Vážení zákazníci,

dovolujeme si Vás upozornit, že na tuto ukázku knihy se vztahují autorská práva, tzv. copyright.

To znamená, že ukázka má sloužit výhradně pro osobní potřebu potenciálního kupujícího (aby čtenář viděl, jakým způsobem je titul zpracován a mohl se také podle tohoto, jako jednoho z parametrů, rozhodnout, zda titul koupí či ne).

Z toho vyplývá, že není dovoleno tuto ukázku jakýmkoliv způsobem dále šířit, veřejně či neveřejně např. umisťováním na datová média, na jiné internetové stránky (ani prostřednictvím odkazů) apod.

redakce nakladatelství BEN – technická literatura
redakce@ben.cz

8 Použití souborů INI a RES

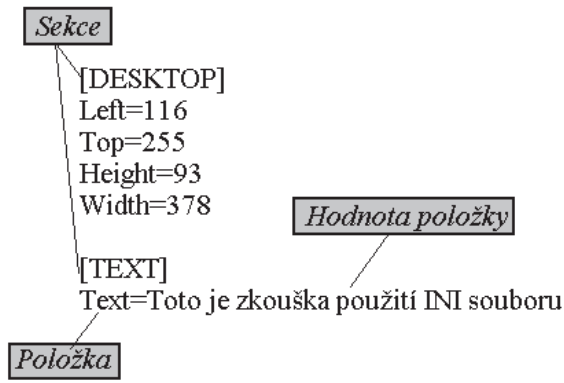
Třebaže je tato kapitola poměrně krátká, obsahuje popis často používaných technik.

8.1 Soubory INI

Přípona (INI) těchto souborů výstižně určuje jejich funkci. Tyto soubory se používají k *inicializaci* aplikace.

Aplikace si při svém spuštění z INI souboru načte například rozměry svého formuláře, cestu k souborům nebo jiné informace. Na konci běhu aplikace zase tyto informace ukládá zpět do INI souboru, aby je při opětovném spuštění mohla použít.

INI soubor sestává ze sekcí (sections) (poznáme je podle toho, že jejich názvy jsou psány mezi **[ ]**), sekce obsahují položky (keys) (poznáme je podle toho, že jejich názvy jsou následovány =) a hodnoty položek (values) (poznáme je podle toho, že jejich názvům předchází =). Viz obr. 8.1.



Obr. 8.1 Význam jednotlivých částí INI souboru

Práce s INI soubory v C++ Builderu je velmi snadná. C++ Builder totiž definuje třídu **TIniFile**, jejíž metody řeší vše potřebné za nás. Jediné co je třeba, je seznámit se s touto třídou.

8.1.1 Třída TIniFile

TIniFile je definována v knihovně: **vcl\inifiles.hpp**.

Datové položky

– **AnsiString FileName** (R/O) název INI souboru, na který se odvoláváme instancí TIniFile.

Metody

- **__fastcall TIniFile(const AnsiString FileName);** konstruktor, **FileName** je jméno INI souboru,
- **bool __fastcall ReadBool(AnsiString Section, AnsiString Ident, bool Default);** načte ze sekce **Section** hodnotu položky **Ident**, pokud se položka nevyskytuje, vezme hodnotu **Default**,
- **long __fastcall ReadInteger(AnsiString Section, AnsiString Ident, long Default);** to samé pro int,
- **AnsiString __fastcall ReadString(AnsiString Section, AnsiString Ident, AnsiString Default);** to samé pro AnsiString,
- **void __fastcall WriteBool(AnsiString Section, AnsiString Ident, bool Value);** zapíše do sekce **Section** položku **Ident** s hodnotou **Value**,
- **void __fastcall WriteInteger(AnsiString Section, AnsiString Ident, long Value);** to samé pro int,
- **void __fastcall WriteString(AnsiString Section, AnsiString Ident, AnsiString Value);** to samé pro AnsiString,
- **void __fastcall DeleteKey(AnsiString Section, AnsiString Ident);** vymaže ze sekce **Section** položku **Ident**,
- **void __fastcall EraseSection(AnsiString Section);** vymaže sekci **Section**,
- **void __fastcall ReadSections(TStrings* Strings);** načte názvy všech sekcí do seznamu **Strings**,
- **void __fastcall ReadSection(AnsiString Section, TStrings* Strings);** načte názvy všech položek sekce **Section** do seznamu **Strings**,
- **void __fastcall ReadSectionValues(AnsiString Section, TStrings* Strings);** načte položky sekce **Section** včetně jejich hodnot do **Strings**.

8.1.2 Příklad

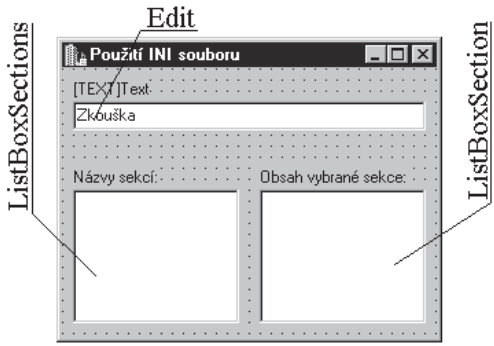
Spusťte C++ Builder, do formuláře umístěte komponenty dle nákresu na obr. 8.2 (komponenty Label není nutné pojmenovat).

Pomocí objekt inspektoru (karta Events) vygenerujte funkce: **FormCreate**, **FormClose** (události OnCreate a OnClose formuláře) a **ListBoxSectionsClick** (událost OnClick komponenty ListBoxSections).

Upravte obsahy zdrojových souborů podle níže uvedených výpisů.

UNIT1.H:

```
.
.
#include <vcl\inifiles.hpp>
//---------------------------------
class TForm1 : public TForm
{
__published:    // IDE-managed Components
.
.
```



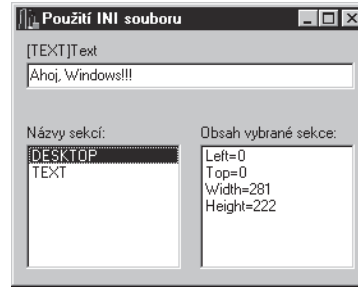
Obr. 8.2 Návrhový formulář

```
void __fastcall ListBoxSectionsClick(TObject *Sender);
private:  // User declarations
TIniFile *IniFile;
public:         // User declarations
__fastcall TForm1(TComponent* Owner);
__fastcall ~TForm1();
};
//---------------------------------
extern TForm1 *Form1;
//---------------------------------
#endif
UNIT1.CPP:
.
.
__fastcall TForm1::TForm1(TComponent* Owner)
: TForm(Owner)
{
     IniFile=new TIniFile("INIFILE.INI")[1];
}
//---------------------------------
__fastcall TForm1::~TForm1()
{
     delete(IniFile);
}
//---------------------------------
void __fastcall TForm1::FormCreate[2](TObject *Sender)
{
```

[1] Otevři soubor INIFILE.INI z adresáře WINDOWS (.INI soubory se standardně umísťují do adresáře WINDOWS).

[2] Událost vytvoření formuláře.

```
     //nastavení hodnot podle obsahu INI souboru:
     Left=IniFile->ReadInteger("DESKTOP","Left",Left);
     Top=IniFile->ReadInteger("DESKTOP","Top",Top);
     Height=IniFile->ReadInteger("DESKTOP","Height",Height);
     Width=IniFile->ReadInteger("DESKTOP","Width",Width);
     Edit->Text=IniFile->ReadString("TEXT","Text",Edit->Text);
     IniFile->ReadSections[1]ListBoxSections->Items);
}
//---------------------------------
void __fastcall TForm1::FormClose[2](TObject *Sender, TCloseAction
&Action)
{
     IniFile->WriteInteger("DESKTOP","Left",Left);
     IniFile->WriteInteger("DESKTOP","Top",Top);
     IniFile->WriteInteger("DESKTOP","Width",Width);
     IniFile->WriteInteger("DESKTOP","Height",Height);
     IniFile->WriteString("TEXT","Text",Edit->Text);
}
//---------------------------------
void __fastcall TForm1::ListBoxSectionsClick[3](TObject *Sender)
{
     ListBoxSection->Clear()[4];
     IniFile->ReadSectionValues[5](
          ListBoxSections->Items
          ->Strings[ListBoxSections->ItemIndex],
          ListBoxSection->Items);
}
```



Obr. 8.3 Formulář jsem umístil přesně do levého horního rohu obrazovky a do editačního pole jsme napsal text

[1] Načtení názvů sekcí do ListBoxSections.

[2] Událost konce aplikace, ulož změny do INI souboru.

[3] Událost výběru položky z ListBoxSections.

[4] Nejdříve smaž staré položky.

[5] Nahraj názvy položek a jejich hodnoty do ListBoxSection.

Přeložte, zkuste měnit polohu formuláře a text v Edit.

Uložte, po novém spuštění si aplikace zachová předchozí nastavení.

Dále si ověřte, že v adresáři WINDOWS vznikl soubor INIFILE.INI (prohlédněte si jej textovým editorem).

Kontrolní výpis obsah souboru INIFILE.INI:

```
[DESKTOP]
Left=0
Top=0
Width=281
Height=222
[TEXT]
Text=Ahoj, Windows!!!
```

8.2 Soubory RES

Úlohu RES souborů jsme stručně popsali již v kapitole 4. Na obr. 4.6 je naznačeno, že resource jsou posledním souborem, ze kterého se spojuje cílový kód aplikace C++ Builderu.

Co vlastně RES soubor obsahuje? Dá se to napsat jednou větou: *RES soubor obsahuje zdroje*. Zdrojem se v našem případě rozumí bitová mapa, kurzor nebo ikona.

Proč se ale RES soubory používají, vždyť všechno výše uvedené lze do programu „dostat" i pomocí standarních souborů (.bmp, .cur, .ico)? Ano, ale je zde podstatný rozdíl! Standardní soubory lze použít za běhu aplikace (takže musí být rozhodně na disku spolu s cílovým kódem aplikace), RES soubory se připojí do cílového kódu souboru a pro běh aplikace je již nepotřebujeme.

8.2.1 Vytvoření resource

RES soubory budeme vytvářet pomocí Image Editoru, jeho spuštění je snadné: Vyberte položku menu **Tools|Image Editor**, spustí se Image Editor, zde vyberte položku menu **File|New|Resource**.

Obr. 8.4 Volba nového elementu pro vložení do resource

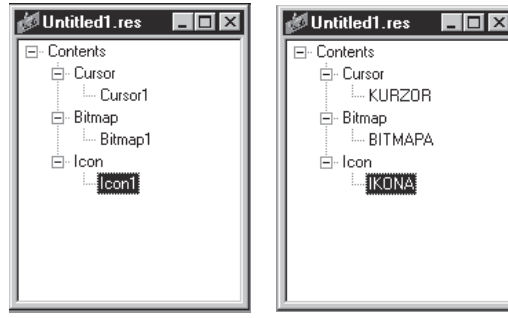
Zásady práce v Image Editoru si rychle osvojíme, protože jsou prakticky stejné, jako v každém jiném grafickém editoru.

Nejdříve klikneme pravým tlačítkem myši na **Contents**, objeví se popup menu v němž můžeme provést výběr elementu, který chceme vytvořit. Viz obr. 8.4.

Vytvořte tedy po jednom elementu bitové mapy, kurzoru a ikony. Všimněte si, že Image Editor přehledně řadí elementy do stromu, podle jejich typu.

Když kliknete pravým tlačítkem, je popup menu bohatší, význam jeho nových položek je snad dostatečně jasný: **Edit** (umožní editovat element), **Rename** (změní jméno elementu), **Delete** (vymaže element).

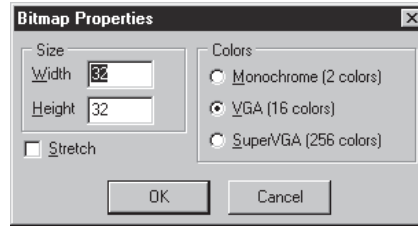
Zkuste pomocí Rename změnit jména na Bitmapa, Kurzor a Ikona[1].



Obr. 8.5 Strom před a po změně jmen elementů

Zkusme nejdříve pracovat s bitovou mapou, zde je situace nejjednodušší. Zvolte tedy z menu **Edit**, nyní můžete kreslit běžným způsobem (bitmapu lze pomocí schránky přetáhnout i z jiné aplikace).

Povšimnutí si zaslouží pouze položka menu **Bitmap|Image Properties**. Touto položkou menu lze změnit rozměry bitové mapy a její barevné rozlišení. Tento dialog se zobrazuje rovněž při vytváření bitové mapy.



Obr. 8.6 Dialogové okno pro změnu vlastností bitové mapy

[1] Velká a malá písmena nehrají roli.

Rovněž i kreslení ikony je běžnou záležitostí. Ale pozor, protože ikona zůstává na pracovní ploše obrazovky, je umožněno volit transparentní a inverzní barvu. V praxi to znamená, že ikona nemůsí být vždy vidět, protože pozadí může mít nevhodnou barvu.

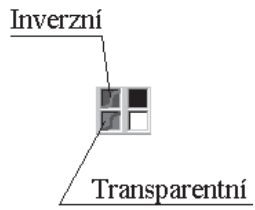
Pokud máme pochybnosti, je velice jednoduché ikonu otestovat pomocí dialogu, který se zobrazí po výběru položky menu **Icon**|**Test**. Zde lze volit barvu pozadí, ovládání je ryze intuitivní.

Nejsložitější se jeví kreslení kurzoru. Je možno volit černou, bílou, transparentní nebo inverzní barvu, viz obr. 8.7.

Pro snadnou zkoušku lze vyvolat položku menu **Cursor**|**Test**, vytvoří se dialog, který umožní provést test.

Důležitá je také definice koncového bodu[1], která se provede pomocí položky menu **Cursor**|**Set Hot Spot**.

Zkuste tedy nakreslit tyto tři elementy a uložit celý RES soubor.



Obr. 8.7 Volba barev kurzoru

8.2.2 Použití resource v aplikaci

Vidíme, že samotné vytvoření RES souboru je hračkou, teď ale zbývá odpovědět na otázku, jak ve zdrojovém kódu sdělíme, že chceme daný element z RES souboru použít.

Tento problém lze vtipně rozdělit na dva dílčí:

1) musíme překladači sdělit název RES souboru, který chceme použít,
2) ve vhodném okamžiku musíme daný element „nahrát“ za pomoci k tomu určených funkcí.

První problém se řeší pomocí direktivy **#pragma**:

```
#pragma resource "název_RES_souboru"
```

Do úvozovek se tedy napíše název RES souboru, který chceme použít (samozřejmně, že pokud není v aktuálním adresáři, musí se zapsat plná cesta).

Druhý problém můžeme vyřešit pomocí dalších funkcí Windows API. Funkce jsou tři a jejich názvy se zdají být dostatečně jasné:

```
HBITMAP LoadBitmap(HINSTANCE hInstance, LPSTR lpBitmapName);
HICON LoadIcon(HINSTANCE hInstance, LPSTR lpIconName);
HCURSOR LoadCursor(HINSTANCE hInstance, LPSTR lpCursorName);
```

[1] Tento bod je rozhodující při aktivaci myší. Obvykle se logicky umísťuje do vrcholu obrazce, který představuje kurzor.

Každá z funkcí vyžaduje dva parametry:

- **hInstance** je handle na instanci aplikace, obvykle se na místo tohoto parametru zapisuje externí proměnná C++ Builderu s označením **HInstance**, takto je definována:
  `extern HINSTANCE HInstance;`
- **LPSTR** je jméno elementu tak, jak jsme jej zapsali v Image Editoru[1].
- *Funkce vrací* handle na daný objekt, pokud element daného jména existuje, jinak vrací **NULL**.

8.2.3 Příklad

Spusťte C++ Builder, pomocí Image Editoru nakreslete elementy popsané v kapitole 8.2.1 a uložte je do souboru RESOURCE.RES (do stejného adresáře jako zdrojové soubory aplikace).

Potom pomocí objekt inspektoru (karta Events) vygenerujte funkce **FormCreate**, **FormPaint**, **FormResize** (události OnCreate, OnPaint, OnResize formuláře). O více se nemusíte starat, pouze upravte zdrojové soubory podle níže uvedených výpisů.

UNIT1.H:

```
.
.
class TForm1 : public TForm
{
__published:    // IDE-managed Components
void __fastcall FormCreate(TObject *Sender);
void __fastcall FormPaint(TObject *Sender);
void __fastcall FormResize(TObject *Sender);
private:  // User declarations
Graphics::TBitmap *Bitmap;
public:         // User declarations
__fastcall TForm1(TComponent* Owner);
__fastcall ~TForm1();
};
//---------------------------------
extern TForm1 *Form1;
//---------------------------------
#endif
```

UNIT1.CPP:

```
//---------------------------------
#include <vcl\vcl.h>
#pragma hdrstop
#include "Unit1.h"
//---------------------------------
#pragma resource "*.dfm"
#pragma resource "resource.res"
TForm1 *Form1;
```

[1] Nerozlišují se velká a malá písmena.