Hrátky s operačním systémem

V této kapitole si ukážeme některé postupy, s jejichž pomocí je možné měnit určitá nastavení operačního systému, manipulovat s objekty pracovní plochy apod.

Při psaní některých programů se můžete dostat do situace, kdy budete chtít změnit některá uživatelská nastavení *Windows*. Tím myslím například změnu tapety, nastavení rychlosti klávesnice, přepnutí klávesnice apod.

Tato kapitola obsahuje několik zajímavých tipů, jak podobné operace uskutečnit. A pro začátek se zaměříme na hlavní panel *Windows*.

Hlavní panel

Hlavní panel je součást *Windows*, která je důvěrně známá všem uživatelům tohoto operačního systému. Bývá umístěn u některého z okrajů obrazovky a obsahuje kromě velice užitečného tlačítka **Start** řadu tlačítek určených k přepínání jednotlivých spuštěných aplikací.

Ukrytí hlavního panelu

V určitých situacích ovšem může hlavní panel na obrazovce překážet a bylo by užitečné umět jej ukrýt. V případě, že byste se i vy do podobné situace dostali, můžete použít následující postup, který využívá API funkci **FindWindow**. Tato funkce hledá v systému okno se stejným názvem třídy nebo okna, jaké je zadáno v jejích parametrech. Tyto parametry jsou celkem dva:

- **lpClassName** – Jméno třídy, která odpovídá hledanému oknu. Pokud má tento parametr hodnotu **nil**, vyhledává se okno podle jeho názvu.
- **lpWindowName** – Jméno okna, které se má vyhledat. Často jméno okna odpovídá textu v záhlaví, ale není to pravidlo a nelze na to spoléhat.

Funkce vrací handle hledaného okna. Pokud okno nebylo nalezeno, vrací nulu. Když si ještě prozradíme, že jméno třídy hlavního panelu je **Shell_TrayWnd,** můžeme napsat kód, který jej skryje:

```
...
procedure TForm1.Button1Click(Sender: TObject);
var Wnd: THandle;
begin
  Wnd:= FindWindow('Shell_TrayWnd', nil);
                                  //nalezení hlavního panelu

  if Wnd <> 0 then
                        //pokud bylo okno nalezeno skrýt jej
       ShowWindow(Wnd, SW_HIDE);

end;
...
```

Ke skrytí okna je použita API funkce **ShowWindow**, která zobrazuje nebo naopak skrývá zadané okno. Její parametry mají následující význam:

- **hWnd** – Handle okna, které se má skrýt nebo zobrazit.
- **nCmdShow** – Parametr určuje, co se má s oknem přesně udělat. Následující tabulka přehledně uvádí všechny hodnoty, které mohou být dosazeny za tento parametr.

Hodnota	*Význam*
SW_HIDE	Skryje okno.
SW_MAXIMIZE	Maximalizuje zadané okno.
SW_MINIMIZE	Minimalizuje okno a aktivuje další okno v pořadí.
SW_RESTORE	Aktivuje a zobrazí zadané okno. Pokud je okno minimalizované nebo maximalizované, obnoví jeho původní velikost.
SW_SHOW	Aktivuje okno a zobrazí ho v jeho aktuální velikosti a pozici.
SW_SHOWDEFAULT	Zobrazí okno ve velikosti a pozici, kterou mělo při vytvoření.
SW_SHOWMAXIMIZED	Aktivuje okno a zobrazí je maximalizované.
SW_SHOWMINIMIZED	Aktivuje okno a zobrazí je minimalizované.
SW_SHOWMINNOACTIVE	Zobrazí okno v minimalizované podobě, ale nezaměří je. To znamená, že aktivní okno zůstane i nadále zaměřené.
SW_SHOWNA	Zobrazí okno v aktuální velikosti a pozici, ale nezaměří je.
SW_SHOWNOACTIVATE	Zobrazí okno v poslední nastavené velikosti, ale nezaměří ho.
SW_SHOWNORMAL	Aktivuje a zobrazí okno. Pokud je okno minimalizované nebo maximalizované, funkce obnoví jeho původní velikost.

Skrytí části hlavního panelu

Pomocí metody **ShowWindow** však lze ukrýt jenom část hlavního panelu, například jenom tlačítka aplikací, nebo tlačítko **Start** nebo systémové ikony v pravém rohu hlavního panelu. Je to možné proto, že všechny tyto části jsou samostatnými dceřinými okny hlavního panelu. Proto stačí zjistit jejich handle a zavolat pro ně funkci **ShowWindow** s příslušným parametrem.

Ovšem na vyhledání dceřiných oken je funkce **FindWindow** krátká a budeme muset použít funkci **FindWindowEx**.Tato funkce opět vrací handle nalezeného okna, ale tentokrát bude nutné zadat hned čtyři parametry:

- **Parent** – Handle okna, jehož dceřinná okna se mají prohledávat.
- **Child** – Handle dceřiného okna, od kterého se má začít s hledáním. Pokud je tento parametr nula, prohledávání začne u prvního dceřiného okna.

- **ClassName** – Jméno třídy hledaného okna.
- **WindowName** – Jméno hledaného okna.

Mají-li se například skrýt ikony v pravé části hlavního panelu, musí se vyhledat okno s názvem třídy **TrayNotifyWnd**:

```
...
procedure TForm1.ButtonClick(Sender: TObject);
var Wnd: THandle;
begin
  Wnd:= FindWindow('Shell_TrayWnd', nil);
  Wnd:= FindWindowEx(Wnd, HWND(0), 'TrayNotifyWnd', nil);

  if Wnd <> 0 then
     ShowWindow(Wnd, SW_HIDE);
end;
...
```

O tom, že lze opakovaným voláním prohledávat okna i na další úrovni, nejlépe svědčí kód, který ukryje systémové hodiny na hlavním panelu. Připomeňme jen, že třída tohoto okna se jmenuje **TrayClockWClass** :

```
...
procedure TForm1.ButtonClick(Sender: TObject);
var Wnd: THandle;
begin
  Wnd:= FindWindow('Shell_TrayWnd', nil);
  Wnd:= FindWindowEx(Wnd, HWND(0), 'TrayNotifyWnd', nil);
  Wnd:= FindWindowEx(Wnd, HWND(0), 'TrayClockWClass', nil);

  if Wnd <> 0 then
     ShowWindow(Wnd, SW_HIDE);
end;
...
```

*Všechna okna, která byla skryta tímto způsobem, je možné znovu zobrazit volání funkce **ShowWindow** s parametrem **SW_SHOW**.*

Pracovní plocha

Protože budeme pracovat s plochou, měli bychom si ji nastavit tak, aby se nám co nevíce líbila. A proto si jako první ukážeme, jak změnit nastavení tapety.

Změna pozadí pracovní plochy

Natavení tapety na pracovní ploše lze provést pomocí API funkce **SystemParametrsInfo**. Tato funkce slouží k nastavení nebo zjištění stavu mnoha systémových nastavení. Co bude

přesně dělat, závisí na hodnotě jejího prvního parametru. V našem případě bude mít tento parametr hodnotu **SPI_SETDESKWALLPAPER**, další dva parametry mají v tomto konkrétním případě hodnotu 0, jméno obrázku, který se má použít jako tapeta, a poslední parametr bude opět nula.

Zde je kód, který při stisku tlačítka umožní vybrat obrázek pomocí komponenty **OpenPictureDialog1** a nastavit ho na pozadí pracovní plochy:

```
...
procedure TForm1.ButtonClick(Sender: TObject);
begin
  if OpenPictureDialog1.Execute then
    SystemParametersInfo(SPI_SETDESKWALLPAPER, 0,
                    PChar(OpenPictureDialog1.FileName), 0);
end;
...
```

Skrytí ikon na pracovní ploše

Nová tapeta je nastavená a na pracovní ploše jí to moc sluší, ovšem dokonalý estetický zážitek kazí ikony rozmístěné na pracovní ploše. Proto si nyní ukážeme, jak je ukrýt, aby nepřekážely.

Kdo tuší, že použijeme stejný postup jako v případě ukrývání hlavního panelu, tuší správně. Ikony na pracovní ploše jsou zobrazeny v okně, jehož třída se jmenuje **ShellDll_DefView,** a které je dceřiným oknem třídy **Progman**:

```
...
procedure TForm1.Click(Sender: TObject);
var Wnd: THandle;
begin
  Wnd:= FindWindow('Progman', nil);
  Wnd:= FindWindowEx(Wnd, HWND(0), 'ShellDll_DefView', nil);

  if Wnd <> 0 then
     ShowWindow(Wnd, SW_HIDE);
end;
...
```

Zjištění rozměrů pracovní plochy

Způsobů, jak zjistit nastavené rozlišení pracovní plochy, je hned několik. Ale ten nejjednodušší, a podle mého názoru nejelegantnější, je použitím globální proměnné **Screen**. Tato proměnná je typu **TScreen**, vytváří se automaticky a je přístupná všem modulům v aplikaci. Třída **TScreen** obsahuje celou řadu zajímavých informací o obrazovce. Mezi jinými zde naleznete proměnné **Width** a **Height**, jejichž hodnoty odpovídají nastavené šířce a výšce pracovní plochy. Pokud tedy potřebujete vytvořit formulář tak, aby byl za každé okolnosti roztažený přes celou obrazovku, použijte následující kód:

```
...
procedure TForm1.FormCreate(Sender: TObject);
begin
  Width:= Screen.Width;        //nastavení rozměrů a polohy
                                                  //formuláře
  Height:=Screen.Height;
  Left:=0;
  Top:=0;
  FormStyle:=fsStayOnTop;       //nastavení stylu formuláře
end;
...
```

*Vlastnosti **Width** a **Height** třídy **TScreen** jsou pouze pro čtení, a proto je nelze použít ke změně rozlišení.*

Zachycení změny rozlišení

V případě, že by měla velikost formuláře vždy kopírovat velikost pracovní plochy, je nutné počítat s tím, že se může rozlišení změnit i během činnosti programu. V takovém případě by se o tom jistě měl náš formulář dozvědět.

Řešení tohoto problému spočívá ve vytvoření obsluhy zprávy **WM_DISPLAYCHANGE**, kterou operační systém posílá oknům, pokud se změní rozlišení. Do definice formuláře je tedy nutné přidat následující metodu:

```
...
  public
    procedure WMDisplayChange(var Msg : TWMDisplayChange);
message WM_DISPLAYCHANG;
...
```

V těle této funkce je opět nutné změnit velikost a polohu formuláře podle nového rozlišení. Také se nesmí zapomenout na volání funkce předka:

```
...
procedure TForm1.WMDisplayChange(var Msg: TWMDisplayChange);
begin
  inherited;
  Width:= Screen.Width;
  Height:=Screen.Height;
  Left:=0;
  Top:=0;
end;
...
```

Globální proměnná Screen

Třída **TScreen**, která nám tolik pomohla při zjišťování rozlišení, obsahuje ještě mnoho dalších vlastností, které se týkají obrazovky, ale i věcí, které byste možná na první pohled nečekali. Pomocí proměnné **Screen** můžete zjistit, kolik je na obrazovce formulářů, kolik je v systému instalovaných fontů apod. Následující tabulka ukazuje přehled nejdůležitějších vlastností třídy **TScreen**.

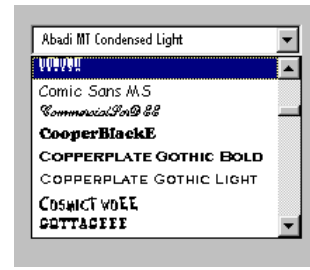
Vlastnost	*Význam*
ActiveControl	Obsahuje referenci na prvek, který je v danou chvíli aktivní.
ActiveCustomForm	Obsahuje referenci na potomka třídy **TCustomForm**, který je aktivní.
ActiveForm	Obsahuje referenci na aktivní formulář.
Cursor	Kurzor myši.
Cursors	Pole dostupných kurzorů, které může aplikace používat.
CustomFormCount	Počet formulářů a editorů vlastností zobrazených na obrazovce.
CustomForms	Seznam referencí na zobrazené formuláře.
DefaultKbLayout	Obsahuje handle na rozložení klávesnice, které bylo aktivní v okamžiku, kdy byla aplikace spuštěna.
DesktopHeight	Výška virtuální pracovní plochy. S virtuálními pracovními plochami se pracuje ve vícemonitorových systémech.
DesktopLeft	Souřadnice levého okraje virtuální pracovní plochy.
DesktopRect	Oblast ohraničující virtuální pracovní plochu.
DesktopTop	Horní okraj virtuální pracovní plochy.
DesktopWidth	Šířka virtuální pracovní plochy.
Fonts	Seznam jmen fontů instalovaných v systému.
FormCount	Počet formulářů zobrazených na obrazovce.
Forms	Seznam referencí na formuláře zobrazené na obrazovce.
Height	Výška obrazovky.
HintFont	Font používaný pro bublinovou nápovědu.
IconFont	Font používaný pro popis ikon.
MenuFont	Font používaný v nabídkách.
MonitorCount	Počet použitých monitorů.
Monitors	Seznam použitých monitorů.
Width	Šířka obrazovky.

Rozvírací nabídka s fonty

V následujícím příkladu si ukážeme další praktické použití proměnné **Screen**. V tomto případě nám poslouží jako zdroj informací o jménech nainstalovaných fontů, které umístíme do rozevírací nabídky. Vytvoření seznamu je velmi jednoduché a při vytváření formuláře, na kterém je nabídka umístěna, stačí provést následující kód:

```
...
procedure TForm1.FormCreate(Sender: TObject);
begin
   with ComboBox1 do
    begin
      Items:=Screen.Fonts;
      Style:=csDropDownList;
       ItemIndex := 0;
    end;
end;
...
```

Aby byl příklad o něco zajímavější, upravíme nyní nabídku tak, aby jednotlivé názvy fontů v seznamu byly napsány příslušným fontem, tak jak to bývá zvykem u některých textových editorů.



Nejprve je nutné nastavit vlastnost **Style** u rozevírací nabídky na hodnotu **csOwnerDrawFixed**:

```
...
procedure TForm1.FormCreate(Sender: TObject);
begin
   with ComboBox1 do
    begin
       ...
      Style:=csOwnerDrawFixed;
    end;
end;
...
```

Tím jsme si zajistili, že při každém překreslení položky seznamu bude vyvolána událost **OnDraw** třídy **TComboBox**. V této metodě můžeme změnit použitý font a vykreslit jméno fontu na určené místo:

```
...
procedure TForm1.ComboBox1DrawItem(Control: TWinControl;
Index: Integer;
                  Rect: TRect; State: TOwnerDrawState);

var combo:TComboBox;
    oldPColor, oldBColor:TColor;
      oldFont: TFont;
```

```
begin
  combo:=(Control as TComboBox);
                    //přetypovat ovládací prvek na ComboBox
  if combo = nil then
     exit;

  oldFont:=combo.Font;            //ulozit nastavení plátna
  oldBColor:= combo.Canvas.Brush.Color;
  oldPColor:= combo.Canvas.Pen.Color;

  combo.Canvas.Pen.Color:=clWhite;    //nastavit barvu pera

  if odSelected in State then              //nastavit barvu
                                           //pozadí polozky
    combo.Canvas.Brush.Color:=clActiveCaption
  else
    combo.Canvas.Brush.Color:=clWhite;

  combo.Canvas.Rectangle(Rect);  //vykreslit pozadí polozky

  combo.Canvas.Font.Name:= combo.Items[ Index] ;
                                //nastavit font podle jména

                                        //vykreslit polozku
  combo.Canvas.TextOut(
              Rect.Left+5, Rect.Top, combo.Items[ Index] );

  combo.Canvas.Brush.Color:= oldBColor;
                                //obnovit původní nastavení
  combo.Canvas.Pen.Color:= oldPColor;
  combo.Font:=oldFont;
end;
...
```

V případě nějakých nejasností porovnejte uvedený kód s kódem u vytváření uživatelem překreslovaných nabídek v páté kapitole.

Vypnutí a restart počítače

Programátoři programující aplikace, které provádějí časově náročné operace, nebo instalační programy apod. jistě využijí funkci, která dokáže vypnout nebo restartovat počítač, případně odhlásit uživatele.

Všech těchto akcí lze dosáhnout pomocí jediné API funkce, která se jmenuje **ExitWindowsEx**. Chování této funkce se mění podle zadaných parametrů. Jsou dva. Na činnost funkce má však vliv jen první z nich:

- **uFlags** – Parametr obsahuje příznak nebo kombinaci příznaků specifikujících, jakým způsobem mají být Windows ukončeny. Následující tabulka udává přehled možných hodnot:

Hodnota	*Význam*
EWX_LOGOFF	Ukončí všechny běžící procesy a odhlásí aktuálního uživatele.
EWX_POWEROFF	Ukončí *Windows* a vypne počítač.
EWX_REBOOT	Funkce vypne systém a restartuje *Windows* (nikoli počítač).
EWX_SHUTDOWN	Vypne systém a připraví jej tak, aby bylo možné počítač bezpečně vypnout. To znamená, že veškeré informace byly uloženy na disk a všechny běžící procesy byly zastaveny.

V kombinaci s předchozími hodnotami se mohou v parametru objevit ještě tyto příznaky:

Hodnota	*Význam*
EWX_FORCE	Funkce vypne běžící procesy, aniž by je o tom nějak informovala a dala jim možnost operaci přerušit. Tento postup může ale způsobit ztrátu dat, a proto byste jej měli používat pouze ve skutečně opodstatněných případech.
EWX_FORCEIFHUNG	Tento příznak zajistí, že jsou násilím ukončeny procesy, které nereagují na požadavek na ukončení. Pokud je současně nastaven příznak **EWX_FORCE**, je tento příznak ignorován (příznak funguje pouze ve *Windows 2000* a vyšších).

- **dwReserved** – Druhý parametr funkce je ignorován.

Následující kód způsobí vypnutí počítače s tím, že všechny procesy dostanou příležitost uložit všechna data a nereagující procesy budou ukončeny násilím:

```
...
procedure TForm1.Button2Click(Sender: TObject);
begin
  if MessageDlg('Opravdu chcete vypnout počítač?',
        mtConfirmation, [mbYes, mbNo], 0) = mrYes then

    ExitWindowsEx(EWX_POWEROFF or EWX_FORCEIFHUNG, 0);
end;
...
```

*Funkce **ExitWindowsEx** bude pracovat pochopitelně jen tehdy, pokud počítač, na kterém program běží, umožňuje softwarové vypínání.*

V programech používajících *Windows NT* nebo *Windows 2000* je navíc možné používat funkci **LockWorkStation**, která nemá žádné parametry a zamkne pracovní stanici.

Automatické spuštění programu

Následující příklad navazuje na ten předchozí, protože když nějaká aplikace potřebuje během své činnosti restart počítače, bylo by dobré, aby se po restartu sama spustila a mohla pokračovat v započaté práci.

Tuto funkci, známou z instalačních programů, je možné celkem snadno naprogramovat s využitím registrů *Windows*. Autoři *Windows* totiž s podobnou možností počítali a přidali do systémového registru klíč, do něhož je možné zapsat jména programů, které se mají spustit po startu *Windows*.

Tento klíč se nachází v části **HKEY_CURRENT_USER** a má název: **SOFTWARE\Microsoft\Windows\CurrentVersion\Run**. Do něj je možné zapisovat některým způsobem popsaným v kapitole šest.

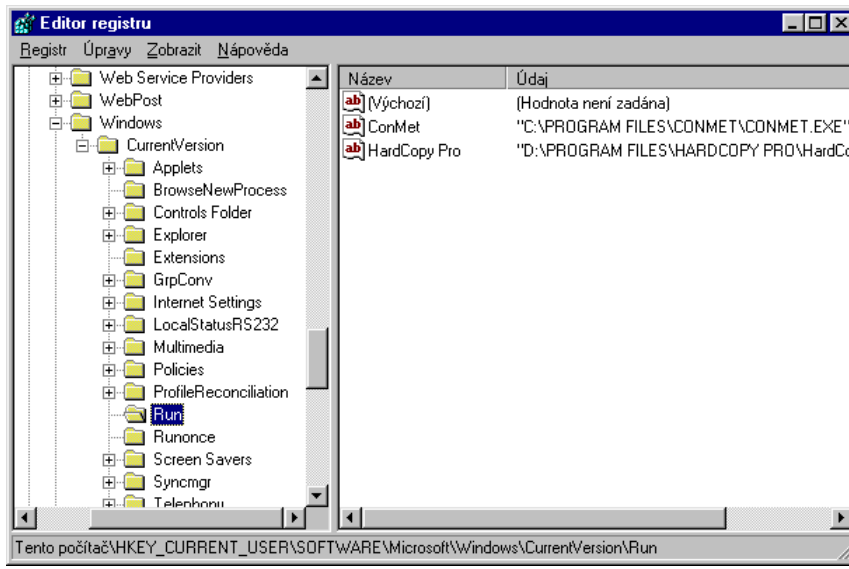
Jestliže se program nemá spouštět vždy, ale pouze při příštím startu, je výhodnější použít klíč registru s názvem **SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce**. Po startu *Windows* jsou totiž spuštěny všechny programy zapsané v tomto klíči, a potom je obsah klíče automaticky vymazán, takže se nemusíte o nic starat.

Následující kód ukazuje, jak může program restartovat počítač a po novém naběhnutí systému pokračovat ve své práci:

```
...
procedure TForm1.ButtonClick(Sender: TObject);
var
  Reg: TRegistry;
begin
  if MessageDlg('Opravdu chcete vypnout počítač?',
        mtConfirmation, [mbYes, mbNo], 0) = mrYes then
  begin
    Reg := TRegistry.Create;
    try
      Reg.RootKey := HKEY_CURRENT_USER;
      if Reg.OpenKey(
       '\Software\Microsoft\Windows\CurrentVersion\RunOnce',
          True) then
          begin
            Reg.WriteString('MyApp','"'
            + Application.ExeName + '„');
            Reg.CloseKey;
          end;
    finally
      Reg.Free;
    end;

   ExitWindowsEx(EWX_POWEROFF, 0);

  end;
end;
...
```



Aplikace uvedené v tomto klíči se spustí automaticky při každém startu Windows.