قرار دادن کدها در یونیت

یکی از قابلیتهای مهم دلفی استفاده از یونیتها می باشد شما میتوانید توابع و رویه هایی که در برنامه ها زیاد با ان سرو کار دارید را درون یک unit قرار داده و به راحتی در برنامه های خود استفاده کنید برای این کار یک پروژه جدید ایجاد کنید از گزینه File/New/Unit یک Unit به پروژه اضافه کنید unit فوق را با نام Sample ذخیره کنید به قسمت InterFace یونیت رفته و با استفاده از کلمه کلیدی Uses یونیتهای مورد نیاز مانند Classesو Sysutils را به unit اضافه میکینم

```
unit sample;


interface
uses
SysUtils,Classes;


Implementation
```

النون به قسمت Implementation رفته و توابع مورد نظر را پیاده سازی میکنیم هر تابعی که پیاده سازی میکینم اعلان ان تابع را قبل از implementation قرار دهید

```
unit sample;


interface


uses
SysUtils,Classes;


 function number(str:string):string;
 function _3digit(s:string):string;


implementation


function number(str:string):string;
var
i:integer;
```

```
st:string;
begin
for I := 0 to length(str) do
  if str[i] in['0'..'9'] then
  st:=st+str[i];
  Result:=st;
end;

function _3digit(s:string):string;
var
str:string;
begin
str:=number(s);
Result:=FormatFloat('#,',StrToInt64(str));
end;
end.
```

در این یونیت ۲ تابع وجود دارد اولی بنام Number که مقداری از نوع String گرفته و اعداد انرا جدا میکند و به صورت رشته ای از اعداد به خروجی پاس میدهد و دومی تابعی بنام _3digit که مقداری به عنوان String دریافت و بعد با استفاده از تابع number اعداد را از متن جدا کرده و با استفاده از تابع FloatString اعداد را ۳ رقم ۳ رقم جدا کرده و به خروجی پاس میدهد یونیت Sample را ۲باره ذخیره کنید حالا نوبت به استفاده از این یونیت در برنامه می شود نام یونیت را در لیست uses پروژه قرار دهید

```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls,sample;
```

حالا میتوانید از توابع داخل یونیت به طور مستقیم استفاده کنید ۲ تا Edit روی فورم قرار دهید میخواهیم با استفاده از توابع داخل یونیت کاری کنیم که Edit اولی فقط اعداد را در خود نگه دارد و Edit دومی مقدار پولی را در خود حفظ کند

```
procedure TForm1.Edit1Change(Sender: TObject);
begin
if edit1.Text<>'' then
```

```
begin
edit1.Text:=number(edit1.Text);
edit1.SelStart:=length(edit1.Text)+1;
end;
end;

procedure TForm1.Edit2Change(Sender: TObject);
begin
if edit2.Text<>'' then
begin
  edit2.Text:=_3digit(edit2.Text);
  edit2.SelStart:=length(edit2.Text)+1;
end;
end;
```

استفاده از متد SelStart باعث می شود تا نشانگر موس همواره در انتهای اعداد قرار گیرد

*** استفاده از یونیت کلیه مزایای استفاده از زیر برنامه ها را دارند به علاوه اینکه برای استفاده از انها نیاز به کپی کردن انها از برنامه قبلی به برنامه جدید نیست زیرا فقط کافیست انها را در یک unit قرار داده و با کپی یونیت رون پوشه پروژه و افزودن نام ان به uses از کلیه توابع درون ان استفاده کنید

فصل ۱۳

کدهای اسمبلی در دلفی

در برنامه نویسی حرفه ای گاهی برنامه نویس به خاطر بعضی مسائل مجبور به استفاده از کدهایی به زبانهای دیگه ، بالاخص اسمبلی ، می شه. اگرچه درک کردن کدهای اسمبلی خیلی سخته ، ولی بعضی مواقع کار رو خیلی راحتتر می کنه. بهرحال من اینجا قصد ندارم که کاربردهای این شیوه رو مطرح کنم. فقط طریقه استفاده از این کدها رو عنوان می کنم. در حالت کلی به دو روش مختلف میشه کدهای اسمبلی رو بین کدهای زبان پاسکال قرار داد.

روش اول : استفاده از دستور asm

در این روش کدهای اسمبلی رو داخل بلوکی که با کلمه کلیدی asm شروع و به end ختم می شه ، می نویسیم. مثلا دو کد زیر رو در نظر بگیرید:

مثال:VCL-GUI

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i:Integer;
begin
i:=StrToInt(Edit1.Text);
asm
mov eax,i;
add eax,100;
mov i,eax;
end;
ShowMessage(IntToStr(i));
end;
```

مثال:console

```
program test1;
begin
    asm
        mov ah,08h
        int 21h
        mov ah,02h
        mov dl,al
        int 21h
   end;
end.
```

این دو برنامه ساده که اولی مقدار Edit1.text را با ۱۰۰ جمع میکند ودر یک showmessage نمایش میدهد و دومی یک کاراکتر از کاربر دریافت و چاپ می کنه. البته لازم به توضیحه که شما می تونید از متغیرهای تعریف شده خودتون هم بین کدها استفاده کنید. مثلا

```
program test2;
uses  crt;
var
  c : char;
begin
     asm
         mov ah,08h
         int 21h
         mov c,al
    end;
    showmessage(' char is : ' + c);
end.
```

کد اسمبلی بالا یه کاراکتر از کاربر دریافت می کنه و بدون اینکه چاپش کنه تو c قرار می ده. در ضمن حتما باید به علامت h بعد از اعداد توجه کنید ، که نشون می ده اعداد در مبنای ۱۶ هستن.

شما می تونید تو این روش از برچسبها (Label) هم استفاده کنید ، اما باید برای مشخص شدن برچسب علامت @ رو قبلش قرار بدید. مثلا:

```
program test3;
uses  crt;
var
     n , sum : integer;
begin
     write('Enter n :');
     readln(n);
     asm
         mov cx,n
         mov ax,0
@       BEG:
```

```
        add ax,cx
        loop @BEG
        mov sum,ax
    end;
     writeln(' sum = ' + sum);
      readln;
end.
```

این برنامه عدد n رو می گیره و مجموع اعداد طبیعی از یک تا n رو حساب می کنه.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i:Integer;
begin
i:=StrToInt(Edit1.Text);
asm
mov eax,i;
cmp eax,200;
je @else
jne @endif
@else:
mov eax,500;
mov i,eax;
@endif:
add eax,1000;
mov i,eax;
end;
ShowMessage(IntToStr(i));
end;
```

در مثال فوق اگر مقدار i برابر ۲۰۰ نباشد مقدار i با ۱۰۰۰ جمع می شود و اگر برابر باشد مقدار ۵۰۰ در Eax قرار میگیرد و با ۱۰۰۰ جمع می شود

از بلوک asm داخل روالها و توابع هم می شه استفاده کرد. مخصوصا اینکه کل تابع یا روال رو بخواین با کدهای اسمبلی بنویسید. مثل روال زیر:

```
procedure putch(c : char);assembler;
asm
    mov ah,02h
    mov dl,c
    int 21h
end;
```

به کلمه کلیدی assembler که آخر تعریف روال اومده ، توجه کنید.

روش دوم : روالهای خارجی

در این روش یه روال یا تابع رو که قبلا به زبان اسمبلی نوشته شده با کمک برنامه tasm یا برنامه های مشابه ، به فایل .obj تبدیل کرده و به برنامه توسط دستور {$L} ملحق می کنیم. مثلا:

```
CODE SEGMENT BYTE PUBLIC
ASSUME CS:CODE
PUBLIC SUM
SUM PROC FAR
    PUSH BP
    MOV BP,SP
    MOV AX,[BP+08]
    ADD BX,[BP+06]
    POP BP
    RET 4
SUM ENDP
CODE ENDS
END
```

این قطعه کد رو با اسم testasm.asm کامپایل کنید و بعد می تونید تو برنامه های پاسکال استفاده کنید:

```
program test4;
uses crt;
{$F+}

{$L testasm}
function sum (i,j : integer) : integer; external;
begin
      writeln(' 5 + 9 = ',sum(5,9));
      readln;
end.
```

توی این قطعه کدها باید به موارد زیادی توجه کنید ، از جمله external، FAR، PUBLIC، ASSUME و ...

استفاده از کدهای c و++c در دلفی

البته میتوان کد های C++ Builder را به صورت فایلهایی با پسوند obj.کامپایل میکنیم و بعد فایلهای obj موردنظر را با کمک Compiler Directive مربوطه ($L) در دلفی قرار دهیم با استفاده از برنامه COFF2OMF میتوانید فایلهای object تولید شده در c یا ++c را به object فایلهای قابل استفاده در Delphi تبدیل کنید و از انها استفاده نمود

مثال:

```
//COBJ Example
//This is an example of an OBJ created with Borland C++ that is linked
//into an EXE (DAPP.EXE) created with Delphi.
#include

//Declaration
extern "C"   void _stdcall COBJ_Function;()
```

```
void _stdcall COBJ_Function()
{
MessageBox(NULL,  "Hello from a Borland C++ OBJ,"!
        "Success", MB_OK | MB_TASKMODAL);
return;
}
```

یک تابع بنام COBJ_Function که هیچ پارامتر ورودی و خروجی ندارد و تابع MessageBox ویندوز را فراخوانی میکند

```
unit Simple;

interface

uses
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,Dialogs,StdCtrls;

type
TMain = class(TForm)
Button1: TButton;
Label1: TLabel;
procedure Button1Click(Sender: TObject);
private
{ Private declarations}
public
{ Public declarations }
end;

var
Main: TMain;
implementation
```

```
{$R *.DFM}
{Specify the name of the OBJ containing the function.}
{$L cobj.obj}

procedure COBJ_Function; StdCall; far; external;

procedure TMain.Button1Click(Sender: TObject);
begin
COBJ_Function;
end;
end.
```

ما تابع COBJ_Function درون فایل cobj.obj را به صورت

```
procedure COBJ_Function; StdCall; far; external;
```

تعریف کردیم و همانند توابع داخل دلفی از ان استفاده میکنیم

فصل ۱۴

طراحی زمان اجرا

زبان برنامه نویسی دلفی این توان رو داره که مولفه هایی رو حین اجرای برنامه به اون اضافه یا ازش کم کنه. مثالی هم زده شد که یه فرم خالی را با اعداد جدول ضرب پر می کردیم. حالا ممکنه سوال کنین چطور می شه به این کنترلها رویداد نسبت داد؟ مثلا OnClick یا OnCreate و ...

اگه یه شی کنترلی (مثلا Button) رو تو یه فرم دلفی قرار بدین و روی اون دوبل کلیک کنین ، یه تابع برای اختصاص رویداد OnClick واسه اون Button به شکل زیر درست می شه:

```
procedure TForm1.Button1Click(Sender: TObject);
```

این تابع یه ورودی به اسم Sender داره که در واقع شی فراخواننده تابع رو بهش منتقل می کنه (هر شیئی تو دلفی به طور مستقیم یا غیر مستقی از TObject مشتق می شه)! خوب حالا این سوال پیش می یاد که چه نیازی به ارسال این پارامتر هست. مگه برنامه نمی دونه این تابع مال

کدوم شیء کنترلیه؟ جواب هم بله هست و هم نه!! در واقع قدرت واقعی طراحی زمان اجرا همین جا معلوم می شه.

قطعه کد زیر رو در نظر بگیرید:

```
procedure TForm1.FormCreate(Sender: TObject);
Var
  Btn : TButton;
  i : Integer;
begin
  For i:=1 To 5 Do
  Begin
   Btn:=TButton.Create(Self);
   Btn.Parent:=Self;
   Btn.Top:=i*30;
   Btn.Left:=20;
   Btn.Caption:='0;'
   Btn.OnClick:=BtnClicks;
  End;
end;
```

این کد ۵ تا Button رو زمانی که فرم داره ایجاد می شه ، روی فرم قرار می ده اما یه دستور اضافی داره که وظیفه مهمی هم داره:

```
Btn.OnClick:=BtnClicks;
```

این دستور ، تابع BtnClicks رو به رویداد OnClick همون Button نسبت می ده (تعریف این تابع رو بعدا می بینید). در واقع همه Button ها از یه تابع برای رویداد OnClick استفاده می کنن. اما این تابع از کجا تشخیص می ده که توسط کدوم Button فراخوانی شده؟ قطعه کد مربوط به تابع BtnClicks رو بخونید تا متوجه بشید. البته بهتره به تشابه شیوه تعریف این تابع با تابع Button1Click بالا ، دقت کنید:

```
procedure TForm1.BtnClicks(Sender: TObject);
```

```
var
  x : Integer;
  Btn : TButton;
begin
  Btn:=TButton(Sender);
  x:=StrToInt(Btn.Caption);
  Inc(x);
  Btn.Caption:=Format('%d',[x]);
end;
```

فکر می کنم متوجه جواب سوال شدید : یا استفاده از Sender. تابع بالا هر بار که فراخوانی می شه ، مقدار Caption مربوطه رو یه واحد اضافه می کنه.

ما در زمان ایجاد Button در متد ;Btn:=TButton.Create(Self) پارامتر را Self در نظر گرفتیم. هنگام ایجاد یك شیئ در زمان اجرا (Run-Time) که از کلاس TControl مشتق شده مانند یك فرم (TForm) ، متد Create در انتظار یك پارامتر به عنوان "مالك" یا Owner کلاس هست. چه مقداری برای این پارامتر به عنوان مالك می‌بایست ارسال شود؟

فرض کنید فرمی به نام TMyForm در برنامه‌تان دارید، و می‌خواهید در زمان اجراء یك شیئ از این کلاس بسازید و استفاده کنید. به کلاس پدر TMyForm یعنی کلاس TForm برمی‌گردیم، این کلاس متدی به نام Create دارد ، که وظیفه ی ساخت و مقداردهی اولیه یك شیئ جدید که از TForm یا هر کلاس دیگری که از TForm مشتق شده را دارد:

مالك(Owner):

در اینجا، پارامتر AOwner، مالك شیئ هست. مالك(Owner) فرم ، مسئول آزاد سازی حافظه تخصیص داده شده به شیئ فرم در موقع لزوم هست. زمانیکه مالك فرم از بین برود ، به صورت خودکار فرم نیز از بین می‌رود و این موضوع برای تمامی کلاسها صادق ست.

نکته مهم اینست که ، کدام یکی از مقادیر Application، Self وnil را میبایست به عنوان پارامتر ارسال کنید؟

برای رسیدن به جواب ، ابتدا باید معنی و مفهوم کلمات application, nil, self را بدانیم.

nil : مشخص می‌کند که هیچ شیئ مالك فرم نیست و بنابراین ، برنامه نویس (شما) ، وظیفه دارد فرم ساخته شده را آزاد کند. مثلاً زمانی که دیگر نیازی به فرم ندارید می‌توانید با دستور myForm.Free ، حافظه مورد استفاده فرم را آزاد کنید.

درباره ی nil : nil يك ثابت خاص، جهت تخصيص به انواع اشاره‌گرهاست. nil، اختصار كلمه ی لاتين Nihil هست، كه معنی آن هيچی يا صفر هست، بعضی‌ها هم می‌گويند nil به معنی Not In List هست. اشاره‌گری كه مقدار nil دارد، هرگز به حافظه ی معتبری اشاره نمی‌كند، اما چون اين اشاره‌گر تعريف شده و برای آن حافظه‌ای رزرو شده، بعضی از متدها می‌توانند آن را تست كنند (مانند تابع assigned). نمی‌توانيد تفاوتی بين يك اشاره‌گر كه مقداردهی اوليه نشده و يك اشاره‌گر كه مقداردهی اوليه ی آن انجام گرفته بيابيد، در واقع راهی برای تشخيص وجود ندارد. در دلفی nil دارای مقدار ۰/صفر هست و اشاره به اولين بايت حافظه دارد كه ظاهراً اين بايت در اختيار كدهای دلفی قرار نمی‌گيرد.

Self : مشخص كننده شيئی ست كه متد Create را فراخوانی كرده (Self اشاره‌گری ست به كلاس جاری) ، فرض كنيد در فرمی با نام MainForm هستيد و می‌خواهيد شيئ MyForm را بسازيد، روی دكمه ای كليك می‌كنيد و كد مورد نظر را می‌نويسيد، در اينجا Self به كلاس MainForm اشاره دارد و نه دكمه‌ای كه كد را روی آن نوشتيد(در واقع دكمه، فيلدی از كلاس MainForm هست). بنابراين Self مساوی ست با MainForm ، پس هر زمانيكه MainForm از بين برود(آزاد شود) ، MyForm نيز از بين می‌رود(آزاد می‌شود).

Application : مشخص كننده يك متغير عمومی از نوع كلاس TApplication هست و زمانی ايجاد می‌شود كه برنامه‌تان را اجرا می‌كنيد و در زمان خاتمه برنامه نيز به همراه تمام اشيائی كه مالكشان هست از بين می‌رود. ايجاد و حذف آن بر عهده ی شما نيست و از اين بابت نگران نباشيد. اين كلاس و شيئ به ترتيب در يونيت Forms تعريف و در يونيت Controls ساخته می‌شود.كلاس TApplication هم يك كامپوننت هست اما در زمان طراحی نمی‌توانيد از آن استفاده كنيد. بعضی از ويژگی‌های Application را می‌توانيد مستقيماً در صفحه Application فرم Project Options تنظيم كنيد، برای مابقی تنظيمات هم، می‌بايست از كد استفاده كنيد.

مثال :

۱ - Modal form ها (فرمهايی كه تا وقتی بسته نشوند ، ادامه اجرای برنامه مقدور نمی‌باشد و اين به معنی مرگ برنامه نيست، بلكه كاربر می‌بايست/می‌تواند فرم را ببندد - متد ShowModal) : زمانی كه بايد نمايش داده شوند ، ساخته می‌شوند و بعد از بستن فرم توسط كاربر ، حافظه مربوطه آزاد می‌شود. در اين نوع فرمها می‌توان از پارامتر nil به عنوان مالك (Owner) استفاده كرد ، چون بعد از بستن فرم ، آزاد كردن حافظه بر عهده برنامه نويس هست :

```
var
  myForm : TMyForm;
begin
  myForm := TMyForm.Create(nil);
```

```
  try
    myForm.ShowModal;
  finally
    myForm.Free;
  end;
end;
```

۲ - Modaless Form ها (زمانی که می‌بایست از چند فرم به طور همزمان استفاده کرد، کاربرد دارند - متد Show) : در این نوع فرمها می‌توانید از پارامتر Application به عنوان مالك (Owner) استفاده کنید :

```
var
  myForm : TMyForm;
...
  myForm := TMyForm.Create(Application);
```

در زمان خاتمه ی برنامه ، شیئ Application حافظه مربوط به myForm را آزاد خواهد کرد.

چرا و چه موقع استفاده از TMyForm.Create(Application) پیشنهاد نمی‌شود ؟

می‌توانید application را پاس دهید، ولی وقتی این کار را انجام می‌دهید هر کامپوننت و فرمی که به طور مستقیم و غیر مستقیم ، application، مالك آن هست، می‌بایست از این عمل با خبر شود، پس نوعی آگاه سازی نسبت به این کامپوننتها و فرمها صورت میگیرد و اگر تعداد شان زیاد باشد (مثلاً چندین فرم و هزار کامپوننت) با تاخیر زمانی قابل توجهی در هنگام نمایش فرم روبرو خواهیم شد. در این جور مواقع بهتر است که از nil به جای application استفاده کنیم تا هم فرم سریعتر ظاهر شود و هم تاثیری در کد نخواهد داشت.

۳ – چه وقتی از self استفاده کنیم ؟ اگر فرم مورد نیاز شما از نوع Modal نیست و همچنین فرم اصلی برنامه هم نیست، اگر از self به عنوان مالك(Owner) استفاده کنید، با بسته شدن مالك(Owner)، فرم ساخته شده نیز فوراً آزاد می‌شود. معمولاً هنگام ایجاد یك فرم ، نباید از self استفاده کنید، در عوض از application یا nil استفاده کنید ، تفاوتی هم ندارد که modal یا modaless باشد. اما، از self زمانی استفاده کنید که نمی‌خواهید فرم ، طول عمری بیشتر از سازنده یا مالکش داشته باشد.

البته باید به نکته های زیر توجه کنید:

۱- تعریف اولیه تابع BtnClicks رو توی کلاس TForm1 قرار بدید:

```
public
  Procedure BtnClicks(Sender : TObject);
```

۲- یونیت StdCtrls رو به برنامه اضافه کنید (برای استفاده کردن از TButton).

فصل ۱۵

فایل های INI

فایل های .INI دارای ساختاری بر اساس فایلهای متنی هستند و برای نگهداری اطلاعات پیکر بندی برنامه های کاربردی استفاده میشوند که هم براحتی بوسیله ما قابل ویرایش هستند و هم بوسیله یک ساختار ساده در هر برنامه ساده قابل دسترسی هستند .

بدلیل اینکه ویندوز داری Registry هست کسانی که از ویندوز استفاده میکنند آشنایی کمی با فایلهای .ini دارند ولی در ویندوز هنوز هم از فایلهای .ini استفاده میشود. مثل Win.ini و System.ini . ویندوز از این فایلها برای ذخیره اطلاعات مهمی از جمله اطلاعات پیکربندی استفاده میکند که براحتی قابل پاک شدن ، ویرایش و دیدن هستند. بسیاری از برنامه های تحت ویندوز برای ذخیره اطلاعات پیکربندی خود از Registry استفاده میکنند در حالیکه استفاده از فایلهای .ini هم سریعتر و هم ایمن تر است . یک مثال ساده برای استفاده از فایلهای .ini ذخیره اندازه ، حالت و موقعیت فرم برنامه شماست . بطور کلی هر چیزی که شما در رجیستری ذخیره میکنید میشود در فایلهای .ini ذخیره کرد .

ساختار فایلهای .ini

فایلهای .ini نوعی فایل متنی هستند که به بخشهای محدود به ۶۴ کیلو بایت (Section) تقسیم شدند که هر بخش میتواند دارای چند کلید (Key) باشد و هر کلید میتواند دارای صفر یا چند مقدار (Value) باشد . مثال:

```
[SectionName]
keyname=value
;comment
keyname=value
```

نام هر بخش درون کروشه قرار گرفته و در باید در خط اول هر بخش قرار داشته باشد نام بخشها و نام کلیدها نمیتوانند کاراکتر فاصله داشته باشند. بعد از نام کلیدها علامت = قرار میگیرد که میتواند قبل و بعد از آن کاراکتر فاصله قرار بگیرد . اگر بخشهایی با نام یکسان در یک فایل یا کلیدهایی با نام یکسان در یک بخش قرار داشته باشند مقدار آخر بر بقیه مقدارهای یکسان غالب است .

یک کلید میتواند دارای مقادیری از نوعهای String , Integer , Boolean باشد. دلفی از فایلهایINI در خیلی وضعیتها استفاده میکند. برای مثال فایلهای .SDK نوعی فایل هستند مانند ini ها .

کلاس TiniFile

دلفی برای ذخیره و بازیابی فایلهای ini. کلاس TiniFile را در اختیار ما قرار داده است. این کلاس در یونیت inifiles.pas قرار گرفته است. قبل از کار کردن با فایلهای .ini لازم است یک مثال راجع به استفاده از این کلاس ببینیم.

```
uses inifiles;
...
var
  IniFile : TIniFile;
begin
  IniFile := TIniFile.Create('myapp.ini');
```

این کد یه فایل .ini ایجاد میکند و این فایل را به myapp.ini ارجاع میدهد . البته این کد فایل را درون پوشه ویندوز ایجاد میکند ولی بهتر این است که برای ذخیره کردن اطلاعاتی از برنامه فایل .ini را درون پوشه برنامه ایجاد کنید . برای این کار باید آدرس کامل فایل را بنویسید . مثال :

```
IniFile := TIniFile.Create('C:\Program Files\AppFolder\myapp.ini');
```

البته میتوانیم از تابع ChangeFileExt هم استفاده کنیم که در این صورت یک فایل با نام فایل برنامه و درون پوشه برنامه ایجاد میکنیم .

```
IniFile := TIniFile.Create(ChangeFileExt(Application.ExeName,'.ini'));
```

خواندن از فایلهای .ini

کلاس TiniFileچندین متد برای خواندن از فایلهای .ini دارند . متد ReadString برای خواندن مقدارهای رشته ای از یک کلید استفاده میشود. متد ReadInteger, ReadFloat ومتدهای مشابه برای خواندن مقدارهای عددی استفاده میشوند . همه این متدها یک مقدار پیش فرض دارند که وقتی فایل مورد نظر یا کلید و مقدار مورد نظر موجود نباشد استفاده میشود. مثلا ReadString به این صورت بیان میشود.

```
function ReadString(const Section, Ident ,Default: String): String; override;
```

Section نام بخش ، Ident نام کلید و Default نشان دهنده مقدار پیش فرض است .

نوشتن در فایلهای .ini

برای هر متد خواندن یک متد متناظر برای نوشتن وجود دارد . مثلا WriteString, WriteBool, WriteInteger و غیره

فرض کنید میخواهیم برنامه ای بنویسیم که تاریخ آخرین استفاده و آخرین موقعیت فرم برنامه را ذخیره کند . پس لازم است یک فایل.ini با دو بخش داشته باشیم . یک بخش با نام Date برای ذخیره تاریخ و یک بخش با نام Position برای ذخیره آخرین موقیت برنامه. بخش Date شامل کلید Last و بخش Position شامل کلیدهای Top, Left, width, Height .

کلید Last باید از نوع TDateTime و کلیدهای بخش Position باید از نوع عددی باشند.

برای نوشتن برنامه رویداد OnCreate فرم اصلی برنامه را بصورت زیر مینویسیم . (فراموش نکنید در بخش Uses یونیت TIniFiles را اضافه کنید.)

```
procedure TForm1.FormCreate(Sender: TObject);
var
  MyIniFile  : TIniFile;
  LastDate : TDateTime;
begin
  MyIniFile := TIniFile.Create(ChangeFileExt(Application.ExeName,'.ini'));
  LastDate := MyIniFile.ReadDate('Date', 'Last', Date);
  ShowMessage('This program was previously used on' + DateToStr(LastDate));
  Form1.Top := MyIniFile.ReadInteger(' Position','Top', Form1.Top);
  Form1.Left := MyIniFile.ReadInteger(' Position','Left', Form1.Left);
  Form1.Width := MyIniFile.ReadInteger('Position','Width', Form1.Width);
```

```
  Form1.Height := MyIniFile.ReadInteger('Position','Height', Form1.Height);
  MyIniFile.Free;
end;
```

با این کد در صورتی که هنگام اجرای برنامه فایل .ini مورد نظر وجود داشته باشد آخرین تاریخ استفاده از برنامه نشان داده میشود و فرم در آخرین موقعیت قبلی قرار میگیرد.

برای ذخیره شدن آخرین تاریخ و موقیت فرم ، رویداد OnClose فرم اصلی برنامه را به این صورت مینویسیم :

```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
var
  MyIniFile  : TIniFile;
begin
  MyIniFile := TIniFile.Create(ChangeFileExt(Application.ExeName,'.ini'));
  MyIniFile.WriteDate('Date', 'Last', Date);
  With MyIniFile, Form1 do
  begin
    WriteInteger('Position','Top', Top);
    WriteInteger('Position','Left', Left);
    WriteInteger('Position','Width', Width);
    WriteInteger('Position','Height', Height);
  end;
  MyIniFile.Free;
end;
```

این کد باعث میشود در هنگام بسته شدن برنامه تاریخ و موقعیت فرم در فایل .ini ذخیره شود.

کار کردن با بخشها

خواندن لیست گروه ها (Sections) :

منظور از گروه همان Section ای است که برای نوشتن اطلاعات وارد نموده اید ،

متد های ReadSection نام کلیدهای یک بخش و متد ReadSections نام بخشهای یک فایل را در یک TStringList قرار میدهد.

به عنوان مثال اگر سه داده String در فایل نوشته باشید و نام Section هر سه آنها را " MySection " گذاشته باشید ، می توانید ، آن ۳ مقدار را در یک TStrings ، قرار دهید

ساختار این متدها به شرح زیر می باشد :

```
ReadSection( Section : String;  Strings : TStrings) ;
ReadSectionValues( Section : String;  Strings : TStrings) ;
```

با استفاده از متد ReadSections میتوانید ، لیست تمام گروه های موجود در فایل INI را دریافت نمایید و در یک TStrings قرار دهید :

```
ReadSections( Strings : TStrings );
```

متد EraseSection :

با استفاده از این متد میتوانید یک Section خاص را از فایل حذف نمایید (اطلاعات آن را پاک کنید) :

```
EraseSection( Section : String );
```

متد DeleteKey :

به وسیله این متد می توانید یک مقدار را از فایل حذف نمایید ، این متد دو پارامتر برای نام Section و نام Ident دارد :

```
DeleteKey( Section : String;   Ident : String) ;
```

تابع SectionExists :

با استفاده از این تابع میتوانید چک کنید که آیا یک Section خاص در فایل قرار دارد یا خیر ،مقدار برگشتی این تابع از نوه Boolean می باشد :

```
SectionExists( Section : String ) : Boolean;
```

تابع ValueExists :

به وسیله این تابع میتوانید چک کنید که آیا یک مقدار در فایل وجود دارد یا خیر ، مقدار برگشتی این تابع از نوع Boolean می باشد :

```
ValueExists( Section : String;  Ident : String);
```

متد UpdateFile :

این متد وظیفه بروز رسانی فایل INI و پاک کردن بافر استفاده شده فایل INI را دارد و بهتر است بعد از چندین عمل خواندن و نوشتن فراخوانی شود تا فایل INI برای کار مجدد آماده شود.

خاصیت FileName :

این خاصیت فقط خواندنی می باشد (ReadOnly) و به وسیله آن می توانید نام (و آدرس) فایل INI را نمایش دهید ...

کلاسهای دیگری هم در یونیت Registry وجود دارند از جمله TRegIniFile برای دسترسی ساده به سیستم رجیستری ویندوز بصورت فایلهای ini که استفاده از آنها ساده است.

محدودیتها و راه حل ها

بدلیل اینکه کلاس TIniFile از Windows API استفاده میکند به محدودیت ۶۴ کیلو بایتی به فایلهای ini تحمیل میشود. در صورتی که احتیاج دارید اطلاعاتی بیشتر از ۶۴ کیلو بایت در فایل ذخیره کنید باید بجای استفاده از TIniFile از TMemIniFile استفاده کنید که در این صورت مشکل محدودیت ۶۴ کیلو بایتی را ندارید.