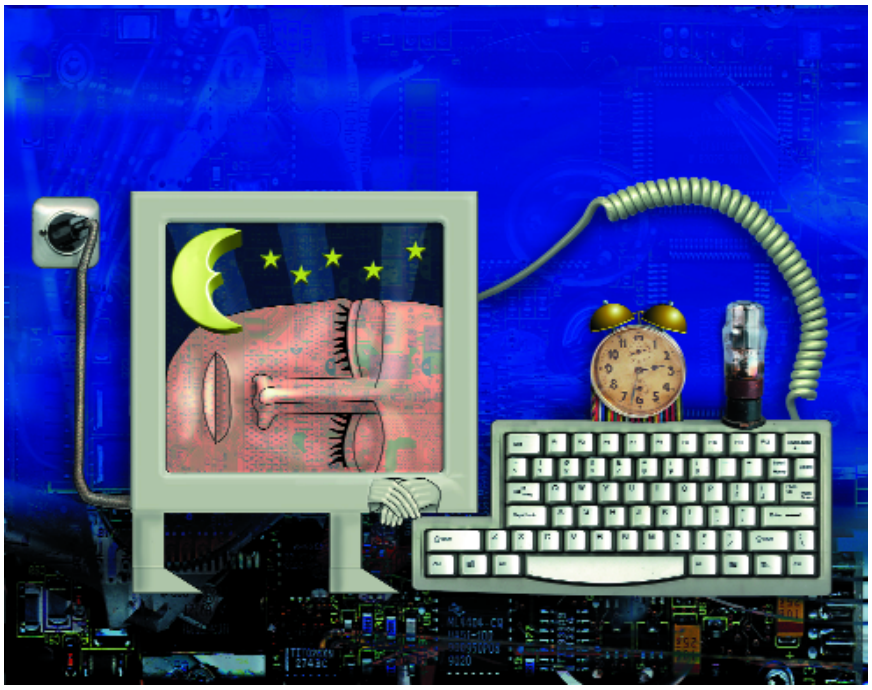


marek tomasik

Wygaszacz ekranu może zrobić każdy

Śpij spokojnie...

Wygaszacze ekranu bywają niedocenianym elementem naszej komputerowej codzienności. Niewielkim nakładem pracy można to jednak zmienić – screensaver stanie się naszą dumą i chlubą (nie wspominając o umilaniu nam przerw w pracy).

Artur Janowski

Pierwotnym przeznaczeniem wygaszaczy była i jest do dziś ochrona monitora przed nadmiernym (i nierównomiernym) zużyciem warstwy luminoforu wyścielającego wewnętrzną część jego kineskopu. Sytuacja taka ma miejsce przy długotrwałej ekspozycji statycznych obrazów, a efekty jej szkodliwego działania spotęgują się przy dużym zróżnicowaniu kontrastowym poszczególnych elementów obrazu (np. białego prostokąta na czarnym tle). I właśnie w celu zminimalizowania ryzyka szybkiej utraty właściwości świetlnych monitora systemy operacyjne „pozwalają" wygaszaczom przejmować kontrolę nad obrazem przez niego wyświetlanym. Następuje to zwykle po pewnym okresie bezczynności komputera (a dokładniej braku akcji ze strony użytkownika).

Wraz z upływem czasu wygaszacze zaczęły spełniać i inne funkcje. Dziś już nikogo nie dziwią ich skomplikowane animacje, wykorzystywane coraz częściej w celach reklamowych. Ciekawa multimedialna prezentacja silniej bowiem oddziałuje na klienta oczekującego na obsłużenie w siedzibie firmy niż folder informacyjny w standardowej (czyli analogowej) postaci.

Osoby ceniące swoją prywatność oraz poufność danych również będą zadowolone, używając wygaszaczy. Ustawione hasło i odpowiednio krótki czas oczekiwania przez system na włączenie screensavera mogą zapobiec wielu niezręcznym sytuacjom. Nikt bowiem nie lubi ostentacyjnie blokować stacji roboczej, w momencie gdy opuszcza stanowisko pracy. Odpowiednio dobrany wygaszacz może wreszcie stanowić „tło" dla relaksujących rozmyślań w czasie przerwy na kawę czy inną niezbyt zdrową używkę.

Jak działa wygaszacz?

Screensaver w Windows jest w zasadzie zwykłą aplikacją. Aby system mógł koordynować jej pracę, program taki musi spełniać przynajmniej cztery założenia:

- rozszerzeniem nazwy pliku aplikacji powinno być SCR w miejsce standardowego EXE;
- wygaszacz musi być wyposażony w zabezpieczenie przed wielokrotnym uruchomieniem własnej instancji w tym samym czasie;
- „rasowy" wygaszacz powinien również umożliwiać własną konfigurację zapisywaną na stałe w systemie;
- przerywać działanie w wyniku zaistnienia jakiegokolwiek zdarzenia związanego z użyciem klawiatury bądź myszy.

Problem rozszerzenia nazwy pliku nie jest trudny do rozwiązania. Wystarczy je zmienić po kompilacji z **.exe** na **.scr**. Wymagający Czytelnicy mogą jednak wprost nakazać kompilatorowi Delphi tworzenie takiego pliku. W tym celu uzupełniamy kod pliku projektu dyrektywą kompilatora **{$E scr}** lub zmieniamy domyślne rozszerzenie w polu **Target file extension** (opcja **Project | Options**, zakładka **Application**).

Nieco trudniej jest zapobiec wielokrotnemu uruchomieniu się tego samego programu. Musi on mianowicie w jakiś sposób zakodować w systemie informacje o swoim działaniu. Jeśli taki „sygnał" odnajdzie kolejna uruchomiona instancja wygaszacza, powinna ona przerwać swoje działanie. Najprostszym rozwiązaniem wydaje się tu użycie tzw. muteksów – globalnych obiektów Windows. Są one reprezentowane przez uchwyty THandle i, co najważniejsze, dostępne dla wszystkich wątków uruchomionych aktualnie w systemie.

Funkcja tworząca mutex, CreateMutex(), wymaga następujących parametrów:

- wskaźnik (do zmiennej typu Psecurity_attributes) służący do opisu praw dostępu; przyjęcie ustawień standardowych uzyskuje się przez użycie wskaźnika pustego (Nil);
- wartość logiczna (typu Boolean), decydująca, czy wątek właśnie tworzący mutex ma być jego właścicielem (True) czy też nie (False);
- nazwa (typu Pchar) identyfikująca mutex w systemie.

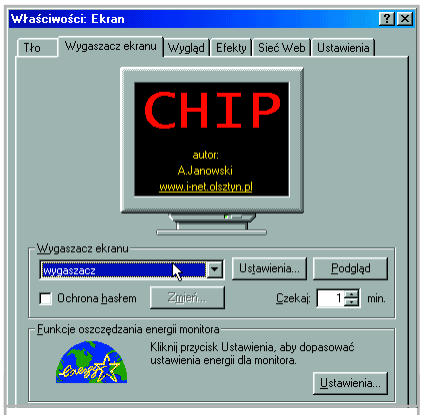
CreateMutex po stwierdzeniu braku muteksu o podanej nazwie w systemie tworzy go, a funkcja GetLastError zwraca wartość **0**. Jeśli mutex taki już istnieje, CreateMutex tworzy do niego nowy uchwyt, a funkcja GetLastError zwraca wartość ERROR_ALREADY_EXISTS.

Opisany sposób unikania wielokrotnego uruchamiania tego samego programu można zaimplementować na przykład tak:

146 »

```
program Project1;
uses
 Forms,
 Unit1 in 'Unit1.pas' {Form1},
 Unit2 in 'Unit2.pas' {Form2},
 Unit3 in 'Unit3.pas' {Form3},
 SysUtils,
 Windows;

{$R *.RES}
{$E scr}
var
Mutex:THandle;
begin
 Mutex:= CreateMutex(nil, True,
   'wygaszacz A.J.2002');
 if (Mutex <> 0) and
   (GetLastError = 0) then
 begin
 if pos('/S',UpperCase
   (ParamStr(1)))>0 then
 begin
 Application.Initialize;
 Application.Title:= 'Screen Saver
    A.Janowski 2002';
 Application.CreateForm(TForm2,
    Form2);
 Application.CreateForm(TForm1,
    Form1);
 end;
 if pos('/C',UpperCase
    (ParamStr(1)))>0 then
 begin
 Application.Initialize;
 Application.CreateForm(TForm1,
    Form1);
 Application.CreateForm(TForm2,
    Form2);
 Application.CreateForm(TForm3,
    Form3);
 end;
 if pos('/P',UpperCase
    (ParamStr(1)))>0 then
 begin
 Application.Initialize;
 Application.CreateForm(TForm3,
    Form3);
 end;
 Application.Run;
 if Mutex <> 0 then
    CloseHandle(Mutex);
 end;
end.
 Application.Run;
 if Mutex <> 0 then
    CloseHandle(Mutex);
 end;
end.
```



Nasz program musi WSPÓŁPRACOWAĆ Z MECHANIZMAMI WINDOWS.

Kod ten został nieco rozbudowany w stosunku do opisu, o czym za chwilę.

Dlaczego tyle okien?

Nasza aplikacja składa się z trzech formularzy: Form1 – okienka konfiguracji, Form2 – właściwego okna wygaszacza, oraz Form3 – okienka podglądu. Skąd więc to całe zamieszanie w kodzie pliku projektu omawianej aplikacji? Wynika ono z faktu, że screensaver ekranu powinien wykonywać różne czynności w zależności od wartości parametrów towarzyszących jego wywołaniu. I tak, jeśli wartość pierwszego parametru jest równa „/p”, oznacza to, że został on uruchomiony jako podgląd z okienka **Właściwości Ekranu** Windows. W takiej sytuacji wygaszacz może dokonać dowolnej swojej prezentacji na ekranie umownego małego monitora okienka Właściwości Ekranu. Z tego też względu przy wywołaniu aplikacji jako drugi parametr przekazywany jest do niego uchwyt „okienka z monitorem”. Uchwyt okienka pozwala na „podpięcie” procedury obsługi Form3.

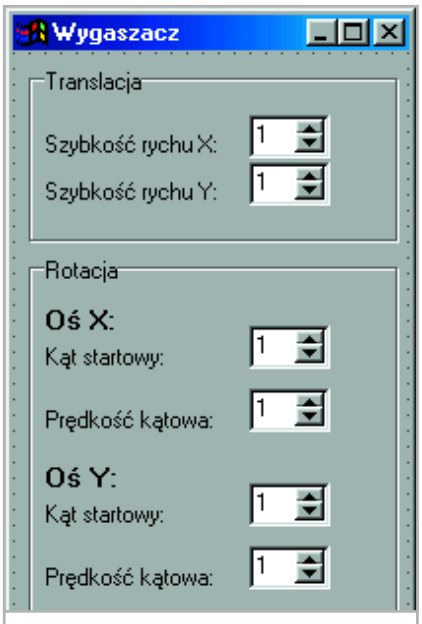
```
Procedure Tform3.FormCreate(Sender:
     TObject);
var
Preview: hWnd;
begin
 SetWindowLong(Application.
 Handle,GWL_EXSTYLE,WS_EX_TOOLWINDOW
  and not WS_EX_APPWINDOW or
  WS_EX_TOPMOST);
 Preview:= StrToIntDef
   (ParamStr(2), 0);
 SetWindowLong(self.handle,
   GWL_STYLE,
 GetWindowLong(Preview, GWL_STYLE)
    or WS_DISABLED or WS_CHILD);
 SetWindowLong(self.handle,
    GWL_EXSTYLE,
 GetWindowLong((Preview),
    GWL_EXSTYLE));
 Windows.SetParent(self.handle,
    Preview);
 self.Top:= 0;
 self.Left:= 0;
end;
```

Jak wynika z kodu pliku projektu, Form3 tworzony jest tylko w przypadku przesłania do wygaszacza pierwszego parametru o wartości /p. N ależy jednak pr zedtem dos tosować parametry tego formularza, aby odpowiadały one wymaganiom okna Właściwości Ekranu – wszystkie elementy z Form3 będą bowiem na nim wizualizowane.

Dopasowany do systemu

W celu uproszczenia całej procedury najłatwiej jest nadać Form3 wymagane parametry: ClientHeight=112, ClientWidth=152 oraz BorderStyle=bsNone. Ustawienia tych parametrów można uniknąć, używając przeskalowania okienka aplikacji, jednak dla uproszczenia konstrukcji programu proces ten został tu pominięty. Tak przygotowany formularz może stanowić osnowę prezentacji w ramach monitora z zakładki **Wygaszacze** okienka Właściwości Ekranu.

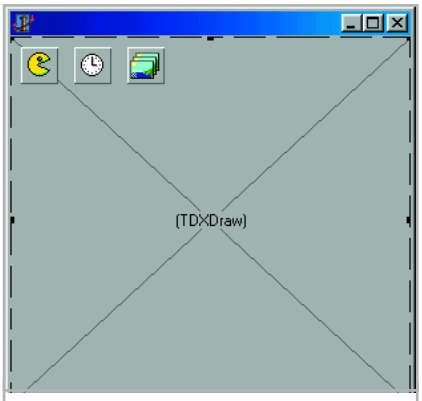
Aby uruchomić screensaver w trybie konfiguracji, Windows wywołuje go z parametrem /c. W ówczas głó wnym f ormularzem naszego programu staje się Form1 (świadczy o tym kolejność tworzenia formularzy – patrz: plik projektu). Umożliwia on 148»



Warto zadbać o WYGODNĄ KONFIGURACJĘ naszej aplikacji.

wprowadzenie zmian wartości sześciu parametrów wykorzystywanych w animacji wizualizowanej na Form2.

Wszystkie parametry są liczbami całkowitymi, wybieranymi z określonego zakresu odpowiednich TSpinEdit, znajdujących się na zakładce Sample. Najważniejszy jest jednak nie dobór konkretnych komponentów, lecz możliwość zmiany i sposób zapamiętania ich wartości w systemie (poza czasem aktywności wygaszacza), tak aby po uruchomieniu programu mogły być one prawidłowo zainicjowane. Do najpopularniejszych metod rozwiązania tego problemu należy użycie plików INI lub rejestrów. W prezentowanej aplikacji zostało wykorzystane drugie z wymienionych rozwiązań. Czym jest drzewo Rejestrów w Windows,



Główne okno wygaszacza prezentuje się skromnie. Do utworzenia animacji użyjemy KOMPONENTÓW DELPHIX.

nie trzeba chyba nikomu tłumaczyć. Procedura zapisująca wartość klucza o podanej nazwie (wykorzystująca klasę TRegistry) może wyglądać w sposób następujący:

```
procedure zapisz_rejestr(na
zwa:string;wartosc:integer);
var
Rejestr:TRegistry;
begin
 Rejestr:=TRegistry.Create;
 try
 Rejestr.OpenKey('Software\
    Wygaszacz',true);
 Rejestr.WriteInteger(nazwa,
    wartosc);
 finally
 Rejestr.Free;
 end;
end;
```

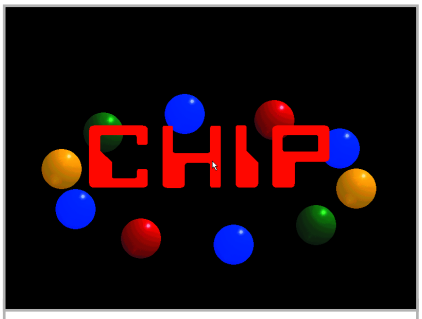
Cóż ona takiego robi? Po pobraniu parametrów: nazwy klucza (łańcuch znaków) i jego wartości (liczba całkowita), utworzeniu obiektu Rejestr (TRegister) otwiera w nim podklucz Software\Wygaszacz. Podklucz ten jest podkluczem klucza głównego Rootkey (domyślnie ustawiony na HKEY_CURRENT_USER). Funkcja OpenKey klasy TRegister poza nazwą klucza zawiera również parametr typu logicznego CanCreate odpowiedzialny za „zachowanie" funkcji w sytuacji braku klucza o podanej nazwie i tylko przy jego wartości równej **True** klucz taki zostanie utworzony.

Dopiero teraz możliwe jest umieszczenie w kluczu konkretnych wartości danych. W zależności od ich typów można posłużyć się jedną z odpowiednich metod (WriteBinaryData, WriteBool, WriteCurrency, WriteDate, WriteDateTime, WriteExpandString, WriteFloat, WriteInteger, WriteString, WriteTime). Ważną sprawą jest ujęcie całości operacji na Rejestrach w blok **try...finally** celem poprawnego zachowania się aplikacji w sytuacjach wyjątkowych:

```
function czytaj_rejestr
  (nazwa:string):integer;
var
Rejestr:TRegistry;
begin
 Rejestr:=TRegistry.Create;
 try
 czytaj_rejestr:=-1;
 if Rejestr.OpenKey('Software\
    Wygaszacz',false) then
czytaj_rejestr := Rejestr.
    ReadInteger(nazwa);
 finally
 Rejestr.Free;
 end;
end;
```

Procedura **zapisz_rejestr()** wywoływana jest kilkakrotnie (dla wszystkich możliwych do ustawienia parametrów wygaszacza) po każdorazowym użyciu przycisku Ok znajdującego się na Form1.

Procedura **czytaj_rejestr()** służy zadaniu odwrotnemu, tj. pobraniu odpowiednich wartości klucza, i opiera swoje działanie na jednej z wielu metod odczytujących wartości kluczy (ReadBinaryData, ReadBool, ReadCurrency, ReadDate, ReadDateTime, ReadFloat, ReadInteger, ReadString, ReadTime). Jest ona, podobnie jak zapisz_rejestr, wywoływana wielokrotnie w procedurze, podczas tworzenia formularza Form1. Procedura ta służy do zainicjowania właściwymi wartościami kontrolek TSpinEdit. Odpowiadają one określonym parametrom wykorzystywanym przez prostą animację, stanowiącą główną część wygaszacza, a tworzoną w obrębie Form2.



Efekty multimedialne prezentowane przez wygaszacz ZALEŻĄ JEDYNIE OD NASZEJ INWENCJI I UMIEJĘTNOŚCI.

Pole do popisu

„Treścią" wygaszacza może być dowolny obrazek czy animacja i właściwie jedynymi ograniczeniami są tu zmysł estetyczno-plastyczny oraz umiejętności programisty. Dlatego też przykładowy wygaszacz zawiera prostą animację wykorzystującą mechanizm DirectX. Zostały w niej wykorzystane komponenty z pakietu DelphiX (można je skopiować z płyty CHIP-CD). Do wykonania efektu przedstawionego w załączonym do artykułu programie wystarczy zapoznanie się z przykładowymi źródłami dołączonymi do instalatora pakietu. Wiele informacji na temat tych komponentów można również odnaleźć w Internecie – choćby pod adresami podanymi w ramce „Info".

Aby wygaszacz spełnił wszystkie cztery wymagania wymienione na wstępie artykułu, powinien również odpowiednio reagować na akcje użytkownika. Program musi więc kończyć swoje działanie po otrzymaniu jakiegokolwiek sygnału o użyciu myszy bądź klawiatury. Zamknięcie aplikacji musi następować po każdorazowym wywołaniu procedury obsługi zdarzeń **OnMouseMove** i **OnKeyDown**. ■

INFO

INFORMACJE O DELPHIX

http://community.borland.com/article/1,1410,10503,00.html

http://www.softhard.com.pl/~sulek/delphix/eng/dxtutor1.html

Na płycie CD, w dziale **Porady | Programowanie w Delphi**, można znaleźć kod programu opisywanego w artykule oraz komponenty DelphiX.