEDATA - program obsahuje jeden datový segment (program je knihovna DLL) bit 1: MULTIPLEDATA - program obsahuje více datových segmentů (program je spustitelná aplikace) pokud ani jeden z bitů 0, 1 není nastaven, program je NOAUTODATA (neobsahuje automatic data segmenr) bit 2: globální inicializace (?) bit 3: pouze chráněný mód bit 4: instrukce 8086 bit 5: instrukce 80286 bit 6: instrukce 80386 bit 7: instrukce 80x87
0Dh	byte	Aplication flags: bity 0-2 typ aplikace (0 - full screen/konzolová aplikace, 1 - kompatibilní s Windows API, 2 - používá WinAPI) bit 3: aplikace OS/2 bit 4: první segment v programu obsahuje kód, který načítá aplikaci bit 5: linker zjistil při sestavování programu chybu, přesto byl EXE vytvořen bit 6: rezervováno bit 7: soubor je DLL knihovna
0Eh	byte	Číslo automatic data segmentu
10h	word	Počáteční velikost haldy
12h	word	Počáteční velikost zásobníku
14h	dword	Vstupní bod programu (CS:IP, CS je index v segmentové tabulce)
18h	dword	Počáteční ukazatel na zásobník (SS:SP, SS je index v segmentové tabulce)
1Ch	word	Počet prvků segmentové tabulky
1Eh	word	Počet prvků v module-reference tabulce
20h	word	Počet bytů v nonresident-name tabulce
22h	word	Offset segmentové tabulky (od začátku NE hlavičky)
24h	word	Offset resource tabulky (od začátku NE hlavičky)
26h	word	Offset resident-name tabulky (od začátku NE hlavičky)

28h word Offset module-reference tabulky (od začátku NE hlavičky)
2Ah word Offset imported-name tabulky (pole řetězců, ukončené řetězcem s délkou 0) (od začátku NE hlavičky)
2Ch dword Offset nonresident-name tabulky (od začátku souboru)
30h word Počet přesunovatelných položek ve vstupní tabulce
32h word **File alignment size shift count**
0 is equivalent to 9 (default 512-byte pages)
34h word Počet zdrojů v resource tabulce
36h byte Cílový operační systém
bit 0: neznámý
bit 1: OS/2
bit 2: Windows
bit 3: European MS-DOS 4.x
bit 4: Windows 386
bit 5: BOSS (Borland Operating System Services)

37h byte Doplňkové informace
bit 0: Podporuje dlouhá jména souborů
bit 1: Aplikace pro Windows 2.x, která běží v chráněném módu 3.x
bit 2: Aplikace pro Windows 2.x, která podporuje proporcionální fonty
bit 3: Program obsahuje fast-load prostor
38h word Offset fast-load prostoru (v sektorech od začátku souboru)
3Ah word Délka fast-load prostoru (v sektorech)
3Ch word Rezervováno
3Eh byte Očekávaná verze Windows.

Segmentová tabulka

Položky popisují segmenty použité v programu.

Offset Délka Popis
00h word Offset datového segmentu (v sektorech) (0 - datový segment neexistuje)
02h word Délka segmentu (v bytech) (0 - délka je 64kB)
04h word Obsah EXE
bit 0: datový segment
bit 1: paměť pro segment alokována
bit 2: segment je načten
bit 3: rezervováno
bit 4: segment je typu MOVEABLE (0 - FIXED)
bit 5: segment je typu PURE nebo SHAREABLE (0 - segment je IMPURE nebo NONSHAREABLE)
bit 6: segment je typu PRELOAD (0 - LOADONCALL)
bit 7: pokud je tento bit nastaven a: segment je kódový, pak je to typ EXECUTEONLY, pokud je segment datový, pak je to typ READONLY
bit 8: segment obsahuje relokační data
bit 9: rezervováno
bit 10: rezervováno
bit 11: rezervováno
bit 12: segment je typu DISCARDABLE
bit 13: rezervováno
bit 14: rezervováno
bit 15: rezervováno

06h word Minimální prostor pro alokaci segmentu (v bytech) (0 - prostor je 64 kB)

Tabulka zdrojů (Resource Table)

Tabulka má tento formát:

WORD rscAlignShift; Alignment shift pro data, tato hodnota použitá jako exponent dvou dá faktor (v bytech) pro výpočet pozice zdroje v souboru.
TYPEINFO rscTypes[]; Pole informací o typech zdrojů (jedna položka pro každý typ použitý v programu)
WORD rscEndTypes; Určuje konec definice typů. Musí být nula.
BYTE rscResourceNames[]; Jména zdrojů uložených v souboru, položky jsou uloženy za sebou - první byte položky určuje počet následujících znaků.
BYTE rscEndNames; Určuje konec definice jmen. Musí být nula.

Struktura TYPEINFO

```
typedef struct _TYPEINFO {
    WORD        rtTypeID;
    WORD        rtResourceCount;
    DWORD       rtReserved;
    NAMEINFO    rtNameInfo[];
} TYPEINFO;
```

Položky:
rtTypeID Určuje typ zdroje. Je to buď konstanta typu zdroje (viz níže) nebo offset ve jménu resource-type. Jestliže je nastaven nejvyšší bit (0x8000), typ je určen jednou z konstant:

Hodnota	Typ
RT_ACCELERATOR	Tabulka akcelerátorů
RT_BITMAP	Bitmapa
RT_CURSOR	Kurzor
RT_DIALOG	Dialogový box
RT_FONT	Font
RT_FONTDIR	Adresář fontu
RT_GROUP_CURSOR	Adresář kurzoru
RT_GROUP_ICON	Adresář ikony
RT_ICON	Ikona
RT_MENU	Menu
RT_RCDATA	Uživatelská data
RT_STRING	Tabulka řetězců

Pokud nejvyšší bit není nastaven, hodnota určuje offset

If the high bit of the value in this member is not set, the value represents an offset, in bytes relative to the beginning of the resource table, to a name in the rscResourceNames member.

rtResourceCount Specifies the number of resources of this type in the executable file.
rtReserved Reserved.
rtNameInfo Specifies an array of NAMEINFO structures containing information about individual resources. The
rtResourceCount member specifies the number of structures in the array.

Viz také: MS Windows Software Development Kit 3.1

COM
EXE (DOS)

Internet Shortcut (URL)

Hynek Sládeček

URL je formát používaný ve Windows95 pro záznam odkazu na URL adresu. Umožňuje přímé spuštění prohlížeče a načtení dané stránky/odeslání pošty/apod.

Všechny informace jsou zapsány v textovém formátu.

Hlavička

[InternetShortcut]

Data

URL= (URL)
WorkingDirectory= (pracovní adresář)
ShowCommand= (okno: 1 = normální, 3 = maximalizované, 7 = minimalizované)
IconIndex= (pořadí ikony v souboru)
IconFile= (soubor s ikonou)
HotKey= (decimálně kód:

Řídící klávesa:
Shift 0x0100
Ctrl 0x0200
Alt 0x0400
+
Běžná klávesa:
A 0x0041
B 0x0042
C 0x0043
...

Příklad:
Ctrl + Shift + A
0x0200 + 0x0100 + 0x0041 = 0x0341 = 833)

Příklad:

[InternetShortcut]
URL= http://www.krovina.cz/studna
WorkingDirectory= C:\Windows
ShowCommand= 7
IconIndex=13
IconFile=C:\Windows\System\shell32.dll
HotKey=833

Windows Cabinet File (CAB)

Hynek Sládeček

Struktura souboru CAB:

- Hlavička cabinetu
- Rezervovaný prostor (jestliže je nastaven přepínač CAB_FLAG_RESERVE (flags))
- Jméno předcházejícího cabinetu (jestliže je nastaven přepínač CAB_FLAG_HASPREV)
- Jméno předcházejícího disku (jestliže je nastaven přepínač CAB_FLAG_HASPREV)
- Jméno následujícího cabinetu (jestliže je nastaven přepínač CAB_FLAG_HASNEXT)
- Jméno následujícího disku (jestliže je nastaven přepínač CAB_FLAG_HASNEXT)
- Hlavičky adresářů (podle počtu cFolders)
- Hlavičky souborů (podle počtu cFiles)
- Data souborů (offset: coffCabStart v hlaviččce adresáře)

Hlavička cabinetu

Offset	Velikost	Jméno	Obsah
00h	DWORD	sig	signatura: "MSFC"
04h	DWORD	csumHeader	kontrolní součet (0 - nepoužito)
08h	DWORD	cbCabinet	velikost souboru
0Ch	DWORD	csumFolders	kontrolní součet adresářů (0 - nepoužito)
10h	DWORD	coffFiles	offset prvního záznamu
14h	DWORD	csumFiles	kontrolní součet souborů (0 - nepoužito)
18h	WORD	version	verze (0x0103)
1Ah	WORD	cFolders	počet adresářů
1Ch	WORD	cFiles	počet souborů
1Eh	WORD	flags	přepínače: CAB_FLAG_HASPREV 0x0001 (existuje předchozí cabinet) CAB_FLAG_HASNEXT 0x0002 (existuje další cabinet) CAB_FLAG_RESERVE 0x0004 (existuje rezervovaný prostor)
20h	WORD	setID	identifikační číslo **sady**
22h	WORD	iCabinet;	číslo cabinetu (počínaje 0)

Struktura reserved bloku

00h	DWORD	length	délka rezervovaných dat
04h	length	reserved	rezervovaná data

Jména (předchozího/následujícího cabinetu/disku) jsou řetězce ukončené nulou (znakem číslo 0).

Hlavička adresáře

Offset	Velikost	Jméno	Obsah
00h	DWORD	coffCabStart	offset počátku dat adresáře
04h	WORD	cCFData	???
06h	WORD	typeCompress	typ komprese (podle tcomp* defin. v FDI.H)

Hlavička záznamu

00h	DWORD	cbFile	velikost souboru (nekomprimovaného)
04h	DWORD	uoffFolderStart	offset souboru po dekompresi
08h	WORD	iFolder	id souboru CAB_FILE_FIRST 0x0000 CAB_FILE_NEXT 0x0001 CAB_FILE_SPLIT 0xFFFE CAB_FILE_CONTINUED 0xFFFD
0Ah	WORD	date	datum vytvoření souboru (ve formátu DOSu)

0Ch	WORD	time	čas vytvoření souboru (ve formátu DOSu)		
0Eh	WORD	attribs	atributy souboru		
			CAB_ATTRIB_READONLY	0x0001	
			CAB_ATTRIB_HIDDEN	0x0002	
			CAB_ATTRIB_SYSTEM	0x0004	
			CAB_ATTRIB_VOLUME		0x0008
			CAB_ATTRIB_DIRECTORY	0x0010	
			CAB_ATTRIB_ARCHIVE		0x0020

Windows Icon (ICO)

Hynek Sládeček

Hlavička

Soubor ikony začíná definicí adresáře ikon obsažených v daném souboru.

```
typedef struct {
    WORD           idReserved;
    WORD           idType;
    WORD           idCount;
    ICONDIRENTRY   idEntries[1];
} ICONDIR, *LPICONDIR;
```

idReserved	rezervováno, musí být 0
idType	typ prostředku (pro ikony 1)
idCount	počet ikon v souboru
idEntries	pole hlaviček ikon (pro každou ikonu jeden)

Hlavička ikony

```
typedef struct {
    BYTE  bWidth;
    BYTE  bHeight;
    BYTE  bColorCount;
    BYTE  bReserved;
    WORD  wPlanes;
    WORD  wBitCount;
    DWORD dwBytesInRes;
    DWORD dwImageOffset;
} ICONDIRENTRY, *LPICONDIRENTRY;
```

bWidth	šířka obrázku
bHeight	výška obrázku
bColorCount	počet barev v obrázku (0, pokud je zde 8 nebo více bitů na pixel - 256 barev)
bReserved	rezervováno
wPlanes	počet barevných rovin
wBitCount	počet bitů na pixel
dwBytesInRes	délka tohoto prostředku v bytech
dwImageOffset	offset obrázku od počátku souboru

Data ikony

```
typedef struct {
   BITMAPINFOHEADER icHeader;
   RGBQUAD          icColors[1];
   BYTE             icXOR[1];      // DIB bits for XOR mask
   BYTE             icAND[1];      // DIB bits for AND mask
} ICONIMAGE, *LPICONIMAGE;
```

Struktura **icHeader** je stejná jako u formátu BMP (BITMAPINFOHEADER). Použity jsou však pouze proměnné **biSize**, **biWidth**, **biHeight**, **biPlanes**, **biBitCount**, **biSizeImage** (v tomto popisu BMP jsou to HeaderSize, ImageWidth, ImageHeight, NumberOfImagePlanes, BitsPerPixel, SizeOfBitmap). Ostatní

musí být 0. Proměnná biHeight určuje výšku XOR masky + výšku AND masky.
Tabulka barev **icColors** definuje barvy pro masku XOR. Maska AND je monochromatická, stejných rozměrů jako maska XOR. Formát masky XOR i AND odpovídá formátu bitové mapy, jak je definován u BMP.

Poznámka
Při vykreslování Windows nejprve aplikuje masku AND pomocí bitové operace AND, potom masku XOR pomocí bitové operace XOR.

Windows95 Animated Cursor File (ANI)

Hynek Sládeček

Struktura souboru

"RIFF" {délka souboru}
"ACON"
"LIST" {délka seznamu}
"INFO"
"INAM" {délka řetězce Jméno kurzoru} {Data}
"IART" {délka řetězce Autor} {Data}
"fram"
"icon" {délka dat ikony} {Data}
...
"icon" {délka dat ikony} {Data}
"anih" {délka hlavičky ANI (36 bytů)} {Data}
"rate" {délka bloku prodlev} {Data}
"seq " {délka bloku pořadí} {Data}

Položky v bloku "seq " udávají pořadí zobrazování ikon. Pokud tento blok chybí, jsou zobrazovány popořadě.
Položky v bloku "rate" udávají prodlevu po zobrazení každého obrázku (podle bloku "seq "). Prodleva se uvádí v jednotkách Jiffies. Každý Jiffie znamená 1/60 sekundy.
Počet ikon odpovídá proměnné cFrames z hlavičky. Data ikony jsou ve formátu ICO.

Bloky **"ACON"**, **"anih"**, **"rate"** a **"seq "** mohou být v libovolném pořadí.
Bloky "LIST", "rate" a "seq " jsou nepovinné.
Všechny délky jsou ve 4bytovém formátu (dword).
Počet datových prvků v blocích **"rate"** a **"seq "** odpovídá proměnné cSteps v hlavičce. Prvky jsou typu dword, délka těchto bloků je tedy 4*cSteps.

Hlavička ANI

```
struct tagANIHeader {
    DWORD cbSizeOf;
    DWORD cFrames;
    DWORD cSteps;
    DWORD cx, cy;
    DWORD cBitCount, cPlanes;
    DWORD JifRate;
    DWORD flags;
} ANIHeader;
```

cbSizeOf počet bytů hlavičky AniHeader (36)
cFrames počet ikon v tomto souboru
cSteps počet zobrazovaných obrázků před opakování cyklu (některé ikony mohou být zobrazeny vícekrát - viz blok "seq ")
cx, cy rezervováno, musí být 0
cBitCount rezervováno, musí být 0
cPlanes rezervováno, musí být 0
JifRate pokud chybí blok "rate", udává jednotnou prodlevu mezi obrázky
flags identifikátor prostředku (1)

Formáty MS Office

Hynek Sládeček

Formáty programů balíku MS Office (Word, Excel, Powerpoint, ...) nejsou volně přístupné. Microsoft je poskytuje členům MSDN (MS Developer Network). Jejich získání je však vázáno určitými podmínkami - např. je zakázáno je dále šířit.
Na Internetu se mi podařilo najít popis formátu DOC, ovšem pochybné kvality (http://wwwwbs.cs.tu-berlin.de/~schwartz/pmh/elser/word6/format.html).

Členství v MSDN je zdarma, proto máte pomocí internetu možnost dostat se k originální dokumentaci velice snadno .
V následujících krocích je popsán postup, jak se dostat k popisu formátů:

1. Načtěte adresu MSDN (**http://www.microsoft.com/msdn/library**). Pokud nejste členy MSDN, zaregistrujte se.
2. Klikněte na MSDN Library Online.
3. Klikněte na zarážku Library Online pob Member Area.
4. Poklepejte na Microsoft Office Development.
5. Poklepejte na Office.
6. Poklepejte na Microsoft Office 97 Binary File Formats
7. Vyberte si formát, který vás zajímá.

Windows Resource File (RES)

Hynek Sládeček

Soubor prostředků (.RES) neobsahuje žádnou hlavičku, jen po sobě následující záznamy.
Formát RES souborů je odlišný pro 16-bitová a 32-bitová Windows.

32-bitový formát

Soubor začíná prázdným záznamem o 32 bytech:

00000000 20000000 FFFF0000 FFFF0000 00000000 00000000 00000000 00000000

Každý záznam má hlavičku:

Velikost	Jméno	Obsah
4 byty	DataSize	Délka dat následujících po hlavičce
4 byty	HeaderSize	Délka hlavičky (minimálně 16)
?	Type	Typ prostředku
?	Name	Jméno prostředku
4 byty	DataVersion	Verze formátu pro daný prostředek, většinou 0
2 byty	Flags	Přepínače: Discardable (1000h)
2 byty	Language	identifikátor primárního a sekundárního jazyka. 0 znamená jazykově neutrální, kódy viz dokumentace k Windows. Výsledná hodnota se počítá podle vzorce: (secondary << 10 | primary).
4 byty	Version	Verze prostředku
4 byty	Characteristics	Cokoli

Za hlavičkou následují data záznamu.

Typ a jméno prostředku mohou být textové nebo číselné. Pokud jsou první dva byty FFFFh, následující dva byty jsou číselná hodnota. Jinak jsou první dva byty první znak Unicode nulou ukončeného řetězce.

16-bitový formát

Velikost	Jméno	Obsah
?	Type	Typ prostředku
?	Name	Jméno prostředku
2 byty	Flags	Přepínače: Discardable (1000h), Moveable (0010h), Pure (0020h), Preload (0040h)
4 byty	Size	Délka dat následujících po hlavičce

Za hlavičkou následují data záznamu.

Typ a jméno prostředku mohou být textové nebo číselné. Pokud je první byte FFh, následující dva byty jsou číslení hodnota. Jinak je první byte prvním znakem ANSI řetězce.

Typy prostředků

Předdefinované typy pro 16-bitová Windows:

Hodnota	Typ
1	RT_CURSOR

2	RT_BITMAP
3	RT_ICON
4	RT_MENU
5	RT_DIALOG
6	RT_STRING
7	RT_FONTDIR
8	RT_FONT
9	RT_ACCELERATOR
10	RT_RCDATA
12	RT_GROUP_CURSOR
14	RT_GROUP_ICON

Další předdefinované typy pro 32-bitová Windows:

Hodnota	**Typ**
11	RT_MESSAGETABLE
16	RT_VERSION
17	RT_DLGINCLUDE
19	RT_PLUGPLAY
20	RT_VXD
21	RT_ANICURSOR

Windows Cardfile (CRD)

Petr Říha

Byte	Význam
0 - 2	vždy MGC (4D 47 43)
3 - 4	počet karet v souboru

Za prvními pěti byty jsou hlavičky všech karet v souboru. Informace o první kartě začínají na 5. byte souboru, informace o každé další kartě začínají vždy 34 bytes po začátku předchozí karty (druhý záznam začíná na byte 39, třetí na byte 6D, atd.).
Struktura hlavičky:

0 - 5	rezervováno s možným využitím v budoucnu
6 - 9	absolutní pozice dat karty v souboru
A	vždy nula
B - 32	? (nějaký text)
33	pokud předchozí karta neobsahuje text (viz níže), má tento byte hodnotu 0

Po poslední hlavičce následují samotná data. Mohou být zapsána v jednom ze čtyř formátů:

1. graphic & text
2. text only
3. graphic only
4. blank.

Data ve formátu blank se skládají pouze ze čtyř bytů (všechny 4 mají hodnotu null). Ostatní 3 formáty mají následující strukturu:

Graphic & text	Text only	Graphic only	
0 - 1	0 - 1 #	0 - 1	délka bitmapy #
2 - 3	*	2 - 3	šířka obrázku *
4 - 5	*	4 - 5	výška obrázku *
6 - 7	*	6 - 7	X-ová souřadnice obrázku *
8 - 9	*	8 - 9	Y-ová souřadnice obrázku *
A - x	*	A - x	bitmapa *
x+1 - x+2	2 - 3	x+1 - x+2 #	délka textu #
x+3 - y	4 - z	*	text *

x= délka bitmapy + 9
y= délka textu + x + 2
z= délka textu + 3

- pokud není bitmapa/text, tyto byty mají hodnotu null
* - pokud není bitmapa/text, tyto byty neexistují.

Všechna čísla jsou zapsána v hexadecimálním tvaru.

Windows Registration File (REG)

Hynek Sládeček

Soubory REG slouží k zápisu do globálního registru systému. S touto příponou je tato akce asociována, stačí tedy dvojitě kliknout.
Obsah souborů REG je prostý ASCII text, každý řádek je ukončen CRLF.
Můžete se setkat s dvěma verzemi: pro Windows 3.x a Windows 95/NT. V závislosti na odlišnostech struktury registrační databáze v obou systémech je odlišný také formát souborů REG.

REG ve Windows 3.x

Hlavička
V hlavičce je uveden řetězec "REGEDIT".

Klíče
Platné větve:

```
HKEY_CLASSES_ROOT
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
HKEY_USERS
HKEY_CURRENT_CONFIG
HKEY_DYN_DATA
```

Registrační databáze Windows 3.x nepoužívá jiné hodnoty než (Default).
Data se vkládají ve formátu:

```
klíè = data
```

Data jsou textové řetězce (nezadávají se v uvozovkách)..

Příklad:

```
HKEY_CLASSES_ROOT\.tlk = Talk
HKEY_CLASSES_ROOT\Talk = Talk Voice Annotation
```

REG ve Windows 95/NT

Hlavička
V hlavičce je uveden řetězec "REGEDIT4".

Klíče
Platné větve:

```
HKEY_CLASSES_ROOT
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
HKEY_USERS
HKEY_CURRENT_CONFIG
HKEY_DYN_DATA
```

Každý modifikovaný klíč je uveden v samostatné sekci (v hranatých závorkách), např.

```
[HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Windows]
```

Následují hodnoty a jim přiřazovaná data ve formátu:

```
[klíè]
"hodnota1"=data
"hodnota2"=data
```

Pod každým klíčem může být více modifikovaných/přidávaných hodnot.
Při modifikaci hodnoty (Default) se jako hodnota zadává znak "@":

```
[klíè]
@=data
```

Data
Data mohou být definována ve třech formátech:

- textový řetězec (uvádí se v uvozovkách, v konvencích C, např. "Adresář Windows/nC:\\WINDOWS")
- binární řetězec (uvádí se v seznamu hexadecimálních čísel oddělených čárkami, např. a0,ff,b0,ee
- dword (32-bit integer) (uvádí se hexadecimálně po slově dword s dvojtečkou, např. dword:ffffffa0)

Slovníček

klíč - klíčem se rozumí cesta v hierarchii registrační databáze, př.

- HKEY_CURRENT_CONFIG
- HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Windows

hodnota - hodnotou se rozumí jméno datového pole uvnitř některého klíče, př.

- (Default) - hodnota beze jména
- DisplayName
- UninstallString (pod klíčem HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Windows/CurrentVersion/Uninstall/InternetExplorer)

data - informace uložená pod některou hodnotou, př.

- "InternetExplorer 3.02" (hodnota DisplayName pod klíčem HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Windows/CurrentVersion/Uninstall/InternetExplorer)
- ff ff ff ff (hodnota History_Expire_Days pod klíčem HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/InternetExplorer/Main)

CD Audio File (CDA)

Hynek Sládeček

CDA je jeden z multimediálních formátů Microsoftu, struktura je definována pomocí RIFF (Microsoft Resource Interchange File Format).
Schematicky znázorněná struktura souboru vypadá takto:

"RIFF" {délka souboru}
"CDDA"
"fmt " {délka datového bloku}
{datový blok}

Podrobný popis formátu:

Offset	Délka	Popis
00h	4 byty	identifikátor RIFF "RIFF" (52h 49h 46h 46h)
04h	4 byty	délka zbytku souboru
08h	4 byty	identifikátor CDA "CDDA" (43h 44h 44h 41h)
0Ch	4 byty	identifikátor datového bloku "fmt " (66h 6dh 74h 20h)
10h	4 byty	délka datového bloku (v této verzi 24)
14h	24 byty	datový blok

Datový blok:

Offset	Délka	Popis
00h	2 byty	Verze CDA souborů (nyní 1 - tomu odpovídají následující data).
02h	2 byty	Číslo stopy.
04h	4 byty	Sériové číslo disku (to, které je zloženo v souboru CDPLAYER.INI).
08h	4 byty	Začátek stopy v HSG formátu.
0Ch	4 byty	Délka stopy v HSG formátu.
10h	4 byty	Začátek stopy v Red-Book formátu.
14h	4 byty	Délka stopy v Red-Book formátu.

HSG formát

čas = minut * 4500 + sekund * 75 + frame

Red-Book formát

Offset	Délka	Popis
00h	1 byte	frame
01h	1 byte	sekundy
02h	1 byte	minuty
03h	1 byte	nepoužito

Viz také: CDPLAYER.INI

Windows Help Contents File (CNT)

Hynek Sládeček

Soubor s definicí obsahu HLP souboru. Umožňuje hierarchické uspořádání položek nápovědy.
Soubor CNT musí mít stejné jméno jako HLP soubor (např. myhelp.hlp + myhelp.cnt).
Textový soubor ve formátu ASCII, řádky jsou ukončeny CRLF.

Hlavička

```
:Base jméno_souboru.hlp>okno
:Title Nadpis v dialogovém oknì s obsahem
```

Parametr ">okno" není povinný. Pokud nebude uvedeno, zobrazí se nápověda v běžném okně.
Pokud je obsah určen pro odkazy na témata z více HLP souborů, následuje jejich výčet:

```
Index: jméno_souboru.hlp
Index: jméno_souboru2.hlp
Index: jméno_souboru3.hlp
...
Index: jméno_souborux.hlp
```

Struktura

Za obsahem následují položky seznamu ve formátu:

```
èíslo_úrovnì Popis tématu=context_string_tématu@jméno_souborux.hlp
```

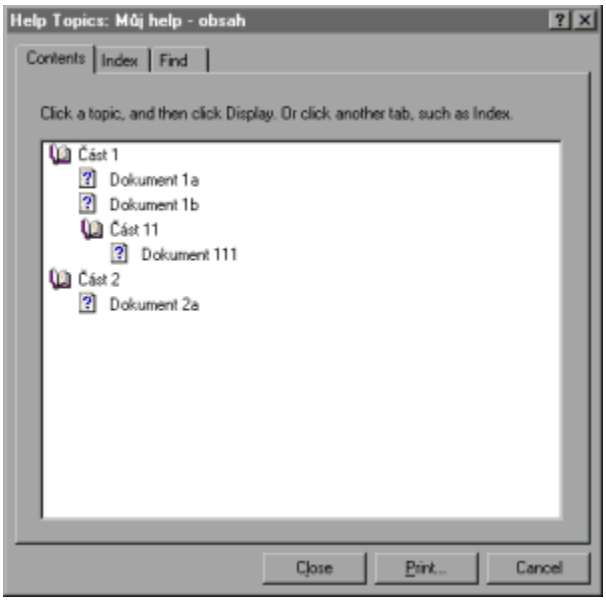
Číslo úrovně definuje umístění položky v seznamu (1= nejvyšší úroveň, další jsou podřazené).
Parametr "@jméno_souborux.hlp" se neuvádí v případě odkazu do souboru uvedeného v :Base.
Pokud není uveden ani parametr "=context_string_tématu", položka bude zobrazena s ikonou knihy a další položky s vyšším číslem úrovně budou spadat pod tuto položku.

Příklad:

```
:Base myhelp.hlp
:Title Mùj help - obsah

1 Èást 1
2 Dokument 1a=dok1
2 Dokument 1b=dok2
2 Èást 11
3 Dokument 111=dok1c1
1 Èást 2
2 Dokument 2a=dok2a
```

Bude interpretováno jako:



Řádky bez čísla úrovně jsou ignorovány.
Pokud je CNT soubor u souboru HLP ve formátu pro Windows 3.x, bude obsah ve Windows 95 zobrazen.

Microsoft Palette (PAL)

Hynek Sládeček

Struktura souboru

"RIFF" {délka souboru}
    "PAL "
    "data" {velikost dat}
    {Data}

Délka souboru je word (nezahrnuje se identifikátor "RIFF" a samotná informace, tedy délka souboru mínus 8 bytů).

Data

word	Version	Číslo verze - mělo by být 0x300 pro Windows 3.0 a pozdější
word	NumEntries	Počet definovaných barev.

Následují záznamy o barvách (podle počtu definovaných barev).

Záznam barvy

byte	Red
byte	Green
byte	Blue
byte	Flag

Red, Green, Blue jsou indexy RGB.
Flag může nabývat následujících hodnot:

0x00 Záznam obsahuje informace o složkách RGB (normální).
0x01 Barva bude použita pro animaci. Tento záznam nebude použit pro přiřazování dalších barev. Barva bude aplikována pouze pokud je v systémové paletě volný záznam.
0x02 Nižší word záznamu barvy (byty Red + Green) určuje index barvy v hardwarové paletě.
0x04 Barva bude při aplikace palety vložena do nepoužitého záznamu systémové palety (místo aby byla přiřazena existující barvě v systémové paletě). Pokud zde žádný prázdný záznam není, bude barva aplikována normálně (viz závorka).

Po posledním záznamu následují čtyři byty mně neznámých informací.
Informace počínaje položkou Data odpovídají struktuře LOGPALETTE v Borland C++ (kromě posledních 4 bytů). Hodnoty v bytu Flag připadají v úvahu při použití palety pro funkce Windows API.

Creative Voice File (VOC)

Bronislav Křístek

Jde o výtvor firmy Creative Labs a svého času asi o nejrozšířenější zvukový formát pod DOSem.

Záhlaví

Na začátku souboru je umístěno záhlaví. Má následující strukturu:

BAJT POPIS

0..19 Obsahuje identifikační řetězec "Creative Voice File" ukončený znakem #26.
20..21 Hodnota typu word - začátek dat. Obvykle hodnota 26.
22..23 Hodnota typu word - verze souboru. Ve vyšším byte je číslo před desetinnou tečkou, v nižším byte číslo verze za desetinnou tečkou.
24..25 Hodnota typu word - číslo verze souboru komplementárně.

Od pozice 26 v souboru začínají jednotlivé datové bloky. První bajt v bloku udává typ bloku, pak většinou následují další bajty, jejichž význam závisí na druhu tohoto bloku. V současné době existuje celkem 10 druhů datových bloků a verze souboru 1.20.

Druh bloku 0 - terminátor

Obsahuje pouze tento jediný bajt s hodnotou 0 a nalézá se zásadně na konci souboru.

Druh bloku 1 - Voice Data

Základní datový blok, obsahuje údaje nutné pro přehrávání vzorku a pak vlastní zvuková data.
Složení bloku 1:
Identifikátor - bajt s hodnotou 1
Délka bloku - tři bajty. První bajt má nejnižší význam, třetí bajt má nejvyšší význam. Použitelná hodnota se vypočítá takto: 1.bajt + 2.bajt * 256 + 3.bajt * 65536. **Stejným způsobem se počítá délka bloku i ve všech ostatních druzích bloků, pokud tyto příslušné tři bajty obsahují.** V takto zjištěné hodnotě je zde zahrnuta jak délka vlastních zvukových dat vyjádřená bajtech, tak i následující dva bajty udávající konstantu pro vzorkovací frekvenci a komprimaci.
TC - bajt obsahující časovou konstantu ve formě předávané DSP SoundBlastru pomocí příkazu 40h. Vztah mezi TC a vzorkovací frekvencí lze vyjádřit takto:

TC = 256 - (1 000 000 / frekvence)

Jedná se samozřejmě o MONO mód. V době vzniku formátu VOC ještě zvukové karty stereo neovládaly.
Pack - bajt udávající druh komprimace vzorku.

0: vzorky 8bitové nekomprimované
1: komprimace na 4 bity
2: komprimace 2,6 bitů
3: komprimace na 2 bity

Voice Data - vlastní zvuková data. Jejich délka v byte se rovná délce bloku zmenšené o 2.

Druh bloku 2 - Voice Continuation

Vyskytuje se v případě, že zvuková data jsou tak dlouhá, že se všechna nevešla do bloku 1.
Složení bloku 2:
Identifikátor - bajt s hodnotou 2.
Délka Bloku - tři bajty. Výpočet stejný jako u bloku 1.
Voice Data - zvuková data. Jejich délku udává předchozí délka bloku.

Druh bloku 3 - Silence

Blok se postará o to, aby byla zvukovka chvíli zticha. U SoundBlastru se dá využít příkaz 40h pro nastavení vzorkovací frekvence a příkaz 80h pro nastavení ticha následovaný dvěma datovými bajty

udávajícími počet vzorků ticha. Na konci ticha vyšle SB přerušení stejné jako na konci přenosu pomocí DMA.
Složení bloku 3:
Identifikátor - bajt s hodnotou 3.
Délka bloku - tři bajty. Délka bloku by měla být rovna třem.
Periody - 2 bajty. Udávají počet vzorků (period) ticha. Výpočet = první bajt + druhý bajt * 256.
TC - jeden bajt. Význam stejný jako u bloku 1.

Druh bloku 4 - Marker
Blok se dá využít jako značka pro synchronizaci obrazu a zvuku.
Složení bloku 4:
Identifikátor - bajt s hodnotou 4.
Délka bloku - tři bajty. Délka bloku je zde vždy rovna hodnotě dvě.
Označení - dva bajty. Nejprve je nižší bajt, pak vyšší bajt. Možné hodnoty jsou 1 až FFFFh.

Druh bloku 5 - ASCII Text
V tomto bloku může být uložen libovolný text. Délka bloku obsahuje délku textu včetně ukončovacího znaku 0h.
Složení bloku 5:
Identifikátor - bajt s hodnotou 5.
Délka bloku - tři bajty.
Text - vlastní text. I přesto, že na vyjádření délky bloku jsou vyhrazeny i zde 3 bajty, dá se předpokládat, že délka textu nebude příliš dlouhá.

Druh bloku 6 - Repeat Loop
Označuje začátek části souboru, která se má přehrávat vícekrát po sobě.
Složení bloku 6:
Identifikátor - bajt s hodnotou 6.
Délka bloku - tři bajty. Délka bloku je zde vždy rovna hodnotě dvě.
Čítač - dva bajty. Nejprve nižší bajt, pak vyšší bajt.

Druh bloku 7 - End Repeat Loop
Označuje konec části souboru, která začínala blokem číslo 6.
Složení bloku 7:
Identifikátor - bajt s hodnotou 7.
Délka bloku - tři bajty s hodnotou nula.

Druh bloku 8 - Extended Block
Tento druh bloku přibyl později s tím, jak se objevily první stereo karty. Pokud se v souboru vyskytuje, bývá umístěn vždy před blokem číslo 1. Parametry obsažené v bloku číslo 1 pak nejsou platné (kromě délky zvukových dat), je třeba se řídit pouze parametry uvedenými v bloku 8.
Složení bloku 8:
Identifikátor - bajt s hodnotou 8.
Délka bloku - tři bajty. Délka bloku je zde vždy rovna hodnotě čtyři.
TC - dva bajty. Nejprve nižší bajt, pak vyšší bajt. Zde je rozdíl proti bloku číslo 1. Tam TC (časová konstanta) zabírala pouze jeden bajt, tady dva. Vztah mezi TC a vzorkovací frekvencí se dá vyjádřit vzorcem TC = 65 536 - (256 000 000 / n * frekvence) , kde n udává počet kanálů (MONO = 1, STEREO = 2). Jak se ale vypořádat s faktem, že SoundBlaster umí přijmout TC pouze jako jednobajtovou hodnotu? Úplně jednoduše - hodnota v tom tvaru, jak ji potřebuje SoundBlaster je zde obsažena ve vyšším bajtu TC a je stejná jako hodnota TC uvedená v bloku číslo 1.

Pack - jeden bajt udávající druh komprimace stejně jako tomu je v bloku 1. Zde by se měl rovnat vždy nule, tedy nekomprimované osmibitové vzorky.
Mode - jeden bajt. 0 = mono, 1 = stereo.

Druh bloku 9 - ???

Nejnovější druh bloku. Přibyl se šesnáctibitovými kartami. Pokud se v souboru vyskytuje, není v souboru ani blok 1, ani blok 8. Na rozdíl od předchozích druhů bloků, jejichž popis se běžně vyskytuje ve volně šiřitelných zdrojích, popis bloku číslo 9 jsem nikde nenašel. Proto neznám ani jeho název a rovněž údaje o jeho složení jsou jen částečné - získal jsem je "pitvou" několika VOC souborů, které jsem vytvořil nahrávacím programem dodaným výrobcem k mému SB16.
Složení bloku 9:
Identifikátor - bajt s hodnotou 9.
Délka bloku - tři bajty.
Frekvence - dva bajty udávající tentokrát ne TC, ale přímo vzorkovací frekvenci. Nejprve nižší bajt, pak vyšší bajt. SB16 totiž umí kromě příkazu 40h pro nastavení TC také příkaz 41h pro přímé nastavení vzorkovací frekvence.
??? - v mých souborech vždy dva nulové bajty neznámého významu.
8/16 bit - jeden bajt. V případě osmibitových vzorků obsahoval hodnotu 8, v případě šesnáctibitových vzorků obsahoval hodnotu 16.
Mode - jeden bajt. V případě monofonních vzorků obsahuje hodnotu 1, v případě stereofonních vzorků obsahuje hodnotu 2.
??? - jeden bajt. Význam neznámý. V případě osmibitových vzorků ať už mono nebo stereo byl vždy nulový, v případě šesnáctibitových vzorků obsahoval vždy hodnotu 4.
??? - dalších 5 bajtů neznámého významu, v mých souborech byly všechny nulové.
Voice data - vlastní zvuková data. Jejich délka vyjádřená v bajtech = délka bloku - 12.

Nakonec uvádím ukázku programu, který na obrazovku vypisuje složení VOC souboru. Program je kvůli délce oprostěn od různých parádiček a kontrol úspěšného otevření souboru atp...

Soubor voc.pas

```
PROGRAM Rozbor;
{* Rozbor souboru *.VOC se zvukovymi daty. *}
USES CRT;
CONST
  hotovo:boolean = false;
TYPE
  Tznaky = array[0..65528] of char;

VAR
  p_znaky: ^Tznaky;
  fr:file;              {* K souboru bude pristupovano jako k netypovemu *}
  jmenosoub:string[12];
  PozVSoub:longint;     {* urceni pozice v souboru *}
  BlokSize: longint;    {* delka nasledujiciho datoveho bloku *}
  frekvence : word;     {* Vzorkovaci frekvence *}
  TimeConst:byte;
  pom: byte;            {* hodnota bajtu nacteneho ze souboru *}
  w: word;
  ID_Text: string[20];  {* Identifikacni retezec VOC souboru *}

FUNCTION GetBlockSize: longint;
{* Funkce vraci velikost aktualniho bloku *}
VAR
```

```
  b:byte;
  k:longint;
begin
  BlockRead(fr,b,1);
  k:= b;
  BlockRead(fr,b,1);
  k:= k + longint(b) * 256;
  BlockRead(fr,b,1);
  GetBlockSize:= k + Longint(b) * 65536;
end;

BEGIN   {***************  HLAVNI  PROGRAM  ****************}
  ClrScr;
  write('Zadej jmeno souboru: '); readln(jmenosoub);
  assign(fr,jmenosoub);
  reset(fr,1);
  {* Nacteni identifikacniho retezce *}
  ID_Text[0]:=char(20);
  BlockRead(fr,ID_Text[1],20);
  writeln('HLAVICKA SOUBORU');
  writeln('Identifikacni retezec souboru = ',ID_Text);
  if ID_Text <> 'Creative Voice File' + #26 then
    begin
      writeln('Tento retezec neodpovida identifikacnimu retezci VOC
formatu!');
      write('Po stisku ENTER se program ukonci ');
      close(fr);
      readln;
      halt;
    end;

  BlockRead(fr,w,2);
  writeln('Zacatek bloku dat na pozici ',w);

  Blockread(fr,w,2);
  writeln('Cislo verze souboru = ',Hi(w),'.',Lo(w));

  BlockRead(fr,w,2);
  writeln('Cislo verze komplementarne = ',w);
  writeln; writeln('DATOVE BLOKY');

  repeat
      BlockRead(fr,pom,1);
      case pom of
        0: begin
             writeln('Typ bloku 0 = TERMINATOR');
             writeln('Toto je konec VOC souboru.');
             hotovo:= true;
           end;
        1: begin
             writeln('Typ bloku 1 = VOICE DATA');
             BlokSize:= GetBlockSize;
             writeln('Delka nasledujiciho bloku = ',BlokSize,' bajtu.');
             BlockRead(fr,pom,1);
             write('Hodnota Time Constant = ',pom);
             frekvence:= 1000000 DIV (256-pom);
             writeln(', coz odpovida frekvenci ',frekvence,'Hz v MONO modu.');
```

```
            BlockRead(fr,pom,1);
            write('Hodnota Pack = ',pom,', data jsou ');
            case pom of
              0: writeln('osmibitova nekomprimovana.');
              1: writeln('ctyrbitove komprimovana.');
              2: writeln('2,6 bitove komprimovana.');
              3: writeln('dvoubitove komprimovana.');
            end;
            writeln('Vlastni delka zvukovych dat je ',BlokSize-2,' bajtu.');
            PozVSoub:= FilePos(fr);
            Seek(fr,PozVSoub + BlokSize - 2);
          end;
        2: begin
            writeln('Typ bloku 2 = VOICE CONTINUATION');
            BlokSize:= GetBlockSize;
            writeln('Nasleduji zvukova data o delce ',BlokSize,' bajtu.');
            PozVSoub:= FilePos(fr);
            Seek(fr,PozVSoub + BlokSize);
          end;
        3: begin
            writeln('Typ bloku 3 = SILENCE');
            BlokSize:= GetBlockSize;
            writeln('Delka bloku 3 = ',BlokSize,' bajtu');
            BlockRead(fr,pom,1);
            w:= pom;
            BlockRead(fr,pom,1);
            w:= w + word(pom)*256;
            write('Nasleduje ',w,' period ticha ');
            BlockRead(fr,pom,1);
            writeln(', Time Const = ',pom);
          end;
        4: begin
            writeln('Typ bloku 4 = MARKER');
            BlokSize:= GetBlockSize;
            writeln('Delka bloku = ',BlokSize,' bajty.');
            BlockRead(fr,pom,1);
            w:= pom;
            BlockRead(fr,pom,1);
            w:= w + word(pom)*256;
            writeln('Znacka cislo ',w);
          end;
        5: begin
            writeln('Typ bloku 5 = ASCII TEXT');
            BlokSize:= GetBlockSize;
            writeln('Delka bloku ',BlokSize,' znaku vcetne ukoncovaciho znaku
0h.');
            getmem(p_znaky,BlokSize);
            BlockRead(fr,p_znaky,BlokSize);
            writeln('Na dalsim radku nasleduje text:');
            for w:= 0 to BlokSize-2 DO write(p_znaky^[w]);
            writeln;
            FreeMem(p_znaky,BlokSize);
          end;
        6: begin
            writeln('Typ bloku 6 = REPEAT LOOP');
            BlokSize:= GetBlockSize;
            writeln('Delka bloku = ',BlokSize,' bajty.');
```

```
          BlockRead(fr,pom,1);
          w:= pom;
          BlockRead(fr,pom,1);
          w:= w + word(pom)*256;
          writeln('Tento blok se bude opakovat pri prehravani ',w,' krat');
        end;
      7: begin
          writeln('Typ bloku 7 = END REPEAT LOOP');
          BlokSize:= GetBlockSize;
          writeln('Delka bloku = ',BlokSize,' bajtu.');
        end;
      8: begin
          writeln('Typ bloku 8 = EXTENDED BLOCK');
          BlokSize:= GetBlockSize;
          writeln('Delka bloku = ',BlokSize,' bajtu');
          BlockRead(fr,pom,1);
          w:=pom;
          BlockRead(fr,pom,1);
          w:= w+ word(pom)*256;
          writeln('Time Constant = ',w,' vyssi byte = ',Hi(w));
          frekvence:= 256000000 DIV (65536-w);
          write('Pro MONO mod to odpovida frekvenci ',frekvence,'Hz');
          frekvence:= 128000000 DIV (65536-w);
          writeln(', pro STEREO mod frekvenci ',frekvence,'Hz.');
          BlockRead(fr,pom,1);
          write('Pack hodnota = ',pom,', coz odpovida datum ');
          case pom of
               0: writeln('8bitovym nekomprimovanym.');
               1: writeln('komprimovanym na 4 bity.');
               2: writeln('komprimovanym 2,6 bitove');
               3: writeln('komprimovanym na 2 bity.');
          end;
          BlockRead(fr,pom,1);
          if pom = 0 then writeln('Vzorky jsou MONO.')
            else writeln('Vzorky jsou STEREO.');
        end;
      9: begin
          writeln('Typ bloku 9 = ????');
          BlokSize:= GetBlockSize;
          writeln('Velikost bloku je ',BlokSize,' bajtu.');
          BlockRead(fr,pom,1);
          w:= pom;
          BlockRead(fr,pom,1);
          w:= w+ word(pom)*256;
          writeln('Vzorkovaci frekvence je ',w,' Hz');
          BlockRead(fr,w,2);
          BlockRead(fr,pom,1);
          if pom=8 then write('Vzorky 8bitove ')
            else if pom=16 then write('Vzorky 16bitove ');
          BlockRead(fr,pom,1);
          if pom=1 then writeln('MONO')
            else if pom=2 then writeln('STEREO');
          PozVSoub:= FilePos(fr);
          Seek(fr,PozVSoub+BlokSize-6);
        end;
    end;
  until hotovo;
```

```
  close(fr);
  writeln('To byl cely soubor, stiskni enter');
  readln;
END.
```

Text602 (602)

Marek Jurák

Hlavička súboru formátu T602 ver. 3.0 (platí aj pre ver. 3.1):

Poznámka: Ak nie je napísané, tak "n" je celé číslo.

@CT n, kde n = 0 (ak vstupno-výstupný kód = 1), 1 (ak vstupno-výstupný kód = 2), 2 (ak vstupno-výstupný kód = 3)
@LM n, kde n = ľavý okraj
@RM n, kde n = pravý okraj
@PL n, kde n = počet riadkov na stranu
@TB môžu byť znaky "-" a "T", kde "T" je umiestnenie tabelátora
@MT n, kde n = horný okraj
@MB n, kde n = dolný okraj
@PO n, kde n = ľavý stĺpec (pri tlači)
@PN n, kde n = číslovanie od (nemusí byť žiadne "n")
@OP n, kde n = číslovať stánky (nie)
@PG n, kde n = číslovať stánky (áno)

Telo súboru (v texte):

@LH n, kde n = riadkovanie (n = 6 alebo n = 4 alebo n = 3) (označuje sa len začiatok zmeny riadkovania)
@HE n, kde n = záhlavie hore (n = text)
@FO n, kde n = záklavie dole (n = text)
@PI n, kde n = vloženie obrázku
@PA n, kde n = tvrdý koniec stránky

Ďalej v texte môžu byť aj **riadiace kódy** (prvé - začiatok, druhé - koniec):

1 - Elite
2 - Tučné
3 - Condens
4 - Kurzíva
14 - Pica
15 - Široké
16 - Vysoké
19 - Podčiarknuté
20 - Horný index
22 - Dolný index
29 - Veľké

10 - Nový riadok
11 - Obojstranný formát
13 - Tvrdý CR
141 - Mäkký CR
173 - Mäkké delenie
254 - Tvrdá medzera

AUTORUN.INF

Hynek Sládeček

Program: Microsoft Windows 95
Umístění: Kořenový adresář disku

Pokud je soubor AUTORUN.INF umístěn v kořenovém adresáři CD-ROM, při vložení do mechaniky se ve Windows 95 automaticky spustí program "aplikace.exe" a disk bude zobrazován s ikonou "ikona.ico".
Volbu automatického spouštění vloženého CD-ROM lze uživatelsky vypnout.
Funguje také s pevnými disky.

Formát

Textový soubor ve formátu ASCII, řádky jsou ukončeny CRLF.

```
[autorun]
open=aplikace.exe
icon=ikona.ico, poøadí ikony
položka=nová hodnota
```

Kromě ikony a automaticky spouštěného programu je možné definovat také další položky v menu příslušného disku prostřednictvím úpravy položek v registrační databázi.

`položka` určuje relativní cestu v HKEY_CLASSES_ROOT\AutoRun\číslo disku\

Vlastnosti disku se definují stejným způsobem jako vlastnosti jednotlivých typů souborů (podle přípon).

Struktura položek:

`DefaultIcon\(Default)`
Jméno souboru s ikonou, index ikony.

`Shell\(Default)`
Jméno položky v seznamu, která bude spuštěna při dvojím kliknutí. Pokud žádné nebude uvedeno, bude pužito autorun.

`Shell\jméno\(Default)`
Řetězec, který se zobrazí v menu aktivovaným pravým tlačítkem (pokud před písmenem &, bude aktivní - podtrženo).

`\Shell\jméno\command\(Default)`
Příkaz, který se má provést.

jméno je libovolné (určuje akci). Pro některá jména jsou položky v menu předdefinované, nemusí se uvádět v S`hell\jméno\(Default)`:

autorun Standardní akce pro disky.
open Standardní akce pro soubory.
print Tisk.

Položky open a icon odpovídají těmto relativním cestám:

```
icon  DefaultIcon\(Default)
open  Shell\AutoRun\command\(Default)
```

CDPLAYER.INI

Hynek Sládeček

Program: CD Player (součást Windows 95)
Umístění: Adresář Windows

Soubor pro záznam informací o audio CD (např. autor, jméno CD, jména skladeb).

Formát

Textový soubor ve formátu ASCII, řádky jsou ukončeny CRLF.
Nemá hlavičku.
Následují záznamy.

Záznam

```
[id]
EntryType=1
artist=autor
title=název CD
numtracks=poèet skladeb
0=jméno první skladby
1=jméno druhé skladby
...
order=poøadí pøehrávání skladeb (decimální èísla oddìlená mezerami)
numplay=poèet pøehrávaných skladeb
```

id - hexadecimálně sériové číslo CD (viz CDA). Pro identifikaci CD.

LOGO.SYS

Hynek Sládeček

Program: Microsoft Windows 95
Umístění: Kořenový adresář disku s Windows

Úvodní obrazovka při startu Windows 95. Pokud je tento soubor uložen v kořenovém adresáři disku s Windows (např. C:\), nahrazuje původní obrázek (logo Windows 95).

Formát:

Přejmenovaný BMP soubor.
Rozměry obrázku: 320x400.
Barevná hloubka: 256 barev (8-bit).

LOGOS.SYS

Hynek Sládeček

Program: Microsoft Windows 95
Umístění: Adresář Windows

Obrazovka zobrazovaná při ukončování Windows 95. V anglické verzi obsahuje oranžový text "Now you can turn off your computer." (v české "Nyní můžete počítač bez obav vypnout.").
Soubor je umístněn v adresáři Windows 95 (např. C:\Windows). Narozdíl od souboru LOGO.SYS je zde uložena originální obrazovka (soubor existuje).

Formát:

Přejmenovaný BMP soubor.
Rozměry obrázku: 320x400.
Barevná hloubka: 256 barev (8-bit).

LOGOW.SYS

Hynek Sládeček

Program: Microsoft Windows 95
Umístění: Adresář Windows

Obrazovka zobrazovaná při ukončování Windows 95. V anglické verzi obsahuje text "Please wait while your computer shuts down." na pozadí modrých oblak.
Soubor je umístněn v adresáři Windows 95 (např. C:\Windows). Narozdíl od souboru LOGO.SYS je zde uložena originální obrazovka (soubor existuje).

Formát:

Přejmenovaný BMP soubor.
Rozměry obrázku: 320x400.
Barevná hloubka: 256 barev (8-bit).

WINHELP.INI

Hynek Sládeček

Program: Windows Help
Umístění: Adresář Windows

Definuje adresáře souborů WinHelpu.

Formát

Textový soubor ve formátu ASCII, řádky jsou ukončeny CRLF.

```
[files]
soubor=adresáø, hlášení
soubor=adresáø, hlášení
...
```

soubor Jméno souboru: HLP (soubor nápovědy) nebo DLL (knihovna pro HLP soubor)
cesta Specifikace adresáře, ve kterém se soubor nalézá. Ve Windows 95 mohou být použita dlouhá jména.
hlášení Zpráva zobrazovaná při absenci souboru v definovaném adresáři.

Příklad:

```
[files]
MYHELP.HLP=C:\PROG, Soubor MYHELP.HLP chybí.
MYLIB.DLL=C:\PROG, Nemohu nalézt knihovnu MYLIB.DLL.
```

WININIT.INI

Hynek Sládeček

Program: Microsoft Windows 95
Umístění: Adresář Windows

Umožňuje přejmenovat nebo vymazat soubor při startu Windows 95.
Po provedení (zobrazí se text o instalaci), je soubor přejmenován na WINHELP.BAK.

Formát

Textový soubor ve formátu ASCII, řádky jsou ukončeny CRLF.

```
[rename]
nový_soubor=starý_soubor
```

`nový_soubor` Kompletní cesta k novému souboru. Pokud bude nový soubor "NUL", bude starý soubor smazán.
`starý_soubor` Kompletní cesta ke starému souboru.

Tímto způsobem je možné soubor (nebo adresář) přejmenovat, přesunout nebo smazat.
U dlouhých jmen je nutné použít jejich krátkých ekvivalentů pro DOS (např. C:\PROGRA~1 místo C:\Program Files)

Příklad:

```
[Rename]
NUL=C:\WINDOWS\MYSETUP.EXE
C:\PROG\NEWFILE.TXT=C:\PROG\OLD\FILE.TXT
NUL=C:\PROG\OLD
```

Kromě sekce [rename] se ve WININIT.INI mohou objevit také sekce [CombineVxDs] a [SetupOptions].

INILib 1.01

Hynek Sládeček

Knihovna pro správu souborů INI pro C++ (ve verzi 1.01 pouze čtení). Freeware.

Funkce

```
char *ReadItemName(FILE *fhandle,
            const char *header,
            const char *item,
            char *str,
            int length);
```

- čte položku INI souboru na základě jména položky
- je potřeba mít otevřený soubor f (textový mód)
- žádaný řetězec bude uložen do proměnné na adresu str, stejná adresa je funkcí vrácena
- chybové kódy: v případě neúspěchu vrací NULL

Parametry:
[**header**]
item=str

fhandlehandle otevřeného souboru (v textovém módu)
length maximální délka hledaného řetězce

```
char *ReadItemOrder(FILE *fhandle,
            const char *header,
            int order,
            char *str,
            int length);
```

- čte položku INI souboru na zákadě pořadí položky
- je potřeba mít otevřený soubor f (textový mód)
- žádaný řetězec bude uložen do proměnné na adresu str, stejná adresa je funkcí vrácena
- chybové kódy: v případě neúspěchu vrací NULL

Parametry:
[**header**]
str1
str2
str3

fhandlehandle otevřeného souboru (v textovém módu)
order pořadí řádku (počínaje 0)
length maximální délka hledaného řetězce

Soubor inilib.h

```
// INILib 1.01 (header)
// správa souborù INI
// ----------------------
// © 1997, 1998 Hynek Sládeèek
```

```
#ifndef INILIB
#define INILIB

char *ReadItemName(FILE *fhandle, const char *header, const char *item, char *str, int
length);
char *ReadItemOrder(FILE *fhandle, const char *header, int order, char *str, int length);

#elsif
```

Soubor inilib.cpp

```
// INILib 1.01
// správa souborù INI
// ----------------------
// © 1997, 1998 Hynek Sládeèek

#include <stdio.h>
#include <stdlib.h>
#include "openpdb.h"

char *ReadItemName(FILE *fhandle, const char *header, const char *item, char *str, int
length);
char *ReadItemOrder(FILE *fhandle, const char *header, int order, char *str, int length);

#define cutspace(f,c) while((c=getc(f))==' ')

int SetPosHeader(FILE *f, const char *h);  //nastavi pozici na novy radek po hlavicce v h
int SetPosItem(FILE *f, const char *i);
char *reads(char *str, int l, FILE *f);
int strcmp(const char *s1, const char *s2);
int strlen(const char *s);


char *ReadItemName(FILE *fhandle, const char *header, const char *item, char *str, int
length)
{
  if(SetPosHeader(fhandle,header))
    return NULL;             // nenalezena hlavicka

  if(SetPosItem(fhandle,item))     // nenalezeno jmeno
    return NULL;

  return reads(str,length,fhandle);
}


char *ReadItemOrder(FILE *fhandle, const char *header, int order, char *str, int length)
{

  if(SetPosHeader(fhandle,header))
    return NULL;             // nenalezena hlavicka

  while((str[0]!='[')
      &&(order-->=0))
    reads(str,length,fhandle);

  return str;
}


// ******************** pomocné funkce ************************

char *reads(char *str, int l, FILE *f)
{
  int i=0;
  char c;

  while(((c=getc(f))!='\n')&&(c!=EOF)&&(i<l))
    str[i++]=c;
```

```
  str[i]='\0';

  if(i==l){
    while((c!='\n')&&(c!=EOF))
      c=getc(f);
  }
  return str;
}

int SetPosHeader(FILE *f, const char *h)
{
  register int i=0;
  char str[81];
  int c;
  char end=0;

  fseek(f,0,SEEK_SET);
  while(!end) {
    while((c=getc(f))!='[')  // najde dalsi hlavicku
      if(c==EOF)
       return 1;

    while((c=getc(f))!=']') { // zkopiruje hlavicku do str
      if(c==EOF)               // kbyubychom narazili na konec souboru, chyba
       return 1;

      str[i]=c;
      i++;
    }
    str[i]='\0';         // ukonci retezec
    i=0;

    if(!strcmp(str,h)) {  // jestlize je to hlavicka
      while((c=getc(f))!='\n')  // docte do konce radku a tim padem nastavi na
        if(c==EOF)
          return 1;                   // prvni znak dalsi radky
      return 0;             // vse v poradku
    }
  }
  return 1;
}

int SetPosItem(FILE *f, const char *i)
{
  int c;
  char str[21];
  char n=0;

  cutspace(f,c);
  while((c!='[')&&(c!='\n')&&(c!=EOF)) {
    while((c!='=')&&(c!='\n')&&(c!=EOF)) {
      str[n]=c;
      n++;
      c=getc(f);
    }
    str[n]=0;
    n=0;

    if(!strcmp(str,i))
      return 0;

    while((c!='\n')&&(c!=EOF))
      c=getc(f);
    cutspace(f,c);
  }
  return 1;

}

int strcmp(const char *s1, const char *s2)
{
```

```
  register int i;

  if (strlen(s1)!=strlen(s2))
    return -1;

  for(i=0;i<strlen(s1);i++)
    if(s1[i]!=s2[i])
      return -1;

  return 0;
}

int strlen(const char *s)
{
  int n=0;

  while(s[n]!='\0')
    n++;

  return n;
}
```

SHOW_PCX.PAS

```
{$R-}    {Range checking off}
{$B-}    {Boolean complete evaluation off}
{$S-}    {Stack checking off}
{$I+}    {I/O checking on}
{$N-}    {No numeric coprocessor}

program show_pcx;

{*************************************************************************}
{                                                                         }
{ SHOW_PCX is an example program written in Borland's Turbo Pascal(R) 5.0.  }
{ (Turbo Pascal is a registered trademark of Borland International, Inc.)   }
{ SHOW_PCX doesn't use any of the graphics routines built into Turbo Pascal, }
{ since many programmers won't be using Pascal for their final program.     }
{                                                                         }
{                             PERMISSION TO COPY:                         }
{                                                                         }
{          SHOW_PCX -- (C) Copyright 1989 ZSoft, Corporation.             }
{                                                                         }
{ You are licensed to freely copy SHOW_PCX and incorporate it into your     }
{ own programs, provided that:                                            }
{                                                                         }
{  IF YOU COPY SHOW_PCX WITHOUT CHANGING IT:                              }
{  (1) You must retain this "Permission to Copy" notice, and              }
{  (2) You must not charge for the SHOW_PCX software or                   }
{      documentation; however, you may charge a service fee for           }
{      disk duplication and distribution, so long as such fee is          }
{      not more than $5.00.                                               }
{                                                                         }
{  IF YOU MODIFY SHOW_PCX AND/OR INCORPORATE SHOW_PCX INTO YOUR OWN PROGRAMS }
{  (1) You must include the following acknowledgment notice in the        }
{      appropriate places:                                                }
{                                                                         }
{      Includes portions of SHOW_PCX.                                     }
{      Used by permission of ZSoft Corporation.                           }
{                                                                         }
{                                                                         }
{ ZSoft Corporation reserves all rights to SHOW_PCX except as stated herein. }
{                                                                         }
{                                                                         }
{                  [END OF "PERMISSION TO COPY" NOTICE]                   }
{                                                                         }
{ This program reads a PC Paintbrush PCX file and shows it on the screen.   }
{ The picture must be a 2 color CGA, 4 color CGA, or a 16 color EGA picture. }
{ The picture will be displayed until a key is pressed.                   }
{                                                                         }
{ This program can be run at the DOS prompt - 'SHOW_PCX SAMPLE.PCX'.       }
{                                                                         }
{*************************************************************************}
{                                                                         }
{ Since this program is provided as a service, you are on your own when     }
{ when you modify it to work with your own programs.                      }
{                                                                         }
{ We strive to make every program bug-free. If you find any bugs in this    }
{ program, please contact us on Compuserve (76702,1207)                   }
{ However, this program is provided AS IS and we are not responsible for any }
{ problems you might discover.                                            }
{                                                                         }
{*************************************************************************}
{                                                                         }
{ Remember, some computers and video adapters are NOT 100% compatible, no   }
{ matter what their marketing department may say. This shows up when your   }
{ program runs on everyone's computer EXCEPT a particular clone.          }
{ Unfortunately, there is not much you can do to correct it.              }
{                                                                         }
{ For example, some early VGA cards do not support the BIOS calls to set up  }
{ a VGA palette - so the PCX image may come up all black, or with the wrong  }
```

```
{ colors.                                                                    }
{                                                                            }
{ Also, if you use code that attempts to determine what kind of video card   }
{ is attached to the computer it may lock-up...                              }
{                                                                            }
{****************************************************************************}
{                                                                            }
{ The PCX file format was originally developed in 1982, when there were only }
{ three video addapters: CGA, Hercules, and the Tecmar Graphics Master. Over }
{ the years, as new hardware became available (EGA, VGA, etc.), we had to    }
{ modify the format. Wherever posible, we insure downward compatiblity. This }
{ means, if you follow the suggestions in this program, your own program     }
{ should be able to read 'new' PCX files in the future.                      }
{                                                                            }
{****************************************************************************}

{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
{
  NEEDED ADDITIONS:
  CGA palette - read old and new palette - set screen palette
}
{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~}


uses
  Crt, Dos;

const
   MAX_WIDTH = 4000;    { arbitrary - maximum width (in bytes) of a PCX image }
   COMPRESS_NUM = $C0;  { this is the upper two bits that indicate a count }
   MAX_BLOCK = 4096;

   RED = 0;
   GREEN = 1;
   BLUE = 2;

   CGA4 = $04;          { video modes }
   CGA2 = $06;
   EGA  = $10;
   VGA  = $12;
   MCGA = $13;

type
   str80 = string [80];
   file_buffer = array [0..127] of byte;
   block_array = array [0..MAX_BLOCK] of byte;
   pal_array = array [0..255, RED..BLUE] of byte;
   ega_array = array [0..16] of byte;
   line_array = array [0..MAX_WIDTH] of byte;

   pcx_header = record
        Manufacturer: byte;     { Always 10 for PCX file }

        Version: byte;          { 2 - old PCX - no palette (not used anymore),
                                  3 - no palette,
                                  4 - Microsoft Windows - no palette (only in
                                      old files, new Windows version uses 3),
                                  5 - with palette }

        Encoding: byte;         { 1 is PCX, it is possible that we may add
                                  additional encoding methods in the future }

        Bits_per_pixel: byte;   { Number of bits to represent a pixel
                                  (per plane) - 1, 2, 4, or 8 }

        Xmin: integer;          { Image window dimensions (inclusive) }
        Ymin: integer;          { Xmin, Ymin are usually zero (not always) }
        Xmax: integer;
        Ymax: integer;

        Hdpi: integer;          { Resolution of image (dots per inch) }
```

```
        Vdpi: integer;          { Set to scanner resolution - 300 is default }

        ColorMap: array [0..15, RED..BLUE] of byte;
                                { RGB palette data (16 colors or less)
                                  256 color palette is appended to end of file }

        Reserved: byte;         { (used to contain video mode)
                                  now it is ignored - just set to zero }

        Nplanes: byte;          { Number of planes }

        Bytes_per_line_per_plane: integer;   { Number of bytes to allocate
                                               for a scanline plane.
                                               MUST be an an EVEN number!
                                               Do NOT calculate from Xmax-Xmin! }

        PaletteInfo: integer;   { 1 = black & white or color image,
                                  2 = grayscale image - ignored in PB4, PB4+
                                  palette must also be set to shades of gray! }

        HscreenSize: integer;   { added for PC Paintbrush IV Plus ver 1.0,  }
        VscreenSize: integer;   { PC Paintbrush IV ver 1.02 (and later)     }
                                { I know it is tempting to use these fields
                                  to determine what video mode should be used
                                  to display the image - but it is NOT
                                  recommended since the fields will probably
                                  just contain garbage. It is better to have
                                  the user install for the graphics mode he
                                  wants to use... }

        Filler: array [74..127] of byte;     { Just set to zeros }
        end;

var
   Name: str80;                         { Name of PCX file to load }
   ImageName: str80;                    { Name of PCX file - used by ReadError }
   BlockFile: file;                     { file for reading block data }
   BlockData: block_array;              { 4k data buffer }

   Header: pcx_header;                  { PCX file header }
   Palette256: pal_array;               { place to put 256 color palette }
   PaletteEGA: ega_array;               { place to put 17 EGA palette values }
   PCXline: line_array;                 { place to put uncompressed data }

   Ymax: integer;                       { maximum Y value on screen }
   NextByte: integer;                   { index into file buffer in ReadByte }
   Index: integer;                      { PCXline index - where to put Data }
   Data: byte;                          { PCX compressed data byte }

   PictureMode: integer;                { Graphics mode number }
   Reg: Registers;                      { Register set - used for int 10 calls }


{ ================================= Error ================================== }

procedure Error (s: str80 );

{ Print out the error message and wait, then halt }

var c: char;
    i: integer;

begin
TextMode (C80);
writeln ('ERROR');
writeln (s);
halt;
end;   { Error }


{ ============================== ReadError ================================ }
```

```
procedure ReadError (msg: integer);

{ Check for an i/o error }

begin
if IOresult <> 0 then
   case msg of
   1: Error ('Can''t open file - ' + ImageName);
   2: Error ('Error closing file - ' + ImageName + ' - disk may be full');
   3: Error ('Error reading file - ' + ImageName);

   else
      Error ('Error doing file I/O - ' + ImageName);
   end;   { case }
end;   { ReadError }


{ ========================== VideoMode ============================== }

procedure VideoMode (n: integer);

{ Do a BIOS call to set the video mode }
{ In Turbo Pascal, a '$' means the number is hexadeximal. }

begin
Reg.ah := $00;
Reg.al := n;                          { mode number }
intr ($10, Reg);                      { call interrupt }
end;  { VideoMode }


{ ========================== EGApalette ============================== }

procedure EGApalette (n, R, G, B: integer);

{ Set a single EGA's palette register.
  n is the index of the palette register.
  R, G, and B are 0..255. }

{ This code is never called - it is here as an example }

{ In Turbo Pascal, a '$' means the number is hexadeximal. }

var i: integer;

begin
R := R shr 6;                         { R, G, and B are now 0..3 }
G := G shr 6;
B := B shr 6;
i := (R shl 4) + (G shl 2) + B;

Reg.ah := $10;
Reg.al := 0;                          { set individual palette register }
Reg.bh := i;                          { value }
Reg.bl := n;                          { palette register number }
intr ($10, Reg);                      { call interrupt }
end;  { EGApalette }


{ ========================== VGApalette ============================== }

procedure VGApalette (n, R, G, B: integer);

{ Set a single VGA palette and DAC register pair.
  n is the index of the palette register.
  R, G, and B are 0..255. }

{ This code is never called - it is here as an example }

{ In Turbo Pascal, a '$' means the number is hexadeximal. }
```

```
begin
R := R shr 2;                          { R, G, and B are now 0..63 }
G := G shr 2;
B := B shr 2;

Reg.ah := $10;                         { Set Palette Call }
Reg.al := $0;                          { set individual palette register }
Reg.bl := n;                           { palette register number 0..15, 0..255 }
Reg.bh := n;                           { palette register value }
intr ($10, Reg);                       { call interrupt }

Reg.ah := $10;                         { Set DAC Call }
Reg.al := $10;                         { set individual DAC register }
Reg.bx := n;                           { DAC register number 0..15, 0..255 }
Reg.dh := R;                           { red value 0..63 }
Reg.ch := G;                           { green value 0..63 }
Reg.cl := B;                           { blue value 0..63 }
intr ($10, Reg);                       { call interrupt }
end;  { VGApalette }


{ ========================== EGA16palette ================================ }

procedure EGA16palette;

{ Set the EGA's entire 16 color palette. }
{ In Turbo Pascal, a '$' means the number is hexadeximal. }

var
   i, r, g, b: integer;

begin
for i := 0 to 15 do
   begin
   r := Header.ColorMap [i, RED]   shr 6;       { r, g, and b are now 0..3 }
   g := Header.ColorMap [i, GREEN] shr 6;
   b := Header.ColorMap [i, BLUE]  shr 6;
   PaletteEGA [i] := (r shl 4) + (g shl 2) + b;
   end;
PaletteEGA [16] := 0;                  { border color }

Reg.ah := $10;                         { Set Palette Call }
Reg.al := $02;                         { set a block of palette registers }
Reg.dx := ofs (PaletteEGA);            { offset of block }
Reg.es := seg (PaletteEGA);            { segment of block }
intr ($10, Reg);                       { call interrupt }

end;  { EGA16palette }


{ ========================== VGA16palette ================================ }

procedure VGA16palette;

{ Set the VGA's entire 16 color palette. }
{ In Turbo Pascal, a '$' means the number is hexadeximal. }

var
   i: integer;

begin
for i := 0 to 15 do
   PaletteEGA [i] := i;
PaletteEGA [16] := 0;                  { border color }

Reg.ah := $10;                         { Set Palette Call }
Reg.al := $02;                         { set a block of palette registers }
Reg.dx := ofs (PaletteEGA);            { offset of block }
Reg.es := seg (PaletteEGA);            { segment of block }
intr ($10, Reg);                       { call interrupt }
```

```
for i := 0 to 15 do
   begin                                          { R, G, and B must be 0..63 }
   Palette256 [i, RED]   := Header.ColorMap [i, RED]   shr 2;
   Palette256 [i, GREEN] := Header.ColorMap [i, GREEN] shr 2;
   Palette256 [i, BLUE]  := Header.ColorMap [i, BLUE]  shr 2;
   end;

Reg.ah := $10;                       { Set DAC Call }
Reg.al := $12;                       { set a block of DAC registers }
Reg.bx := 0;                         { first DAC register number }
Reg.cx := 255;                       { number of registers to update }
Reg.dx := ofs (Palette256);          { offset of block }
Reg.es := seg (Palette256);          { segment of block }
intr ($10, Reg);                     { call interrupt }

end;  { VGA16palette }


{ ========================== EntireVGApalette ============================== }

procedure EntireVGApalette;

{ Set the VGA's entire 256 color palette. }
{ In Turbo Pascal, a '$' means the number is hexadeximal. }

var
   i: integer;

begin
for i := 0 to 255 do
   begin                                          { R, G, and B must be 0..63 }
   Palette256 [i, RED]   := Palette256 [i, RED]   shr 2;
   Palette256 [i, GREEN] := Palette256 [i, GREEN] shr 2;
   Palette256 [i, BLUE]  := Palette256 [i, BLUE]  shr 2;
   end;

Reg.ah := $10;                       { Set DAC Call }
Reg.al := $12;                       { set a block of DAC registers }
Reg.bx := 0;                         { first DAC register number }
Reg.cx := 255;                       { number of registers to update }
Reg.dx := ofs (Palette256);          { offset of block }
Reg.es := seg (Palette256);          { segment of block }
intr ($10, Reg);                     { call interrupt }

end;  { EntireVGApalette }


{ ========================== SetPalette ================================ }

procedure SetPalette;

{ Set up the entire graphics palette }

var i: integer;

begin
if PictureMode = MCGA then
   EntireVGApalette
else if PictureMode = VGA then
   VGA16palette
else
   EGA16palette;
end;  { SetPalette }


{ ========================== ShowCGA ================================ }

procedure ShowCGA (Y: integer);

{ Put a line of CGA data on the screen }
```

```
{ In Turbo Pascal, a '$' means the number is hexadeximal. }

var
   i, j, l, m, t: integer;
   Yoffset: integer;
   CGAScreen: array [0..32000] of byte absolute $B800:$0000;

begin
i := 8 div Header.Bits_per_pixel;        { i is pixels per byte }

if (i = 8) then                          { 1 bit per pixel }
   j := 7
else                                     { 2 bits per pixel }
   j := 3;

t := (Header.Xmax - Header.Xmin + 1);    { width in pixels }
m := t and j;                            { left over bits }

l := (t + j) div i;                      { compute number of bytes to display }
if l > 80 then
   begin
   l := 80;                              { don't overrun screen width }
   m := 0;
   end;

if (m <> 0) then                         { we need to mask unseen pixels }
   begin
   m := $FF shl (8 - (m * Header.Bits_per_pixel));   { m = mask }
   t := l - 1;
   PCXline [t] := PCXline [t] and m;     { mask off unseen pixels }
   end;

Yoffset := 8192 * (Y and 1);
Move (PCXline [0], CGAScreen [((Y shr 1) * 80) + Yoffset], l);

end;   { ShowCGA }


{ ============================ ShowEGA ================================ }

procedure ShowEGA (Y: integer);

{ Put a line of EGA (or VGA) data on the screen }
{ In Turbo Pascal, a '$' means the number is hexadeximal. }

var
   i, j, l, m, t: integer;
   EGAplane: integer;
   EGAscreen: array [0..32000] of byte absolute $A000:$0000;

begin
EGAplane := $0100;                       { the first plane to update }
PortW [$3CE] := $0005;                   { use write mode 0 }

{ PortW [$3CE] := $0005;      does port I/O by words. It is the same as:

  Out 03CEh,05h
  Out 03CFh,00h
}

t := (Header.Xmax - Header.Xmin + 1);    { width in pixels }
m := t and 7;                            { left over bits }

l := (t + 7) shr 3;                      { compute number of bytes to display }
if (l >= 80) then
   begin
   l := 80;                              { don't overrun screen width }
   m := 0;
   end;

if (m <> 0) then
```

```
   m := $FF shl (8 - m)                  { m = mask for unseen pixels }
else
   m := $FF;

for i := 0 to Header.Nplanes-1 do
   begin
   j := i * Header.Bytes_per_line_per_plane;
   t := j + l - 1;
   PCXline [t] := PCXline [t] and m;           { mask off unseen pixels }

   PortW [$3C4] := EGAplane + 2;               { set plane number }
   Move (PCXline [j], EGAscreen [Y * 80], l);
   EGAplane := EGAplane shl 1;
   end;

PortW [$3C4] := $0F02;                         { default plane mask }
end;   { ShowEGA }


{ ========================== ShowMCGA ============================== }

procedure ShowMCGA (Y: integer);

{ Put a line of MCGA data on the screen }
{ In Turbo Pascal, a '$' means the number is hexadeximal. }

var
   l: integer;
   MCGAscreen: array [0..64000] of byte absolute $A000:$0000;

begin
l := Header.XMax - Header.Xmin;            { compute number of bytes to display }
if l > 320 then
   l := 320;                               { don't overrun screen width }

Move (PCXline [0], MCGAScreen [Y * 320], l);

end;   { ShowMCGA }


{ ========================== Read256palette ============================== }

procedure Read256palette;

{ Read in a 256 color palette at end of PCX file }

var
   i: integer;
   b: byte;

begin
seek (BlockFile, FileSize (BlockFile) - 769);
BlockRead (BlockFile, b, 1);           { read indicator byte }
ReadError (3);

if b <> 12 then                        { no palette here... }
   exit;

BlockRead (BlockFile, Palette256, 3*256);
ReadError (3);

seek (BlockFile, 128);                 { go back to start of PCX data }

end;  { Read256palette }


{ ========================== ReadHeader ============================== }

procedure ReadHeader;

{ Load a picture header from a PC Paintbrush PCX file }
```

```
label WrongFormat;

begin
{$I-}
BlockRead (BlockFile, Header, 128);         { read 128 byte PCX header }
ReadError (3);

                                            { Is it a PCX file? }
if (Header.Manufacturer <> 10) or (Header.Encoding <> 1) then
   begin
   close (BlockFile);
   Error ('This is not a valid PCX image file.');
   end;

if (Header.Nplanes = 4) and (Header.Bits_per_pixel = 1) then
   begin
   if (Header.Ymax - Header.Ymin) <= 349 then
      begin
      PictureMode := EGA;
      Ymax := 349;
      end
   else
      begin
      PictureMode := VGA;
      Ymax := 479;
      end;
   end
else if (Header.Nplanes = 1) then
   begin
   Ymax := 199;

   if (Header.Bits_per_pixel = 1) then
      PictureMode := CGA2
   else if (Header.Bits_per_pixel = 2) then
      PictureMode := CGA4
   else if (Header.Bits_per_pixel = 8) then
      begin
      PictureMode := MCGA;
      if Header.Version = 5 then
         Read256palette;
      end
   else
      goto WrongFormat;
   end
else
   begin
WrongFormat:
   close (BlockFile);
   Error ('PCX file is in wrong format - It must be a CGA, EGA, VGA, or MCGA image');
   end;

Index := 0;
NextByte := MAX_BLOCK;          { indicates no data read in yet... }

end;  { ReadHeader }


{ ============================ ReadByte ================================= }

procedure ReadByte;

{ read a single byte of data - use BlockRead because it is FAST! }

var
   NumBlocksRead: integer;

begin
if NextByte = MAX_BLOCK then
   begin
   BlockRead (BlockFile, BlockData, MAX_BLOCK, NumBlocksRead);
```

```
   NextByte := 0;
   end;

data := BlockData [NextByte];
inc (NextByte);                          { NextByte++; }
end;  { ReadByte }


{ ========================== Read_PCX_Line ================================ }

procedure Read_PCX_Line;

{ Read a line from a PC Paintbrush PCX file }

var
   count: integer;
   bytes_per_line: integer;

begin
{$I-}

bytes_per_line := Header.Bytes_per_line_per_plane * Header.Nplanes;

                          { bring in any data that wrapped from previous line }
                          { usually none  -  this is just to be safe          }
if Index <> 0 then
   FillChar (PCXline [0], Index, data);    { fills a contiguous block of data }

while (Index < bytes_per_line) do          { read 1 line of data (all planes) }
   begin
   ReadByte;

   if (data and $C0) = compress_num then
      begin
      count := data and $3F;
      ReadByte;
      FillChar (PCXline [Index], count, data);  { fills a contiguous block }
      inc (Index, count);                       { Index += count; }
      end
   else
      begin
      PCXline [Index] := data;
      inc (Index);                              { Index++; }
      end;
   end;

ReadError (3);

Index := Index - bytes_per_line;

{$I+}
end;  { Read_PCX_Line }


{ ========================== Read_PCX ================================ }

procedure Read_PCX (name: str80);

{ Read PC Paintbrush PCX file and put it on the screen }

var
   k, kmax: integer;

begin
{$I-}
ImageName := name;                     { used by ReadError }

assign (BlockFile, name);
reset (BlockFile, 1);                  { use 1 byte blocks }
ReadError (1);
```

```
ReadHeader;                             { read the PCX header }

{ >>>>> No checking is done to see if the user has the correct hardware <<<<<
  >>>>> to load the image. Your program sure verify the video mode is   <<<<<
  >>>>> supported. Otherwise, the computer may lock-up.                 <<<<< }

VideoMode (PictureMode);                { switch to graphics mode }
if Header.Version = 5 then
   SetPalette;                          { set the screen palette, if available }

{ >>>>> Note: You should compute the height of the image as follows. <<<<<
  >>>>> Do NOT just read until End-Of-File!                          <<<<< }

kmax := Header.Ymin + Ymax;
if Header.Ymax < kmax then        { don't show more than the screen can display }
   kmax := Header.ymax;

if (PictureMode = EGA) or (PictureMode = VGA) then
   begin
   for k := Header.Ymin to kmax do          { each loop is separate for speed }
      begin
      Read_PCX_Line;
      ShowEGA (k);
      end;
   end
else if (PictureMode = MCGA) then
   begin
   for k := Header.Ymin to kmax do
      begin
      Read_PCX_Line;
      ShowMCGA (k);
      end;
   end
else                                        { it's a CGA picture }
   begin
   for k := Header.Ymin to kmax do
      begin
      Read_PCX_Line;
      ShowCGA (k);
      end;
    end;

close (BlockFile);
ReadError (2);
{$I+}
end;  { Read_PCX }


{ ========================== DISPLAY_PCX ================================ }

procedure display_pcx (name: str80);

{ Display a PCX picture }

var
   c: char;

begin
Read_PCX (name);              { read and display the file }

while (not KeyPressed) do     { wait for any key to be pressed }
   { nothing };

c := ReadKey;                 { now get rid of the key that was pressed }
if c = #0 then                { handle function keys }
   c := ReadKey;

end;   { display_pcx }


{ ************************** MAIN ****************************** }
```

```
begin
ClrScr;
writeln ('         SHOW_PCX - read and display a PC Paintbrush (R) picture');
writeln;
writeln ('                              PERMISSION TO COPY:');
writeln ('            SHOW_PCX -- (C) Copyright 1989 ZSoft, Corporation.');
writeln;
writeln ('You are licensed to freely copy SHOW_PCX and incorporate it into your');
writeln ('own programs, provided that:');
writeln ('  IF YOU COPY SHOW_PCX WITHOUT CHANGING IT:');
writeln ('  (1) You must retain this "Permission to Copy" notice, and');
writeln ('  (2) You must not charge for the SHOW_PCX software or documentaion;');
writeln ('      however, you may charge a service fee for disk duplication and');
writeln ('      distribution, so long as such fee is not more than $5.00');
writeln ('  IF YOU MODIFY SHOW_PCX AND/OR INCORPORATE SHOW_PCX INTO YOUR OWN PROGRAMS');
writeln ('  (1) You must include the following notice in the appropriate places:');
writeln ('      Includes portions of SHOW_PCX. Used by permission of ZSoft Corporation.');
writeln;
writeln (' ZSoft Corporation reserves all rights to SHOW_PCX except as stated herein.');
writeln (' ZSoft Corporation,  450 Franklin Road, Suite 100,  Marietta, GA 30067');
writeln (' (404) 428-0008');
writeln ('                  [END OF "PERMISSION TO COPY" NOTICE]');
writeln;

if (ParamCount = 0) then           { no DOS command line parameters }
   begin
   writeln ('The image must be a 2 or 4 color CGA, 16 color EGA or VGA,');
   writeln ('or a 256 color MCGA picture');
   writeln;

   write ('Enter name of picture file to display: ');
   readln (name);
   writeln;
   end
else
   Name := ParamStr (1);           { get filename from DOS command line }

if (Pos ('.', Name) = 0) then      { make sure the filename has PCX extension }
   Name := Concat (Name, '.pcx');

display_pcx (Name);

TextMode (co80);                   { back to text mode }

end.  { Show_PCX }
```

PLAYFLI.ASM

```
;FLI Player
;vytvoril Vit Kovalcik (v èeštinì: Vít Kovalèík)
;e-mail : vkovalcik@iname.com
;www : http://www.geocities.com/SiliconValley/Hills/1335/

;Tento zdrojovy kod je freeware. Je mozno jej dale sirit v nezmenene
;forme. Pokud jej budete modifikovat a pouzivat ve svych programech,
;prosim uvedte moje jmeno v 'O programu' nebo podobne (neni podminkou).

;Pokud mate pristup na Internet, muzete me poslat zpravu, jestli se vam
;tento programek hodi nebo pokud najdete nejakou chybu.

;Melo by to jit prelozit TASM & TLINK.
;(tzn. prikazy 'tasm.exe playfli.asm' a 'tlink.exe playfli.obj')
;Vysledny soubor ma v tomto pripade 1953 B.

    DOSSEG
    .286P
    .MODEL SMALL
    .STACK 200h
    .DATA
    .CODE

    PUBLIC START

ErrorMsgs  DB  'Chyba pri otevirani souboru !',13,10,'$'
           DB  'Chyba ve formatu !',13,10,'$'
           DB  'Chyba pri alokaci pameti !',13,10,'$'
           DB  'Snimek je vetsi nez 64 kB !',13,10,'$'
           DB  'Neznamy typ bloku (chunk) !',13,10,'$'
           DB  'Balicku s barvama je prilis mnoho !',13,10,'$'
           DB  'Stejnych radku je prilis mnoho (radku>255 (proc?))',13,10,'$'
           DB  'Menicich se radku je prilis mnoho (radku>255 (proc?))',13,10,'$'
           DB  'Jako parametr musi byt zadano jmeno souboru .FLI',13,10,'$'
           DB  'Toto je soubor typu .FLC !',13,10,'$'
Handle     DW  0             ;Handle souboru
MemSize    DW  0             ;Velikost alokovane pameti
AllocSeg   DW  0             ;Pocatecni alokovany segment
ActFrame   DW  0             ;Aktualni frame
ActPos     DW  0             ;Aktualni pozice
FileName   DB  128 DUP(?)
FLIH       DB 128 DUP(?)

;Protoze se v tom neda vyznat, jsou oznaceny instukce
;'push ..' a 'pop ..', ktere patri k sobe symolem 'p<cislo>'
;(doufam, ze to je oznaceno spravne)


DEALLOCMEM PROC
    mov dx,[AllocSeg]
    cmp dx,0
    jz NoDealloc
    mov es,dx
    mov ah,049H
    int 21H
NoDealloc:
    ret
DEALLOCMEM ENDP

CLOSEFILE PROC
    mov bx,[Handle]
    cmp bx,0
    jz NoClose
    mov ah,03EH
    int 21H
NoClose:
    ret
CLOSEFILE ENDP
```

```
ERRORMSG PROC
    call DoneVGA
    call DeallocMem
    call CloseFile
    mov ax,seg ErrorMsgs
    mov ds,ax
    mov si,offset ErrorMsgs
MsgLoop:
    cmp cl,1
    jz MsgLoopEnd
CharLoop:
    lodsb
    cmp al,'$'
    jnz CharLoop
    dec cl
    jmp MsgLoop
MsgLoopEnd:
    mov dx,si
    mov ah,009H
    int 21H
    mov ax,4C00h             ;Konec programu
    int 21h
    ret
ERRORMSG ENDP


INITVGA PROC
    mov ax,0013H
    int 10H
    ret
INITVGA ENDP

DONEVGA PROC
    mov ax,0003H
    int 10H
    ret
DONEVGA ENDP

START PROC
    mov ah,62H
    int 21H
    mov es,bx
    mov ah,4AH
    mov bx,00FFFH
    int 21H
    mov bx,es                ;Otevreni souboru
    mov di,00081H
    mov al,es:[di]
    cmp al,13
    jz NoParams
    cmp al,0
    jnz ParamYes
NoParams:
    mov cl,9
    call errormsg
ParamYes:
    mov al,es:[di]
    cmp al,32
    jnz NoSpace
    inc di
NoSpace:
    mov bx,seg FileName
    mov si,offset FileName
    mov ds,bx
    push si                  ;p1
NameLoop:
    mov al,es:[di]
    mov ds:[si],al
    inc di
    inc si
```

```
    cmp al,32
    jz EOName
    cmp al,13
    jnz NameLoop
EOName:
    dec si
    dec di
    cmp di,00082H
    jnz NameProbOK
    mov cl,1
    call ErrorMsg
NameProbOK:
    mov al,00H
    mov ds:[si],al
    pop dx                  ;p1
    mov ax,03D02H
    int 21H
    jnc OpenOK
    mov cl,1
    call ErrorMsg
OpenOK:
    mov [Handle],ax

    mov ah,048H             ;Alokace pameti
    mov bx,00FFFH
    int 21H
    jnc AllocOK
    mov cl,3
    call ErrorMsg
AllocOK:
    mov [AllocSeg],ax

    mov bx,[Handle]         ;Cteni hlavicky FLIcka
    mov ax,seg FLIH
    mov ds,ax
    mov dx,offset FLIH
    mov ah,03FH
    mov cx,128
    int 21H

    call InitVGA
FLIPlayLoop:
    mov ax,seg FLIH
    mov es,ax
    mov di,offset FLIH
    mov ax,es:[di+4]
    cmp ax,0AF12H
    jnz FLIMagicOK
    mov cl,10
    call ErrorMsg
FLIMagicOK:
    mov ax,es:[di+6]
    mov [ActFrame],ax

    mov bx,[Handle]         ;Cteni hlavicky prvniho snimku
    mov cx,[AllocSeg]
    mov ds,cx
    xor dx,dx
    mov cx,16
    mov ah,03FH
    int 21H
    mov si,dx


FrameLoop:
    dec [ActFrame]

    mov ax,[si+4]
    cmp ax,0F1FAH
    jz FrameOK
    mov cl,2
```

```
    call ErrorMsg
FrameOK:
    mov ax,[si+6]
    cmp dx,0
    jnz FrameLarger
    jmp FrameSizeOK
FrameLarger:
    cmp [si],0FFFFH
    jc FrameSizeOK
    mov cl,4
    call ErrorMsg
FrameSizeOK:
    mov cx,[si]

    push ax                 ;Cteni ramu + dalsi hlavicky do pameti
                            ;p2
    mov bx,[Handle]
    mov ax,[AllocSeg]
    mov ds,ax
    xor dx,dx
    mov ah,03FH
    int 21H
    pop ax                  ;p2

    mov bl,al               ;bl=MaxChunk
    xor si,si               ;ds:si=AllocSeg
    sub cx,16
    push cx                 ;p3

ChunkLoop:
    cmp bl,0
    jnz NextChunk
    jmp FrameEnd
NextChunk:
    dec bl
    mov cl,[si+4]           ;cl=typ chunku
    mov ax,[si]
    add ax,si
    push ax                 ;p7
    add si,6
    push bx                 ;p4

;(ChunkType11)
    cmp cl,11
    jnz ChunkType12
    mov dh,[si+1]
    cmp dh,0
    jz ColorsOK
    mov cl,6
    call ErrorMsg
ColorsOK:
    mov dl,[si]             ;dl=pocet balicku
    inc si
    inc si
    xor bl,bl
PacketLoop:
    push dx                 ;p5
    add bl,[si]
    mov cl,[si+1]
    inc si
    inc si
    xor bh,bh
    xor ch,ch
    cmp cl,0
    jnz No256ColorsChange
    mov cx,256
No256ColorsChange:
    mov ax,ds
    mov es,ax
    mov ax,01012H
    mov dx,si
```

```
    int 10H
    add bl,cl

Color:
    add si,3
    dec cx
    cmp cx,0
    jnz Color

    pop dx                  ;p5
    dec dl
    cmp dl,0
    jnz PacketLoop
    jmp ChunkEnd


ChunkType12:
    cmp cl,12
    jz ReallyChunkType12
    jmp ChunkType13
ReallyChunkType12:
    mov ax,0A000H
    mov es,ax
    mov bh,[si+1]
    cmp bh,0
    jz LineYSameOK
    mov cl,7
    call ErrorMsg
LineYSameOK:
    mov bh,[si+3]
    jz LineYChangeOK
    mov cl,8
    call ErrorMsg
LineYChangeOK:
    mov bl,[si]             ;YAct
    mov bh,[si+2]           ;YMax
    add si,4
    add bh,bl
    xor dx,dx               ;x
LineLoop:
    push bx                 ;p6
    mov ax,320
    xor bh,bh
    mul bx
    mov di,ax
    mov bh,[si]             ;pocet balicku
    inc si
    cmp bh,0
    jz NoLinePackets
LinePacketLoop:
    dec bh
    mov cl,[si]
    inc si
    xor ch,ch
    add di,cx
    mov cl,[si]
    inc si
    cmp cl,0
    jz OneLinePacketEnd
    cmp cl,128
    jc LOverZero
    mov al,[si]
    inc si

    xor cl,0FFH             ;tyto dva radky: cl:=256-cl
    inc cl

    rep stosb               ;ch musi byt 0
    jmp OneLinePacketEnd
LOverZero:
    rep movsb               ;ch musi byt 0
```

```
OneLinePacketEnd:
    cmp bh,0
    jg LinePacketLoop
NoLinePackets:
    pop bx                  ;p6
    inc bl
    cmp bl,bh
    jnz LineLoop

    jmp ChunkEnd
ChunkType13:
    cmp cl,13
    jnz ChunkType15

    mov ax,0A000H
    mov es,ax
    xor di,di
    mov cx,32000
    mov ax,di               ;toto je rychlejsi nez 'xor ax,ax'
    rep stosw
    jmp ChunkEnd


ChunkType15:
    cmp cl,15
    jnz ChunkType16
    mov ax,0A000H
    mov es,ax
    xor bl,bl               ;y
BrunLineLoop:
    mov ax,320
    xor bh,bh
    mul bx
    mov di,ax

    mov bh,[si]             ;pocet balicku
    inc si
    cmp bh,0
    jz NoPackets
BrunPacketLoop:
    dec bh
    mov cl,[si]
    inc si
    cmp cl,0
    jz OneBrunPacketEnd
    xor ch,ch
    cmp cl,128
    jnc UnderZero
    jz OneBrunPacketEnd
    mov al,[si]
    inc si
    rep stosb
    jmp OneBrunPacketEnd
UnderZero:
    xor cl,0FFH
    inc cl
    rep movsb
OneBrunPacketEnd:
    cmp bh,0
    jnz BrunPacketLoop
NoPackets:
    inc bl
    cmp bl,200
    jnz BrunLineLoop
    jmp ChunkEnd


ChunkType16:
    cmp cl,16
    jnz UnknownChunkType
    xor di,di
```

```
    mov ax,0A000H
    mov es,ax
;    mov cx,64000
;    rep movsb
    mov cx,32000
    rep movsw

    jmp ChunkEnd
UnknownChunkType:
    mov cl,5
    call ErrorMsg
ChunkEnd:
    pop bx                  ;p4
    pop si                  ;p7

    jmp ChunkLoop

FrameEnd:
    pop si                  ;p3
    mov ah,0bH
    int 21H
    cmp al,0FFH
    jz EndOfPrg


    mov dx,seg FLIH
    mov es,dx
    mov di,offset FLIH
    mov cx,es:[di+16]
    cmp cx,0
    jnz Wait1
    mov cx,1
Wait1:
    mov dx,3dah
    in al,dx
    test al,8
    jz Wait1
Wait2:
    mov dx,3dah
    in al,dx
    test al,8
    jnz Wait2
    dec cx
    cmp cx,0
    jnz Wait1


    cmp [ActFrame],0
    jz FLIEnd
    jmp FrameLoop           ;Skok na dalsi snimek
FLIEnd:

    mov ax,4200H
    mov bx,[Handle]
    xor cx,cx
    mov dx,128
    int 21H
    jmp FLIPlayLoop

EndOfPrg:
    call DoneVGA
    call DeallocMem
    call CloseFile

    mov ax,4c00h            ;Konec programu
    int 21h
START ENDP
END START
```

Grafické formáty

Grafické formáty
Branislav Sobota, Ján Milán

Obsah:
1. Technické prostriedky pre počítačovú grafiku
2. Farby na počítači
3. Komprimácia grafických údajov
4. Formátový Babylon
5. Rastrové grafické formáty
6. Vektorové grafické formáty
7. Niektoré možné trendy
8. Záver
9. Prílohy
10. Použitá literatura
11. Register

teorie: barevné modely RGB, CMY, HSB, HLS, UWB, ztrátové a neztrátové komprimace
rastrové formáty: BMP, FLC, FLI, GIF, ICO, IMG, JPEG, MPEG, MSP, PCX, RAW, TIFF
vektorové formáty: DXB, DXF, EPS, HPG, LDF, WMF

disketa: zdrojové texty (C) pro práci s formáty (BMP, PCX, GIF, ICO, IMG, RAW, WMF, DXF); pomocné programy (ukládání obrazovky, ztrátová komprimace, konverzní programy, etc.)

Vydáno v Českých Budějovicích, 1996.
160 stran.
Slovensky.
ISBN 80-85828-58-8

Cena knihy 119,- Kč (včetně DPH)
Cena diskety 69,- Kč (včetně DPH)

Nakladatelství Kopp
Šumavská 3, 370 01 České Budějovice
038/602 43
kopp@kopp.cz
http://www.kopp.cz

Řídící kódy ASCII

	Hex	Ctrl	Name	Význam
0	00	^@	NUL	null (end of string)
1	01	^A	SOH	start of heading
2	02	^B	STX	start of text
3	03	^C	ETX	end of text
4	04	^D	EOT	end of transmission
5	05	^E	ENQ	enquiry
6	06	^F	ACK	acknowledge
7	07	^G	BEL	bell
8	08	^H	BS	backspace
9	09	^I	HT	TAB horizontal tab
10	0a	^J	LF	line feed
11	0b	^K	VT	vertical tab
12	0c	^L	FF	form feed
13	0d	^M	CR	carriage return
14	0e	^N	SO	shift out
15	0f	^O	SI	shift in
16	10	^P	DLE	data line escape
17	11	^Q	DC1	device ctrl 1 (X-ON)
18	12	^R	DC2	device ctrl 2
19	13	^S	DC3	device ctrl 3 (X-OFF)
20	14	^T	DC4	device ctrl 4
21	15	^U	NAK	negative acknowledge
22	16	^V	SYN	synchronous idle
23	17	^W	ETB	end of transmit block
24	18	^X	CAN	cancel
25	19	^Y	EM	end of medium
26	1a	^Z	SUB	substitute
27	1b	^[	ESC	escape
28	1c	^\	FS	file separator
29	1d	^]	GS	group separator
30	1e	^^	RS	record separator
31	1f	^_	US	unit separator

Seznam domén 1. řádu

Generické domény:

ARPA	Arpanet
ARTS	Umìní
COM	Komeèní organizace
EDU	Vzdìlávací organizace
FIRM	Firmy
GOV	Vládní organizace
INFO	Informace
INT	Mezinárodní organizace
MIL	Vojenské organizace
NATO	NATO
NET	Sí
NOM	Osobní stránky
ORG	Neziskové organizace
REC	Rekreace
SHOP	Obchodování po internetu
WEB	

Domény podle státù:

AD	Andora
AE	Spojené arabské emiráty
AF	Afghánistán
AG	Antigua a Barbuda
AI	Anguilla
AL	Albánie
AM	Arménie
AN	Holandské Antily
AO	Angola
AQ	Antarktida
AR	Argentina
AS	Americké Samoa
AT	Rakousko
AU	Austrálie
AW	Aruba
AZ	Azerbajdžán
BA	Bosna a Hercegovina
BB	Barbados
BD	Bangladéš
BE	Belgie
BF	Burkina Faso
BG	Bulharsko
BH	Bahrajn
BI	Burundi
BJ	Benin
BM	Bermudy
BN	Brunei Darussalam
BO	Bolívie
BR	Brazílie
BS	Bahamy

BT	Bhutan
BV	Bouvetùv ostrov
BW	Botswana
BY	Bìlorusko
BZ	Belize
CA	Kanada
CC	Kokosové ostrovy
CF	Støedoafrická republika
CG	Kongo
CH	Švýcarsko
CI	Cote D'Ivoire (Ivory Coast)
CK	Cookovy ostrovy
CL	Chile
CM	Cameroon
CN	Èína
CO	Kolumbie
CR	Kostarika
CS	Èeskoslovensko
CU	Kuba
CV	Cape Verde
CX	Vánoèní ostrov
CY	Kypr
CZ	Èeská republika
DE	Nìmecko
DJ	Džibuti
DK	Dánsko
DM	Dominika
DO	Dominikánská republika
DZ	Alžír
EC	Ekvádor
EE	Estonsko
EG	Egypt
EH	Západní Sahara
ER	Eritrea
ES	Španìlsko
ET	Etiopie
FI	Finsko
FJ	Fidži
FK	Falklandy (Malvíny)
FM	Mikronézie
FO	Faroe Islands
FR	Francie
FX	Francie, Metropolitan
GA	Gabon
GB	Velká Británie
GD	Grenada
GE	Gruzie
GF	Francouzská Guyana
GH	Ghana
GI	Gibraltar
GL	Greenland
GM	Gambie

GN	Guinea
GP	Guadeloupe
GQ	Rovníková Guinea
GR	Øecko
GS	S. Georgia a S. Sandwich Islands
GT	Guatemala
GU	Guam
GW	Guinea-Bissau
GY	Guyana
HK	Hong Kong
HM	Heard a McDonald Islands
HN	Honduras
HR	Chorvatsko
HT	Haiti
HU	Maïarsko
ID	Indonésie
IE	Irsko
IL	Israel
IN	Indie
IO	British Indian Ocean Territory
IQ	Irák
IR	Irán
IS	Island
IT	Itálie
JM	Jamajka
JO	Jordánsko
JP	Japonsko
KE	Keòa
KG	Kyrgyzstán
KH	Kambodža
KI	Kiribati
KM	Komory
KN	Saint Kitts and Nevis
KP	Severní Korea
KR	Jižní Korea
KW	Kuvajt
KY	Kajmanské ostrovy
KZ	Kazachstán
LA	Laos
LB	Libanon
LC	Saint Lucia
LI	Lichtenštejnsko
LK	Sri Lanka
LR	Libérie
LS	Lesotho
LT	Litva
LU	Lucembursko
LV	Lotyšsko
LY	Libye
MA	Maroko
MC	Monako

MD	Moldova
MG	Madagaskar
MH	Marshallovy ostrovy
MK	Makedonie
ML	Mali
MM	Myanmar
MN	Mongolsko
MO	Macau
MP	Northern Mariana Islands
MQ	Martinik
MR	Mauritanie
MS	Montserrat
MT	Malta
MU	Mauritius
MV	Maledivy
MW	Malawi
MX	Mexiko
MY	Malajsie
MZ	Mosambik
NA	Namibie
NC	Nová Kaledonie
NE	Nigerie
NF	Norfolk Island
NG	Nigérie
NI	Nikaragua
NL	Holandsko
NO	Norsko
NP	Nepál
NR	Nauru
NT	Neutrální zóna
NU	Niue
NZ	Nový Zéland
OM	Omán
PA	Panama
PE	Peru
PF	Francouská Polynesie
PG	Papua Nová Guinea
PH	Filipíny
PK	Pákistán
PL	Polsko
PM	Sv. Pierre a Miquelon
PN	Pitcairn
PR	Portoriko
PT	Portugalsko
PW	Palau
PY	Paraguay
QA	Qatar
RE	Reunion
RO	Rumunsko
RU	Rusko
RW	Rwanda

SA	Saudská Arábie
SB	Solomon Islands
SC	Seychelly
SD	Sudán
SE	Švédsko
SG	Singapur
SH	Sv. Helena
SI	Slovinsko
SJ	Svalbard a Jan Mayen Islands
SK	Slovensko
SL	Sierra Leone
SM	San Marino
SN	Senegal
SO	Somálsko
SR	Surinam
ST	Sao Tome and Principe
SU	SSSR
SV	El Salvador
SY	Sýrie
SZ	Svazijsko
TC	Turks a Caicos Islands
TD	Èad
TF	Francouzská jižní teritoria
TG	Togo
TH	Thajsko
TJ	Tádžikistán
TK	Tokelau
TM	Turkmenistán
TN	Tunisko
TO	Tonga
TP	East Timor
TR	Turecko
TT	Trinidad a Tobago
TV	Tuvalu
TW	Taiwan
TZ	Tanzanie
UA	Ukrajina
UG	Uganda
UK	Velká Británie
UM	USA - malé ostrovy
US	USA
UY	Uruguay
UZ	Uzbekistán
VA	Vatikán
VC	Saint Vincent and the Grenadines
VE	Venezuela
VG	Virgin Islands (Britské)
VI	Virgin Islands (U.S.)
VN	Vietnam
VU	Vanuatu
WF	Wallis and Futuna Islands
WS	Samoa

YE	Yemen
YT	Mayotte
YU	Jugoslávie
ZA	Jižní Afrika
ZM	Zambie
ZR	Zair
ZW	Zimbabwe

Scan kódy kláves

1	Esc
2	1
3	2
4	3
5	4
6	5
7	6
8	7
9	8
10	9
11	0
12	- _
13	= +
14	Backspace
15	Tab
16	Q
17	W
18	E
19	R
20	T
21	Y
22	U
23	I
24	O
25	P
26	[{
27	] }
28	Enter
29	Levý Ctrl
30	A
31	S
32	D
33	F
34	G
35	H
36	J
37	K
38	L
39	; :
40	' "
41	` ~
42	Levý Shift
44	Z
45	X
46	C
47	V
48	B
49	N
50	M
51	, <
52	. >
53	/ ?
54	Pravý Shift
55	* na keypadu (numerická klávesnice)

```
56   Levý Alt
57   Mezerník
58   Caps Lock
59   F1
60   F2
61   F3
62   F4
63   F5
64   F6
65   F7
66   F8
67   F9
68   F10
69   Num Lock
70   Scroll Lock
71   7 na keypadu (numerická klávesnice)
72   8 na keypadu (numerická klávesnice)
73   9 na keypadu (numerická klávesnice)
74   - na keypadu (numerická klávesnice)
75   4 na keypadu (numerická klávesnice)
76   5 na keypadu (numerická klávesnice)
77   6 na keypadu (numerická klávesnice)
78   + na keypadu (numerická klávesnice)
79   1 na keypadu (numerická klávesnice)
80   2 na keypadu (numerická klávesnice)
81   3 na keypadu (numerická klávesnice)
82   0 na keypadu (numerická klávesnice)
83   . na keypadu (numerická klávesnice)
87   F11
88   F12
```

Scan kódy rozšířených kláves

```
 28   Enter na keypadu (numerická klávesnice)
 29   Pravý Ctrl
 42   Print Screen
 53   / na keypadu (numerická klávesnice)
 55   Print Screen
 56   Pravý Alt
 71   Home
 72   Šipka nahoru
 73   Page Up
 75   Šipka vlevo
 77   Šipka vpravo
 79   End
 80   Šipka dolù
 81   Page Down
 82   Insert
 83   Delete
111   Macro
```

Virtuální kódy kláves (Windows)

VK_LBUTTON	01	Levé tlaèítko myši
VK_RBUTTON	02	Pravé tlaèítko myši
VK_CANCEL	03	Ctrl-Break
VK_MBUTTON	04	Støední tlaèítko myši
VK_BACK	08	Backspace
VK_TAB	09	Tabulátor
VK_CLEAR	0C	Clear, nepoužito
VK_RETURN	0D	Enter
VK_SHIFT	10	Shift
VK_CONTROL	11	Ctrl
VK_MENU	12	Alt
VK_PAUSE	13	Pause
VK_CAPITAL	14	Caps Lock
VK_ESCAPE	1B	Esc
VK_SPACE	20	Mezerník
VK_PRIOR	21	Page Up
VK_NEXT	22	Page Down
VK_END	23	End
VK_HOME	24	Home
VK_LEFT	25	Šipka vlevo
VK_UP 26	UP	Šipka nahoru
VK_RIGHT	27	Šipka vpravo
VK_DOWN	28	Šipka dolù
VK_SELECT	29	Select, nepoužito
VK_EXECUTE	2B	Execute, nepoužito
VK_SNAPSHOT	2C	Print Screen
VK_INSERT	2D	Insert
VK_DELETE	2E	Delete
VK_HELP	2F	Help, nepoužito
VK_0	30	0
VK_1	31	1
VK_2	32	2
VK_3	33	3
VK_4	34	4
VK_5	35	5
VK_6	36	6
VK_7	37	7
VK_8	38	8
VK_9	39	9
VK_A	41	A
VK_B	42	B
VK_C	43	C
VK_D	44	D
VK_E	45	E
VK_F	46	F
VK_G	47	G
VK_H	48	H
VK_I	49	I
VK_J	4A	J
VK_K	4B	K
VK_L	4C	L
VK_M	4D	M
VK_N	4E	N
VK_O	4F	O

VK_P	50	P
VK_Q	51	Q
VK_R	52	R
VK_S	53	S
VK_T	54	T
VK_U	55	U
VK_V	56	V
VK_W	57	W
VK_X	58	X
VK_Y	59	Y
VK_Z	5A	Z
VK_NUMPAD0	60	0 na keypadu (numerická klávesnice)
VK_NUMPAD1	61	1 na keypadu (numerická klávesnice)
VK_NUMPAD2	62	2 na keypadu (numerická klávesnice)
VK_NUMPAD3	63	3 na keypadu (numerická klávesnice)
VK_NUMPAD4	64	4 na keypadu (numerická klávesnice)
VK_NUMPAD5	65	5 na keypadu (numerická klávesnice)
VK_NUMPAD6	66	6 na keypadu (numerická klávesnice)
VK_NUMPAD7	67	7 na keypadu (numerická klávesnice)
VK_NUMPAD8	68	8 na keypadu (numerická klávesnice)
VK_NUMPAD9	69	9 na keypadu (numerická klávesnice)
VK_MULTIPLY	6A	* na keypadu (numerická klávesnice)
VK_ADD	6B	+ na keypadu (numerická klávesnice)
VK_SEPARATOR	6C	Separator, nepoužito
VK_SUBTRACT	6D	- na keypadu (numerická klávesnice)
VK_DECIMAL	6E	. na keypadu (numerická klávesnice)
VK_DIVIDE	6F	/ na keypadu (numerická klávesnice)
VK_F1	70	F1
VK_F2	71	F2
VK_F3	72	F3
VK_F4	73	F4
VK_F5	74	F5
VK_F6	75	F6
VK_F7	76	F7
VK_F8	77	F8
VK_F9	78	F9
VK_F10	79	F10
VK_F11	7A	F11
VK_F12	7B	F12
VK_F13	7C	F13
VK_F14	7D	F14
VK_F15	7E	F15
VK_F16	7F	F16
VK_F17	80H	F17
VK_F18	81H	F18
VK_F19	82H	F19
VK_F20	83H	F20
VK_F21	84H	F21
VK_F22	85H	F22
VK_F23	86H	F23
VK_F24	87H	F24
VK_NUMLOCK	90	Num Lock
VK_SCROLL	91	Scroll Lock

Kódové stránky

037 EBCDIC
437 MS-DOS (USA)
500 EBCDIC "500V1"
708 Arabská (ASMO 708)
709 Arabská (ASMO 449+, BCON V4)
710 Arabská (Transparent Arabic)
720 Arabská (Transparent ASMO)
737 Øecká (døíve 437G)
775 Baltská
850 MS-DOS Vícejazyèná (Latin I)
852 MS-DOS Slovanská (Latin II)
855 IBM Cyrilice
857 IBM Turecká
860 MS-DOS Portugalská
861 MS-DOS Islandská
862 Hebrejeská
863 MS-DOS Kanadská (Francouzská)
864 Arabská
865 MS-DOS Nordická
866 MS-DOS Ruská (SSSR)
869 IBM Moderní øecká
874 Thajská
875 EBCDIC
932 Japonská
936 Èínská (PRC, Singapur)
949 Korejská
950 Èínská (Taiwan, Hong Kong)
1026 EBCDIC
1200 Unicode (BMP nebo ISO 10646)
1250 Windows 3.1 Východoevropská (Eastern European)
1251 Windows 3.1 Cyrilice
1252 Windows 3.1 US (ANSI)
1253 Windows 3.1 Øecká
1254 Windows 3.1 Turecká
1255 Hebrejská
1256 Arabská
1257 Baltská
1361 Korejská (Johab)
10000 Macintosh Roman
10001 Macintosh Japonská
10006 Macintosh Øecká I
10007 Macintosh Cyrilice
10029 Macintosh Latin 2
10079 Macintosh Islandská
10081 Macintosh Turecká