abc

12

PRZECHOWYWANIE INFORMACJI W REJESTRZE SYSTEMU WINDOWS

Rejestr systemu Windows jest bazą danych, w której system i aplikacje mogą przechowywać swoje dane, w szczególności parametry konfiguracyjne. Rejestr składa się z kilku kluczy głównych, z których z punktu widzenia programisty interesujące są trzy. Pierwszym jest `HKEY_LOCAL_MACHINE`, który przechowuje ustawienia systemowe dotyczące sprzętu, systemu i wszystkich użytkowników. Drugi to `HKEY_CURRENT_USER`. Jest to w istocie alias do jednego z kluczy znajdujących się w kluczu głównym `HKEY_USERS`, mianowicie do tego, którego właścicielem jest aktualnie zalogowany użytkownik. W tym kluczu (tj. w `HKEY_CURRENT_USER`) przechowywane są wszystkie ustawienia użytkownika dotyczące systemu (np. ustawienia interfejsu graficznego) oraz spersonalizowane ustawienia aplikacji.

Trzecim kluczem, o którym chciałbym wspomnieć, jest HKEY_CLASSES_ROOT, w którym zapisane są informacje o skojarzeniach aplikacji z rozszerzeniami pliku. My skupimy się na kluczu HKEY_CURRENT_USER.

Rejestr zorganizowany jest na wzór systemu plików. W każdym z kluczy podstawowych (ang. *root key*) można zakładać własne klucze (ang. *key*), które odpowiadają folderom. W każdym kluczu mogą znajdować się podklucze lub wartości (ang. *value*). Wartości przechowują zwykle łańcuchy (ciągi, typ REG_SZ) lub liczby naturalne (typ REG_DWORD). Inne typy nie będą w tym rozdziale wykorzystywane.

Do ręcznej edycji rejestru służy dołączony do systemu edytor rejestru. Uruchamia się go poleceniem regedit.

Jeżeli zagadnienia dotyczące rejestru wydają się komuś trudne lub niebezpieczne i rozważa przeskoczenie tego rozdziału, to zapewniam, że postępując dokładnie według poniższych opisów, będziemy bezpieczni. Natomiast korzystanie z rejestru nie jest też niczym specjalnie skomplikowanym, a to za sprawą komponentu TRegistry. Tak naprawdę cała sztuka korzystania z rejestru w Delphi polega wyłącznie na poznaniu tego komponentu. No i oczywiście niezbędna jest ogólna orientacja w strukturze rejestru. Tę jednak za chwilę omówimy.

Korzystanie z rejestru zaprezentujemy w dwóch przykładach: przechowywanie opcji aplikacji (na przykładzie pozycji i rozmiaru okna) oraz umieszczanie zapisu powodującego automatyczne uruchamianie aplikacji w momencie logowania użytkownika.

Przechowywanie danych aplikacji w rejestrze

Przygotujmy dwie funkcje, które będą odczytywać z rejestru i zapisywać do niego położenie i rozmiar okna. Jeżeli ich wywołania znajdą się w metodach zdarzeniowych formy, związanych odpowiednio z OnCreate i OnClose, to po uruchomieniu aplikacji forma pojawi się w miejscu, w którym została zamknięta w poprzedniej sesji. Obie funkcje zdefiniujemy w osobnym module, aby umożliwić ich wygodne wykorzystanie w wielu projektach. Funkcje będą zapisywać do lub odczytywać ze wskazanego klucza rejestru cztery wartości typu całkowitego (REG_DWORD) o nazwach Left, Top, Width i Height, odpowiadające własnościom formy określającymi jej położenie i rozmiar. Pierwszym argumentem obu funkcji będzie referencja do obiektu okna, czyli w naszym przypadku Form1 lub Self. Drugim — ścieżka do klucza, w którym zostaną zapisane ustawienia okna.

Nic nie stoi na przeszkodzie, aby użyć tych funkcji dla kilku okien w przypadku aplikacji z wieloma oknami. Należy wówczas pamiętać o podaniu różnych kluczy rejestru. Do automatycznego wykreowania takiego klucza, innego dla każdej formy, można wykorzystać nadany przez programistę tytuł aplikacji oraz nazwę obiektu okna, np. `'\Software\Helion\'+Application.Title+'\'+Self.Name`.

Wszystkie ćwiczenia z tego rozdziału umieścimy we wspólnym projekcie. Stwórzmy wobec tego nowy projekt *VCL Forms Application — Delphi for Win32* lub *VCL Forms Application — Delphi for .NET* i zapiszmy go do osobnego katalogu.

Jak utworzyć nowy moduł na funkcje odczytujące i zapisujące dane do rejestru?

Była już o tym mowa przy okazji tworzenia modułu dla funkcji `Drukuj` w poprzednim rozdziale, ale nie zaszkodzi powtórzyć.

Mając otwarty projekt aplikacji *VCL Forms Application — Delphi for Win32* lub *VCL Forms Application — Delphi for .NET*, dodajemy do niego nowy moduł:

- w przypadku projektu *Delphi for Win32* można z menu *File*, podmenu *New*, wybrać pozycję *Unit — Delphi for Win32*,
- w projekcie *Delphi for .NET* należy natomiast przełączyć się w widok edytora, a wówczas w oknie *Tool Palette* w sekcji *Delphi for .NET Projects | New Files* znajdziemy pozycję *Unit*, którą należy dwukrotnie kliknąć.

Naciskamy *Ctrl+S* i nowy moduł zapisujemy do pliku *Rejestr.pas*. W tym module tworzymy sekcję `uses` (listing 12.1), do której dodajemy moduły `Forms` (zawiera definicję klasy `TForm`), `Registry` (zawiera klasę `TRegistry`), `Windows` (zawiera definicje stałych określających klucze podstawowe, m.in. `HKEY_CURRENT_USER`, oraz stałych określających prawa dostępu do kluczy) i `Dialogs` (zawiera definicję procedury `ShowMessage`). Moduły te będą konieczne do przygotowania funkcji odczytujących i zapisujących dane do rejestru.

Listing 12.1. Moduły dostępne z modułu Rejestr

```
unit Rejestr;

interface

uses Forms, Registry, Windows, Dialogs;

implementation

end.
```

Deklarowanie funkcji w interfejsie modułu

W sekcji `interface` modułu *Rejestr* deklarujemy dwie funkcje `CzytajUstawieniaOkna` i `ZapiszUstawieniaOkna` o sygnaturach widocznych na listingu 12.2. Argumenty w obu przypadkach są jednakowe. Pierwszy to obiekt klasy `TForm`, a więc obiekt okna, a drugi — nazwa klucza rejestru.

Listing 12.2. Deklaracje funkcji w interfejsie modułu

```
unit Rejestr;

interface

uses Forms, Registry, Windows, Dialogs;

function CzytajUstawieniaOkna(okno :TForm;klucz :String; komunikaty: Boolean)
:Boolean;
function ZapiszUstawieniaOkna(okno :TForm;klucz :String; komunikaty: Boolean)
:Boolean;

implementation

end.
```

Tak zadeklarowane funkcje muszą być teraz zdefiniowane. Zrobimy to w następnych dwóch ćwiczeniach.

Jak odczytywać dane z rejestru?

W sekcji `implementation` umieszczamy definicję pierwszej z funkcji, odczytującej z rejestru dane o ustawieniu okna (listing 12.3):

Listing 12.3. Czytanie danych z rejestru

```
unit Rejestr;

interface

uses Forms, Registry, Windows, Dialogs;

function CzytajUstawieniaOkna(okno :TForm;klucz :String; komunikaty: Boolean)
:Boolean;
function ZapiszUstawieniaOkna(okno :TForm;klucz :String; komunikaty: Boolean)
:Boolean;

implementation
```

```
function CzytajUstawieniaOkna(okno :TForm; klucz :String; komunikaty: Boolean)
:Boolean;
var Rejestr :TRegistry;
begin
Result:=False;
Rejestr:=TRegistry.Create;    //tworzenie instancji TRegistry
Rejestr.Access:=KEY_READ;     //nadawanie uprawnień do odczytu rejestru
Rejestr.RootKey:=HKEY_CURRENT_USER;       //wybór klucza głównego

if Rejestr.KeyExists(klucz) then          //czy klucz istnieje?
  if Rejestr.OpenKey(klucz,False) then    //otwieranie klucza
    begin
    //zapisywanie danych
    if Rejestr.ValueExists('Left') then okno.Left:=Rejestr.ReadInteger('Left');
    if Rejestr.ValueExists('Top') then okno.Top:=Rejestr.ReadInteger('Top');
    if Rejestr.ValueExists('Width') then okno.Width:=Rejestr.ReadInteger('Width');
    if Rejestr.ValueExists('Height') then okno.Height:=Rejestr.ReadInteger
    ('Height');
    Rejestr.CloseKey;          //zamknięcie dostępu do klucza rejestru
    Result:=True;
    end
    else
    begin
    if komunikaty then ShowMessage('Klucz "'+klucz+'" nie może być otwarty.');
    end
  else
  if komunikaty then ShowMessage('Klucz "'+klucz+'" nie istnieje.');
Rejestr.Free;                 //usuniecie instancji TRegistry z pamięci
end;

end.
```

Postarajmy się zrozumieć, co ta funkcja robi. Na początek warto przejrzeć się powyższemu listingowi i prześledzić komentarze dające obraz o ogólnej strukturze funkcji.

Przede wszystkim tworzymy obiekt, który pozwala nam na dostęp do rejestru. Jest to obiekt typu `TRegistry`, a jego referencję zapisujemy do zmiennej obiektowej `Rejestr`. Robi to pierwsza linia funkcji tuż za słowem kluczowym `begin`. Po utworzeniu klucza nadawane są mu jedynie uprawnienia do odczytywania danych. W ten sposób nie ma niebezpieczeństwa omyłkowej zmiany danych. Konieczne jest również wybranie klucza głównego, z którego będziemy korzystać. W naszym przypadku jest to klucz związany z bieżącym użytkownikiem, a więc `HKEY_CURRENT_USER`.

Po przygotowaniu obiektu `Register` możemy zająć się odczytaniem danych. Przede wszystkim musimy sprawdzić, czy istnieje w rejestrze klucz wskazany przez argument `klucz`, z którego chcemy odczytać dane. Służy do tego metoda `KeyExists` klasy `TRegistry`, której argumentem jest nazwa klucza. Jeżeli nie istnieje,

wyświetlamy odpowiedni komunikat. Załóżmy jednak, że klucz istnieje. Wówczas próbujemy go otworzyć metodą OpenKey. Może się nam to nie udać, jeżeli nie mamy odpowiednich uprawnień. Jednak jeżeli operujemy w kluczu użytkownika (HKEY_CURRENT_USER), to jest to sytuacja mało prawdopodobna. Co innego, gdybyśmy chcieli zabrać się za klucz HKEY_LOCAL_MACHINE lub HKEY_CLASSES_ROOT. Załóżmy jednak, że funkcja OpenKey zwróciła wartość True, a więc otwarcie klucza powiodło się. Mamy wówczas dostęp do zapisanych w nim wartości. Do ich odczytu służy seria metod. Wybór jednej z nich zależy od typu odczytywanych wartości. Mamy więc metodę ReadInteger odczytującą liczby naturalne, ReadFloat do odczytu danych zmiennoprzecinkowych, ReadBool do odczytu zmiennych logicznych, ReadString do odczytu łańcuchów, ReadDateTime do odczytu daty i czasu oraz wiele innych. W naszym przykładzie do odczytu położenia i rozmiarów okna wykorzystujemy jedynie metodę ReadInteger.

Po odczytaniu danych z klucza należy go zamknąć (metoda CloseKey), a obiekt Rejestr usunąć z pamięci. No, przynajmniej w projektach dla Win32[1], bo w projektach dla .NET można sobie to odpuścić — obiektem zajmie się „kolekcjoner śmieci".

Przykład wykorzystania instrukcji with

Zanim przejdziemy do dalszego omawiania klasy TRegistry i korzystania z rejestru, chciałbym skorzystać z doskonałej okazji, żeby pokazać jeszcze raz, w jaki sposób można wykorzystać instrukcję Delphi with. Zauważmy, że w funkcji CzytajUstawieniaOkna co najmniej kilkanaście razy musieliśmy użyć nazwy obiektu Rejestr z operatorem dostępu, czyli kropką, aby móc wywołać metody tego obiektu. Można tego uniknąć, i jak wiemy z rozdziału trzeciego, pozwala na to właśnie instrukcja with. Listing 12.4 pokazuje, co należy zrobić.

Listing 12.4. Instrukcja with skraca kod, ale dla niektórych czyni go mniej czytelnym

```
function CzytajUstawieniaOkna(okno :TForm; klucz :String; komunikaty: Boolean)
:Boolean;
var Rejestr :TRegistry;
begin
Result:=False;
Rejestr:=TRegistry.Create;       //tworzenie instancji TRegistry
with Rejestr do
  begin
  Access:=KEY_READ;              //nadawanie uprawnień do odczytu rejestru
  RootKey:=HKEY_CURRENT_USER;    //wybór klucza głównego
```

[1] Klasa TRegistry nie jest komponentem (nie dziedziczy z TComponent), czyli nie posiada właściciela. Nie ma więc obiektu, który byłby odpowiedzialny za jej usunięcie w projektach dla Win32 w przypadku, gdy my o tym zapomnimy.

```
  if KeyExists(klucz) then          //czy klucz istnieje?
    if OpenKey(klucz,False) then    //otwieranie klucza
      begin
      //zapisywanie danych
      if ValueExists('Left') then okno.Left:=ReadInteger('Left');
      if ValueExists('Top') then okno.Top:=ReadInteger('Top');
      if ValueExists('Width') then okno.Width:=ReadInteger('Width');
      if ValueExists('Height') then okno.Height:=ReadInteger('Height');
      CloseKey; //zamknięcie dostępu do klucza rejestru
      Result:=True;
      end
      else
      begin
        if komunikaty then ShowMessage('Klucz "'+klucz+'" nie może być otwarty.');
      end
    else
    if komunikaty then ShowMessage('Klucz "'+klucz+'" nie istnieje.');
  Free;    //usuniecie instancji TRegistry z pamięci
  end;     //koniec with-Rejestr
end;
```

Cały kod, w którym używane były pola, metody i własności obiektu `Rejestr`, otoczony został przez poznaną w trzecim rozdziale konstrukcję

```
with obiekt do
  begin
  end;
```

Wewnątrz tego bloku, przy wszystkich odwołaniach do zmiennych, procedur i funkcji, kompilator testuje, czy nie są to przypadkiem nazwy pól, własności lub metod wskazanego obiektu lub struktury. Jeżeli nazwa jest niejednoznaczna, to znaczy mogłaby się odnosić zarówno do zmiennej lub funkcji, jak i do pola obiektu lub jego metody, to pierwszeństwo mają elementy składowe wskazanego obiektu. Jeżeli jednak chcemy wywołać zmienną lub funkcję, możemy to zrobić, wskazując ją jednoznacznie przez podanie nazwy jej modułu, np. `Unit1.`*`zmienna`* lub `Unit1.`*`procedura`*.

Instrukcja `with` bezsprzecznie skraca kod. Nie będę jej jednak stosować w dalszej części tej książki (pozostawiając zadanie ewentualnego użycia Czytelnikowi), ponieważ zmniejsza czytelność kodu — w bloku, który obejmuje instrukcja `with`, może nie być do końca jasne, czy wywoływana jest metoda wskazanego w tej instrukcji obiektu, czy zwykła funkcja (analogicznie wygląda sprawa z własnościami i polami obiektu oraz zwykłymi zmiennymi).

Jak zapisać dane do rejestru?

Teraz zdefiniujmy drugą z zadeklarowanych funkcji (listing 12.5).

Listing 12.5. Funkcja zapisująca dane o geometrii okna do rejestru

```
function ZapiszUstawieniaOkna(okno :TForm; klucz :String; komunikaty: Boolean)
:Boolean;
var Rejestr :TRegistry;
begin
Result:=False;
Rejestr:=TRegistry.Create;
Rejestr.Access:=KEY_WRITE;
Rejestr.RootKey:=HKEY_CURRENT_USER;

if not Rejestr.KeyExists(klucz) then
  if not Rejestr.CreateKey(klucz) then
    begin
    ShowMessage('Klucz "'+klucz+'" nie może być utworzony.');
    Rejestr.Free;
    Exit;
    end;

if Rejestr.OpenKey(klucz,False) then
  begin
  Rejestr.WriteInteger('Left',okno.Left);
  Rejestr.WriteInteger('Top',okno.Top);
  Rejestr.WriteInteger('Width',okno.Width);
  Rejestr.WriteInteger('Height',okno.Height);
  Rejestr.CloseKey;
  Result:=True;
  end
  else ShowMessage('Klucz "'+klucz+'" nie może być otwarty.');

Rejestr.Free;
end;
```

Zwróćmy uwagę na to, że funkcja zapisująca dane do rejestru ma bardzo podobną strukturę do funkcji odczytującej dane. Również teraz musimy stworzyć i skonfigurować obiekt-instancję klasy `TRegistry`. Jednak tym razem obiektowi nadajemy uprawnienia do wprowadzania zmian w rejestrze.

Podobnie jak w poprzedniej funkcji sprawdzamy, czy istnieje klucz rejestru identyfikowany przez łańcuch przekazany w zmiennej `klucz`. Jeżeli nie istnieje, tworzymy go metodą `CreateKey`. Następnie otwieramy klucz, zapisujemy do niego dane, zamykamy klucz i na końcu usuwamy obiekt `Rejestr` z pamięci.

Do zapisywania danych służy zbiór procedur `Write...`, które są jakby zwierciadlanym odbiciem funkcji `Read...`. Mamy więc wykorzystywaną w naszym przykładzie procedurę zapisującą do rejestru liczbę naturalną (`WriteInteger`), zapisującą liczbę zmiennoprzecinkową (`WriteFloat`), zmienną logiczną (`WriteBool`) czy łańcuch (`WriteString`).

Odczyt z rejestru pozycji i rozmiaru okna po uruchomieniu aplikacji i ich zapis przy jej zamknięciu

No to teraz mamy już z górki. Funkcje są gotowe, wystarczy ich użyć.

Wracamy do modułu okna *Unit1.pas* (zakładka *Unit1* na górze okna edytora). Musimy przedstawić mu moduł *Rejestr*. Ale ponieważ jest to moduł wewnętrzny projektu, wystarczy z menu *File* wybrać *Use Unit* i w otwartym oknie kliknąć *OK* (dokładnie tak, jak nauczyliśmy się w rozdziale 8.). Następnie przechodzimy do widoku projektowania (*F12*), i klikając dwukrotnie podgląd formy, tworzymy domyślną metodę zdarzeniową do zdarzenia `OnCreate` formy. Umieszczamy w niej wywołanie funkcji `CzytejUstawieniaOkna` (listing 12.6).

Listing 12.6. Odczytywanie pozycji i rozmiaru formy przy uruchamianiu aplikacji

```
procedure TForm1.FormCreate(Sender: TObject);
begin
CzytajUstawieniaOkna(Self,'\Software\Helion\Przyklad\Okno',False);
end;
```

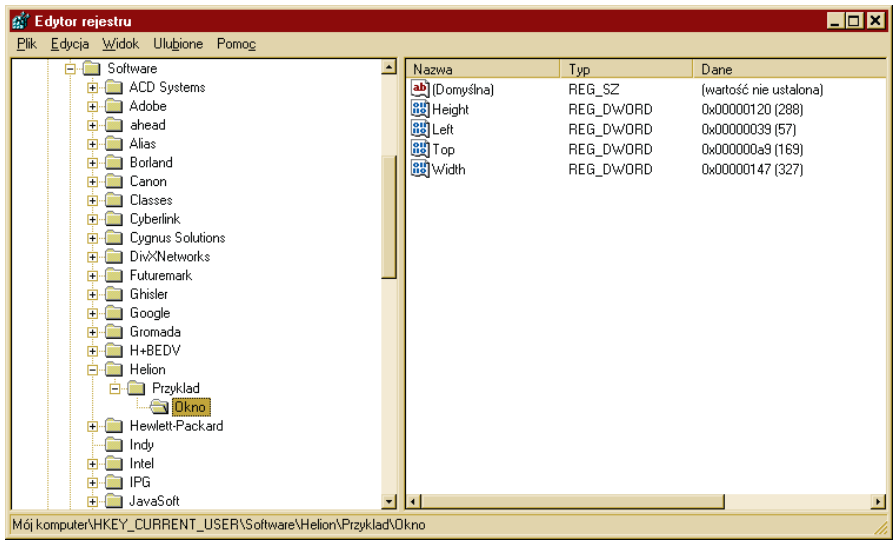
Ważny jest klucz, który wybieramy do przechowywania danych. Do przechowywania danych przez aplikacje przygotowane są klucze *Software* w każdym kluczu głównym. Zgodnie ze zwyczajem, w tym kluczu tworzony jest podklucz identyfikujący producenta programu, a dopiero w nim właściwy klucz aplikacji. Tak też zrobiliśmy my — za nazwę producenta wzięliśmy *Helion*, a aplikacji — *Przyklad*. Ponadto stworzyliśmy dodatkowy podklucz *Okno*, w którym przechowywane będą wartości związane z położeniem i rozmiarem okna.

W analogiczny sposób tworzymy metodę zdarzeniową do zdarzenia `OnClose` i w niej umieszczamy polecenie zapisujące dane do rejestru widoczne na listingu 12.7.

Listing 12.7. Zapisywanie pozycji i położenia przed zamknięciem formy

```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
ZapiszUstawieniaOkna(Self,'\Software\Helion\Przyklad\Okno',True);
end;
```

Aby upewnić się co do położenia klucza z zapisanymi przez funkcję `ZapiszUstawieniaOkna` wartościami i ich poprawności, możemy zajrzeć do rejestru systemowego za pomocą systemowego edytora rejestru uruchamianego poleceniem `regedit` (rysunek 12.1).



Rysunek 12.1. Klucz rejestru z wartościami utworzonymi przez program oglądany w systemowym edytorze rejestru

Zapisywanie położenia i rozmiaru okna jest oczywiście tylko przykładem informacji, jakie można przechowywać w rejestrze. Mogłyby to być na przykład ostatnio otwierane pliki, katalog tymczasowy aplikacji czy opcje skonfigurowane przez użytkownika.

Automatyczne uruchamianie aplikacji w momencie logowania użytkownika

Efekt automatycznego uruchomienia aplikacji w momencie logowania użytkownika można uzyskać na kilka sposobów. Pierwszy z nich to utworzenie skrótu w podmenu *Autostart* menu *Start* użytkownika. Jest to jednak sposób przeznaczony raczej do samodzielnego wykonania przez użytkownika za pomocą myszy, niż przez aplikacje. W przypadku programów bardziej właściwe jest umieszczenie odpowiedniego zapisu w rejestrze w kluczu *Software\Microsoft\Windows\CurrentVersion\Run*. Można to zrobić albo w kluczu prywatnym użytkownika (`HKEY_CURRENT_USER`), albo w kluczu systemowym (`HKEY_LOCAL_MACHINE`). W tym drugim przypadku aplikacja będzie uruchamiana po zalogowaniu każdego z użytkowników. Należy jednak pamiętać, że aby stworzyć zapis w tym kluczu, użytkownik musi mieć odpowiednio wysokie uprawnienia. Poza tym wykorzystanie klucza użytkownika umożliwia łatwą personalizację ustawień.

Przygotujemy procedurę, która będzie umieszczała odpowiedni zapis w rejestrze. Oprócz niej potrzebna będzie również procedura, która pozwoli na jego usunięcie. Przydałaby się również funkcja sprawdzająca, czy zapis już istnieje.

Zapisywanie do rejestru informacji o uruchamianiu aplikacji w momencie logowania użytkownika

Zacznijmy od procedury zapisującej informacje o bieżącej aplikacji do klucza, w którym umieszczane są informacje o uruchamianych automatycznie programach. Jest ona bardzo podobna do zdefiniowanej wyżej funkcji zapisującej dane o położeniu i rozmiarze okna. Może nawet prostsza, bo dodawana będzie tylko jedna wartość, a klucz jest z góry znany. Dodajmy zatem procedurę z listingu 12.8 do modułu z pliku *Rejestr.pas*:

Listing 12.8. Procedura dodająca do rejestru zapis o automatycznym uruchomieniu aplikacji po zalogowaniu użytkownika

```
procedure ZapiszAutostart(nazwa :String; plikEXE :String);
const klucz :String ='\SOFTWARE\Microsoft\Windows\CurrentVersion\Run';
var Rejestr :TRegistry;
begin
Rejestr:=TRegistry.Create;
Rejestr.Access:=KEY_SET_VALUE;
Rejestr.RootKey:=HKEY_CURRENT_USER;
if Rejestr.OpenKey(klucz,False) then
  begin
  Rejestr.WriteString(nazwa,plikEXE);
  Rejestr.CloseKey;
  end
  else ShowMessage('Klucz "'+klucz+'" nie istnieje lub nie może być otwarty.');
Rejestr.Free;
end;
```

Zwróćmy uwagę, że tym razem przyznaliśmy obiektowi `Rejestr` uprawnienia identyfikowane przez stałą `KEY_SET_VALUE`. Są one słabsze niż użyte poprzedni `KEY_WRITE`. Pozwalają tylko na tworzenie, modyfikowanie i usuwanie wartości w istniejących już kluczach. Obiekt nie ma uprawnień do tworzenia lub usuwania samych kluczy.

Usuwanie zapisu o automatycznych uruchamianiu

W zupełnie podobny sposób definiujemy procedurę usuwającą zapis. Do usunięcia wartości używamy metody `DeleteValue` klasy `TRegistry` (listing 12.9).

Listing 12.9. Wyróżnione fragmenty to jedyne miejsce różniące tę procedurę od poprzedniej

```
procedure UsunAutostart(nazwa :String);
const klucz='\SOFTWARE\Microsoft\Windows\CurrentVersion\Run';
var Rejestr :TRegistry;
begin
Rejestr:=TRegistry.Create;
Rejestr.Access:=KEY_SET_VALUE;
Rejestr.RootKey:=HKEY_CURRENT_USER;
if Rejestr.OpenKey(klucz,False) then
  begin
  Rejestr.DeleteValue(nazwa);
  Rejestr.CloseKey;
  end
  else ShowMessage('Klucz "'+klucz+'" nie istnieje lub nie może być otwarty.');
Rejestr.Free;
end;
```

Sprawdzanie, czy zapis o automatycznym uruchomieniu istnieje

Tym razem definiujemy funkcję, a nie procedurę. Będzie ona zwracać typ logiczny, który będzie nas informować o tym, czy istnieje odpowiednia wartość odpowiadająca za automatyczne uruchomienie aplikacji (listing 12.10). Zwróćmy uwagę, że tym razem nadajemy obiektowi `Rejestr` jeszcze mniejsze uprawnienia, które wystarczają jednak do sprawdzania, czy wartość jest w kluczu obecna (stała `KEY_QUERY_VALUE`).

Listing 12.10. Wyróżnione fragmenty różnią tę funkcję od poprzedniej

```
function SprawdzAutostart(nazwa :String) :Boolean;
const klucz='\SOFTWARE\Microsoft\Windows\CurrentVersion\Run';
var Rejestr :TRegistry;
begin
Result:=False;
Rejestr:=TRegistry.Create;
Rejestr.Access:=KEY_QUERY_VALUE;
Rejestr.RootKey:=HKEY_CURRENT_USER;
if Rejestr.OpenKey(klucz,False) then
  begin
  Result:=Rejestr.ValueExists(nazwa);
  Rejestr.CloseKey;
  end
  else ShowMessage('Klucz "'+klucz+'" nie istnieje lub nie może być otwarty.');
Rejestr.Free;
end;
```

Udostępnianie funkcji z modułu

Kopiujemy sygnatury trzech funkcji do sekcji interfejsu modułu, dzięki czemu staną się one widoczne spoza tego modułu (listing 12.11). I w ten sposób zakończyliśmy rozwijanie modułu *Rejestr*.

Listing 12.11. Umieszczanie funkcji w interfejsie modułu

```
unit Rejestr;

interface

uses Forms, Registry, Windows, Dialogs;

function CzytajUstawieniaOkna(okno :TForm;klucz :String; komunikaty: Boolean)
:Boolean;
function ZapiszUstawieniaOkna(okno :TForm;klucz :String; komunikaty: Boolean)
:Boolean;
procedure ZapiszAutostart(nazwa :String; plikEXE :String);
procedure UsunAutostart(nazwa :String);
function SprawdzAutostart(nazwa :String) :Boolean;
```

Test funkcji

Wróćmy do modułu formy (plik *Unit1.pas*) i umieśćmy na niej pole opcji `TCheckBox` z etykietą `Uruchom automatycznie w momencie logowania użytkownika`. Następnie kliknijmy dwukrotnie formę w dowolne miejsce nie zajęte przez inny komponent, co spowoduje utworzenie domyślnej metody zdarzeniowej do jej zdarzenia `OnCreate` wywoływanego tuż po uruchomieniu aplikacji. Umieśćmy w niej polecenie sprawdzające, czy istnieje klucz autostartu o nazwie `Helion` (listing 12.12).

Listing 12.12. Pole opcji pokazuje, czy zapis już istnieje w rejestrze, a jednocześnie pozwoli na jego dodanie lub usunięcie

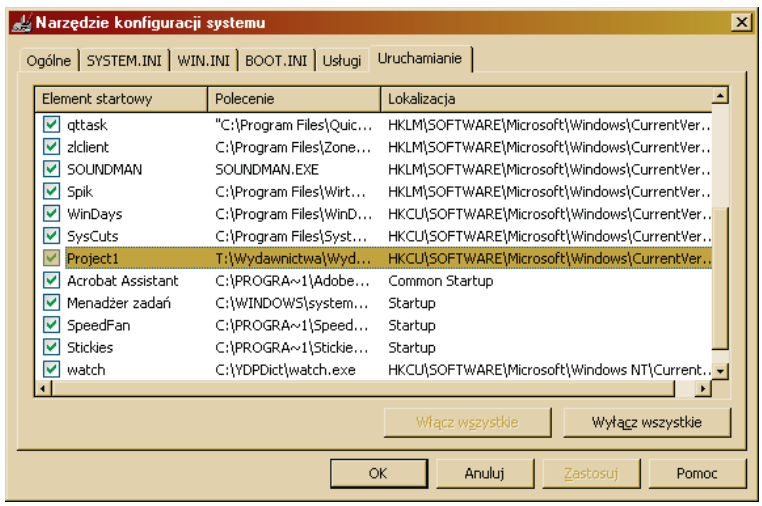
```
procedure TForm1.FormCreate(Sender: TObject);
begin
CzytajUstawieniaOkna(Self,'\Software\Helion\Przyklad\Okno',False);
CheckBox1.Checked:=SprawdzAutostart('Helion');
end;
```

Pozwólmy użytkownikowi tworzyć zapis o autostarcie bieżącej aplikacji w rejestrze poprzez zaznaczanie lub usuwanie zaznaczenia pola opcji. Wówczas wywoływane jest zdarzenie `OnClick` tego komponentu, które musimy odnaleźć w inspektorze obiektów i stworzyć do niego metodę zdarzeniową o treści widocznej na listingu 12.13.

Listing 12.13. Umieszczamy lub kasujemy wartość w rejestrze w zależności od stanu pola opcji CheckBox1

```
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
if CheckBox1.Checked
    then ZapiszAutostart('Helion',Application.ExeName)
    else UsunAutostart('Helion');
end;
```

Możemy uruchomić aplikację i zaznaczyć pole opcji. Aby upewnić się, że zapis został dodany do rejestru, można wykorzystać edytor rejestru lub Narzędzie konfiguracji systemu (rysunek 12.2), ale najlepszym testem będzie zwyczajne wylogowanie bieżącego użytkownika i ponowne jego zalogowanie. Jeżeli wszystko działa prawidłowo — projektowana aplikacja zostanie automatycznie uruchomiona.



Rysunek 12.2. Narzędzie konfiguracji systemu uruchamiane jest poleceniem msconfig

W domu

Przenoszenie modułu Rejestr do innych projektów

Wszystkie funkcje dotyczące rejestru, w szczególności te dotyczące przechowywania położenia i rozmiaru formy, umieściliśmy w osobnym module. Nie powinno być więc żadnego problemu, aby wykorzystać je w innych projektach. Proponuję, aby Czytelnik dodał do naszego projektu przeglądarki plików graficznych pamięć położenia i rozmiaru formy.

Lista ostatnio otwartych plików w rejestrze

Proponuję także trudniejsze zadanie: do menu *Plik* tej aplikacji proszę dodać podmenu *Ostatnio otwierane pliki*, a w nim umieścić listę dziesięciu ostatnio otwartych. Listę tę należy przechowywać w rejestrze w osobnym podkluczu.

Instrukcja with

Chciałbym, aby Czytelnik zastosował konstrukcję `with` w procedurach z listingów 12.5, 12.8, 12.9 i 12.10.