The Complete Reference

راهنمای جامع

Delphi 7 Studio

2003

در این کتاب می‌خوانید :

- دایره المعارف اجزاء سازندهٔ موجود در Delphi 7
- طراحی و ساخت اجزاء سازندهٔ CLX و VCL
- پیاده‌سازی بانک‌های اطلاعاتی به وسیلهٔ DBExpress
- کار با بانک‌های اطلاعاتی ADO
- طراحی و ساخت برنامه‌های کاربردی COM+ و CORBA
- ساخت برنامه‌های سرویس‌دهندهٔ WEB
- ساخت برنامه‌های کاربردی ASP
- طراحی و ساخت برنامه‌های Wireless
- آشنایی با قابلیت‌های .NET و نحوهٔ بکارگیری آن در Delphi 7

تالیف :
مهندس احمد پهلوان تفتی
مهندس مهدی آصفی

بسم الله الرحمن الرحیم

Delphi 7

احمد پهلوان تفتی

مهدی آصفی

پهلوان تختی، احمد
Delphi 7 [دلفی ۷] / احمد پهلوان تفتی، مهدی آصفی . ـ تهران : ناقوس، ۱۳۸۲ .
۵۶۸ ص. : مصور.
ISBN 964-377-041-9 : ۴۵۰۰۰ ریال
فهرستنویسی بر اساس اطلاعات فیپا.
۱. دلفی (فایل کامپیوتر). ۲. نرم‌افزار ـ ـ تولید. ۳. زبان‌های برنامه نویسی تصویری.
الف. آصفی، مهدی، ۱۳۵۳ ـ ب. عنوان.
۹ پ۸د / ۷۶ / QA ۷۶ ۸د۲۵۶ / ۰۰۵
کتابخانهٔ ملی ایران ۸۲-۱۱۳۵م

انتشارات ناقوس
انتشارات اشاره

نام کتاب	:	راهنمای جامع Delphi 7
ناشر	:	انتشارات ناقوس
نویسنده	:	احمد پهلوان تفتی، مهدی آصفی
چاپ دوم	:	پاییز ۱۳۸۲
تیراژ	:	۱۰۰۰
لیتوگرافی	:	سیناگرافیک
چاپ	:	سعدی
صحافی	:	نوری
قیمت	:	۴۵۰۰۰ ریال
شابک	:	۹۶۴-۳۷۷-۰۴۱-۹
ISBN	:	964-377-041-9

www.naghoospress.com

مرکز پخش : انتشارات ناقوس
تهران : خیابان انقلاب ـ خیابان ۱۲ فروردین ـ کوچهٔ‌نوروز ـ پلاک ۲۷
تلفن : ۶۴۰۱۳۹۳ - ۶۴۰۶۸۳۴ - ۶۴۱۱۷۱۵

فهرست مطالب

فصل اول: به دلفی ۷ خوش آمدید

فصل دوم: پاسکال شی‌گرا

فصل سوم: کتابخانهٔ اجزاء سازندهٔ دلفی ۷

فصل چهارم: پیام‌ها در دلفی

فصل پنجم: معماری بانک‌های اطلاعاتی در دلفی

فصل ششم: dbExpress ابزاری برای طراحی بانک‌های اطلاعاتی

فصل هفتم: طراحی بانک‌های اطلاعاتی ADO به کمک dbGO

فصل هشتم: ساخت اجزاء سازنده VCL

فصل نهم: ساخت اجزاء سازنده VCL پیشرفته

فصل دهم: ساخت اجزاء سازنده CLX

فصل یازدهم: بسته‌ها

فصل دوازدهم: بکارگیری Open Tools API

فصل سیزدهم: استفاده از سرویس‌های COM+/MTS در طراحی برنامه‌های تراکنشی

فصل چهاردهم: طراحی و پیاده‌سازی برنامه‌های CORBA

فصل پانزدهم: طراحی سرویس‌های وب بر پایهٔ SOAP

فصل شانزدهم: طراحی برنامه‌های Multitier

فصل هفدهم: طراحی و ساخت برنامه‌های ASP

فصل هجدهم: استفاده از WebSnap در طراحی سرویس‌دهنده‌های وب

فصل نوزدهم: طراحی Wireless در دلفی

مقدمه

فن‌آوری و دانش کامپیوتر با سرعتی باور نکردنی و بدون وقفه در حال تغییر بوده و در این بین طراحی و توسعه سیستم‌های نرم‌افزاری نقشی محوری داشته است. طراحی نرم‌افزارهای قوی نیازمند ابزارهای متعددی است که مهمترین این ابزارها، زبانی قدرتمند برای برنامه‌سازی است.

امروز دلفی به عنوان یک ابزار منحصر بفرد و قدرتمند در زمینهٔ تولید نرم‌افزارهای کاربردی اعم از بانک‌های اطلاعاتی، برنامه‌های گرافیکی، برنامه‌های سرویس‌دهندهٔ وب و ... مطرح می‌باشد و در این میان از مزایای بی‌شماری نسبت به سایر زبان‌های برنامه‌سازی همچون VB ، Visual C و ... برخوردار است.

کتاب حاضر به عنوان مرجع کاملی برای دلفی ۷ به همراه مثال‌ها و برنامه‌های کاربردی متنوع به رشته تحریر در آمده است. موضوعات هر بخش از کتاب به گونه‌ای تدوین شده است که ضمن تشریح دقیق مطالب و نکات تخصصی، مهارت‌های کاربردی جدیدی را نیز به آموخته‌های شما می‌افزاید. بدیهی است مبنای نظری کتاب به همراه راه‌کارهای علمی ارائه شده در آن، تضمین کنندهٔ یادگیری مؤثر این زبان برنامه‌سازی خواهد بود.

در پایان بر خود لازم می‌دانیم از همکاری صمیمانه سرکار خانم مهندس الهام ثاقب حسین‌پور به خاطر ویراستاری علمی کتاب و سرکار خانم‌ها ساناز حقانی و مریم پهلوان تفتی و ستاره ثاقب حسین‌پور که در شکل‌گیری این اثر همراه ما بودند تشکر و سپاسگزاری نمائیم. مراحل فنی چاپ کتاب برعهدهٔ انتشارات ناقوس بوده است که صمیمانه از مدیریت محترم این مؤسسه، جناب آقای کاوه تجملی کمال قدردانی را داریم.

پایان سخن این که، این کتاب تلاشی است بی‌ادعا و ادای دینی است نسبت به جامعهٔ انفورماتیک کشور، بدین لحاظ از خوانندگان محترم این مجموعه تقاضا داریم، خطاها و لغزشهای کتاب را کریمانه اعلام فرمایند تا ان‌شاءالله در چاپهای بعدی مورد تصحیح قرار گیرد.

احمد پهلوان تفتی – مهدی آصفی
زمستان ۱۳۸۱

فصل اول

به دلفی ۷ خوش آمدید

در این فصل می‌خوانید

- **ویژگی‌های دلفی**
- **ملزومات مورد نیاز جهت نصب دلفی ۷**

دلفی یک محصول نرم‌افزاری بی‌نظیر و قدرتمند از شرکت Borland است که ویژگی‌های منحصر به فرد آن در انعطاف‌پذیری، Visual بودن، شیءگرایی و سادگی، آن را نسبت به سایر محیط‌های برنامه‌سازی همچون VB و C++ متمایز می‌نمایاند.

دلفی براساس یک زبان شیءگرای کامل به نام Object Pascal تهیه شده است. با استفاده از دلفی قادر خواهید بود تمام کارهای قابل انجام با سایر زبان‌های برنامه‌نویسی چون C++ و VB را انجام داده و در عین حال با مشکلات کمتری در حین کار مواجه شوید.

دلفی ابزاری است که شما را در تهیه و طراحی برنامه‌های کاربردی متنوع و همه منظوره با عملکرد بالا یاری می‌رساند.

ملزومات سیستم

قبل از هر چیز لازم است تا نرم‌افزار دلفی را بر روی سیستم خود نصب نمائید. چهار نگارش مختلف از دلفی ۷ موجود است. جدول ۱-۱ ملزومات سیستمی مورد نیاز هر یک از این نگارش‌ها را نشان می‌دهد.

جدول ۱-۱ ملزومات مورد نیاز جهت نگارش‌های مختلف دلفی ۷

نگارش	ملزومات
Delphi7 Studio Architect	● پردازنده پنتیوم با سرعت حداقل 233 mhz ● حداقل 64 MB حافظهٔ اصلی (RAM) ● حداقل 520 MB فضای خالی بر روی هارد دیسک ● سیستم عامل Windows XP یا Windows 2000 یا Windows 98

Delphi 7 Studio Enterprise	● پردازنده پنتیوم با سرعت حداقل 233 mhz ● حداقل 64 MB حافظهٔ اصلی ● حداقل 450 MB فضای خالی بر روی هارد دیسک ● سیستم عامل Windows XP یا Windows 2000 یا Windows 98
Delphi7 Studio Professional	● پردازنده پنتیوم با سرعت حداقل 233 mhz ● حداقل 64 MB حافظهٔ اصلی ● حداقل 400 MB فضای خالی بر روی هارد دیسک ● سیستم عامل Windows XP یا Windows 2000 یا Windows 98
Delphi7 Personal	● پردازنده پنتیوم با سرعت حداقل 233 mhz ● حداقل 32 MB حافظه اصلی ● حداقل 160 MB فضای خالی بر روی هارد دیسک ● سیستم عامل Windows XP یا Windows 2000 یا Windows 98

آنچه که در این کتاب مطرح شده است مربوط به نگارش Delphi7 Studio Professional می‌باشد.

مطالبی در مورد این کتاب

این کتاب برای برنامه‌نویسان پیشرفته دلفی و برنامه‌نویسان دیگری تهیه شده است که مایلند از زبان‌هایی همچون VB یا C++ به محیط دلفی روی آورند.

در تمام فصل‌های کتاب، هر کجا که مفاهیم و اصول اولیه (برای برنامه‌نویسان مبتدی دلفی) مورد نیاز بوده است، سعی شده است با اشارهٔ مختصری به این اصول، نیاز این گروه از خوانندگان نیز رفع گردد و پس از آن بیشتر به زمینه‌های حرفه‌ای برنامه‌نویسی دلفی و ارائه مثال‌های متنوع و کاربردی پرداخته شده است. این کتاب همانند پل ارتباطی میان دلفی ۷ و برنامه‌سازانی است که می‌خواهند سطح دانش خود را در زمینه برنامه‌نویسی دلفی ارتقاء دهند.

فصل دوم

پاسکال شی‌گرا

در این فصل می‌خوانید

- **تعریف متغیرها و ثابت‌ها**
- **عملگرها**
- **انواع داده‌ها**
- **نوع داده‌ای Variant**
- **آرایه‌ها، مجموعه‌ها و رکوردها**
- **تعریف روال‌ها و توابع**
- **سازنده‌ها و مخرب‌ها**
- **متدها**
- **استثناها**

پاسکال شی‌گرا زبان برنامه‌سازی است که قدمت زیادی داشته و در ابتدا با عنوان توربو پاسکال ارائه گردیده است. این زبان برنامه‌سازی در طی دوران تکاملی خود و در بین برنامه‌نویسان به عنوان یکی از قوی‌ترین زبانها شناخته شده است. قابلیت شی‌ءگرا از توربو پاسکال نسخهٔ ۵/۵ به بعد به این زبان اضافه گردید و آنچه که در دلفی ۷ خواهید دید نقطهٔ اوج تکاملی برای این زبان بوده است.

در این فصل اصول تشکیل دهنده و ساختار درونی این زبان را ارائه خواهیم نمود.

عناوین توصیفی

عناوین توصیفی در این زبان به سه طریق ارائه شده است. نمونهٔ هر یک از این روشها در زیر آورده شده است.

```
{Comment using curly braces}
(* Comment using paren and asterisk *)
// double backslash comment
```

بارگذاری اضافی

بارگذاری اضافی در برنامه‌نویسی شیءگرا به مفهوم استفادهٔ همزمان از عناوین مشابه در متغیرها، توابع و رویه‌هاست. باید توجه داشته باشید که در بارگذاری اضافی متغیرها، توابع و رویه‌ها، تنها نام آنها می‌تواند یکسان در نظر گرفته شود و لذا نوع و آرگومانهای ارجاعی به آنها متفاوت خواهند بود.

به عنوان مثال در زیر چند نمونه مختلف از رویه Hello ارائه شده است.

```
Procedure Hello (I: Integer); Overload;
Procedure Hello (I: String); Overload;
Procedure Hello (I: Double); Overload;
```

متغیرها

متغیر عنصری است که مقدار آن در طی اجرای برنامه تغییر می‌کند. در پاسکال شیءگرا، تعریف نوع متغیرها از اساسی‌ترین اعمال لازم است. برنامه‌نویسانی که با C یا Java کار می‌کنند کمتر به این روش عادت دارند. بدین معنی که در C یا Java با تخصیص یک مقدار به متغیر می‌توان نوع آن را مشخص نمود. اما در پاسکال شیءگرا، اولین قدم مربوط به تعیین نوع متغیرهاست. برای درک بیشتر این مسئله دو روال به نام‌های foo و FOO را در پاسکال شیءگرا و C نمایش می‌دهیم.

```
Procedure FOO;
{
int X = 1;
X ++;
int Y = 2;
float f;
// ... etc ...
}
```

```
Void foo (Void)
Var
X, Y: Integer;
f: Double;
begin
X: = 1;
inc(X);
Y: 2;
// ... etc ...
end;
```

■ نکته

در پاسکال شیءگرا (همانند بیسیک) حروف کوچک و بزرگ تفاوت نخواهند داشت.

ثابت‌ها

در برخی از برنامه‌ها لازم است تا یک مقدار خاص را بارها مورد استفاده قرار دهید. ثابت‌ها در اینگونه مواقع به‌کار می‌آیند. ثابت‌ها چیزی جز یک نام برای مقادیر مورد استفاده در برنامه‌هایتان نیستند. در زیر نمونه‌ای از تعریف ثابت‌ها ارائه شده است.

```
Const
ADecimalNumber: Double = 3.14;
I: Interger = 10;
ErrorString: String = 'Danger, Danger, Danger!';
```

در پاسکال شی‌گرا روال‌هایی نیز جهت تعریف انواع ویژه‌ای از ثابت‌ها وجود دارد. این روالها عبارتند از Ord()، Chr()، Trunc()، Round()، High()، Low() و Sizeof() که در قطعه کد زیر نحوهٔ به کارگیری آنها را نمایش داده‌ایم.

```
type
  A = array[1..2] of Integer;

const
  w: Word = SizeOf(Byte);

var
  i: Integer = 8;
  j: SmallInt = Ord('a');
  L: Longint = Trunc(3.14159);
  x: ShortInt = Round(2.71828);
  B1: Byte = High(A);
  B2: Byte = Low(A);
  C: char = Chr(46);
```

عملگرها

شیوهٔ مقایسه و ارزشیابی متغیرها و ثابت‌ها در زبان‌های برنامه‌سازی به وسیلهٔ عملگرها انجام می‌پذیرد. در نتیجه مقایسه و ارزشیابی عبارت‌ها در زبان‌های برنامه‌سازی به روش استفاده از عملگرها وابسته است.

عملگر انتساب

از عملگر انتساب جهت تخصیص مقدار به متغیرها و یا عبارتها استفاده می‌شود. این عملگر در زبان C و Java متناظر با '=' و در زبان پاسکال شی‌گرا متناظر با ':=' است. به عنوان مثال عبارت N: =1; مقدار عددی یک را به متغیر N نسبت می‌دهد.

عملگرهای مقایسه‌ای

عملگرهای مقایسه‌ای برای مقایسه دو متغیر مورد استفاده قرار می‌گیرند. این عملگرها به صورت >، = و < و یا ترکیبی از آنها مورد استفاده واقع می‌شوند.

عملگرهای منطقی

عملگرهای منطقی در رابطه با مقایسه یا پردازش مقادیر در سطح TRUE یا FALSE می‌باشند.

جدول ۱-۲ عملگرهای انتسابی، مقایسه‌ای و منطقی

Operator	*Pascal*	*Java/C*	*Visual Basic*
Assignment	:=	=	=
Comparison	=	==	= or Is*
Not equal to	<>	!=	<>
Less than	<	<	<
Greater than	>	>	>
Less than or equal to	<=	<=	<=
Greater than or equal to	>=	>=	>=
Logical and	and	&&	And
Logical or	or	||	Or
Logical not	not	!	Not

**The* `Is` *comparison operator is used for objects, whereas the = comparison operator is used for other types.*

فهرست عملگرهای مقایسه‌ای، انتسابی و منطقی در سه زبان Basic ، C و پاسکال در جدول ۱-۲ ارائه شده است.

عملگرهای حسابی

عملگرهای حسابی امکان انجام عملیات بر روی یک یا دو عملوند را فراهم می‌آورند. این عملگرها عموماً به صورت عبارتهای جبری استفاده می‌شوند. فهرست این عملگرها در جدول ۲-۲ آورده شده است.

جدول ۲-۲ فهرست عملگرهای حسابی

Operator	*Pascal*	*Java/C*	*Visual Basic*
Addition	+	+	+
Subtraction	-	-	-
Multiplication	*	*	*
Floating-point division	/	/	/
Integer division	div	/	\
Modulus	mod	%	Mod
Exponent	*None*	*None*	

عملگرهای بیتی

این عملگرها در رابطه با شیفت دادن و یا مقایسه مقادیر در سطح بیت‌ها هستند. جدول ۳-۲ فهرست

جدول ۳-۲ فهرست عملگرهای بیتی

Operator	*Pascal*	*Java/C*	*Visual Basic*
And	`and`	`&`	`And`
Not	`not`	`~`	`Not`
Or	`or`	`|`	`Or`
Xor	`xor`	`^`	`Xor`
Shift+left	`shl`	`<<`	*None*
Shift+right	`shr`	`>>`	*None*

این عملگرها را نشان می‌دهد.

افزایش و کاهش تک واحدی

در اکثر زبان‌های برنامه‌سازی روال‌هایی برای افزایش یا کاهش تک واحدی مقادیر وجود دارد. عمل افزایش تک واحدی در پاسکال شی‌گرا توسط روال ()Inc و عمل کاهش تک واحدی به وسیلهٔ روال ()Dec انجام می‌پذیرد. البته در روالهای فوق امکان تعریف میزان افزایش یا کاهش نیز وجود دارد. به طور مثال عبارت Inc(Varicble, 3) باعث افزایش ۳ واحدی متغیر ذکر شده می‌شود. لیست مقایسه‌ای این روال‌ها در سه زبان Basic ، C و پاسکال در جدول ۴-۲ ارائه شده است.

جدول ۴-۲ عملگرهای افزایش و کاهش تک واحدی

Operator	*Pascal*	*Java/C*	*Visual Basic*
Increment	`Inc()`	`++`	*None*
Decrement	`Dec()`	`--`	*None*

انواع داده‌ها در پاسکال شی‌گرا

شکل داده‌ها نقش مهمی در چگونگی درک آنها ایفا می‌کند. با وجود محدود بودن گنجایش حافظه می‌بایست روشی اتخاذ گردد که حافظه هدر نرود.

پاسکال شی‌گرا به عنوان زبانی که انواع داده‌ها را به دقت رعایت می‌کند شناخته شده است. این بدان معناست که پاسکال شی‌گرا تضمین می‌کند که انواع مختلف داده‌ها می‌توانند با روشی ساخت یافته با یکدیگر ارتباط برقرار کنند. پاسکال شی‌گرا چندین نوع از داده‌ها را پشتیبانی می‌کند. انواع این داده‌ها در جدول ۵-۲ شرح داده شده است.

جدول ۵-۲ انواع داده‌ها در زبان‌های **Basic**، **Java** و **C** و پاسکال شی‌گرا

Type of Variable	*Pascal*	*Java*	*C/C++*	*Visual Basic*
8-bit signed integer	`ShortInt`	`byte`	`char`	None
8-bit unsigned integer	`Byte`	None	`BYTE`, `unsigned short`	`Byte`
16-bit signed integer	`SmallInt`	`short`	`short`	`Short`
16-bit unsigned integer	`Word`	None	`unsigned short`	None
32-bit signed integer	`Integer`, `Longint`	`int`	`int`, `long`	`Integer`, `Long`
32-bit unsigned integer	`Cardinal`, `LongWord`	None	`unsigned long`	None
64-bit signed integer	`Int64`	`long`	`__int64`	None
4-byte floating point	`Single`	`float`	`float`	`Single`
6-byte floating point	`Real48`	None	None	None
8-byte floating point	`Double`	`double`	`double`	`Double`
10-byte floating point	`Extended`	None	`long. double`	None
64-bit currency	`currency`	None	None	`Currency`
8-byte date/time	`TDateTime`	None	None	`Date`
16-byte variant	`Variant`, `OleVariant`, `TVarData`	None	`VARIANT**` `Variant†`, `OleVariant†`	`Variant(Default)`
1-byte character	`Char`	None	`char`	None
2-byte character	`WideChar`	`char`	`WCHAR`	
Fixed-length byte string	`ShortString`	None	None	None
Dynamic string	`AnsiString`		`AnsiString†`	`String`
Null-terminated string	`PChar`	None	`char *`	None
Null-terminated wide string	`PWideChar`	None	`LPCWSTR`	None

جدول ۵-۲ ادامه

Dynamic 2-byte string	`WideString`	`String**`	`WideString†`	None
1-byte Boolean	`Boolean`, `ByteBool`	`boolean`	(Any 1-byte)	None
2-byte Boolean	`WordBool`	None	(Any 2-byte)	`Boolean`
4-byte Boolean	`BOOL`, `LongBool`	None	`BOOL`	None

†A proprietary Borland C++Builder class that emulates the corresponding Object Pascal type
***Not a language element proper, but a commonly used structure or class*

نوع کاراکتری

هر کاراکتر یک بایت است. قطعاً می‌دانید که ۲۵۶ نوع کاراکتر مختلف را می‌توان در متغیری از نوع کاراکتری ذخیره نمود (۲۵۶ = ۲^۸) در پاسکال شی‌گرا سه نوع داده از نوع کاراکتری قابل تعریف می‌باشد که در زیر به شرح آنها خواهیم پرداخت.

- Ansichar : یک کاراکتر ۸ بیتی ANSI می‌باشد.
- Widechar : یک نوع کاراکتری ۱۶ بیتی کامل است.
- char : در حال حاضر با Ansichar برابر است ولی ممکن است در نگارش‌های بعدی دلفی تغییر نماید.

نوع رشته‌ای

داده‌های رشته‌ای عموماً به صورتی مفیدتر از داده‌های نوع کاراکتری عمل می‌کنند. غالباً رشته‌ها را به صورت آرایه‌ای از کاراکترها می‌شناسند. فهرست انواع رشته‌ای در پاسکال شی‌گرا به شرح زیر می‌باشد.

- Ansistring : طول آن تا حداکثر ۳ مگا بایت قابل تعریف بوده و دربرگیرندهٔ کاراکترهای Ansi می‌باشد. ضمناً با کاراکتر Null ختم خواهد شد.
- Shortstring : طول آن حداکثر ۲۵۵ کاراکتر و دربرگیرنده کاراکترهای Ansi است. در این نوع رشته‌ای کاراکتر Null برای خاتمه رشته استفاده نمی‌شود.
- Widestring : طول آن حداکثر ۱/۵ گیگا بایت و حاوی کاراکترهای Widechar می‌باشد. این نوع رشته‌ای با کاراکتر Null خاتمه می‌یابند.
- Pchar : اشاره‌گری به یک رشته (از نوع رشته‌های ختم شونده به کاراکتر Null) می‌باشد.
- PAnsichar : اشاره‌گری به رشته‌ای از نوع Ansichar می‌باشد.
- Pwidechar : اشاره‌گری به یک رشته از نوع Widechar می‌باشد.

توجه داشته باشید که نوع پیش‌فرض برای انواع رشته‌ای نوع Ansistring می‌باشد. یعنی چنانچه

نوع رشته را مشخص نکنید، پاسکال شی‌گرا آن را به عنوان Ansistring خواهد شناخت.

```
Var
S: String // S is an AnsiString
```

پاسکال شی‌گرا از رشته‌های طولانی نیز پشتیبانی می‌کند. این پشتیبانی با استفاده از رهنمای کامپایلر $H فعال می‌شود. چنانچه از رهنمای کامپایلر $H- استفاده کنید رشته‌ها به صورت shortstring در نظر گرفته می‌شوند.

```
Var
{$H-}
S1 : string ; // S1 is a shortstring
{$H+}
S2: String ; // S2 is an Ansistring
```

عملگرهای رشته‌ای

منظور از عملگرهای رشته‌ای، عملگرهایی است که بر روی رشته اعمال می‌شوند. یکی از عمده‌ترین این عملگرها، عملگر + است که از آن جهت الحاق دو رشته استفاده می‌شود. در پاسکال شی‌گرا برای انجام این عملیات هم می‌توانید از عملگر + استفاده کنید و هم تابع Concat را به کار ببرید. به مثال زیر در این رابطه توجه کنید.

```
{ using + }
var
  S, S2: string
begin
  S:= 'Cookie ':
  S2 := 'Monster';
  S := S + S2;   { Cookie Monster }
end.
```

```
{ using Concat() }
var
  S, S2: string;
begin
  S:= 'Cookie ';
  S2 := 'Monster';
  S := Concat(S, S2);  { Cookie Monster }
end.
```

تخصیص حافظه

Ansistring رشته‌ای است که به Null ختم شده و تخصیص فضای آن به صورت پویا انجام می‌شود. به محض این که رشته‌های طولانی‌تری در این گونه متغیرها قرار می‌دهید، دلفی عمل تخصیص فضا را

مجدداً انجام می‌دهد. اگر در نظر دارید که طول رشته موردنظر خود را به طور دلخواه تغییر دهید از تابع SetLenght() برای تخصیص حافظه تقریبی مورد نیاز استفاده نمائید. برای درک این مسئله به قطعه کد زیر توجه کنید.

```
var
  S: string;          // string initially has no length
begin
  S := 'Doh!';        // allocates at least enough space for string literal
  { or }
  S := OtherString    // increases ref count of OtherString
                      // (assume OtherString already points to a valid string)
  { or }
  SetLength(S, 4);    // allocates enough space for at least 4 chars
end;
```

توجه داشته باشید که در صورت استفاده از ایندکس (شمارهٔ خانه در یک رشته) باید حتماً مقدار حافظهٔ تخصیص داده شده را مشخص نمائید. برای مثال به کد زیر توجه کنید.

```
var
  S: string;
begin
  S[1] := 'a';  // Won't work because S hasn't been allocated!
end;


var
  S: string;
begin
  SetLength(S, 1);
  S[1] := 'a';        // Now S has enough space to hold the character
end;
```

قابلیت‌های Win32

در اکثر مواقعی که روتین‌های سیستم فرا خوانده می‌شوند مثلاً Win32 API ، می‌بایست از رشته‌هایی که به تهی ختم می‌شوند استفاده شود. لذا در صورتی که از رشته‌های نوع قدیمی مانند Short String استفاده می‌کنید لازم است تا قبل از فراخوانی API عملیات تبدیل را انجام دهید.

اگر قصد دارید در برنامه‌هایتان از نوع Pchar (اشاره‌گر) استفاده نمائید لازم است تا عملیات تخصیص حافظه را با دقت کافی انجام دهید. قبل از ارائه مثال در رابطه با نوع Pchar ضروری می‌دانیم تا توابع مربوط به تخصیص و بازپس‌گیری حافظه را در پاسکال شی‌گرا معرفی کنیم. این توابع در جدول ۶-۲ ارائه شده است.

جدول ٦-۲ توابع مربوط به تخصیص و بازپس‌گیری حافظه

تخصیص حافظه	بازپس‌گیری حافظه
AllocMem()	FreeMem()
GlobalAlloc()	GlobalFree()
GetMem()	FreeMem()
New()	Dispose()
StrAlloc()	StrDispose()
StrNew()	StrDispose()
VirtualAlloc()	VirtualFree()

به قطعهٔ کد زیر که در رابطه با روش استفاده از Pchar و عملیاتهای تخصیص حافظه می‌باشد توجه کنید.

```
var
  P1, P2: PChar;
  S1, S2: string;
begin
  P1 := StrAlloc(64 * SizeOf(Char));  // P1 points to an allocation of 63 Chars
  StrPCopy(P1, 'Delphi 6 ');          // Copy literal string into P1
  S1 := 'Developer''s Guide';         // Put some text in string S1
  P2 := StrNew(PChar(S1));            // P1 points to a copy of S1
  StrCat(P1, P2);                     // concatenate P1 and P2
  S2 := P1;                      // S2 now holds 'Delphi 6 Developer's Guide'
  StrDispose(P1);                     // clean up P1 and P2 buffers
  StrDispose(P2);
end.
```

نوع Variant

نوع Variant یکی از انواع داده‌های پیشرفته است که در دلفی وجود دارد. این نوع می‌تواند یک عدد صحیح، رشته یا مقادیر عددی با ممیز شناور را حفظ کند و کاربرد آن به اندازه‌ای است که نام آن مشخص می‌کند. نوع Variant یک ساختار داده‌ای ۱۶ بیتی است که علاوه بر مقدار، نوع اطلاعات موجود در خود را نیز ذخیره می‌کند. ارزش واقعی استفاده از این نوع داده در زمینهٔ اتوماسیون OLE[1] است.

نوع Variant به صورت پویا عمل می‌کند

یکی از دلایل استفاده Variant ها عدم تشخیص نوع داده‌ها در زمان کامپایل می‌باشد. این بدان معناست که مقداردهی نوع Variant در زمان اجرا می‌تواند به صورت متفاوت و پویا انجام گیرد. قطعه کد صفحه بعد نحوهٔ کارآیی انواع داده‌ای Variant را در عمل نشان می‌دهد.

۱- Object Linking and Embedding

```
var
  V: Variant;
begin
  V := 'Delphi is Great!';   // Variant holds a string
  V := 1;                    // Variant now holds an Integer
  V := 123.34;               // Variant now holds a floating point
  V := True;                 // Variant now holds a boolean
  V := CreateOleObject('Word.Basic'); // Variant now holds an OLE object
end;
```

همانطور که ملاحظه می‌کنید Variant انواع متنوعی از داده‌ها را پشتیبانی می‌کند.

ساختار Variant

ساختار نوع Variant در یونیت System به صورت زیر تعریف شده است.

```
TVarType = Word;
PVarData = ^TVarData;
{$EXTERNALSYM PVarData}
TVarData = packed record
  VType: TVarType;
  case Integer of
    0: (Reserved1: Word;
        case Integer of
          0: (Reserved2, Reserved3: Word;
              case Integer of
                varSmallInt: (VSmallInt: SmallInt);
                varInteger:  (VInteger: Integer);
                varSingle:   (VSingle: Single);
                varDouble:   (VDouble: Double);
                varCurrency: (VCurrency: Currency);
                varDate:     (VDate: TDateTime);
                varOleStr:   (VOleStr: PWideChar);
                varDispatch: (VDispatch: Pointer);
                varError:    (VError: LongWord);
                varBoolean:  (VBoolean: WordBool);
                varUnknown:  (VUnknown: Pointer);
                varShortInt: (VShortInt: ShortInt);
                varByte:     (VByte: Byte);
                varWord:     (VWord: Word);
                varLongWord: (VLongWord: LongWord);
                varInt64:    (VInt64: Int64);
                varString:   (VString: Pointer);
                varAny:      (VAny: Pointer);
                varArray:    (VArray: PVarArray);
                varByRef:  (VPointer: Pointer);
              );
          1: (VLongs: array[0..2] of LongInt);
```

```
      );
    2: (VWords: array [0..6] of Word);
    3: (VBytes: array [0..13] of Byte);
end;
```

همانطور که ملاحظه می‌کنید ساختمان TvarData بالغ بر ۱۶ بایت از حافظه را به خود اختصاص می‌دهد. در پیاده‌سازی نوع COM انواع داده‌ای Variant ، ساختمان ذکر شده در بالا بـر روی اشـیاء COM به صورت زیر نگاشت خواهد شد.

```
{ Variant type codes (wtypes.h) }

  varEmpty    = $0000; { vt_empty       }
  varNull     = $0001; { vt_null        }
  varSmallint = $0002; { vt_i2          }
  varInteger  = $0003; { vt_i4          }
  varSingle   = $0004; { vt_r4          }
  varDouble   = $0005; { vt_r8          }
  varCurrency = $0006; { vt_cy          }
  varDate     = $0007; { vt_date        }
  varOleStr   = $0008; { vt_bstr        }
  varDispatch = $0009; { vt_dispatch    }
  varError    = $000A; { vt_error       }
  varBoolean  = $000B; { vt_bool        }
  varVariant  = $000C; { vt_variant     }
  varUnknown  = $000D; { vt_unknown     }
//varDecimal  = $000E; { vt_decimal     } {UNSUPPORTED}
                       { undefined  $0f } {UNSUPPORTED}
  varShortInt = $0010; { vt_i1          }
  varByte     = $0011; { vt_ui1         }
  varWord     = $0012; { vt_ui2         }
  varLongWord = $0013; { vt_ui4         }
  varInt64    = $0014; { vt_i8          }
//varWord64   = $0015; { vt_ui8         } {UNSUPPORTED}

  { if adding new items, update Variants' varLast, BaseTypeMap and OpTypeMap }
  varStrArg   = $0048; { vt_clsid    }
  varString   = $0100; { Pascal string; not OLE compatible }
  varAny      = $0101; { Corba any }
  varTypeMask = $0FFF;
  varArray    = $2000;
  varByRef    = $4000;
```

یادداشت

همانطور که در کد بالا مشاهده می‌کنید، از انواع داده‌ای Variant نمی‌توان برای ارجاع به اشاره‌گرها و کلاس‌ها استفاده نمود. ▲

چرخهٔ حیات انواع داده‌ای Variant

عملیات تخصیص و بازپس‌گیری حافظه به انواع داده‌ای Variant در دلفی، به صورت خودکار انجام می‌پذیرد. به عنوان مثال با اجرای قطعه کد زیر ملاحظه می‌کنید که نوع Variant به یک رشتهٔ کاراکتری ارجاع داده خواهد شد.

```
Procedure ShowVariant (S: String);
Var
V: Variant
begin
V: S;
ShowMessage (V);
end;
```

مقداردهی اولیه انواع داده‌ای Variant در دلفی به صورت مقادیر صحیح بدون علامت در نظر گرفته می‌شود و در طول تخصیص مقادیر به آن، عمل تخصیص حافظه متناظر با نوع اختصاص داده شده صورت می‌پذیرد. عمل تخصیص و بازپس‌گیری حافظه در زیر روال‌هایی که با انواع داده‌ای Variant کار می‌کنند توسط بلوک try ... Finally تعیین می‌گردد. برای درک این مفهوم به قطعه کد زیر توجه کنید.

```
procedure ShowVariant(S: string);
var
  V: Variant
begin
  V := Unassigned;  // initialize variant to "empty"
  try
    V := S;
    ShowMessage(V);
  finally
    // Now clean up the resources associated with the variant
  end;
end;
```

```
procedure ChangeVariant(S: string);
var
  V: Variant
begin
  V := S;
  V := 34;
end;
```

به کارگیری Variant ها در عبارات

دلفی به شما امکان می‌دهد انواع داده‌ای Variant را به همراه عملگرهای + ، − ، * ، / ، div ، mod ، shl ، shr ، and ، or ، xor ، not ، := ، < ، > ، <> ، <= و >= در عبارتهای حسابی، منطقی یا بیتی به کاربرید.

خود دلفی چگونگی انجام عمل بر روی عملوندهای Variant را تشخیص می‌دهد. به طور مثال اگر دو Variant از نوع صحیح با هم جمع زده شوند، حاصل جمع هم در یک مقدار از نوع صحیح درج خواهد شد. به قطعه کد زیر توجه کنید.

```
Var
V1, V2, V3 : Variant;
begin
V1: = '100'; // A String Type
V2: = '50'; // A String Type
V3: = 200; // An Interger Type
V1: = V1 + V2 + V3;
end;
```

در نگاه اول به قطعه کد بالا به نظر می‌رسد که خروجی برابر با عدد 350 خواهد بود. اما با کمی دقت در خواهید یافت که متغیرهای V1 و V2 از نوع رشته‌ای می‌باشند و از آنجا که در عبارات V1: = V1 + V2 + V3 ابتدا بر حسب تقدم عبارت V1 + V2 محاسبه می‌شود لذا مقدار V1 + V2 برابر با '10050' خواهد شد. از آنجا که قرار است این مقدار با یک مقدار عددی به نام V3 جمع شود لذا عملیات تبدیل نوع به صورت خودکار انجام شده و نتیجه برابر با 10050 + 200 = 10250 خواهد شد.

توجه داشته باشید که چنانچه انواع ذکر شده با هم سازگار نباشند استثنایی[1] تحت عنوان invalid variant type conversion به وقوع می‌پیوندد. نمونه‌ای از این استثناء در زیر ارائه شده است.

```
Var
V1, V2: Variant;
begin
V1: = 77;
V2: = 'hello';
V1: = V1/V2 ; // Raises an exception
end;
```

آرایه‌ها و انواع داده‌ای Variant

در پاسکال شی‌گرا امکان تعریف آرایه‌ها از نوع Variant نیز وجود دارد. در پاسکال شی‌گرا این عملیات به وسیلهٔ توابع VarArrayCreate(), VarArrayof() انجام می‌پذیرد.

VarArrayCreate()

این تابع به صورت زیر تعریف شده و از آن برای ایجاد آرایه‌های نوع Variant استفاده می‌شود.

۱- Exception

Function VarArrayCreate (ConstBounds: arrayofInteger; VarType: Integer): Variant;

به عنوان مثال قطعه برنامه زیر یک آرایه از نوع Variant را ایجاد کرده و به هر کدام از عناصر این آرایه یک مقدار عددی را اختصاص می‌دهد.

```
var
  V: Variant;
begin
  V := VarArrayCreate([1, 4], varInteger); // Create a 4-element array
  V[1] := 1;
  V[2] := 2;
  V[3] := 3;
  V[4] := 4;
end;
```

در تابع VarArrayCreage() امکان ایجاد آرایه‌های چند بعدی نیز وجود دارد. به عنوان مثال برای ایجاد یک آرایه با ابعاد [5 ... 1, 4 ... 1] می‌توانید این تابع را به صورت زیر استفاده کنید.

```
V: = VarArrayCreate ([1, 4, 1, 5], VarInteger);
```

VarArrayOf()

از این تابع جهت به کارگیری یک آرایه تک بعدی از نوع Variant استفاده می‌شود. تابع VarArrayof() به صورت زیر استفاده می‌شود.

```
Funtion VarArrayof (Const Values: arrayofvariant): variant;
```

به عنوان مثال با به کارگیری تابع VarArrayof() به صورت زیر یک آرایه با سه عنصر از نوع صحیح، رشته‌ای و ممیز شناور ایجاد می‌شود.

```
V: = VarArrayof ([1, 'Delphi', 2.2]);
```

توابع و روال‌های آرایه‌های نوع Variant

علاوه بر توابع VarArrayCreate() و VarArreyof() ، توابع و روال‌های دیگری نیز در پاسکال شی‌گرا جهت پشتیبانی از آرایه‌های Variant وجود دارد. لیست این توابع و روالها در زیر آورده شده است.

```
procedure VarArrayRedim(var A: Variant; HighBound: Integer);
function VarArrayDimCount(const A: Variant): Integer;
function VarArrayLowBound(const A: Variant; Dim: Integer): Integer;
function VarArrayHighBound(const A: Variant; Dim: Integer): Integer;
function VarArrayLock(const A: Variant): Pointer;
procedure VarArrayUnlock(const A: Variant);
function VarArrayRef(const A: Variant): Variant;
function VarIsArray(const A: Variant): Boolean;
```

از روال VarArrayRedim() جهت تغییر اندازهٔ آرایه‌ها استفاده می‌شود. تابع VarArrayDimcount() وظیفهٔ ارجاع تعداد ابعاد آرایه‌ها را به عهده دارد. از توابع VarArrayLowBound() و VarArrayHighBound() نیز جهت تعیین حد بالا و پایین آرایه‌ها استفاده می‌شود. از تابع VarArray Ref() جهت ارجاع آرایه‌های Variant به سرورهای OLE دلفی استفاده می‌شود. تابع VarIsArray() یک مقدار منطقی (False یا True) متناظر با ارجاع پارامترهای نوع Variant به آرایه‌های نوع Variant را برمی‌گرداند.

سایر توابع و روالهای انواع Variant به شرح زیر می‌باشد.

```
procedure VarClear(var V: Variant);
procedure VarCopy(var Dest: Variant; const Source: Variant);
procedure VarCast(var Dest: Variant; const Source: Variant; VarType: Integer);
function VarType(const V: Variant): Integer;
function VarAsType(const V: Variant; VarType: Integer): Variant;
function VarIsEmpty(const V: Variant): Boolean;
function VarIsNull(const V: Variant): Boolean;
function VarToStr(const V: Variant): string;
function VarFromDateTime(DateTime: TDateTime): Variant;
function VarToDateTime(const V: Variant): TDateTime;
```

OleVariant

نوع Olevariant همانند داده‌های Variant می‌باشد با این تفاوت که از این نوع تنها در اتوماسیون OLE دلفی استفاده می‌شود.

قابلیت‌های ساختارهای داده‌ای

انواع داده‌هایی که تاکنون معرفی نمودیم برای ذخیره‌سازی یک مقدار منفرد به کار برده می‌شوند. ساختارهای داده‌ای معرف گروهی از اقلام داده‌ای مرتبط به هم در حافظه هستند. پاسکال شی‌گرا امکان ایجاد ساختارهای داده‌ای خاص را با نمونه‌برداری از انواع ساده موجود فراهم نموده است. در این قسمت به معرفی ساختارهای داده‌ای در پاسکال شی‌گرا و موارد استفاده آنها خواهیم پرداخت.

آرایه‌ها

آرایه‌ها روشی را ارائه می‌کنند که با استفاده از آن می‌توانید یک نام منفرد را با مجموعه‌ای از داده‌ها مرتبط نمائید. به عنوان مثال یک آرایهٔ ۸ عنصری از اعداد صحیح در زبان پاسکال شی‌گرا به صورت زیر تعریف می‌شود.

```
Var
A: Array [0...7] of Integer;
```

این آرایه در زبان C به صورت Int A[8]; و در زبان بیسیک به صورت Dim A(8) as Integer

تعریف می‌شود. حوزهٔ تعریف آرایه‌ها در پاسکال شی‌گرا با تعریف در سایر زبانها متفاوت است، به طور مثال در پاسکال شی‌گرا خانهٔ حافظه‌ای مربوط به شروع آرایه (اندیس آغازین آرایه) از یک عدد خاص شروع نمی‌شود. بنابراین تعریف آرایه مطابق با کد زیر در پاسکال شی‌گرا صحیح می‌باشد.

```
Var
A: Array [28 ... 30] of Integer;
```

در صورت به کارگیری الگوی فوق می‌بایست از توابع Low() و High() برای تعیین حد پایین و بالای آرایه‌ها در حلقه‌های For استفاده کنید. با کمی دقت در قطعه کد زیر متوجه این مفهوم خواهید شد.

```
Var
A: array [28 ... 30] of Integer;
i: Integer;
begin
for i: Low(A) to High(A) do
A[i]: = i;
end;
```

آرایه‌هایی که در بالا به آنها اشاره شد، آرایه‌های یک بعدی بودند بدین معنی که یک سطر از داده‌ها به صورت چند ستون در آنها قرار گرفته بودند. در پاسکال شی‌گرا نیز همچون اکثر زبان‌های برنامه‌سازی امکان تعریف آرایه‌های چند بعدی وجود دارد. به عنوان مثال یک آرایهٔ دوبعدی در پاسکال شی‌گرا به صورت زیر تعریف می‌گردد.

```
Var
// Two_dimensional array of Integer;
A: array [1 ... 2, 1 ... 2] of Integer;
```

تعریف آرایه‌ها به صورت پویا

هنگامی که تخمین درستی از وضعیت و تعداد خانه‌های مورد لزوم، آرایه‌ها در دست نباشد، لازم است تا آرایه‌ها را به صورت پویا تعریف نمود. تعریف آرایه‌ها به صورت پویا به معنی تخصیص حافظه در زمان اجرای آنها می‌باشد. به نمونه‌ای از این تعریف که در زیر ارائه شده است توجه کنید:

```
Var
// dynamic array of string;
SA: array of string;
```

در صورت نیاز می‌توانید با استفاده از روال SetLength()، طول مشخصی را به آرایه‌تان اختصاص دهید.

```
begin
// allocate room for 33 elements;
setlength (SA, 33);
```

آرایه‌های پویا چرخهٔ حیات مربوط به خود را دارا می‌باشند. بدین معنی که پس از خاتمه کار می‌بایست حافظه تحت اختیار آنها را به روشی آزاد نمود. این کار توسط تخصیص مقدار **nil** به آنها انجام خواهد پذیرفت.

```
SA: nil; // release SA
```

رکوردها

ساختار داده‌ای دیگری که بسیار متداول و در عین حال مفید است، رکورد می‌باشد. همانند آرایه‌ها هر رکورد نیز برای ذخیره کردن گروهی از اطلاعات مرتبط به کار می‌رود، اما برخلاف آرایه‌ها، در رکوردها نیازی به یکسان بودن داده‌ها نمی‌باشد. رکوردها برای ذخیره‌سازی اطلاعات مواردی همچون اشخاص یا ... استفاده می‌شوند. برای مثال می‌توانید نام، نشانی، آدرس و ... هر شخص را در یک رکورد ذخیره و نگهداری نمائید. به تعریف رکوردها در زبانهای پاسکال، C و بیسیک که در زیر به آنها اشاره شده است توجه کنید.

```
{ Pascal }
Type
  MyRec = record
    i: Integer;
    d: Double;
  end;
```

```
/* C */
typedef struct {
  int i;
  double d;
} MyRec;
```

```
'Visual Basic
Type MyRec
  i As Integer
  d As Double
End Type
```

برای دسترسی به عناصر رکوردها از علامت '.' استفاده می‌شود.

```
Var
N: MyRec;
begin
N.i: = 23;
N.d: = 3.4;
end;
```

قابلیت تعریف رکوردهای Variant نیز در پاسکال شی‌گرا وجود دارد. نمونه‌ای از نحوهٔ تعریف

اینگونه از رکوردها را در قطعه کد زیر ملاحظه می‌کنید.

```
type
TVariantRecord = record
NullStrField: Pchar;
IntField: Integer;
Case Integer of
0: (D: Double);
1: (I: Interger);
2: (C: Char);
end;
```

مجموعه‌ها

هر مجموعه گروهی از عناصر است که می‌بایست با تنها یک نام مرتبط باشند و می‌توان وجود یا عدم وجود مقادیر موردنظر در مجموعه را با مقایسه با این عناصر مشخص ساخت. در زیر، نمونه‌هایی از تعریف مجموعه‌ها در پاسکال شی‌گرا ارائه شده است.

```
type
  TCharSet = set of char;       // possible members: #0 - #255

  TEnum = (Monday, Tuesday, Wednesday, Thursday, Friday);
  TEnumSet = set of TEnum;  // can contain any combination of TEnum members

  TSubrangeSet = set of 1..10; // possible members: 1 - 10
  TAlphaSet = set of 'A'..'z'; // possible members: 'A' - 'z'



type
  TIntSet = set of Integer;  // Invalid: too many elements
  TStrSet = set of string;   // Invalid: not an ordinal type
```

تخصیص مقادیر به مجموعه‌ها نیز به سادگی امکان‌پذیر است. برای آشنایی بیشتر با این عملیات به قطعه کد زیر توجه فرمائید.

```
type
  TCharSet = set of char;       // possible members: #0 - #255

  TEnum = (Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday);
  TEnumSet = set of TEnum;  // can contain any combination of TEnum members

var
  CharSet: TCharSet;
  EnumSet: TEnumSet;
  SubrangeSet: set of 1..10; // possible members: 1 - 10
  AlphaSet: set of 'A'..'z'; // possible members: 'A' - 'z'
```

```
begin
  CharSet := ['A'..'J', 'a', 'm'];
  EnumSet := [Saturday, Sunday];
  SubrangeSet := [1, 2, 4..6];
  AlphaSet := [];  // Empty; no elements
end;
```

اشیاء

چنانچه در سالهای اخیر نگاهی به مقاله‌ها و نشریات رایانه‌ای انداخته باشید، متوجه شده‌اید که تا چه حد بر روی موضوع برنامه‌نویسی شیءگرا تأکید شده است. برنامه‌نویسی شیءگرا شیوه‌ای برای سازماندهی منطقی برنامه‌های کاربردی است. به طور کلی باید گفت شیءگرایی یک الگوی طراحی و پیاده‌سازی را به نمایش می‌گذارد.

واژه کلاس اساس و پایه برنامه‌سازی شیءگرا است. هر کلاس متشکل از یکسری داده و رویه‌هاست که داده‌های مربوط به کلاس را اصطلاحاً "فیلد" و رویه‌های مربوط به کلاس را "متد" می‌نامند. پس از تعریف یک کلاس می‌بایست یک نمونه اجرایی از آن را ایجاد کنید. این نمونه اجرایی را "شیء" نام‌گذاری نموده‌اند. الگوی تعریف شیء در پاسکال شیءگرا را در زیر مشاده می‌نمائید.

```
Type
Tchildobject = class (Tparentobject);
SomeVar: Interger;
Procedure SomeProc;
end;
```

ساختار دلفی یک ساختار شیءگرا است. تمام کلاس‌ها در دلفی، کلاس Tobject را به عنوان والد خود دارند. این بدین معنی است که اگر در دلفی والد کلاسی را مشخص نکنند باز هم طبق پیش‌فرض، کلاس موردنظر از کلاس Tobject مشتق خواهد شد. پس از تعریف داده‌های یک شیء، مهمترین بخش ایجاد و طراحی متدهایی است که تعیین کننده رفتار شیء می‌باشند. تعریف متدهای اشیاء در پاسکال شیءگرا برابر با الگوی زیر صورت می‌پذیرد.

```
Procedure TchildObject.SomeProc;
begin
{Procedure code goes here}
end;
```

اشاره‌گرها

اشاره‌گرها در واقع آدرسهایی از حافظه هستند که در آن محل‌ها، مقادیر عناوینی همچون متغیرها، توابع، رویه‌ها، رکوردها و ... نگهداری می‌شوند. اشاره‌گرها همچنین در مواردی که با متغیرهای پویا سروکار

داشته باشیم مورد استفاده قرار می‌گیرند. نمونه‌ای از تعریف اشاره‌گرها را در زیر مشاهده می‌کنید.

```
Type
  PInt = ^Integer;          // PInt is now a pointer to an Integer
  Foo = record              // A record type
    GobbledyGook: string;
    Snarf: Real;
  end;
  PFoo = ^Foo;              // PFoo is a pointer to a foo type
var
  P: Pointer;               // Untyped pointer
  P2: PFoo;                 // Instance of PFoo
```

به خاطر داشته باشید که اشاره‌گرها تنها محلی برای ذخیره‌سازی آدرسهای حافظه در سیستم می‌باشند. عملیات تخصیص حافظه به اشاره‌گرها توسط روال New() انجام می‌پذیرد. برای آشنایی بیشتر شما با نحوهٔ به کارگیری اشاره‌گرها، مثالی را در این قسمت ارائه کرده‌ایم. به نحوهٔ تخصیص و آزادسازی حافظه در این مثال توجه کنید.

```
Program PtrTest;

Type
  MyRec = record
    I: Integer;
    S: string;
    R: Real;
  end;
  PMyRec = ^MyRec;

var
  Rec : PMyRec;
begin
  New(Rec);       // allocate memory for Rec
  Rec^.I := 10;  // Put stuff in Rec. Note the dereference
  Rec^.S := 'And now for something completely different.';
  Rec^.R := 6.384;
  { Rec is now full }
  Dispose(Rec);  // Don't forget to free memory!
end.
```

■ نکته

چنانچه اشاره‌گر تعریف شده به هیچ محلی از حافظه اشاره ننماید، مقدار آن برابر صفر خواهد بود که اصطلاحاً آن را nil یا null می‌نامند.

گزاره‌های شرطی

گزاره‌های شرطی امکان کنترل شرطی اجرای برنامه‌ها را فراهم می‌آورند. یعنی براساس نتیجهٔ یک شرط،

بخش خاصی از کد برنامه اجرا خواهد گردید.

عبارت شرطی if

عبارت if نیز به عنوان یکی از گزاره‌های شرطی، امکان کنترل برنامه برحسب نتایج شرطها را فراهم می‌آورد.

```
if X = 4 then Y: = X;
```

نکته

چنانکه قرار باشد از عبارت شرطی if برای پیاده‌سازی شرطهای چندگانه (ترکیبی) استفاده کنید لازم است تا از الگوی فرضی مشابه با الگوی زیر پیروی نمائید.

```
if (X = 7) and (Y = 8) then
```

توجه داشته باشید که به کار بردن الگوی زیر ممکن است کامپایلر را به خطا بیاندازد.

```
if X = 7 and Y = 8 then
```

بخش دوم شرطها در عبارت if به وسیلهٔ عبارت else قابل پیاده‌سازی می‌باشند. ساختار if–else به صورت ترکیبی هم به کار برده می‌شود. نمونه‌ای از این ساختار را در زیر مشاهده می‌کنید.

```
if X = 100 then
someFunction
else if X = 200 them
else begin
something else;
Entirely;
end;
```

عبارت Case

اگرچه عبارت if مفید به نظر می‌رسد، اما در مواردی که با شرایط چند سطحی سروکار داریم و ناچار به استفاده از چندین عبارت else در برنامه می‌شویم، برنامه خوانایی خود را از دست می‌دهد، ضمن این که ویرایش‌های بعدی برنامه را نیز دشوار خواهد نمود.

در این گونه موارد، عبارت Case جانشین خوبی برای عبارت if است. الگوی استفاده از عبارت Case در زیر ارائه گردیده است.

```
Case SomeInteger Variable of
101: DoSomthing;
202: begin
DoSomething;
```

```
end;
303: Do Anotherthings;
else Do The Defauit;
end;
```

حلقه‌ها

روش دیگری که برای کنترل اجرایی برنامه در پاسکال شی‌ءگرا وجود دارد، استفاده از حلقه‌هاست. حلقه‌ها برای تکرار بلوکی از برنامه‌ها تا زمان برقراری شرط یا شرایط ویژه به کار برده می‌شوند. در پاسکال شی‌ءگرا سه نوع حلقه وجود دارد که در ذیل به آنها اشاره می‌کنیم.

حلقهٔ while

حلقهٔ while یک یا تعدادی از گزاره‌ها را تا زمان وقوع یک شرط خاص تکرار می‌کند. برخلاف حلقهٔ For که تعداد دفعات تکرار مشخص می‌باشد، حلقهٔ while در زمانی که تعداد دفعات تکرار نامعلوم است به کار می‌رود. به عنوان مثال، حلقهٔ while را می‌توان برای خواندن محتویات یک فایل تا زمان رسیدن به انتهای آن به کار برد. به مثالی که در همین رابطه ارائه شده است توجه کنید.

```
Program FileIt;

{$APPTYPE CONSOLE}

var
  f: TextFile;  // a text file
  s: string;
begin
  AssignFile(f, 'foo.txt');
  Reset(f);
  while not EOF(f) do begin
    readln(f, S);
    writeln(S);
  end;
  CloseFile(f);
end.
```

حلقهٔ Repeat..until

حلقهٔ Repeat ... Until تقریباً شبیه حلقهٔ while است. اما از آنجا که در این حلقه، پس از اجرای بدنهٔ حلقه، شرط موجود در آن بررسی می‌شود، لذا حداقل یکبار حلقه اجرا خواهد شد. مثال زیر چگونگی به کارگیری این حلقه را نمایش می‌دهد.

```
Var
X: Integer;
begin
X: 1;
```

```
repeat
inc(X);
until X>100;
end.
```

روال Break()

Break() روالی است که از آن برای خروج فوری از حلقه‌های for ، while و repeat ... until استفاده می‌شود. این روال کنترل اجرای برنامه را به انتهای حلقه منتقل می‌سازد. به مثال زیر توجه کنید. در این مثال، حلقهٔ for تنها برای ۵ بار متوالی طی خواهد شد.

```
var
  i: Integer;
begin
  for i := 1 to 1000000 do
  begin
    MessageBeep(0);          // make the computer beep
    if i = 5 then Break;
  end;
end;
```

روال Continue()

از روال Continue() نیز برای انتقال کنترل برنامه در داخل حلقه‌ها استفاده می‌شود. در مثال زیر، برای گام اول در حلقه For یک گریز ایجاد شده است.

```
var
  i: Integer;
begin
  for i := 1 to 3 do
  begin
    writeln(i, '. Before continue');
    if i = 1 then Continue;
    writeln(i, '. After continue');
  end;
end;
```

روال‌ها و توابع

در هنگام برنامه‌نویسی ممکن است زیر برنامه‌ای بارها و بارها تکرار شود. به عنوان مثال در یک برنامهٔ انبارداری ممکن است چندین بار کد مربوط به عملیات ورود کالا به انبار یا ثبت کالا در جاهای مختلف برنامه به کار رفته باشد. یعنی در یک برنامه چندین بار از یک کد خاص استفاده شود. در این صورت بهتر است به جای تکرار مجموعه‌ای عبارات از یک روال یا تابع استفاده گردد.

برنامهٔ ارائه شده در لیست ۱-۲ نمونه‌ای از نحوهٔ تعریف و به کارگیری توابع و روال‌ها را نمایش می‌دهد.

لیست ۱-۲ مثالی از تعریف و به کارگیری روال‌ها و توابع در پاسکال شی‌ءگرا

```
Program FuncProc;

{$APPTYPE CONSOLE}

procedure BiggerThanTen(i: Integer);
{ writes something to the screen if I is greater than 10 }
begin
  if I > 10 then
    writeln('Funky.');
end;

function IsPositive(I: Integer): Boolean;
{ Returns True if I is 0 or positive, False if I is negative }
begin
  if I < 0 then
    Result := False
  else
    Result := True;
end;
var
  Num: Integer;
begin
  Num := 23;
  BiggerThanTen(Num);
  if IsPositive(Num) then
    writeln(Num, 'Is positive.')
  else
    writeln(Num, 'Is negative.');
end.
```

ارسال پارامترها

شیوهٔ ارسال پارامترها، به عنوان مهمترین نکته روال‌ها مطرح می‌باشند. ارسال با مقدار، ارسال به وسیله ارجاع و ارسال به صورت ثابت به عنوان شیوه‌های ارسال پارامترها در پاسکال شی‌ءگرا به کار گرفته می‌شوند.

پارامترهای مقداری

یک پارامتر مقداری در واقع یک کپی از پارامتر اصلی ارسال شده از روال فراخوانی است. از آنجائی که مقدار پارامتر تنها یک رونوشت از پارامتر اصلی می‌باشد، اگر در درون زیر روال مقدار جدیدی به پارامتر نسبت داده شوند، پارامتر اصلی همچنان مقدار اولیهٔ خود را حفظ می‌کند. نمونه‌ای از فراخوانی روال‌ها با پارامترهای مقداری را در زیر مشاهده می‌کنید.

```
Procedure Foo (S: String);
```

پارامترهای ارجاعی

چنانچه قصد دارید پارامتری را به یک زیر روال ارسال داشته و مقدار پس از فراخوانی آن تغییر پیدا کند لازم است تا پارامتر را به صورت ارجاعی تعریف نمائید. نام دیگر پارامترهای ارجاعی در پاسکال شیءگرا پارامترهای متغیر است. یک پارامتر متغیر به نشانی حافظه اصلی محل نگهداری پارامتر اصلی مراجعه می‌کند و از این رو هر تغییری که در درون زیر برنامه بر روی پارامتر صورت بگیرد، در روال فرا خوانده نیز منعکس خواهد شد.

برای تعریف پارامترها از این نوع از کلمهٔ کلیدی Var استفاده می‌شود. به مثال زیر توجه کنید.

```
Procedure changeMe (Var x: Longint);
begin
x: = 2; {x is now changed in the calling procedure}
end;
```

پارامترهای ثابت

با قرار دادن کلمهٔ کلیدی Const هنگام ارسال پارامترها به روال‌ها، می‌توانید مطمئن شوید که پارامترها به صورت اتفاقی مقدار نمی‌گیرند.

```
Procedure Goon (Const S: String);
```

یونیت‌ها

مهمترین فایلی که به طور مرتب در دلفی به کار خواهید برد، فایل متن برنامه است که یونیت نامیده می‌شود. در دلفی برای هر فرم یک یونیت وجود دارد. بخش Interface ، بخش Implementation ، Initialization و Finalization به عنوان بخش‌های یک یونیت در دلفی مطرح می‌باشند. بخش Interface هر یونیت شامل تعاریف داده‌ها، متغیرها و ... است. توجه داشته باشید که بخش Interface فاقد دستورات اجرایی است و تنها حاوی تعریف داده‌ها، متغیرها و ثابتهایی است که برای استفاده یونیت‌های دیگر مورد لزوم خواهند بود.

بخش Implementation بخشی است که حاوی دستورات اجرایی برنامه است. تعریف متغیرها، ثابت‌ها و ... نیز در بخش Implementation امکان‌پذیر است. توجه داشته باشید که انواع داده‌هایی که در بخش Interface تعریف می‌شوند در خارج از یونیت قابل استفاده بود، اما تعریف داده‌ها در بخش Implementation باعث می‌گردد که داده‌ها مربوط تنها در درون یونیت مورد استفاده باشند.

دستورات موجود در بخش Initialization هر یونیت، قبل از دیگر دستورات یونیت اجرا می‌شوند. از بخش Finalization ممکن است به همراه بخش Initialization استفاده شود. به طور مثال از این بخش برای اجرای دستورات پایانی همچون آزادسازی حافظه استفاده خواهد شد. در نسخهٔ دلفی 1.0 تابعی به نام ()AddExitProc وظیفه بخش Finalization را به عهده داشته است و این بخش از نسخهٔ

۲ دلفی به بعد به عنوان بخشی از یک یونیت پیاده‌سازی گردیده است.

عبارت Uses

Uses با فرمان دادن به کامپایلر اجازه می‌دهد تا دستورات موجود در یونیت‌های فهرست شده را به یونیت‌های جاری بیفزاید. استفاده از عبارت Uses در هر دو بخش Interface و Implementation مجاز می‌باشد. با آنچه که ذکر شده، هر یونیت می‌تواند الگویی مشابه با الگوی ارائه شده ذیل داشته باشد.

```
Unit FooBar;

interface
uses BarFoo;

  { public declarations here }

implementation

uses BarFly;

  { private declarations here }

initialization
  { unit initialization here }
finalization
  { unit clean-up here }
end.
```

Package

اگر مایلید قسمتی از برنامهٔ خود را بین سایر برنامه‌های کاربردی به اشتراک بگذارید و یا از امکانات آن در سایر برنامه‌ها، ماژول‌ها و ... استفاده کنید، لازم است تا آن را به عنوان Package معرفی نمائید. Package ها در دلفی در حقیقت مجموعی از چندین یونیت هستند که در یک فایل مشابه با فایل‌های DLL (فایل BPL[1]) قرار می‌گیرند. فایل‌های BPL می‌توانند تحت عناوین فایل‌های DLL و EXE نیز کامپایل شوند. الگوی تعریف Package در دلفی به صورت زیر است.

```
Package Packagename
requires Package1, Package2, ...;
Contains
unit1 in 'unit1.pas',
unit2 in 'unit2.pas',
...;
end.
```

۱- Borland Package Library

سازنده‌ها

در دلفی عمل تخصیص حافظه به اشیاء توسط سازنده‌ها انجام می‌پذیرد. این تخصیص حافظه به مفهوم ایجاد نمونه‌ای از شیء موردنظر جهت امور اجرایی برنامه است. مثالی از به کارگیری تابع سازندهٔ Create() را در زیر مشاهده می‌کنید.

```
FooObject. = TFooObject.Create;
```

مخرب‌ها

بازپس‌گیری حافظه تخصیص یافته به اشیاء در دلفی توسط مخرب‌ها صورت می‌پذیرد.

```
FooObject.FREE;
```

متدها

همانطور که گفته شد، متدها در حقیقت توابع و روال‌هایی هستند که رفتار شیء را در محیط برنامه مشخص می‌سازند. هر متد به عنوان جزئی از یک شیء تعریف می‌شود، ضمن این که هر متد دارای کد و محتویات مربوط به خود است به عنوان مثال به متد DoThHustle که برای شیء TBoojieNights تعریف شده است توجه کنید.

```
type
  TBoogieNights = class
    Dance: Boolean;
    procedure DoTheHustle;
  end;

procedure TBoogieNights.DoTheHustle;
begin
  Dance := True;
end;
```

انواع متدها

متدها در دلفی تحت ۴ عنوان ایستا، مجازی، پویا و پیام تعریف می‌شوند.

```
TFoo = class
  procedure IAmAStatic;
  procedure IAmAVirtual; virtual;
  procedure IAmADynamic; dynamic;
  procedure IAmAMessage(var M: TMessage); message wm_SomeMessage;
end;
```

متدهای ایستا

نوع پیش‌فرض متدها در دلفی، نوع ایستا است. به طور کلی متدهای ایستا متدهایی هستند که به عنوان متد پویا یا مجازی تعریف نمی‌شوند.

متدهای مجازی

متدهای مجازی به یک Vritual Method Table (VMT) افزوده شده و به هنگام تعیین رویه‌هایی که باید فراخوانده شوند، از آنها استفاده می‌شود. یک VMT در هر کلاس برای یافتن یک نگارش خاص از یک روال مجازی نگهداری می‌شود.

متدهای پویا

متدهای پویا به یک Dynamic Method Table (DMT) اضافه می‌شود. DMT نیز در هر کلاس به طور جزئی نگهداری شده و به تمام DMT های موجود در کلاس والد خود دسترسی می‌یابد.

متدهای پیام

از متدهای پیام، برای پاسخ‌دهی خودکار به پیام‌های سیستم عامل (ویندوز) در برنامه استفاده می‌شود.

Tobject والد تمامی اشیاء است

Tobject کلاس ریشهٔ اصلی برای تمام زیر کلاس‌های دلفی است، یعنی تمام کلاس‌ها در دلفی از Tobject مشتق می‌شوند. این کلاس در یونیت system وجود داشته و به صورت زیر تعریف شده است.

```
type
  TObject = class
    constructor Create;
    procedure Free;
    class function InitInstance(Instance: Pointer): TObject;
    procedure CleanupInstance;
    function ClassType: TClass;
    class function ClassName: ShortString;
    class function ClassNameIs(const Name: string): Boolean;
    class function ClassParent: TClass;
    class function ClassInfo: Pointer;
    class function InstanceSize: Longint;
    class function InheritsFrom(AClass: TClass): Boolean;
    class function MethodAddress(const Name: ShortString): Pointer;
    class function MethodName(Address: Pointer): ShortString;
    function FieldAddress(const Name: ShortString): Pointer;
    function GetInterface(const IID: TGUID; out Obj): Boolean;
    class function GetInterfaceEntry(const IID: TGUID): PInterfaceEntry;
    class function GetInterfaceTable: PInterfaceTable;
    function SafeCallException(ExceptObject: TObject;
      ExceptAddr: Pointer): HResult; virtual;
    procedure AfterConstruction; virtual;
    procedure BeforeDestruction; virtual;
    procedure Dispatch(var Message); virtual;
    procedure DefaultHandler(var Message); virtual;
    class function NewInstance: TObject; virtual;
```

```
  procedure FreeInstance; virtual;
  destructor Destroy; virtual;
end;
```

مدیریت ساخت یافته در برخورد با استثناها

استثناها در حقیقت روتین‌هایی هستند که اشکالات موجود در رفتار برنامه‌ها را برطرف می‌کنند. در دلفی، روش‌های برخورد با استثناها به صورت زیر کلاس‌هایی از کلاس Exception پیاده‌سازی می‌شوند. بلوک try..finally و try..except روش برخورد با استثناها و امکان ادامهٔ برنامه را فراهم می‌سازند.

در لیست ۳-۲ یک برنامهٔ نمونه را که برای برخورد با استثناهای ورودی/ خروجی طراحی شده است، مشاهده می‌نمائید. این برنامه با استفاده از بلوک try..finally..except پیاده‌سازی گردیده است.

لیست ۳-۲ ورودی/ خروجی فایل و به کارگیری استثناها

```
Program FileIO;

uses Classes, Dialogs;

{$APPTYPE CONSOLE}

var
  F: TextFile;
  S: string;
begin
  AssignFile(F, 'FOO.TXT');
  try
    Reset(F);
    try
      ReadLn(F, S);
    finally
      CloseFile(F);
    end;
  except
    on EInOutError do
      ShowMessage('Error Accessing File!');
  end;
end.
```

توجه داشته باشید که دستورات لازم برای حل مسأله در بخش try دستوراتی که به عدم توانایی حل مسأله مربوط می‌شوند در قسمت except پیاده‌سازی خواهند شد. عبارت on Excpetion do باعث برخورد با استثناء خاصی می‌شود که توسط Exception مشخص گردیده است (به لیست ۳-۲ مراجعه کنید).

برای آشنایی بیشتر با استثناها و روشهای مدیریت آنها به برنامهٔ ارائه شده در لیست ۴-۲ مراجعه نمائید.

لیست ۴-۲ بلوک try..Except

```
Program HandleIt;

{$APPTYPE CONSOLE}

var
  R1, R2: Double;
begin
  while True do begin
  try
    Write('Enter a real number: ');
    ReadLn(R1);
    Write('Enter another real number: ');
    ReadLn(R2);
    Writeln('I will now divide the first number by the second...');
    Writeln('The answer is: ', (R1 / R2):5:2);
  except
    On EZeroDivide do
      Writeln('You cannot divide by zero!');
    On EInOutError do
      Writeln('That is not a valid number!');
  end;
  end;
end.
```

الگوی کلی تعریف استثناها با استفاده از بلوک try..Except به صورت زیر می‌باشد.

```
try
statements
except
on EsomExecption do something;
{do some default exception handling}
end;
```

کلاس Exception

کلاس Excepetion والد تمامی کلاس‌های مربوط به استثناهاست، به این مفهوم که تمامی کلاس‌های مرتبط با استثناها و روشهای برخورد با آنها از کلاس Exception مشتق می‌شوند. تعریف این کلاس در دلفی به صورت زیر می‌باشد.

```
type
  Exception = class(TObject)
  private
    FMessage: string;
    FHelpContext: Integer;
  public
```

```
  constructor Create(const Msg: string);
  constructor CreateFmt(const Msg: string; const Args: array of const);
  constructor CreateRes(Ident: Integer); overload;
  constructor CreateRes(ResStringRec: PResStringRec); overload;
  constructor CreateResFmt(Ident: Integer; const Args: array of const);
    overload;
  constructor CreateResFmt(ResStringRec: PResStringRec;
    const Args: array of const); overload;
  constructor CreateHelp(const Msg: string; AHelpContext: Integer);
  constructor CreateFmtHelp(const Msg: string; const Args: array of const;
    AHelpContext: Integer);
  constructor CreateResHelp(Ident: Integer; AHelpContext: Integer); overload;
  constructor CreateResHelp(ResStringRec: PResStringRec;
    AHelpContext: Integer); overload;
  constructor CreateResFmtHelp(ResStringRec: PResStringRec;
    const Args: array of const;
    AHelpContext: Integer); overload;
  constructor CreateResFmtHelp(Ident: Integer; const Args: array of const;
    AHelpContext: Integer); overload;
  property HelpContext: Integer read FHelpContext write FHelpContext;
  property Message: string read FMessage write FMessage;
end;
```

خلاصه

در این فصل با مفاهیم اساسی پاسکال شی‌گرا همچنین اشیاء سازنده‌ها و مخرب‌ها، متدها، استثناها و ... آشنا شدید. آموختن این فصل در به کارگیری تکنیک‌های پیشرفته ارائه شده در فصول آتی کتاب مؤثر و نقش ضروری دارد. مثال‌های این فصل به گونه‌ای طراحی شده بودند که ضمن آشنایی با نحوهٔ به کارگیری عبارت‌ها در دلفی قادر باشید از هم اکنون بر روی کدهای برنامه‌هایتان تمرکز نمائید.

فصل سوم

کتابخانه اجزاء سازنده دلفی ۷

در این فصل می‌خوانید

- **آشنایی با کتابخانه اجزاء سازندهٔ موجود در دلفی ۷**
- **اجزاء سازندهٔ Visual و NonVisual**
- **بکارگیری اجزاء سازنده در پروژه‌های نرم‌افزاری**

در این فصل با ابزارهای موجود در کتابخانهٔ اجزاء سازندهٔ Visual[۱] دلفی ۷ آشنا خواهید شد. دلفی ۷ دربرگیرندهٔ کتابخانهٔ کاملی از اجزاء سازندهٔ Visual است. این کتابخانه متشکل از شی‌های از پیش آماده شده‌ای است که ویژگیهای مورد استفادهٔ متداول در برنامه‌های کاربردی ویندوز را برایتان فراهم می‌سازند. اجزاء سازندهٔ موجود در دلفی ۷ با استفاده از پاسکال شیءگرا نوشته شده‌اند و با فراهم کردن قابلیت طراحی Visual برنامه‌های کاربردیتان، شما را در کاهش زمان پیاده‌سازی و آزمایش برنامه‌ها یاری می‌دهند. در انتهای این فصل تفاوت بین اجزاء سازندهٔ Visual و NonVisual شرح داده شده است.

اجزاء سازندهٔ Visual موجود در دلفی خواص خاص خود را دارند. این خواص به شما امکان می‌دهند که بسیاری از موارد مربوط به شکل، اندازه، رنگ، موقعیت و ... اجزاء را تغییر دهید. برای دستیابی به این خواص از Object Inspector دلفی استفاده می‌شود. علاوه بر این، اجزاء سازندهٔ دلفی تعدادی رویداد[۲] یا روتین پاسخگویی به رویداد مرتبط با خود دارند. همانطور که می‌دانید دستورات موجود در این روتین‌ها، هنگامی اجرا خواهند شد که رویداد ویژه‌ای نظیر فشردن دکمهٔ ماوس یا کلید Enter رُخ دهد. توجه داشته باشید که برای تغییر خواص اجزاء سازنده از برگهٔ[۳] Properties و برای تعریف رویدادها از برگه Event در Object Inspector استفاده می‌نمائیم.

۱- Visual Component Library ۲- Event ۳- Tab

کتابخانهٔ اجزاء سازندهٔ دلفی ۷

کتابخانهٔ اجزاء سازندهٔ دلفی ۷ به ۲۸ برگه تقسیم‌بندی می‌شود. برگه‌های موجود به شرح زیر می‌باشند:

- Standard
- Additional
- Common Controls (win 32)
- System
- Data Access
- Data Controls
- dbExpress
- Datasnap
- BDE
- ADO
- InterBase
- InterBase Admin
- Web Services
- Internet Express
- Internet
- websnap
- Decision cube
- Dialogs
- win 3.1
- Samples
- ActiveX
- Com+
- Indy Clients
- Indy Servers
- Indy Misc
- Indy intercepts
- Indy I/O Handlers
- Servers

جزئیات مربوط به تمامی اجزاء سازندهٔ دلفی ۷ در این کتاب آورده نشده است. راهنمای online دلفی بزرگترین کمک برای تعیین کاربرد هر یک از اجزاء سازنده خواهد بود. برای این کار کافی است نشان ماوس را بر روی جزء سازنده برده و کلید F1 را بفشارید تا صفحات راهنما، اطلاعاتی دربارهٔ هر یک از اجزاء سازنده و شیوهٔ استفاده از آنها در اختیارتان قرار دهند.

اکنون تمامی برگه‌های موجود در کتابخانهٔ اجزاء سازندهٔ دلفی ۷ را بررسی خواهیم نمود.

برگهٔ Standard

اجزاء سازندهٔ مورد استفادهٔ متداول برنامه‌های کاربردی را در این برگه خواهید یافت. برگه Standard، برگه پیش‌فرض آغازین دلفی است. این برگه شامل ۱۶ جزء سازنده است که در جدول ۱-۳ شرح داده شده‌اند.

جدول ۱-۳ اجزاء سازندهٔ موجود در برگه Standard

جزء سازنده	توضیح
Frames	امکان باز نمودن فریم‌های موجود در برنامهٔ کاربردی را فراهم می‌سازد. این کار با نمایش لیستی از فریم‌های موجود و امکان انتخاب فریم مورد نظر انجام می‌پذیرد.
MainMenu	امکان ایجاد و طراحی منوی اصلی و منوهای کشویی فرم رافراهم می‌آورد.
PopupMenu	امکان ایجاد و طراحی منوهای باز شو را فراهم می‌کند. این منوها با فشردن دکمهٔ سمت راست ماوس ظاهر می‌شوند.
Label	امکان درج متن در فرم‌ها و سایر اجزاء دربرگیرندهٔ اجزاء دیگر را فراهم می‌سازد.

جدول ۱-۳ ادامه

جزء سازنده	توضیح
Edit	برای دریافت متن یک خطی از کاربر مورد استفاده قرار می‌گیرد.
Memo	برای درج یا نمایش متن‌های چند خطی مورد استفاده قرار می‌گیرد.
Button	امکان ایجاد دکمه‌هایی را فراهم می‌آورد که کاربر برای انتخاب گزینه‌ها از آن استفاده می‌نماید.
CheckBox	برای فراهم نمودن امکان انتخاب گزینه‌ها یا خارج نمودن آنها از حالت انتخاب مورد استفاده قرار می‌گیرد.
RadioButton	برای انتخاب مجموعه‌ای از گزینه‌ها مورد استفاده قرار می‌گیرد. توجه داشته باشید که در این حالت تنها یک گزینه قابل انتخاب خواهد بود.
ListBox	امکان ایجاد فهرست‌های قابل انتخاب را فراهم می‌آورد.
ComboBox	این جزء سازنده همانند ListBox بوده با این تفاوت که امکان ویرایش را نیز فراهم می‌آورد.
ScrollBar	برای پیمایش فرمها یا شیءهای کنترلی مورد استفاده قرار می‌گیرد.
GroupBox	این جزء سازنده یکی از اجزاء دربرگیرندهٔ اجزاء دیگر است که برای گروه‌بندی شیءهای کنترلی مرتبط به هم (مانند تعدادی CheckBox) مورد استفاده قرار می‌گیرد.
RadioGruop	این جزء سازنده ترکیبی از GroupBox و RadioButton ها می‌باشد و تنها برای گروه‌بندی اجزاء سازندهٔ RadioButton کاربرد دارد.
Panel	این جزء سازنده، یکی دیگر از اجزاء دربرگیرنده، اجزاء دیگر است که برای گروه‌بندی شیءهای کنترلی مورد استفاده قرار می‌گیرد.
ActionList	این جزء سازنده فهرستی از اجزاء سازندهٔ Action را ذخیره می‌کند. ActionList ویراستاری برای زمان طراحی دارد که برای مدیریت Action ها مورد استفاده قرار می‌گیرد. این جزء سازنده در زمان ساخت و به طور پویا قابل ایجاد می‌باشد.

برگهٔ Additional

برگهٔ Additional در برگیرندهٔ ۲۸ جزء سازنده می‌باشد. فهرست این اجزاء در جدول ۲-۳ ارائه شده است.

جدول ۲-۳ اجزاء سازندهٔ موجود در برگه Additional

جزء سازنده	توضیح
BitBtn	برای ایجاد یک دکمه با یک تصویر (از نوع نقش بیتی) بر روی آن مورد استفاده قرار می‌گیرد. (به عنوان مثال یک دکمهٔ cancel با علامت "X").
SpeedButton	برای ایجاد جعبه ابزارها یا دکمه‌های ویژه‌ای که به صورت فشرده باقی می‌مانند استفاده می‌شوند. بکارگیری این دکمه‌ها، زیبایی خاصی را به برنامه‌های کاربردیتان می‌افزاید.

جدول ۲-۳ ادامه

جزء سازنده	توضیح
MaskEdit	امکان قالب‌بندی صحیح داده‌ها را فراهم می‌آورد.
StringGrid	امکان نمایش جدولی داده‌ها را فراهم می‌آورد. با استفاده از این جزء سازنده قادر خواهید بود داده‌های موجود در برنامه‌هایتان را در یک جدول نمایش دهید.
DrawGrid	امکان نمایش جدولی داده‌های غیرمتنی (نظیر تصاویر) را فراهم می‌سازد.
Image	برای نمایش تصاویر گرافیکی، نمادهای گرافیکی، نقش‌های بیتی مورد استفاده قرار می‌گیرد.
Shape	برای رسم شکلهای هندسی همچون مربع، مستطیل، دایره و ... به‌کار می‌رود.
Bevel	برای رسم مستطیلی به‌کار گرفته می‌شود که می‌تواند به صورت تورفته یا بیرون آمده باشد.
ScrollBox	این جزء سازنده، ویژگی‌های یک کادر فهرست و یک کادر انتخاب را با هم ارائه می‌دهد.
CheckListBox	این جزء سازنده همانند ScrollBox ویژگی‌های یک کادر فهرست و یک کادر انتخاب را با هم ارائه می‌دهد. با این تفاوت که در آن از CheckBox استفاده شده است.
Splitter	امکان ایجاد نواحی با قابلیت تغییر اندازه را فراهم می‌آورد.
StaticText	یک متن غیرقابل ویرایش بوده و موارد استفاده‌ای نظیر Label ها دارد. این جزء سازنده از ویژگی‌های خاصی نسبت به Label ها برخوردار است.
ControlBar	از این جزء سازنده برای درج اجزاء در باندهای دارای قابلیت تغییر اندازه (مانند Toolbar) استفاده می‌شود.
ApplicationEvents	این جزء سازنده در مدیریت رویدادهای یک برنامه کاربرد دارد. ApplicationEvants ایجاد روتین پاسخگویی به رویدادهای برنامه‌های کاربردی را سهولت می‌بخشد.
ValueListEditor	امکان نمایش لیستهای دو ستونی حاوی نام و مقدار داده‌ها را فراهم می‌آورد. با استفاده از این جزء سازنده می‌توانید داده‌ها را به گونه‌ای ذخیره کنید که به صورت زوج‌هایی متشکل از نام و مقدار قابل ارائه باشند.
LabeledEdit	ترکیبی از اجزاء سازندهٔ Label و Edit است.
ColorBox	امکان انتخاب رنگ توسط کاربر را فراهم می‌سازد. این کار با نمایش یک ComboBox از رنگ‌های موجود در سیستم انجام می‌گیرد.
Chart	امکانات کنترلی بر روی داده‌های گرافیکی را فراهم می‌سازد.
ActionManager	امکان مدیریت و کنترل Action های برنامه را فراهم می‌آورد. این شیء دارای یک ویراستار برای مدیریت منوها و میله‌های ابزار (Toolbar) می‌باشد.
ActionMainMenuBar	این جزء سازنده، منویی است که خود وظیفهٔ نمایش سایر منوها و زیرمنوها را به عهده دارد.
ActionToolBar	از این جزء سازنده برای نمایش میلهٔ ابزاری استفاده می‌شود که در برگیرندهٔ

جدول ۲-۳ ادامه

جزء سازنده	توضیح
	Action های برنامه است.
CustomizeDlg	این جزء سازنده در برگیرندهٔ یک کادر محاوره‌ای است که در مدیریت و کنترل میله‌های ابزار و منوها کاربرد دارد.
XpColorMap	استفاده از XpColorMap را در طراحی منوها امکان‌پذیر می‌سازد.
TwilightColorMap	استفاده از TwilightColorMap را در طراحی منوها امکان‌پذیر می‌سازد.
LCDNumber	این جزء سازنده به صورت یک کادر ویرایشی برای متون و اعداد بوده با این ویژگی که در حالت digital (همانند صفحات LCD) نمایش داده می‌شود.
Timer	برای فعال کردن رویه‌ها، توابع و رویدادها در فواصل زمانی معین مورد استفاده قرار می‌گیرد.
PaintBox	برای ایجاد ناحیهٔ قابل رنگ‌آمیزی در فرم مورد استفاده قرار می‌گیرد.

برگهٔ Common Controls (win 32)

برگهٔ win 32 شامل ۲۵ جزء سازنده به شرح جدول ۳-۳ می‌باشد.

جدول ۳-۳ اجزاء سازندهٔ موجود در برگهٔ win 32

جزء سازنده	توضیح
TabControl	یکی از اجزایی است که به شما امکان می‌دهد برگه‌هایی را برای انتخاب کاربر به یک فرم اضافه کنید.
PageControl	از این جزء سازنده برای ایجاد صفحه‌هایی استفاده می‌شود که بهینه‌سازی فضای Desktop کاربرد دارند.
ImageList	امکان کار با فهرست تصاویر را فراهم می‌سازد.
RichEdit	یک کادر ویرایشی است که امکان استفاده از چند رنگ، فونت و ... را فراهم می‌آورد.
TrackBar	به عنوان یک تنظیم کننده عمل می‌نماید. برای توضیحات بیشتر به مستندات دلفی ۷ مراجعه نمائید. عموماً برای تنظیم مقادیری چون بلندی صدا، تیرگی و روشنی تصویر و ... مفید می‌باشد.
ProgressBar	این جزء سازنده، نمایانگر یک خط راهنمای پیشرفت کار می‌باشد.
UpDown	یک دکمهٔ شماره انداز به معنی افزایش یا کاهش تک واحدی مقادیر عددی می‌باشد.
HotKey	امکان پشتیبانی از کلیدهای فوری را فراهم می‌آورد.
Animate	برای پخش انیمیشن، فایل‌های AVI و MOV مورد استفاده قرار می‌گیرد.
DataTimePicker	امکان انتخاب داده‌های موجود بر روی یک تقویم را فراهم می‌آورد. (مانند روز، ساعت، سال و ...). در این جزء سازنده قابلیت ویرایشی نیز وجود دارد.
MonthCalendar	برای نمایش یک تقویم به صورت ماهانه (ماه شمار) مورد استفاده قرار می‌گیرد.
TreeView	جزء سازنده‌ای است که داده‌ها را به شکل سلسله مراتبی نمایش می‌دهد.

جدول ۳-۳ ادامه

جزء سازنده	توضیح
ListView	جزء سازنده‌ای که فهرست موجود را در یک یا چند ستون نمایش می‌دهد.
HeaderControl	امکان ایجاد چند عنوان (header) قابل جابجایی را انجام می‌دهد.
Aminate(CLX)	به توضیحات ارائه شده برای Animate مراجعه نمائید.
StatusBar	امکان نمایش اطلاعات وضعیت جاری برنامه (یا سیستم) در چندین قالب مختلف را فراهم می‌سازد.
ToolBar	برای ایجاد جعبه ابزارهایی استفاده می‌شود که در دستیابی به ویژگی‌های متداول برنامه کاربرد دارند.
CoolBar	برای قرار دادن اجزاء، در میله‌های ابزار مورد استفاده قرار می‌گیرد.
PageScroller	امکان پیمایشی یک صفحه را فراهم می‌آورد.
ComboBoxEx	برای نمایش رشته‌های کاراکتری مرتبط با تصاویر مورد استفاده قرار می‌گیرد.
TextViewer	برای نمایش فایل‌های متن یا صفحات HTML[1] مورد استفاده قرار می‌گیرد.
TextBrowser	برای نمایش فایل‌های متن یا صفحات HTML مورد استفاده قرار می‌گیرد. امکان پشتیبانی از Link های موجود بر روی صفحات HTML نیز در این جزء سازنده گنجانیده شده است.
SpinEdit	امکان درج یا ویرایش خط به خط یک فایل متنی را فراهم می‌آورد.
IconView	برای نمایش گرافیکی (icon) عناوین، داده‌ها، فهرستها و ... مورد استفاده قرار می‌گیرد.

برگهٔ System

برگهٔ System مشتمل بر ۸ جزء سازنده می‌باشد. فهرست این اجزاء در جدول ۴-۳ آورده شده است.

جدول ۴-۳ اجزاء سازندهٔ موجود در برگهٔ System

جزء سازنده	توضیح
Timer	برای فعال کردن رویه‌ها، رویدادها و توابع در فواصل زمانی معین مورد استفاده قرار می‌گیرد.
PaintBox	برای ایجاد ناحیهٔ قابل رنگ‌آمیزی در فرمها مورد استفاده قرار می‌گیرد.
MediaPlayer	برای پخش صدا، فایل‌های ویدئویی و چند رسانه‌ای مورد استفاده قرار می‌گیرد.
OleContainer	برای ایجاد یک ناحیهٔ سرویس‌گیرندهٔ OLE[2] در برنامه‌های کاربردی مورد استفاده قرار می‌گیرد.
DdeClientConv	به منظور برقراری ارتباط با یک سرویس‌دهندهٔ DDE (توسط یک سرویس‌گیرندهٔ DDE) مورد استفاده قرار می‌گیرد.

۱- Hypertext Markup Language

۲- Object Linking and Embedding (تکنولوژی برای انتقال و به اشتراک گذاشتن اطلاعات در بین برنامه‌های کاربردی)

جدول ۴-۳ ادامه

جزء سازنده	توضیح
DdeClientItem	برای مشخص نمودن داده‌های ارسالی سرویس‌گیرنده (حین ارتباط با سرویس‌دهندهٔ DDE) مورد استفاده قرار می‌گیرد.
DdeServerConv	به منظور برقراری ارتباط با یک سرویس‌گیرندهٔ DDE (توسط یک سرویس‌دهندهٔ DDE) مورد استفاده قرار می‌گیرد.
DdeServerItem	برای مشخص نمودن داده‌های ارسالی سرویس‌دهنده (حین ارتباط با سرویس‌گیرندهٔ DDE) مورد استفاده قرار می‌گیرد.

همانطور که ملاحظه خواهید نمود، برگهٔ System حاوی اجزاء سازندهٔ دیگری به جزء اجزاء سازندهٔ ذکر شده است. این موارد جزو اجزاء سازندهٔ CLX[1] بوده و عبارتند از: DirectorTreeView ، FileListView ، FileIconView ، FileEdit ، FileHistoryComboBox و FilterComboBox که موارد استفاده آنها نیز با توجه به نام هر کدام از آنها مشخص می‌باشد. توصیه می‌شود برای دریافت توضیحات بیشتر در مورد این اجزاء سازنده به راهنمای دلفی ۷ مراجعه کنید.

برگهٔ Data Access

برگهٔ Data Access در دلفی ۷ شامل ۶ جزء سازنده است. فهرست این اجزاء را در جدول ۵-۳ مشاهده می‌کنید.

جدول ۵-۳ اجزاء سازندهٔ موجود در برگهٔ Data Access

جزء سازنده	توضیح
DataSource	برای برقراری ارتباط بین اجزاء سازندهٔ Table یا Query مورد استفاده قرار می‌گیرد.
ClientDataSet	برای دستیابی یک Client به یک بانک اطلاعاتی مورد استفاده قرار می‌گیرد.
DataSetProvider	برای برقراری ارتباط بین یک سرویس‌دهندهٔ بانک اطلاعاتی راه دور و یک برنامهٔ کاربردی سرویس‌گیرنده مورد استفاده قرار می‌گیرد.
XMLTransform	برای تبدیل سندهای XML[2] به بسته‌های داده‌ای حاوی جداول بانک اطلاعاتی یا همان سندهای XML و یا بالعکس مورد استفاده قرار می‌گیرد.
XMLTransformClient	برای تبدیل سندهای XML در طرف سرویس‌گیرنده مورد استفاده قرار می‌گیرد.

برگهٔ Data Controls

برگهٔ Data Controls دربرگیرندهٔ ۱۵ جزء سازنده است. فهرست این اجزاء در جدول ۶-۳ ارائه شده است.

۱- Component Libray for Crass-Platform ۲- Extrnsible Markup Language

جدول ٦-٣ اجزاء سازندهٔ موجود در برگهٔ Data Controls

جزء سازنده	توضیح
DBGrid	برای نمایش داده‌های یک بانک اطلاعاتی در یک جدول مورد استفاده قرار می‌گیرد.
DBNavigator	برای ایجاد یک شیء کنترل کننده بانک اطلاعاتی مورد استفاده قرار می‌گیرد. این شیء کنترلی، قابلیت پیمایش و ویرایش داده‌های بانک اطلاعاتی را دارا می‌باشد.
DBText	یک نسخه از دادهٔ موجود در بانک اطلاعاتی که به صورت Label نمایش داده می‌شود.
DBEdit	یک نسخه از دادهٔ موجود در بانک اطلاعاتی که در کادر ویرایشی نمایش داده می‌شود.
DBMemo	یک نسخه از داده‌های موجود در بانک اطلاعاتی که به صورت Memo نمایش داده می‌شوند.
DBImage	یک نسخه از داده‌های تصویری موجود در بانک اطلاعاتی است.
DBListBox	نسخه‌ای از داده‌های موجود در بانک اطلاعاتی که در قالب ListBox نمایش داده می‌شوند.
DBComboBox	نسخه‌ای از داده‌های موجود در بانک اطلاعاتی که در قالب ComboBox نمایش داده می‌شود.
DBCheckBox	نسخه‌ای از داده‌های موجود که در قالب CheckBox مورد استفاده قرار می‌گیرد.
DBRadioGroup	نسخه‌ای از داده‌های موجود که در قالب RadioGroup مورد استفاده قرار می‌گیرد.
DBLookupListBox	برای ایجاد یک ListBox جهت جستجوی داده‌ها در بانک اطلاعاتی مورد استفاده قرار می‌گیرد.
DBLookUpComboBox	برای ایجاد یک ComboBox جهت جستجوی داده‌ها در بانک اطلاعاتی مورد استفاده قرار می‌گیرد.
DBRichEdit	برای ایجاد یک فیلد RichEdit مورد استفاده قرار می‌گیرد.
DBCtrlGrid	برای ایجاد ابزار نمایشی جدولی داده‌ها مورد استفاده قرار می‌گیرد.
DBChart	برای نمایش داده‌ها بانک اطلاعاتی در قالب نمودار مورد استفاده قرار می‌گیرد.

برگهٔ dbExpress

برگهٔ dbExpress شامل ۷ جزء سازنده به شرح جدول ۷-۳ می‌باشد.

جدول ۷-۳ اجزاء سازندهٔ موجود در برگه dbExpress

جزء سازنده	توضیح
SQLConnection	امکان ارتباط dbExpress با سرویس‌دهندهٔ بانک اطلاعاتی را فراهم می‌آورد.
SQLDataset	برای دستیابی dbExpress به داده‌های بانک اطلاعاتی مورد استفاده قرار می‌گیرد.
SQLQuery	برای ایجاد یک پرس‌وجوی قابل اجرا توسط dbExpress مورد استفاده قرار می‌گیرد.

جدول ۷-۳ ادامه

جزء سازنده	توضیح
SQLStoredProc	برای ایجاد یک روال از پیش طراحی شده و قابل اجراء توسط dbExpress مورد استفاده قرار می‌گیرد.
SQLTable	برای ایجاد جداول (Table) قابل دسترسی توسط dbExpress مورد استفاده قرار می‌گیرد.
SQLMonitor	برای تشخیص داده‌های ارسال/ دریافت شده هنگام ارتباط با بانک اطلاعاتی مورد استفاده قرار می‌گیرد.
SimpleDataset	این جزء سازنده، از اجزاء سازندهٔ DatasetProvider و SQLDataset برای دسترسی به بانک اطلاعاتی بهره می‌گیرد.

برگهٔ Datasnap

اجزاء سازندهٔ موجود در برگهٔ Datasnap از پروتکل‌های گوناگونی برای اتصال به کامپیوترهای راه دور پشتیبانی می‌کنند. به‌طورکلی از اجزاء سازندهٔ موجود در Datasnap برای اتصال به دستگاههای سرویس‌دهنده استفاده می‌شود. برگهٔ Datasnap مشتمل بر ۷ جزء سازنده است که در جدول ۸-۳ ارائه شده‌اند.

جدول ۸-۳ اجزاء سازندهٔ موجود در برگهٔ Datasnap

جزء سازنده	توضیح
DCOMConnection	از اتصال DCOM مایکروسافت به کامپیوتر راه دور پشتیبانی می‌کند. توجه داشته باشید که DCOM باید در کامپیوتر راه دور نصب شده باشد.
SocketConnection	از اتصال TCP/IP به یک برنامهٔ کاربردی سرویس‌دهندهٔ راه دور پشتیبانی می‌کند. در این حالت scktsrvr.exe باید در سرویس‌دهندهٔ راه دور در حال اجرا باشد.
SimpleObjectBroker	امکان برقراری ارتباط با سرویس‌دهنده‌های قابل دسترس (معتبر) را فراهم می‌آورد.
WebConnection	با استفاده از Http ، امکان اتصال به یک برنامهٔ کاربردی سرویس‌دهندهٔ راه دور را فراهم می‌سازد. در این حالت، wininet.dll باید در کامپیوتر سرویس‌گیرنده نصب شده باشد.
ConnectionBroker	امکان جمع‌آوری (تمرکز) پروتکل‌های مختلف جهت برقراری اتصال بین سرویس‌دهنده‌ها و سرویس‌گیرنده‌ها را فراهم می‌سازد.
SharedConnection	برای اتصال به سرویس‌دهنده‌هایی مورد استفاده قرار می‌گیرد که دارای چندین زیرمجموعه (چندین سرویس‌دهنده) هستند.
LocalConnection	برای دسترسی به برنامه‌های multi-tiered مورد استفاده قرار می‌گیرد.

برگهٔ BDE[1]

اجزاء سازندهٔ متداول در بانک‌های اطلاعاتی را در برگه BDE خواهید یافت. برگه BDE دربرگیرندهٔ ۸

۱- Broland DataBase Engine

جزء سازنده می‌باشد. فهرست این اجزاء سازنده را در جدول ۹-۳ مشاهده می‌کنید.

جدول ۹-۳ اجزاء سازندهٔ موجود در برگهٔ BDE

جزء سازنده	توضیح
Table	برای برقراری ارتباط مابین یک جدول از بانک اطلاعاتی و برنامهٔ کاربردی مورد استفاده قرار می‌گیرد.
Query	برای ایجاد و اجرای پرس‌وجوهای (از نوع SQL) یک بانک اطلاعاتی مورد استفاده قرار می‌گیرد.
StoredProc	برای اجرای رویه‌هایی که در یک سرویس‌دهندهٔ SQL ذخیره شده‌اند به‌کار می‌رود.
Database	برای برقراری ارتباط با سرویس‌دهنده‌های بانک اطلاعاتی (راه دور) مورد استفاده قرار می‌گیرد.
Session	امکان کنترل بر ارتباط‌های بانک اطلاعاتی یک برنامهٔ کاربردی را فراهم می‌آورد.
BatchMove	به کاربران امکان می‌دهد که به طور محلی بر روی جداول کار کنند و سپس جداول بروز رسانده شده را به سرویس‌دهنده ارسال نمایند. (معمولاً در تهیه پشتیبان کاربرد دارند).
UpdateSQL	امکان بروز رسانی یک بانک اطلاعاتی SQL را فراهم می‌آورد.
NestedTable	امکان درج و بازیابی داده‌ها با قالب تو در تو (Nested) را در جداول بانک اطلاعاتی فراهم می‌آورد.

برگهٔ ADO[1]

دلفی ۷ شیءهای کنترلی دارد که امکان استفاده از ADO را فراهم می‌سازند. این اجزاء سازنده در برگهٔ ADO قرار گرفته‌اند. فهرست این اجزاء سازنده در جدول ۱۰-۳ ارائه شده است.

جدول ۱۰-۳ اجزاء سازندهٔ موجود در برگهٔ ADO

جزء سازنده	توضیح
ADOConnection	امکان ارتباط با بانک‌های اطلاعاتی ADO را فراهم می‌سازد.
ADOCommand	امکان اجرای فرامین SQL (بدون خروجی) را بر روی بانک‌های اطلاعاتی ADO فراهم می‌آورد.
ADODataset	امکان کار (ویرایش، درج، حذف و ...) بر روی بانک‌های اطلاعاتی ADO را فراهم می‌آورد. این جزء سازنده می‌تواند با اجزاء سازندهٔ ADOTable ، ADOQuery و ADOStoredProc ارتباط برقرار نماید.
ADOTable	برای برقراری ارتباط مابین یک جدول از بانک‌های اطلاعاتی ADO و برنامهٔ کاربردی مورد استفاده قرار می‌گیرد.

۱- ActiveX Data Objects

جدول ۱۰-۳ ادامه

جزء سازنده	توضیح
ADOQuery	برای ایجاد و اجرای پرس و جوهای (از نوع SQL) یک بانک اطلاعاتی ADO مورد استفاده قرار می‌گیرد.
ADOStoredProc	برای اجرای رویه‌های ذخیره شدهٔ برای کار با ADO مورد استفاده قرار می‌گیرد.
RDSConnection	امکان مدیریت داده‌ها در بانک‌های اطلاعاتی multi-tier را فراهم می‌آورد.

برگهٔ InterBase

امکان استفاده از بانک‌های اطلاعاتی InterBase در دلفی ۷ فراهم شده است. اجزاء سازندهٔ مرتبط با این کار در برگهٔ InterBase وجود دارند. این برگه شامل ۱۳ جزء سازنده است که فهرست آنها را در جدول ۱۱-۳ مشاهده خواهید نمود.

جدول ۱۱-۳ اجزاء سازندهٔ موجود در برگهٔ InterBase

جزء سازنده	توضیح
IBTable	برای برقراری ارتباط مابین یک جدول از بانک‌های اطلاعاتی InterBase و برنامهٔ کاربردی مورد استفاده قرار می‌گیرد.
IBQuery	برای ایجاد و اجرای پرس‌وجوهای یک بانک اطلاعاتی InterBase مورد استفاده قرار می‌گیرد.
IBStoredProc	برای اجرای رویه‌های ذخیره شده برای کار با بانک‌های اطلاعاتی InterBase مورد استفاده قرار می‌گیرد.
IBTransaction	امکان کنترل تراکنش‌ها مابین چندین بانک اطلاعاتی InterBase را فراهم می‌آورد.
IBUpdateSQL	امکان بروزرسانی بانک اطلاعاتی را فراهم می‌سازد.
IBDataset	امکان کار بر روی بانک‌های اطلاعاتی InterBase را فراهم می‌سازد.
IBSQL	برای اجرای فرامین SQL بر روی بانک‌های اطلاعاتی InterBase مورد استفاده قرار می‌گیرد.
IBDatabaseInfo	برای بازیابی برخی از اطلاعات مربوط به بانک اطلاعاتی (مانند نسخهٔ نرم‌افزاری، حافظهٔ استفاده شده مورد استفاده و ...) قرار می‌گیرد.
IBSQLMonitor	امکان کنترل اجرای فرامین SQL را فراهم می‌آورد.
IBExtract	امکان مدیریت رویدادها در سرویس‌دهندهٔ InterBase را فراهم می‌سازد.
IBEctract	برای بازیابی اطلاعاتی نظیر اطلاعات شاخص‌های (Index) بانک، دیدگاهها، role ها و ... مورد استفاده قرار می‌گیرد.
IBClientDataset	برای دستیابی یک Client به یک بانک اطلاعاتی InterBase مورد استفاده قرار می‌گیرد.

برگهٔ InterBase Admin

برگهٔ InterBase Admin حاوی شیءهای کنترلی می‌باشد که در اعمال مدیریتی بانک‌های اطلاعاتی

InterBase کاربرد دارند. این برگهٔ مشتمل بر ۱۱ جزء سازنده به شرح جدول ۱۲-۳ می‌باشد.

جدول ۱۲-۳ اجزاء سازندهٔ موجود در برگهٔ InterBase Admin

جزء سازنده	توضیح
IBConfigService	برای پیکربندی پارامترهای بانک اطلاعاتی مورد استفاده قرار می‌گیرد.
IBBackupService	امکان پشتیبان‌گیری از بانک اطلاعاتی را فراهم می‌آورد.
IBRestoreService	امکان بازیابی اطلاعات (در صورت تهیه فایل‌های پشتیبانی) بانک اطلاعاتی را فراهم می‌آورد.
IBValidationService	امکان فعال نمودن (معتبر نمودن) بانک اطلاعاتی برای انجام تراکنش‌ها را فراهم می‌آورد.
IBStatisticalService	امکان گزارش‌گیری از (نمایش) اطلاعات آماری بانک اطلاعاتی را فراهم می‌سازد.
IBLogService	جهت بازیابی اطلاعات موجود در فایل interbase.Log مورد استفاده قرار می‌گیرد. این فایل حاوی جزئیات انجام شده بر روی بانک اطلاعاتی است.
IBSecurityService	امکان مدیریت کاربران (برای برقراری امنیت) بانک اطلاعاتی را فراهم می‌آورد.
IBServerProperties	جهت بازیابی اطلاعات سرویس‌دهندهٔ بانک اطلاعاتی مورد استفاده قرار می‌گیرد.
IBLicensingService	جهت پیکربندی پارامترهای خاص مورد استفاده قرار می‌گیرد.
IBInstall	جهت تنظیم و نصب اجزاء بانک اطلاعاتی InterBase کاربرد دارد.
IBUnInstall	جهت حذف (یا غیرفعال نمودن) اجزاء بانک اطلاعاتی InterBase کاربرد دارد.

برگهٔ Web Services

در دلفی ۷ امکان بکارگیری و استفاده از پروتکل SOAP[1] وجود دارد. این پروتکل در سیستم‌های توزیع شده (distributed) و برای تبادل اطلاعات مابین آنها کاربرد دارد. شیءهای کنترلی یا اجزاء سازندهٔ مربوط به این کار در برگهٔ Web Services دلفی ۷ گنجانیده شده‌اند. فهرست اجزاء سازنده برگهٔ Web services در جدول ۱۳-۳ ارائه شده است.

جدول ۱۳-۳ اجزاء سازندهٔ موجود در برگهٔ Web Services

جزء سازنده	توضیح
HTTPRIO	از پیام‌های http جهت فراخوانی پروتکل (یا اشیاء) SOAP استفاده می‌کند.
HTTPRegResp	جهت اجرای یک روال در طرف سرویس‌دهنده مورد استفاده قرار می‌گیرد.
OPToSoapDomConvert	جهت مدیریت روال‌های SOAP مورد استفاده قرار می‌گیرد.
HTTPSoapDispatcher	جهت پاسخگویی به روال‌های SOAP و ارسال آنها مورد استفاده قرار می‌گیرد.

۱- Simple Object Access Protocol

جدول ۱۳-۳ ادامه

جزء سازنده	توضیح
WSDLHTMLPublish	از سندهای مبتنی بر WSDL[1] پشتیبانی می‌کند.
HTTPSoapPascalInvoker	جهت تفسیر و اجرای پیام‌های SOAP کاربرد دارد.

برگهٔ Intenet Experss

اجزاء سازندهٔ موجود در این برگهٔ جهت کار با ابزار Internet Express دلفی ۷ کاربرد دارند. فهرست اجزاء سازندهٔ موجود در برگهٔ Internet Express در جدول ۱۴-۳ ارائه شده است.

جدول ۱۴-۳ اجزاء سازندهٔ موجود در برگهٔ Internet Express

جزء سازنده	توضیح
XMLBroker	جهت بازیابی و فراخوانی سندهای XML موجود در طرف سرویس‌دهنده، مورد استفاده قرار می‌گیرد.
INETXPageProducer	جهت ایجاد صفحات HTML و پشتیبانی از سندهای XML کاربرد دارد.

برگهٔ Internet

برگهٔ Internet دلفی ۷ دربرگیرندهٔ ۱۳ جزء سازنده است. این اجزاء سازنده برای طراحی برنامه‌های کاربردی مبتنی بر اینترنت و TCP/IP کاربرد دارند. فهرست این اجزاء سازنده در جدول ۱۵-۳ ارائه گردیده است.

جدول ۱۵-۳ اجزاء سازندهٔ موجود در برگهٔ Internet

جزء سازنده	توضیح
ClientSocket	برای برقراری اتصال با کامپیوتر دیگری از شبکه مورد استفاده قرار می‌گیرد. برای اتصال از پروتکل TCP/IP استفاده می‌شود.
ServerSocket	برای پاسخگویی به درخواست‌های سرویس‌گیرنده مورد استفاده قرار می‌گیرد. (با استفاده از پروتکل TCP/IP)
WebDispatcher	برای تبدیل قالب داده‌ها به صفحات HTML مورد استفاده قرار می‌گیرد.
PageProducer	برای تبدیل صفحات HTML به رشته‌ای از کدهای HTML که توسط مرورگرهای اینترنتی قابل مشاهده باشند، به کار می‌رود.
DataSetTableProducer	برای ایجاد یک صفحهٔ HTML از روی رکوردهای یک Dataset مورد استفاده قرار می‌گیرد. (با استفاده از شیء TDataset).

۱- Web Service Definition Language

جدول ۱۵-۳ ادامه

جزء سازنده	توضیح
DatasetPageProducer	برای تبدیل صفحات HTML به کدهای HTMLکاربرد دارد.
QueryTableProducer	برای ایجاد یک صفحهٔ HTML از روی رکوردهای یک Dataset مورد استفاده قرار می‌گیرد. (با استفاده از شیء TQuery)
SQLQueryTableProducer	برای ایجاد یک صفحهٔ HTML از روی رکوردهای یک Dataset مورد استفاده قرار می‌گیرد. (با استفاده از شیء TSQLQuery)
TcpClient	برای برقراری ارتباط با کامپیوترهای سرویس‌گیرنده از طریق اینترنت (TCP/IP) مورد استفاده قرار می‌گیرد.
TcpServer	برای برقراری ارتباط با کامپیوترهای سرویس‌دهنده از طریق اینترنت (TCP/IP) مورد استفاده قرار می‌گیرد.
UpdSocket	برای برقراری اتصالات UDP[1] در یک شبکه مورد استفاده قرار می‌گیرد.
XMLDocument	امکان خواندن صفحات و سندهای XML را فراهم می‌آورد.
WebBrowser	امکان اتصال به مرورگرهای اینترنتی را فراهم می‌آورد. برای این کار فایل SHDOCVW.DLL باید در فهرست محل نصب ویندوز وجود داشته باشد.

برگهٔ Websnap

اجزاء سازنده موجود در این برگهٔ برای طراحی برنامه‌های سرویس‌دهندهٔ مبتنی بر وب مورد استفاده قرار می‌گیرند. فهرست این اجزاء را در جدول ۱۶-۳ مشاهده می‌کنید.

جدول ۱۶-۳ اجزاء سازندهٔ موجود در برگهٔ websnap

جزء سازنده	توضیح
Adapter	امکان مدیریت رویدادها و اجرای فرامین در برنامه‌های کاربردی مبتنی بر وب را فراهم می‌سازد.
PageAdapter	برای نمایش حجم وسیعی از داده‌های مبتنی بر وب (HTML) مورد استفاده قرار می‌گیرد.
DatasetAdapter	امکان کارآیی یک Dataset در محیط‌های[2] Stateless را فراهم می‌آورد.
LoginFormAdaptr	امکان طراحی فرم ورودی (Login) را فراهم می‌آورد.
StringsValuesList	امکان نمایش و ذخیره‌سازی داده‌ها به صورت زوج‌هایی متشکل از نام و مقدار آنها را فراهم می‌آورد.
DatasetValuesList	امکان نمایش داده‌های موجود در یک Dataset را به صورت زوج‌هایی متشکل از نام و مقدار آنها را فراهم می‌آورد.
WebAppcomponents	برای مدیریت برنامه‌های کاربردی websnap مورد استفاده قرار می‌گیرد.

۱- User Datagram Protocol

۲- در این رابطه به فصل ۱۳ مراجعه نمایید.

جدول ۱۶-۳ ادامه

جزء سازنده	توضیح
ApplicationAdapter	برای مدیریت و کنترل شیءهای فعال برنامهٔ کاربردی مورد استفاده قرار می‌گیرد.
EnduserAdapter	برای بازیابی (یا جمع‌آوری) اطلاعات مربوط به کاربران نظیر نام، حدود دسترسی و ... مورد استفاده قرار می‌گیرد.
EndUserSessionAdapter	امکان کنترل ارتباطهای کاربران با برنامه‌های کاربردی را فراهم می‌سازد.
PageDispatcher	امکان توزیع و ارسال درخواستهای http به صفحات وب را فراهم می‌سازد.
AdapterDispatcher	برای پردازش و دستکاری صفحات HTML مورد استفاده قرار می‌گیرد.
LocateFileService	امکان کنترل و نگهداری فهرستها و فایل‌های موقتی (در زمان اجرا) را فراهم می‌سازد.
SessionService	برای ذخیره‌سازی اطلاعات متداول کاربران و کنترل ارتباطهای انجام شده مورد استفاده قرار می‌گیرد.
WebUserList	برای ذخیره‌سازی فهرست کاربران شبکه (به همراه کد شناسایی و کلمهٔ رمز) مورد استفاده قرار می‌گیرد.
XSLPageProducer	برای تبدیل سندهای XML به XSL[1] مورد استفاده قرار می‌گیرد.
AdapterPageProducer	برای ایجاد صفحات HTML و Javascript مورد استفاده قرار می‌گیرد.

برگهٔ Decision Cube

برگهٔ Decision Cube شامل ۶ جزء سازنده برای رسم نمودارها و تصاویر گرافیکی چند بُعدی است. فهرست این اجزاء سازنده را در جدول ۱۷-۳ ملاحظه خواهید نمود.

جدول ۱۷-۳ اجزاء سازندهٔ موجود در برگهٔ Decision Cube

جزء سازنده	توضیح
DecisionCube	یک مکان چند بُعدی برای نگهداری داده‌ها است و برای ارزیابی داده‌ها از یک Dataset هم می‌تواند مورد استفاده قرار گیرد.
DecisionQuery	نسخهٔ ویژه‌ای از TQuery است که برای استفاده از DecisionCube طراحی شده است.
DecisionSource	برای تعریف و تعیین وضعیت جاری اجزایی چون DecisionGroup و DecisionGrid مورد استفاده قرار می‌گیرد.
DecisionPivot	برای باز و بسته نمودن فیلدهای Decisioncube مورد استفاده قرار می‌گیرد.
DecisionGrid	برای نمایش داده‌های Decisioncube در قالب جداول مورد استفاده قرارمی‌گیرد.
DecisionGroup	برای نمایش داده‌های Decisioncube در قالب نمودار و تصاویر گرافیکی مورد استفاده قرار می‌گیرد.

۱- Extensible Stylesheet Language

برگهٔ Dialogs

اجزاء سازندهٔ موجود در برگهٔ Dialogs ، جهت ایجاد کادرهای مکالمه در برنامه‌های کاربردی تحت ویندوز مورد استفاده قرار می‌گیرند. استفاده از کادرهای مکالمه در یکسان‌سازی و هماهنگی اجزایی برنامه‌هایتان نقش بسزایی خواهد داشت.

این برگهٔ شامل ۱۰ جزء سازنده است که فهرست آنها در جدول ۱۸-۳ ارائه شده است.

جدول ۱۸-۳ اجزاء سازندهٔ موجود در برگهٔ Dialogs

جزء سازنده	توضیح
OpenDialog	برای ایجاد یک کادر مکالمهٔ Open File (باز کردن فایل) مورد استفاده قرار می‌گیرد.
SaveDialog	برای ایجاد یک کادر مکالمهٔ Save File (ذخیره کردن فایل) مورد استفاده قرار می‌گیرد.
OpenPictureDialog	برای ایجاد یک کادر مکالمهٔ Open Picture (باز کردن فایل‌های گرافیکی) مورد استفاده قرار می‌گیرد.
SavePictureDialog	برای ایجاد یک کادر مکالمهٔ Save Picture (ذخیره کردن فایل‌های گرافیکی) مورد استفاده قرار می‌گیرد.
FontDialog	برای ایجاد یک کادر مکالمهٔ Font (تعیین فونت) مورد استفاده قرار می‌گیرد.
ColorDialog	برای ایجاد یک کادر مکالمهٔ Color (تعیین رنگ) مورد استفاده قرار می‌گیرد.
PrintDialog	برای ایجاد یک کادر مکالمهٔ Print (تعیین ویژگیهای چاپ) مورد استفاده قرار می‌گیرد.
PrinterSetupDialog	برای ایجاد یک کادر مکالمهٔ Printer Setup (تنظیم چاپگر) مورد استفاده قرار می‌گیرد.
FindDialog	برای ایجاد یک کادر مکالمهٔ Find (جستجو) مورد استفاده قرار می‌گیرد.
ReplaceDialog	برای ایجاد یک کادر مکالمهٔ Replace (جایگذاری) مورد استفاده قرار می‌گیرد.

برگهٔ Win 3.1

برگهٔ Win 3.1 شامل ۱۱ جزء سازنده است که به منظور سازگاری با نگارشهای قبلی دلفی مورد استفاده قرار گرفته و امکان انتقال برنامه‌های کاربردی را میسر می‌سازند. فهرست این اجزاء سازنده در جدول ۱۹-۳ ارائه شده است.

جدول ۱۹-۳ اجزاء سازندهٔ موجود در برگهٔ win 3.1

جزء سازنده	توضیح
DBLookupList	یک شیء کنترلی در ویندوز 3.1 است که برای جستجوی یک مقدار در یک جدول مورد استفاده قرار می‌گیرد. این کار با نمایش یک کادر فهرست انجام می‌پذیرد.
DBLookupCombo	همانند DBLookup بوده با این تفاوت که برای جستجو از کادر Combo استفاده خواهد نمود.
TabSet	برای ایجاد برگه‌های چند صفحه‌ای مورد استفاده قرار می‌گیرد.

جدول ۱۹-۳ ادامه

جزء سازنده	توضیح
OutLine	امکان نمایش درختواره داده‌ها را با قالب سلسله مراتبی فراهم می‌سازد.
TabbedNotebook	امکان ایجاد فرم‌های چند صفحه‌ای به وسیلهٔ برگه‌ها را فراهم می‌آورد.
Notebook	برای ایجاد چندین صفحهٔ قابل استفاده با Tabset مورد استفاده قرار می‌گیرد.
Header	امکان نمایش متن‌ها در قسمتهای قابل تغییر اندازه را فراهم می‌آورد.
FileListBox	امکان ایجاد کادر فهرست برای نمایش فایل‌ها را فراهم می‌آورد.
DirectoryListBox	امکان ایجاد یک فهرست برای نمایش دایرکتوری‌ها را فراهم می‌آورد.
DriveComboBox	امکان ایجاد یک کادر Combo برای انتخاب درایو را فراهم می‌آورد.
FilterComboBox	امکان ایجاد یک کادر Combo برای انتخاب یا نمایش فیلتر فایل‌ها را فراهم می‌سازد.

برگهٔ Samples

برگهٔ Samples دلفی ۷ شامل ۱۱ جزء سازنده است. این اجزاء تنها به عنوان نمونه ارائه شده‌اند و دلفی مستنداتی برای آنها ارائه نداده است. یونیت‌های این اجزاء را می‌توانید در فهرست source\samples\ از محل نصب دلفی ۷ پیدا کنید. فهرست این اجزاء در جدول ۲۰-۳ ارائه شده است.

جدول ۲۰-۳ اجزاء سازندهٔ موجود در برگهٔ Sample

جزء سازنده	توضیح
IBEventAlerter	برای هشدار دادن در مورد رویدادهای یک بانک اطلاعاتی مورد استفاده قرار می‌گیرد.
SpinButton	برای ایجاد دکمه‌های شماره‌انداز مورد استفاده قرار می‌گیرد.
SpinEdit	برای ایجاد یک کادر ویرایشی با ویژگی‌های یک شماره‌انداز مورد استفاده قرارمی‌گیرد.
ColorGrid	برای ایجاد یک جدول شطرنجی از رنگهای گوناگون مورد استفاده قرار می‌گیرد.
Gauge	برای ایجاد راهنمای پیشرفت کار مورد استفاده قرار می‌گیرد.
DirectoryOutLine	برای نمایش ساختار دایرکتوری‌ها مورد استفاده قرار می‌گیرد.
Calendar	برای نمایش یک تقویم مورد استفاده قرار می‌گیرد.
ShellFreeViwer	امکان نمایش درختواره فایل‌ها و فهرستها را فراهم می‌آورد.
ShellComboBox	امکان نمایش یک کادر Combo را فراهم می‌آورد.
ShellListView	امکان نمایش یک کادر فهرست را فراهم می‌آورد.
ShellChangeNotifier	برای هشدار دادن در هنگام تغییرات در برنامه‌های کاربردی مورد استفاده قرار می‌گیرد.

برگهٔ ActiveX

برگهٔ ActiveX دلفی ۷ شامل ۴ جزء سازندهٔ ActiveX است. فهرست این اجزاء سازنده در جدول ۲۱-۳ ارائه شده است.

جدول ۲۱-۳ اجزاء سازندهٔ موجود در برگهٔ ActiveX

جزء سازنده	توضیح
Chartfx	برای افزودن قابلیت رسم نمودار در برنامه‌های کاربردی مورد استفاده قرار می‌گیرد.
VSSPell	برای بررسی درستی املای کلمات مورد استفاده قرار می‌گیرد.
F1Book	امکان افزودن صفحه گسترده (spreadsheet) به برنامه‌های کاربردی را فراهم می‌سازد.
Vtchart	برای افزودن قابلیت رسم نمودارهای سه بُعدی در برنامه‌های کاربردی مورد استفاده قرار می‌گیرد.

برگهٔ COM+[1]

Com مبنایی را تشکیل می‌دهد که تکنولوژی OLE و ActiveX از آن شکل گرفته‌اند. برگهٔ COM+ دلفی ۷ شامل یک جزء سازنده است. توضیحات مربوط به این جزء سازنده در جدول ۲۲-۳ ارائه شده است.

جدول ۲۲-۳ اجزاء سازندهٔ موجود در برگهٔ COM+

جزء سازنده	توضیح
COMAdminCatalog	امکان استفاده از سرویس‌های COM را در برنامه‌های کاربردی فراهم می‌آورد.

برگهٔ Indy[2] Clients

این برگهٔ عموماً شامل اجزاء سازنده‌ای است که از پروتکل‌های اینترنت پشتیبانی می‌کند. برگهٔ Indy client دلفی ۷ دربرگیرندهٔ ۳۳ جزء سازنده است. فهرست این اجزاء در جدول ۲۳-۳ ارائه شده است.

جدول ۲۳-۳ اجزاء سازندهٔ موجود در برگهٔ Indy client

جزء سازنده	توضیح
IdTCPClient	برای ایجاد برنامه‌های سرویس‌گیرنده براساس پروتکل TCP مورد استفاده قرار می‌گیرد.[3]
IdUDPClient	برای ایجاد برنامه‌های سرویس‌گیرنده براساس پروتکل UDP مورد استفاده قرار می‌گیرد.
IdDayTime	از پروتکل DayTime (RFC 867) برای برقراری ارتباط باکامپیوترهای سرویس‌گیرنده پشتیبانی می‌کند.
IdDayTimeUDP	از پروتکل DayTime UDP برای برقراری ارتباط باکامپیوترهای سرویس‌گیرنده پشتیبانی می‌کند.

۱- Component Object Model

۲- Internet Direct

جدول ۲۳-۳ ادامه

جزء سازنده	توضیح
IdDNSResolver	برای ثبت جستجوهای انجام شده در مورد DNS[۱] ها مورد استفاده قرار می‌گیرد.
IdEcho	از پروتکل Echo (RFC 862) برای برقراری ارتباط با کامپیوترهای سرویس‌گیرنده پشتیبانی می‌کند.
IdEchoUDP	برای ایجاد برنامه‌های سرویس‌گیرنده براساس پروتکل Echo UDP مورد استفاده قرار می‌گیرد.
IdFinger	برای ایجادبرنامه‌های سرویس‌گیرنده براساس پروتکل Finger[۲] (RFC 1288) مورد استفاده قرار می‌گیرد.
IdFTP	برای انتقال فایل در بین کامپیوترها مورد استفاده قرار می‌گیرد. این کار با استفاده از پروتکل FTP[۳] انجام می‌پذیرد.
IdGopher	از پروتکل Gopher (RFC 1436) پشتیبانی می‌کند.
IdHTTP	برای اتصال به سرویس‌دهنده‌های وب مورد استفاده قرار می‌گیرد. این کار با استفاده از پروتکل HTTP انجام می‌پذیرد.
IdIcmpClient	برای ارسال پیام‌های کنترلی در اینترنت مورد استفاده قرار می‌گیرد. این کار با استفاده از پروتکل ICMP[۴] انجام می‌پذیرد.
IdIdent	از پروتکل Ident[۵] پشتیبانی می‌کند.
IdIMAP4	از پروتکل IMAP4[۶] پشتیبانی می‌کند.
IdIPMCastClient	امکان مدیریت و کنترل IP در برنامه‌های کاربردی مبتنی بر اینترنت یا سرویس‌گیرنده/ سرویس‌دهنده را فراهم می‌آورد.
IdIRC	برای ایجاد برنامه‌های سرویس‌گیرنده براساس پروتکل IRC[۷] مورد استفاده قرار می‌گیرد.
IdLPR	برای ایجاد برنامه‌های سرویس‌گیرنده براساس پروتکل LPR[۸] مورد استفاده قرار می‌گیرد.
IdNNTP	برای ایجاد برنامه‌های سرویس‌گیرندهٔ خبری مورد استفاده قرار می‌گیرد. این کار با استفاده از پروتکل NNTP[۹] انجام می‌پذیرد.
IdPOP3	برای اتصال به سرویس‌دهنده‌های POP3[۱۰] جهت کار با پست الکترونیکی مورد استفاده قرار می‌گیرد.
IdQOTD	برای ایجاد برنامه‌های سرویس‌گیرنده براساس پروتکل QUTOD[۱۱] مورد استفاده قرار می‌گیرد.
IdQOTDUDP	برای ایجاد برنامه‌های سرویس‌گیرنده براساس پروتکل QOTD UDP مورد استفاده قرار می‌گیرد.
IdRexec	از پروتکل Rexec[۱۲] پشتیبانی می‌کند.
IdRSH	از پروتکل RSH[۱۳] پشتیبانی می‌کند.

۱- Domain Name Server
۲- Finger user Information Protocol
۳- File Transfer Protocol
۴- Internet Control Message Protocol
۵- Identification (RFC 1413) Protocol
۶- Internet Message Access Protocol
۷- Internet Relay Chat Protocol
۸- Line Printer Daemon Protocol
۹- Network News Transfer Protocol
۱۰- Post Office Protcol Version3
۱۱- Quote of the Day Protocol
۱۲- Remote Execution Protocol
۱۳- Unix Execution Protocol

جدول ۲۳-۳ ادامه

جزء سازنده	توضیح
IdSMTP	برای ارسال پست الکترونیکی مورد استفاده قرار می‌گیرد. برای این‌کار از پروتکل SMTP[1] استفاده می‌کند.
IdSNMP	از پروتکل SNMP[2] پشتیبانی می‌کند.
IdSNPP	از پروتکل SNPP[3] پشتیبانی می‌کند.
IdSNTP	از پروتکل SNTP[4] پشتیبانی می‌کند.
IdSysLog	از پروتکل BSD Syslog پشتیبانی می‌کند.
IdTelnet	از پروتکل Telnet پشتیبانی می‌کند.
IdTime	از پروتکل Time (RFC 868) پشتیبانی می‌کند.
IdTimeUDP	از پروتکل Time UDP پشتیبانی می‌کند.
IdTrivialFTP	از پروتکل Trival File Transfer (RFC 1783, RFC 1350, RFC 1782) پشتیبانی می‌کند.
IdWhois	از پروتکل whois (یا Nickname) پشتیبانی می‌کند.

برگهٔ Indy Servers

دلفی ۷ شامل کلاس‌ها و اجزاء سازندهٔ متنوعی است که به شما امکان می‌دهد تا با بسیاری از پروتکل‌های متداول برنامه‌نویسی نمائید. این اجزاء سازنده، شیءهای کنترلی زیادی برای سرویس‌دهنده‌ها و سرویس‌گیرنده‌ها دارند که از پروتکل‌هایی نظیر TCP/IP ، HTTP ، UDP ، GoPher ، Echo ، Finger ، POP3 ، SMTP ، NNTP ، Telnet و Ftp پشتیبانی می‌کنند. برخی از این اجزاء سازنده را در برگهٔ Indy Clients که عموماً مربوط به برنامه‌های سرویس‌گیرنده بودند مشاهده نمودید. اجزاء سازنده برای برنامه‌های سرویس‌دهندهٔ از این قبیل نیز در برگهٔ Indy Servers گنجانیده شده‌اند. فهرست این اجزاء سازنده را در جدول ۲۴-۳ ملاحظه خواهید نمود.

جدول ۲۴-۳ اجزاء سازندهٔ موجود در برگهٔ Indy Servers

جزء سازنده	توضیح
IdTCPServer	امکان ایجاد برنامه‌های کاربردی سرویس‌دهنده براساس پروتکل TCP را فراهم می‌سازد.
IdUDPServer	امکان ایجاد برنامه‌های کاربردی سرویس‌دهنده براساس پروتکل UDP را فراهم می‌سازد.
IdChargenServer	امکان ایجاد برنامه‌های کاربردی سرویس‌دهنده براساس پروتکل Chargen[5] را فراهم می‌سازد.
IdChargenUDPServer	امکان ایجاد برنامه‌های کاربردی سرویس‌دهنده براساس پروتکل UDP Chargen

۱- Simple Mail Transfer Protocol
۲- Simple Network Management Protocol
۳- Simple Network Paging Protocol
۴- Simple Network Time Protocol
۵- Character Generator Protocol

جدول ۲۴-۳ ادامه

جزء سازنده	توضیح
	را فراهم می‌سازد.
IdDayTimeServer	امکان ایجاد برنامه‌های کاربردی سرویس‌دهنده براساس پروتکل DayTime (RFC 867) را فراهم می‌آورد.
IdDayTimeUDPServer	امکان پیاده‌سازی پروتکل DayTime در قالب پروتکل UDP را فراهم می‌سازد.
IdDICTServer	امکان پشتیبانی از پروتکل DICT[1] (RFC 2229) را فراهم می‌آورد.
IdDISCARDServer	امکان ایجاد برنامه‌های کاربردی سرویس‌دهنده براساس پروتکل Discard (RFC 863) را فراهم می‌سازد.
IdECHOServer	امکان ایجاد برنامه‌های کاربردی سرویس‌دهنده براساس پروتکل Echo (RFC 862) را فراهم می‌سازد.
IdEchoUDPServer	امکان پشتیبانی از پروتکل EchoUDP را فراهم می‌سازد.
IdFingerServer	امکان ایجاد برنامه‌های کاربردی سرویس‌دهنده براساس پروتکل Finger (RFC 1288) را فراهم می‌سازد.
IdFTPServer	امکان ایجاد برنامه‌های کاربردی سرویس‌دهنده براساس پروتکل FTP را فراهم می‌آورد.
IdGopherServer	امکان ایجاد برنامه‌های کاربردی سرویس‌دهنده براساس پروتکل GoPher را فراهم می‌آورد.
IdHTTPServer	امکان ایجاد برنامه‌های کاربردی سرویس‌دهنده براساس پروتکل HTTP را فراهم می‌سازد.
IdIdentServer	امکان ایجاد برنامه‌های کاربردی سرویس‌دهنده براساس پروتکل Ident را فراهم می‌سازد.
IdIMAP4Server	امکان ایجاد برنامه‌های کاربردی سرویس‌دهنده براساس پروتکل IMAP4 را فراهم می‌سازد.
IdIPMCastServer	امکان مدیریت و کنترل IP را در برنامه‌های کاربردی مبتنی بر اینترنت یا سرویس‌گیرنده/ سرویس‌دهنده را فراهم می‌آورد.
IdIRCServer	از پروتکل IRC پشتیبانی می‌کند.
IdMappedFTP	جهت ارتباط با سایر سرویس‌دهنده‌ها در شبکه به منظور انتقال فایل مورد استفاده قرار می‌گیرد.
IdMappedPOP3	برای اتصال به سایر سرویس‌دهنده‌های POP3 جهت کار با پست الکترونیکی مورد استفاده قرار می‌گیرد.
IdMappedPortTCP	برای اتصال به سایر سرویس‌دهنده‌های TCP در شبکه مورد استفاده قرار می‌گیرد.
IdMappedPortUDP	برای اتصال به سایر سرویس‌دهنده‌های UDP در شبکه مورد استفاده قرار می‌گیرد.
IdMappedTelnet	برای اتصال به سایر سرویس‌دهنده‌های Telnet در شبکه مورد استفاده قرارمی‌گیرد.
IdNNTPServer	برای ایجاد برنامه‌های کاربردی سرویس‌دهندهٔ خبری براساس پروتکل NNTP مورد استفاده قرار می‌گیرد.
IdPOP3Server	برای ایجاد برنامه‌های کاربردی سرویس‌دهندهٔ پست الکترونیکی براساس پروتکل POP3 مورد استفاده قرار می‌گیرد.

۱- Dictionary Server Protocol

جدول ۲۴-۳ ادامه

جزء سازنده	توضیح
dIdQOTDServer	برای ایجاد برنامه‌های کاربردی سرویس‌دهنده براساس پروتکل QUOTD مورد استفاده قرار می‌گیرد.
IdQOTDUDPServer	برای ایجاد برنامه‌های کاربردی سرویس‌دهنده براساس پروتکل UDP QUOTD مورد استفاده قرار می‌گیرد.
IdRexecServer	از پروتکل Rexec پشتیبانی می‌کند.
IdRSHServer	از پروتکل RSH پشتیبانی می‌کند.
IdSimpleServer	برای ایجاد برنامه‌های کاربردی سرویس‌دهنده براساس پروتکل TCP (بدون ایجاد Thread) مورد استفاده قرار می‌گیرد.
IdSMTPServer	برای ایجاد برنامه‌های کاربردی سرویس‌دهندهٔ پست الکترونیکی (ارسال پست الکترونیکی) مورد استفاده قرار می‌گیرد. برای این کار از پروتکل SMPT استفاده می‌کند.
IdSyslogServer	از پروتکل BSD Syslog پشتیبانی می‌کند.
IdTelnetServer	برای ایجاد برنامه‌های کاربردی سرویس‌دهنده براساس پروتکل Telnet مورد استفاده قرار می‌گیرد.
IdTimerServer	از پروتکل Time (RFC 868) پشتیبانی می‌کند.
IdTimeUDPServer	از پروتکل Time UDP پشتیبانی می‌کند.
IdTrivialFTPServer	از پروتکل Trivial File Tranfer پشتیبانی می‌کند.
IdTunnelMaster	از پیاده‌سازی پروتکل TCP در قالب Tunnel[1] پشتیبانی می‌کند. (در طرف سرویس‌دهنده‌ها)
IdTunnelSlave	از پیاده‌سازی پروتکل TCP در قالب Tunnel پشتیبانی می‌کند. (در طرف سرویس‌گیرنده)
IdWhoIsServer	از پروتکل Whois (یا Nickname) پشتیبانی می‌کند.

برگهٔ Indy Misc

فهرست اجزاء سازندهٔ این برگهٔ در جدول ۲۵-۳ ارائه شده است.

جدول ۲۵-۳ اجزاء سازندهٔ موجود در برگهٔ Indy Misc

جزء سازنده	توضیح
IdSocksInfo	از پیاده‌سازی پروتکل SOCKS پشتیبانی می‌کند. این جزء توسط اجزاء سازنده Indy Clients مورد استفاده قرار می‌گیرد.
IdAntiFreeze	در هنگام اجرای یک فرآیند در شبکه، امکان نمایش یک پیام (نمایانگر

۱ - Tunnel به معنی پنهان کردن یا پیچیدن بسته‌ها و پیامهای یک پروتکل در بسته‌های یک پروتکل دیگر است. این روش انتقال، برای اجتناب از محدودیتهای پروتکل‌ها مورد استفاده قرار می‌گیرد.

	پیشرفت کار) را فراهم می‌آورد.
IdCookieManger	امکان مدیریت Cookie[1] ها را فراهم می‌آورد.
IdEncoderMine	برای کد کردن داده‌های باینری (دودویی) به متن (Text) مورد استفاده قرار می‌گیرد. برای این کار از روش کدگذاری MIME استفاده می‌کند.
IdEncoderUUE	برای کد کردن داده‌های باینری به متن مورد استفاده قرار می‌گیرد. برای این کار از روش کدگذاری UUE استفاده می‌کند.
IdEncoderXXE	برای کد کردن داده‌های باینری به متن مورد استفاده قرار می‌گیرد. برای این کار از روش کدگذاری XXE استفاده می‌کند.
IdEncoderQuotedPritable	برای کد کردن داده‌های بالیزی به متن مورد استفاده قرار می‌گیرد. برای این کار از روش کدگذاری Qouted Printable استفاده می‌کند.
IdDateTimeStamp	برای پردازش و تبدیل فیلدهای تاریخ در پروتکل‌های مختلف مورد استفاده قرار می‌گیرد.
IdDecoderMime	برای کدگشایی داده‌های کدگذاری شده به روش MIME مورد.استفاده قرار می‌گیرد.
IdDecoderUUE	برای کدگشایی داده‌های کدگذاری شده به روش UUE مورد استفاده قرار می‌گیرد.
IdDecoderXXE	برای کدگشایی داده‌های کدگذاری شده به روش Quoted Prinable مورد استفاده قرار می‌گیرد.
IdIpWatch	برای تعیین وضعیت IP و نگهداری سوابق آن مورد استفاده قرار می‌گیرد.
IdHL7	از پروتکل Minimal Lower Layer براساس استاندارد HL7[2] پشتیبانی می‌کند.
IdLogDebug	برای جلوگیری از برقراری ارتباط توسط یک رویداد مورد استفاده قرار می‌گیرد.
IdMailbox	برای دریافت اطلاعات مربوط به صندوقهای پستی اشخاص (برابر پروتکل IMAP4) مورد استفاده قرار می‌گیرد.
IdMessage	از پروتکل‌های POP3 ، SMTP و NNTP و MME[3] پشتیبانی می‌کند.
IdMessageDecoderMIME	از کدگذاری به روش MIME RFC-822 پشتیبانی می‌کند.
IdNetworkCalculator	برای تشخیص و تعیین آدرسهای معتبر در شبکه مورد استفاده قرار می‌گیرد.
IdSyslogMessage	از پروتکل syslog پشتیبانی می‌کند.
IdThreadComponent	برای ایجاد اجزاء سازندهٔ Visual مورد استفاده قرار می‌گیرد.
IdThreadMgrDefault	برای مدیریت Thread ها در شبکه مورد استفاده قرار می‌گیرد.
IdThreadMgrPool	برای مدیریت Thread ها در شبکه مورد استفاده قرار می‌گیرد.
IdUserManage	برای مدیریت account[4] ها در شبکه مورد استفاده قرار می‌گیرد. این جزء توسط اجزاء Indy servers مورد استفاده قرار می‌گیرد.
IdVacard	برای پردازش و مدیریت کارتهای مجازی (Virtual Card) مربوط به تجارت الکترونیکی مورد استفاده قرار می‌گیرد.

---

۱ - Cookie ها مجموعه داده‌های کوچکی هستند که در طرف سرویس‌گیرنده‌هایی که مرورگر در آنها اجرا می‌شود، نوشته می‌شوند. Cookie ها برای حفظ اطلاعات در فواصل زمانی بین جلسات کاری مرورگر مورد استفاده قرار می‌گیرند.

۲- Health Level 7 ۳- Multiple Mail Extensions Protocol

۴ - مکانیزمی که در شبکه‌ها و سیستم عامل‌های چند کاربرد برای حفظ اطلاعات کاربران مجاز استفاده می‌شود. accountها در یک شبکه توسط مدیر شبکه ایجاد شده و برای بررسی اعتبار کاربران و سیاستهای کاری (مثلاً مجوزها) مورد استفاده قرار می‌گیرند.

برگهٔ Indy Intercepts

فهرست اجزاء سازندهٔ موجود در برکه Indy Intercepts دلفی ۷ در جدول ۲۶-۳ ارائه شده است.

جدول ۲۶-۳ اجزاء سازندهٔ موجود در برگهٔ Indy Intercepts

جزء سازنده	توضیح
ISBlockCipherIntercept	برای کد بندی داده‌ها (encryption) جهت جلوگیری از دستیابی غیرمجاز (به ویژه در طی انتقال داده‌ها) و همچنین رمزگشایی داده‌ها مورد استفاده قرار می‌گیرد.
IdConnectionIntercept	برای برقراری یک ارتباط پایدار جهت انتقال داده‌ها مورد استفاده قرار می‌گیرد. به عبارت دیگر این جزء سازنده، پایداری ارتباط را گارانتی می‌نماید.
IdCompressionIntercept	برای فشرده‌سازی داده‌ها مورد استفاده قرار می‌گیرد.
IdLogDebug	برای جلوگیری از برقراری ارتباط مورد استفاده قرار می‌گیرد.
IdLogStream	برای مدیریت و کار با stream[1] ها مورد استفاده قرار می‌گیرد.

برگهٔ Indy I/O Handlers

جدول ۲۷-۳ اجزاء سازندهٔ موجود در برگهٔ Indy I/O Handlers دلفی ۷ را نشان می‌دهد.

جدول ۲۷-۳ اجزاء سازندهٔ موجود در برگهٔ Indy I/O Handlers

جزء سازنده	توضیح
IdIOHandlerSocket	امکان ایجاد روتین‌های اداره‌کنندهٔ اعمال ورودی/ خروجی برای یک Socket[2] در شبکه را فراهم می‌سازد.
IdIOHandlerStream	امکان ایجاد روتین‌های ادارهٔ کنندهٔ ورودی/ خروجی که از یک Stream استفاده می‌کنند فراهم می‌آورد.
IdIOHandlerThrottle	برای ایجاد یک روتین ورودی/ خروجی جهت بالا بردن گذردهی سیستم مورد استفاده قرار می‌گیرد.
IdServerIOHandlerSocket	امکان ایجاد روتین‌های ادارهٔ کنندهٔ اعمال ورودی/ خروجی برای یک Socket در شبکه را فراهم می‌آورد.
IdServerIOHandlerSSL	امکان ایجاد روتین‌های ورودی/ خروجی با استفاده از روش SSL[3] را فراهم می‌آورد.
IdSSLIOHandlerSocket	برای پیاده‌سازی روتین‌های ورودی/ خروجی SSL جهت Socket های شبکه مورد استفاده قرار می‌گیرد.

۱ - streamها به معنای انتقال پیوستهٔ داده‌ها، از ابتدا تا انتها با جریان یکنواخت می‌باشد. انتقال پیوستهٔ داده‌ها در اینترنت به کاربران امکان می‌دهد تا پیش از انتقال کامل یک فایل بتوانند به آن دستیابی داشته باشند.

۲ - شناسه‌ای برای یک سرویس خاص در گرهٔ خاصی از یک شبکه.

۳- Secure Socket Layer

برگهٔ Servers

اجزاء سازندهٔ موجود در برگهٔ Servers در برگیرندهٔ فایل‌های ویژه‌ای هستند که واسط یک سرویس‌دهندهٔ اتوماسیون[1] را مشخص می‌کنند. این اجزاء، از برنامه‌های کاربردی موجود در نرم‌افزار office شرکت میکروسافت می‌باشند که با استفاده از آنها می‌توان به ارتباط بین برنامه‌ای نائل شد. توجه داشته باشید که بسیاری از برنامه‌های کاربردی میکروسافت از جمله Word ، Access ، PowerPoint ، Excel ، FrontPage ، Outlook و ... سرویس‌دهنده‌های اتوماسیون می‌باشند، بدین معنی که می‌توانید از این برنامه‌ها به عنوان سرویس‌دهنده‌های برنامه‌های کاربردی خود استفاده کنید.

روشهای مختلفی برای استفاده از سرویس‌دهنده‌های اتوماسیون وجود دارد که آسانترین آنها استفاده از اجزاء سازندهٔ موجود در تب Servers است.

فهرست مهمترین وکاربردی‌ترین اجزاء سازندهٔ موجود در این تب را در جدول ۲۸-۳ مشاهده خواهیدنمود. (برای اطلاعات بیشتر در خصوص سایر اجزاء سازندهٔ موجود در این برگهٔ به راهنمای دلفی مراجعه کنید.)

جدول ۲۸-۳ اجزاء سازندهٔ موجود در برگهٔ Servers

جزء سازنده	توضیح
WordApplication	سرویس‌دهندهٔ اتوماسیون word می‌باشد.
WordDocument	به عنوان رابط سندهای word عمل می‌کند.
WordFont	امکان مدیریت فونت‌ها (تغییر اندازه، شکل و رنگ) را فراهم می‌آورد.
WordParagraphFormat	امکان مدیریت پاراگراف‌ها (تورفتگی متون و ...) را فراهم می‌آورد.
WordLettercontent	امکان ایجاد و مدیریت برنامه‌ها (فهرست گیرندگان، رونوشتها و ...) را فراهم می‌آورد.
Binder	از افزودن سندهای برنامه‌های کاربردی office پشتیبانی می‌کند.
ExcelQueryTable	امکان بازیابی داده‌ها از یک Recordset خارجی را فراهم می‌سازد.
ExcelApplication	سرویس‌دهندهٔ اتوماسیون Excel می‌باشد.
ExcelChart	از رسم نمودار پشتیبانی می‌کند.
Excelworksheet	از کاربرگهٔای Excel پشتیبانی می‌کند.
Excelworkbook	امکان نمایش کاربرگهٔای یک فایل را فراهم می‌آورد.
Exceloleobject	به عنوان یک رابط برای یک شیء OLE عمل می‌نماید.
AccessHyperLink	نمایانگر روابط HyperLink در Access می‌باشد.
AccessForm	نمایانگر رابط Form در Access می‌باشد. از این رابط جهت طراحی رابط‌های گرافیکی برای کاربران استفاده می‌شود.
AccessReport	نمایانگر رابط Report در Access می‌باشد.
AccessRefrencess	امکان مراجعه به کتابخانهٔ خارجی Access را فراهم می‌سازد.
Class	امکان پیاده‌سازی یک کلاس Visual Basic را فراهم می‌آورد.

۱- Automation

جدول ۲۸-۳ ادامه

جزء سازنده	توضیح
PowerPointApplication	سرویس‌دهندهٔ اتوماسیون PowerPoint می‌باشد.
PowerPointSlide	نمایانگر یک Slide در PowerPoint می‌باشد.
PowerPointPresentation	نمایانگر فایل نمایشی ساخته شده از Slide‌ها در PowewrPoint می‌باشد.
OutlookApplication	سرویس‌دهندهٔ اتوماسیون outlook می‌باشد.

اجزاء سازندهٔ Visual و NonVisual

اجزاء سازندهٔ موجود در دلفی به دو نوع Visual و NonVisual تقسیم‌بندی می‌شوند. هر دو نوع برای آماده‌سازی رابط کاربر مورد استفاده قرار می‌گیرند و در زمان طراحی برنامه‌های کاربردی قابل رؤیت می‌باشند. اما اجزاء سازندهٔ NonVisual در زمان اجرای برنامه، قابل رؤیت نمی‌باشند. از نوع Visual می‌توان به عنوان نمونه اجزاء سازندهٔ Button ، Edit و ... و از نوع Non Visual اجزاء سازندهٔ OpenDialog ، Timer ، MainMenu و ... را نام برد.

یادداشت

تاکنون اجزاء سازندهٔ بسیاری را بررسی کردید اما هنوز به آنها نزدیک نشده‌اید. همانگونه که در ابتدای این فصل بیان شد، موارد مهم دیگری نظیر خواص، متدها و ... در مورد اجزاء سازنده وجود دارند که باید به تفصیل بررسی شوند. ما در این فصل تنها به ارائه فهرست اجزاء سازندهٔ موجود در دلفی ۷ به همراه شرح مختصری از آنها اکتفا نمودیم و در ادامهٔ فصول این کتاب به آماده‌سازی و تهیه برنامه‌های متنوع کاربردی خواهیم پرداخت. شما می‌توانید از این فصل به عنوان دایرة المعارف اجزاء سازندهٔ موجود در دلفی ۷ استفاده کنید.

▲

خلاصه

در این فصل با VCL و موارد کاربرد آن آشنا شدید. همانطور که ملاحظه کردید اجزاء سازندهٔ موجود در دلفی ۷ برای طراحی برنامه‌های کاربردی بی‌نظیر می‌باشند. شیءگرایی، قابلیت گسترش و تنوع این اجزاء سازنده، تولید نرم‌افزارهای خوب را سهولت می‌بخشد و کیفیت خارق‌العاده‌ای را به برنامه‌های کاربردیتان می‌افزاید.

فصل چهارم

پیام‌ها در دلفی

در این فصل می‌خوانید

- **مفاهیم و اصول اولیه پیام‌ها در ویندوز**
- **انواع پیام‌ها**
- **نحوهٔ کارکرد پیام‌ها**
- **پیام‌های دلفی**
- **پیام‌های غیراستاندارد**
- **روابط بین پیام‌ها و رویدادها**

یک پیام در حقیقت یک فرمان است. در دلفی برای هر جزء سازنده VCL چندین پیام تعریف می‌شود. به عنوان یک برنامه‌نویس دلفی، لازم است تا آشنایی ویژه‌ای با پیام‌های موجود در دلفی داشته باشید و از این طریق گام‌های مؤثری را در کیفیت محصولات نرم‌افزاری خود بردارید.

پیام‌ها و رویدادها در دلفی ارتباط و وابستگی خاصی با یکدیگر دارند و از این رو یادگیری مفاهیم بنیادی پیام‌ها و نحوهٔ به کارگیری آنها از ضروریات حرفه‌ای برنامه‌نویسان دلفی است.

در این فصل قصد داریم شما را با این مفاهیم و همچنین نحوهٔ به کارگیری و مدیریت پیام‌ها در دلفی آشنا سازیم.

تذکر

آنچه در این فصل ارائه می‌گردد، مربوط به اجزاء سازندهٔ VCL است. مدیریت پیام‌ها در اجزاء سازنده CLX در فصل‌های آتی همین کتاب ارائه گردیده است.

پیام چیست؟

یک پیام در حقیقت یک فرمان است که توسط سیستم عامل به برنامهٔ کاربردی ارسال می‌گردد. به طور مثال فشردن دکمهٔ ماوس، بزرگ کردن اندازهٔ پنجرهٔ یک برنامه و ... در یک برنامهٔ کاربردی مبتنی بر

ویندوز، باعث ارسال یک پیام از طرف ویندوز به برنامهٔ کاربردی خواهد شد.

ساختار یک پیام همچون ساختار یک رکورد بوده و حاوی اطلاعاتی نظیر نوع رویداد، مشخصات مورد لزوم هر پیام و سایر اطلاعات ضروری آنهاست. به عنوان مثال در یک پیام مرتبط با فشردن دکمهٔ ماوس علاوه بر اطلاعات ضروری اطلاعاتی نظیر مختصات ماوس در صفحه نیز وجود دارد.

پیام‌ها در دلفی از یک ساختار عمومی پیروی می‌کنند. رکورد هر پیام (ساختار پیامی که از طرف سیستم عامل ویندوز به سوی برنامهٔ کاربردی ارسال می‌گردد) در دلفی TMsg نام داشته و در یونیت windows قرار دارد. این رکورد به صورت زیر تعریف شده است.

```
type
  TMsg = packed record
    hwnd: HWND;      // the handle of the Window for which the message
                     // is intended
    message: UINT;   // the message constant identifier
    wParam: WPARAM; // 32 bits of additional message-specific information
    lParam: LPARAM; // 32 bits of additional message-specific information
    time: DWORD;     // the time that the message was created
    pt: TPoint;      // Mouse cursor position when the message was created
  end;
```

hwnd این عنصر معرف دستگیرهٔ پنجره‌ای است که پیام مربوطه را نمایش می‌دهد. (منظور از پنجره کادرهای مکالمه، کادرهای ویرایشی، پنجره‌های ویندوز و ... است)

message این عنصر حاوی یک مقدار ثابت بوده و به محتوای پیام اشاره دارد.

wParam این عنصر حاوی اطلاعات یک پیام و یا مشخصات سایر پنجره‌های مرتبط با پیام‌ها می‌باشد.

LParam این عنصر اشاره‌گری به داده‌های موجود در حافظه می‌باشد. این داده‌ها با نوع و محتوای پیام‌ها ارتباط دارند.

انواع پیام‌ها

مقادیر ثابت و منحصر به فردی در توابع API ویندوز برای هر پیام تعریف شده است. این مقادیر ثابت توسط پارامتر Message از رکورد TMsg مشخص می‌شوند. تعدادی از پیام‌های متداول ویندوز به همراه مقادیر ثابت آنها در جدول ۱-۴ ارائه شده است.

جدول ۱-۴ پیام‌های متداول در ویندوز

پیام	مقدار ثابت معرف پیام	توضیح
WM_Activate	$0016	این پیام فعال بودن یا غیرفعال بودن یک پنجره را اعلام می‌کند.
WM_Char	$0102	این پیام، خود معرف ارسال پیام‌های WM_KeyUp و WM_KeyDown می‌باشد که در هنگام فشردن کلیدها بر روی صفحه کلید به وجود می‌آیند.
WM_Close	$0010	نمایانگر پایان کار پیام می‌باشد.

جدول ۱-۴ ادامه

پیام	مقدار ثابت معرف پیام	توضیح
WM_KeyDown	$0100	این پیام نمایانگر فشرده شدن یک کلید بر روی صفحه کلید می‌باشد.
WM_KeyUp	$0101	این پیام نمایانگر آزادسازی یک کلید بر روی صفحه کلید می‌باشد.
WM_LButtonDown	$0201	این پیام نمایانگر فشرده شدن دکمهٔ سمت چپ ماوس می‌باشد.
WM_MouseMove	$0200	این پیام معرف حرکت ماوس بر روی صفحهٔ نمایش می‌باشد.
WM_PAINT	$000F	این پیام نمایانگر تعریف مجدد سطوح برنامهٔ کاربردی می‌باشد. (برای اطلاعات بیشتر به راهنمای دلفی مراجعه کنید).
WM_Timer	$0113	این پیام معرف بروز رویدادهای زمان‌سنج سیستم می‌باشد.
WM_Quit	$0012	این پیام نمایانگر اعلام کاربر برای پایان کار برنامهٔ کاربردی است.

پیام‌ها چگونه کار می‌کنند؟

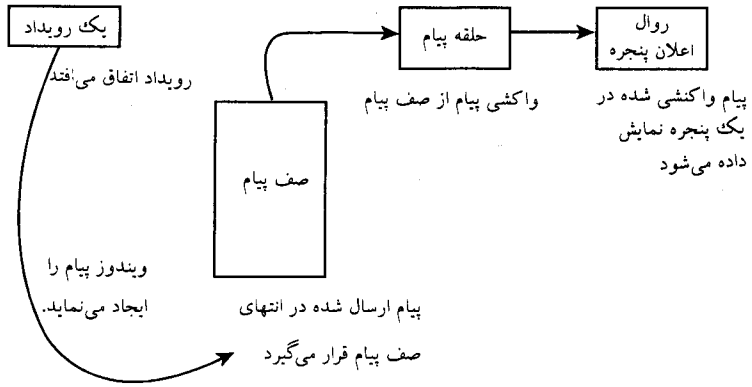
سیستم مدیریت پیام‌ها در برنامه‌های کاربردی تحت ویندوز از سه جزء به شرح زیر تشکیل شده است.

- صف پیام: سیستم عامل ویندوز برای هر برنامهٔ کاربردی یک صف پیام ایجاد می‌نماید. پیام‌ها به ترتیب ورود به صف به برنامهٔ کاربردی در حال اجرا ارسال خواهند شد.
- حلقهٔ پیام: این حلقه دارای مکانیزمی است که از آن برای واکشی پیام‌ها از صف پیام و ارسال آن به پنجرهٔ مربوط به نمایش پیام‌ها استفاده می‌شود.
- روال اعلان پنجره: این روال وظیفه دریافت پیام از حلقه پیام و نمایش محتوای آن را به عهده دارد. پس از پردازش و کار بر روی پیام از طرف برنامه کاربردی، این روال یک مقدار عددی متناسب با این پردازش را به سیستم عامل ویندوز ارسال می‌دارد.

پردازش پیام‌ها در ویندوز نیز براساس ۵ مرحله انجام می‌پذیرد. ضمناً توجه داشته باشید که هر پیام با توجه به یک رویداد به وجود خواهد آمد. مراحل پردازش پیام‌ها در ویندوز به شرح زیر است.

۱- یک رویداد در سیستم روی می‌دهد.
۲- ویندوز این رویداد را به یک پیام تبدیل نموده و آن را در صف پیام قرار می‌دهد.
۳- برنامهٔ کاربردی این پیام را از صف پیام واکشی نموده و آن را در رکورد TMsg قرار می‌دهد.
۴- برنامهٔ کاربردی پیام ساخته شده را در یک پنجرهٔ متناسب با آن به نمایش در می‌آورد.
۵- کاربر عملی را برای پاسخ به پیام ظاهر شده انجام داده و کار خاتمه می‌یابد.

- مراحل ۳ و ۴ مربوط توسط مکانیزم موجود در حلقه پیام انجام می‌شود. نمودار مربوط به این مراحل در شکل ۱-۴ نمایش داده شده است.



شکل ۱-۴ سیستم مدیریت پیام‌ها در ویندوز

پیام‌ها در دلفی

در این قسمت قصد داریم پیام‌ها مربوط به اجزاء سازنده VCL در دلفی را تشریح نمائیم. لازم است یادآور شویم که سه جزء صف پیام، حلقه پیام و روال اعلان پنجره در تمام محیط‌های نرم‌افزاری تحت ویندوز عمومیت دارد و دلفی نیز یکی از این نرم‌افزارهاست.

حلقهٔ پیام در داخل یونیت فرم اجزاء سازندهٔ VCL قرار دارد. دلفی از یک رکورد خاص به نام TMessage برای ذخیره‌سازی اطلاعات TMsg استفاده می‌کند. ساختار این رکورد در زیر ارائه شده است.

```
type
  TMessage = record
    Msg: Cardinal;
    case Integer of
      0: (
        WParam: Longint;
        LParam: Longint;
        Result: Longint);
      1: (
        WParamLo: Word;
        WParamHi: Word;
        LParamLo: Word;
        LParamHi: Word;
        ResultLo: Word;
        ResultHi: Word);
  end;
```

ملاحظه می‌کنید که در ساختار TMessage فیلد جدیدی به نام Result وجود دارد. همانطورکه در ادامهٔ این فصل توضیح خواهیم داد از این فیلد برای ارجاع مقادیر مرتبط با پاسخ داده شده به پیام‌ها استفاده می‌شود.

رکورد حاوی اطلاعات اختصاصی پیام‌ها

رکورد TMessage تنها حاوی اطلاعات ضروری هر پیام است. برخی از پیام‌ها حاوی یکسری اطلاعات اختصاصی بوده لذا می‌بایست رکوردی برای ذخیره‌سازی این اطلاعات نیز وجود داشته باشد. نمونه‌ای از این پیام‌ها، پیام‌های مربوط به رویدادهای ماوس بر روی صفحهٔ نمایش می‌باشند. به عنوان مثال رکورد مربوط به اینگونه پیام‌ها را در زیر مشاهده می‌کنید.

```
type
  TWMMouse = packed record
    Msg: Cardinal;
    Keys: Longint;
    case Integer of
      0: (
        XPos: Smallint;
        YPos: Smallint);
      1: (
        Pos: TSmallPoint;
        Result: Longint);
  end;
```

سایر پیام‌ها مرتبط با رویدادهای ماوس نظیر WM_LBUTTONDOWN و WM_RBUTTONUP نیز مشابه با این ساختار تعریف خواهند شد.

```
TWMRButtonUp = TWMMouse;
TWMLButtonDown = TWMMouse;
```

دستکاری پیام‌ها

دستکاری یا پردازش پیام‌ها به مفهوم تعیین عملکرد یک برنامهٔ کاربردی در زمان قبل و بعد از اعلان پیام‌هاست. در حالت عادی این کار به وسیلهٔ "روال اعلان پنجره" انجام می‌پذیرد، اما هنگامی که چندین پیام در یک زمان اعلام شوند، برنامه‌نویس یا حتی کاربر برنامهٔ کاربردی را با شکل مواجه می‌سازند.

دلفی به نسبت سایر زبان‌های برنامه‌سازی، روشهای بهینه‌تری را برای مدیریت پیام‌های چندگانه ارائه نموده است. در دلفی هر پیام دارای یک روال مختص به خود می‌باشد که از آن طریق اعلان پیام و همچنین روش پاسخ‌دهی به آن را کنترل می‌کند.

نوع و نحوهٔ دستکاری پیام‌ها به وسیلهٔ روال‌ها تعریف می‌گردند. اینگونه روال‌ها در دلفی تحت الگویی خاص پیاده‌سازی می‌شوند. این الگو به شرح زیر می‌باشد:

- روال موردنظر به صورت متدی از شیء موجود تعریف می‌شود.
- یک متغیر از نوع TMessage (با سایر رکوردهای خاص مربوط به پیام‌ها) در این روال به کار گرفته شود.
- از یک مقدار ثابت برای هدایت پیام در اینگونه روال‌ها استفاده شود.

به عنوان مثال به روال زیر که برای WM_PAINT نوشته شده است توجه کنید.

```
Procedure WMPaint (Var Msg: TWMPaint); message WM_PAINT;
```

اکنون برای دستکاری این پیام روالی را تعریف می‌کنیم که تنها با به صدا در آوردن بلندگوی سیستم وقوع این پیام را گزارش کند. (پس از ایجاد فرم اولیه یک پروژه، قطعه کد زیر را در آن تعریف کنید.)

```
Procedure TForm1.WMPaint (Var Msg: TWMPaint);
begin
Beep;
inherited;
end;
```

از عبارت **inherited** در اینجا برای ارسال پیام به کلاس والد شیء (WM_PAINT) استفاده شده است. در لیست ۱-۴ برنامه‌ای برای پردازش پیام WM_PAINT ارائه شده است. برای پیاده‌سازی آن یک فرم جدید ایجاد کرده و کد موجود در لیست ۱-۴ را عیناً در آن وارد نمائید.

لیست ۱-۴ GetMess مثالی از پردازش یک پیام

```
unit GMMain;

interface

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

type
  TForm1 = class(TForm)
  private
    procedure WMPaint(var Msg: TWMPaint); message WM_PAINT;
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.WMPaint(var Msg: TWMPaint);
begin
  MessageBeep(0);
  inherited;
end;

end.
```

تخصیص فیلد Result برای پیام‌ها

ساختار برخی از پیام‌ها به گونه‌ای است که پس از پردازش آنها، یک مقدار به عنوان نتیجهٔ انجام آنها برگردانده می‌شود. در این حالت سیستم عامل براساس مقدار برگردانده شده، عملیات مورنظر کاربر را به انجام می‌رساند. در این قسمت مثالی از پیام WM_CTLCOLOR را مطرح می‌کنیم. هنگامی که این پیام پردازش می‌شود، دستگیرهٔ پنجرهٔ موردنظر (جهت رنگ‌آمیزی) به عنوان مقدار خروجی این پیام برای سیستم عامل ارسال می‌شود. منظور از پنجره در اینجا پنجره یک کادر محاوره‌ای یا پنجرهٔ یک کادر کنترلی در ویندوز است.

به قطعهٔ کد زیر توجه کنید.

```
procedure TForm1.WMCtlColor(var Msg: TWMCtlColor);
var
  BrushHand: hBrush;
begin
  inherited;
  { Create a brush handle and place into BrushHand variable }
  Msg.Result := BrushHand;
end;
```

رویداد OnMessage

یک راه دیگر برای پردازش و به کارگیری پیام‌ها، استفاده از رویداد OnMessage است که در TApplication قرار دارد. OnMessage را در لیست فرم اصلی برنامه‌تان خواهید یافت. رویداد Application.OnMessage از نوع TMessageEvent و با الگوی اولیه زیر تعریف می‌شود:

```
Procedure SomeObject.AppMessageHandler (Var Msg: Tmsg; var Handled:
    Boolean);
```

برای استفاده از رویداد OnMessage می‌بایست جزء سازندهٔ TApplicationEvent را از قسمت Additional انتخاب کرده و آن را به فرم اصلی برنامه‌تان بیافزائید. توجه داشته باشید که رویداد OnMessage متناسب با پیام‌های موجود در صف پیام انجام وظیفه می‌نماید و هیچ پردازشی بر روی پیام‌هایی که مستقیماً توسط سیستم عامل ارسال می‌شوند، ندارد.

برای آشنایی بیشتر با به کارگیری رویداد OnMessage، به قطعه کد زیر توجه نمائید.

```
var
  NumMessages: Integer;

procedure TForm1.ApplicationEvents1Message(var Msg: tagMSG;
  var Handled: Boolean);
begin
  Inc(NumMessages);
  Handled := False;
end;
```

ارسال پیام‌های شخصی

برخی از برنامه‌نویسان تصور می‌کنند که تنها سیستم عامل ویندوز قابلیت ارسال پیام‌ها به برنامه‌های کاربردی مبتنی بر خود را داراست. (این کار در سیستم عامل ویندوز توسط توابع API انجام می‌پذیرد.)

اما اکثر برنامه‌نویسان توابع شخصی را برای ارسال و کنترل پیام‌ها می‌پسندند. در دلفی روشهای مختلفی برای انجام این کار وجود دارد. لازم به ذکر است که توابع موجود در دلفی به صورت مستقل از توابع API ویندوز پیاده‌سازی شده و به صورتی مستقل هم کار می‌کنند. متد ()Perform و توابع ()SendMessage و ()PostMessage نمونه‌ای از توابع دلفی برای ارسال و کنترل پیام‌های شخصی می‌باشند.

متد ()Perform

متد ()Perform از متدهای طراحی شده برای جزء سازنده TControl بوده و شما را قادر می‌سازد تا پیام‌های خود را به اشیاء یا فرم‌های برنامه‌تان ارسال نمائید. متد ()Perform به صورت زیر تعریف می‌شود.

```
function TControl.Perform(MSg: Cardinal; WParam, LParam: Longint): Longint;
```

توابع ()SendMessage و ()PostMessage

همانطور که می‌دانید محیط‌های مبتنی بر ویندوز از تنوع بالایی برخوردارند. در یک برنامهٔ کاربردی تحت ویندوز ممکن است از اشیاء و پنجره‌هایی با الگوهای متفاوت استفاده شود. متد ()Perform عملیات ارسال پیام برای اشیاء، فرم‌ها و پنجره‌هایی را انجام می‌دهد که در دلفی طراحی و پیاده‌سازی شده باشند. اگر می‌خواهید پیامی را برای اشیاء و پنجره‌های طراحی شده توسط سایر زبان‌های برنامه‌سازی ارسال دارید، لازم است تا از توابع ()SendMessage و ()PostMessage استفاده کنید. اما فراموش نکنید که در این روش نیز لازم است تا به دستگیرهٔ پنجرهٔ موردنظر، برای نمایش پیام دسترسی داشته باشید.

()SendMessage و ()PostMessage شباهت الگویی زیادی با یکدیگر داشته اما نحوهٔ کارآیی آنها متفاوت است.

()SendMessage تابعی است که از آن جهت ارسال پیام به "روال اعلان پنجره" استفاده می‌شود و ()PostMessage تابعی است که وظیفهٔ ارسال و دریافت پیام از "صف پیام" را به عهده دارد. تعریف این روال‌ها در زیر ارائه شده است.

```
function SendMessage(hWnd: HWND; Msg: UINT; wParam: WPARAM;
  lParam: LPARAM): LRESULT; stdcall;
function PostMessage(hWnd: HWND; Msg: UINT; wParam: WPARAM;
  lParam: LPARAM): BOOL; stdcall;
```

- hwnd : دستگیرهٔ مربوط به پنجره‌ای که با پیام موردنظر در ارتباط است.
- Msg : شماره شناسایی پیام توسط این پارامتر مشخص می‌شود.
- WParam : این پارامتر حاوی برخی از اطلاعات اختصاصی پیام موردنظر است. اندازه حافظه‌ای این پارامتر برابر ۴۲ بیت است.
- LParam : این پارامتر حاوی برخی از اطلاعات اختصاصی پیام موردنظر است. اندازه حافظه‌ای این پارامتر برابر ۴۲ بیت است.

نکته

مقادیر خروجی توابع ()SendMessage و ()PostMessage با هم متفاوت خواهند بود. خروجی تابع ()SendMessage مقدار Result است که نمایانگر شروع پردازش پیام می‌باشد. اما خروجی تابع ()PostMessage یک مقدار منطقی است که وضعیت درج پیام در صف پیام را نشان می‌دهد.

پیام‌های آگاهی دهنده

پیام‌های آگاهی دهنده، آن دسته از پیام‌هایی هستند که تنها بر وضعیت کارکرد خود شیء یا پنجره دلالت داشته و آن را کنترل می‌کنند. بدین معنی که این گونه پیام‌ها به وسیلهٔ خود اشیاء صادر و به وسیلهٔ همان اشیاء پردازش می‌شود.

اینگونه پیام‌ها تنها منحصر به اجزاء کنترلی موجود در ویندوز، نظیر کادرهای ویرایشی، دکمه‌ها، combo boxها و list boxها است.

لیست برخی از پیام‌های آگاهی دهنده به همراه مفاهیم آنها در جدول ۲-۴ ارائه شده است.

جدول ۲-۴ لیست برخی از پیام‌های آگاهی دهنده

پیام	معنی و مفهوم
پیام‌های مربوط به دکمه‌ها	
BN_CLICKED	کاربر دکمهٔ موجود بر روی فرم را فشرده است.
BN_DISABLE	دکمه به صورت غیرفعال در آمده است.
BN_DOUBLECLICKED	کاربر دکمهٔ موجود بر روی فرم را برای دو مرتبه متوالی فشرده است.
BN_HILITE	کاربر اشاره‌گر ماوس را بر روی دکمه برده است (دکمه در وضعیت Highlight است)
BN_UNHILITE	حالت Highlight می‌بایست برداشته شود.
پیام‌های مربوط به Highlightها	
CBN_CLOSEUP	Listbox مربوط به Combo box ، بسته شده است.
CBN_DBLCLK	کاربر یک رشتهٔ کاراکتری خاص از رشته‌های موجود در Combo box را انتخاب نموده است.
CBN_DROPDOWN	List box مربوط به Combo box در وضعیت نمایش قرار دارد.

جدول ۲-۴ ادامه

پیام	معنی و مفهوم
CBN_ERRSPACE	Combo box موردنظر در حافظه موجود سیستم جای نمی‌گیرد.
CBN_SELCHANGE	یک عنوان جدید از لیست Combo box انتخاب شده است.
CBN_SELENDOK	انتخاب کاربر غیرمعتبر تشخیص داده شده است.
پیام‌های مربوط به کادرهای ویرایشی	
EN_CHANGE	کادر نمایشی، پس از تغییر متن، بروز در آورده می‌شود.
EN_ERRSPACE	کادر ویرایشی موردنظر در حافظه موجود سیستم جای نمی‌گیرد.
EN_HSCROLL	کاربر اشاره‌گر ماوس را بر روی نوار پیمایش افقی کادر قرار داده است.
EN_VSCROLL	کاربر اشاره‌گر ماوس را بر روی نوار پیمایش عمودی کادر قرار داده است.
پیام‌های مربوط به List boxها	
LBN_DBLCLK	کاربر یک رشته کاراکتری خاص از رشته‌های موجود در List box را انتخاب نموده است.
LBN_ERRSPACE	List box موردنظر در حافظه موجود سیستم جای نمی‌گیرد.

پیام‌های داخلی و اجزاء سازنده VCL

علاوه بر آنچه که در بالا ذکر شده، اجزاء سازنده VCL دارای یکسری پیام‌های داخلی نیز می‌باشند. این پیام‌های داخلی بر خصوصیات و ویژگی‌های اجزاء سازنده VCL نظیر رنگ، وضعیت نمایش و ... دلالت دارند.

پیام‌هایی که به وسیلهٔ کاربران تعریف می‌شوند

تابحال به این سؤال فکر کرده‌اید که "چرا گاهی اوقات به جای فراخوانی یک روال، از پیام‌ها استفاده می‌کنیم؟" جواب بسیار ساده است. یکی از مزایای به کارگیری پیام‌ها، طبیعت چند شکلی آنهاست. این خاصیت چند شکلی به پیام‌ها، قابلیت انعطاف منحصر به فردی اعطا می‌کند که بدون نیاز به دانستن ساختار پارامترهای ورودی محیط، می‌توان آنها را کنترل و اجرا نمود. یکی دیگر از دلایل انتخاب پیام‌ها، وجود متدهای متنوع برای دستکاری آنهاست. آنچه که ذکر شد تنها به دلایلی از فواید به کارگیری پیام‌ها اشاره داشت، اما در یک کلام باید گفت که استفاده از پیام‌ها در بهینه‌سازی قابلیت خوانایی، انعطاف و قابلیت اعتماد برنامه‌هایتان کارآیی به سزایی دارد.

پیام‌های داخلی یک برنامهٔ کاربردی

طراحی یک برنامهٔ کاربردی به صورتی که یکسری پیام‌ها را در داخل خود ارسال و کنترل نماید بسیار ساده است. برای این کار از توابع SendMessage(), PostMessage() و متد Perform() استفاده می‌شود. پارامتر Msg توابع فوق در این حالت در بازهٔ WM_USER + 100 تا $7FFF قرار می‌گیرد. یعنی روش به کارگیری توابع SendMessage() و PostMessage() مطابق با الگوی ارائه شده زیر خواهد بود.

```
const
 SX_MYMESSAGE = WM_USER + 100;

begin
  SomeForm.Perform(SX_MYMESSAGE, 0, 0);
  { or }
  SendMessage(SomeForm.Handle, SX_MYMESSAGE, 0, 0);
  { or }
  PostMessage(SomeForm.Handle, SX_MYMESSAGE, 0, 0);
  .
  .
  .
end;
```

که در این صورت برای دستکاری و پردازش پیام‌ها در یک فرم از الگویی مشابه با الگوی زیر استفاده خواهد شد.

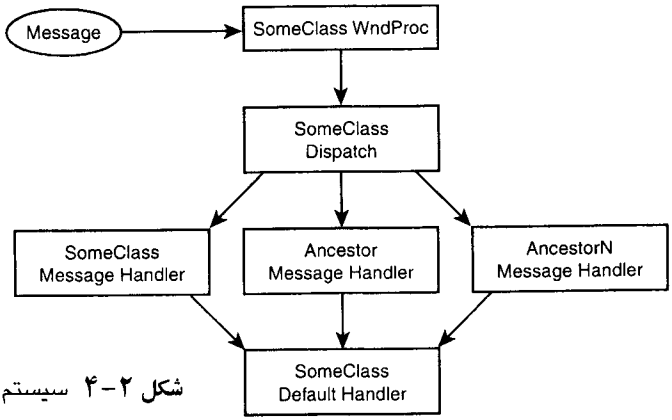
```
TForm1 = class(TForm)
  .
  .
  .
private
  procedure SXMyMessage(var Msg: TMessage); message SX_MYMESSAGE;
end;

procedure TForm1.SXMyMessage(var Msg: TMessage);
begin
  MessageDlg('She turned me into a newt!', mtInformation, [mbOk], 0);
end;
```

ارسال پیام بین چند برنامه کاربردی

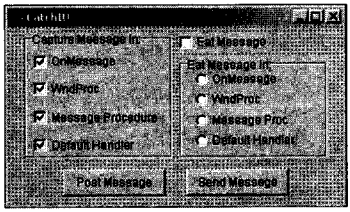
چنانچه قرار باشد مابین چند برنامهٔ کاربردی تحت ویندوز عملیات ارسال و دریافت پیام‌ها را انجام دهید، بهترین روش استفاده از توابع API ویندوز است. آن هم به این دلیل که ممکن است هر کدام از برنامه‌های کاربردی با زبان و محیطی خاص ایجاد شده باشند، ولی از آنجا که همگی تحت سیستم عامل ویندوز قابل اجرا می‌باشند، لذا برای توابع API نیز قابل شناسایی خواهند بود. RegisterwindowMessage() یکی از توابع API ویندوز است که از آن جهت ارسال، پردازش و کنترل پیام‌ها استفاده می‌شود.

ساختار و سیستم پردازش پیام‌ها در اجزاء سازندهٔ VCL

سیستم ارسال، توزیع، پردازش و دستکاری پیام‌ها در اجزاء سازندهٔ VCL براساس یک ساختار خاص پیاده‌سازی می‌شود. شکل ۲-۴ شما را با این ساختار آشنا می‌سازد. (به نحوهٔ توزیع پیام‌ها در این ساختار توجه کنید!).



شکل ۲-۴ سیستم ارسال و پردازش پیام‌ها در اجزاء سازندهٔ VCL



شکل ۴-۴ فرم اصلی برنامهٔ CatchIt

برای درک بیشتر این ساختار و فرآیندهای مرتبط با آن، یک برنامهٔ کاربردی مربوط به پردازش و دستکاری پیام‌ها را در این قسمت ارائه خواهیم نمود. در این برنامه به کاربر امکان داده می‌شود که روش پردازش را به میل خودش انتخاب نماید. منظور از این روشها استفاده از رویداد Application.OnMessage یا به کارگیری تابع WndProc() و یا ... می‌باشد. فرم اصلی این برنامه را در شکل ۴-۴ مشاهده می‌کنید.

همانطور که ملاحظه می‌کنید، دو دکمه به نام‌های SendMessage و postMessage بر روی فرم اصلی برنامه قرار داده شده است که در یکی از آنها از تابع SendMessage() و در دیگری از تابع PostMessage() استفاده می‌شود.

کد مربوط به رویداد Click دکمه‌های Send Message و Post Message را در زیر مشاهده می‌کنید.

```
procedure TMainForm.PostMessButtonClick(Sender: TObject);
{ posts message to form }
begin
  PostMessage(Handle, SX_MYMESSAGE, 1, 0);
end;
procedure TMainForm.SendMessButtonClick(Sender: TObject);
{ sends message to form }
begin
  SendMessage(Handle, SX_MYMESSAGE, 0, 0); // send message to form
end;
```

لیست ۲-۴ کد کامل برنامهٔ مربوط به فرم CatchIt را نشان می‌دهد.

لیست ۲-۴ برنامهٔ CIMain.Pas

```
unit CIMain;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, ExtCtrls, Menus;

const
  SX_MYMESSAGE = WM_USER;                  // User-defined message value
  MessString = '%s message now in %s.';    // String to alert user

type
  TMainForm = class(TForm)
    GroupBox1: TGroupBox;
    PostMessButton: TButton;
    WndProcCB: TCheckBox;
    MessProcCB: TCheckBox;
    DefHandCB: TCheckBox;
    SendMessButton: TButton;
    AppMsgCB: TCheckBox;
    EatMsgCB: TCheckBox;
    EatMsgGB: TGroupBox;
    OnMsgRB: TRadioButton;
    WndProcRB: TRadioButton;
    MsgProcRB: TRadioButton;
    DefHandlerRB: TRadioButton;
    procedure PostMessButtonClick(Sender: TObject);
    procedure SendMessButtonClick(Sender: TObject);
    procedure EatMsgCBClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure AppMsgCBClick(Sender: TObject);
  private
    { Handles messages at Application level }
    procedure OnAppMessage(var Msg: TMsg; var Handled: Boolean);
    { Handles messages at WndProc level }
    procedure WndProc(var Msg: TMessage); override;
    { Handles message after dispatch }
    procedure SXMyMessage(var Msg: TMessage); message SX_MYMESSAGE;
    { Default message handler }
    procedure DefaultHandler(var Msg); override;
  end;

var
  MainForm: TMainForm;

implementation
```

لیست ۲-۴ ادامه

```
{$R *.DFM}

const
  // strings which will indicate whether a message is sent or posted
  SendPostStrings: array[0..1] of String = ('Sent', 'Posted');

procedure TMainForm.FormCreate(Sender: TObject);
{ OnCreate handler for main form }
begin
  // set OnMessage to my OnAppMessage method
  Application.OnMessage := OnAppMessage;
  // use the Tag property of checkboxes to store a reference to their
  // associated radio buttons
  AppMsgCB.Tag := Longint(OnMsgRB);
  WndProcCB.Tag := Longint(WndProcRB);
  MessProcCB.Tag := Longint(MsgProcRB);
  DefHandCB.Tag := Longint(DefHandlerRB);
  // use the Tag property of radio buttons to store a reference to their
  // associated checkbox
  OnMsgRB.Tag := Longint(AppMsgCB);
  WndProcRB.Tag := Longint(WndProcCB);
  MsgProcRB.Tag := Longint(MessProcCB);
  DefHandlerRB.Tag := Longint(DefHandCB);
end;

procedure TMainForm.OnAppMessage(var Msg: TMsg; var Handled: Boolean);
{ OnMessage handler for Application }
begin
  // check to see if message is my user-defined message
  if Msg.Message = SX_MYMESSAGE then
  begin
    if AppMsgCB.Checked then
    begin
      // Let user know about the message.  Set Handled flag appropriately
      ShowMessage(Format(MessString, [SendPostStrings[Msg.WParam],
        'Application.OnMessage']));
      Handled := OnMsgRB.Checked;
    end;
  end;
end;

procedure TMainForm.WndProc(var Msg: TMessage);
{ WndProc procedure of form }
var
  CallInherited: Boolean;
begin
  CallInherited := True;           // assume we will call the inherited
  if Msg.Msg = SX_MYMESSAGE then   // check for our user-defined message
```

لیست ۲-۴ ادامه

```
 begin
   if WndProcCB.Checked then       // if WndProcCB checkbox is checked...
   begin
     // Let user know about the message.
     ShowMessage(Format(MessString, [SendPostStrings[Msg.WParam],
       'WndProc']));
     // Call inherited only if we are not supposed to eat the message.
     CallInherited := not WndProcRB.Checked;
   end;
 end;
 if CallInherited then inherited WndProc(Msg);
end;

procedure TMainForm.SXMyMessage(var Msg: TMessage);
{ Message procedure for user-defined message }
var
  CallInherited: Boolean;
begin
  CallInherited := True;           // assume we will call the inherited
  if MessProcCB.Checked then       // if MessProcCB checkbox is checked
  begin
    // Let user know about the message.
    ShowMessage(Format(MessString, [SendPostStrings[Msg.WParam],
      'Message Procedure']));
    // Call inherited only if we are not supposed to eat the message.
    CallInherited := not MsgProcRB.Checked;
  end;
  if CallInherited then Inherited;
end;

procedure TMainForm.DefaultHandler(var Msg);
{ Default message handler for form }
var
  CallInherited: Boolean;
begin
  CallInherited := True;           // assume we will call the inherited
  // check for our user-defined message
  if TMessage(Msg).Msg = SX_MYMESSAGE then   begin
    if DefHandCB.Checked then      // if DefHandCB checkbox is checked
    begin
      // Let user know about the message.
      ShowMessage(Format(MessString,
        [SendPostStrings[TMessage(Msg).WParam], 'DefaultHandler']));
      // Call inherited only if we are not supposed to eat the message.
      CallInherited := not DefHandlerRB.Checked;
```

لیست ۲-۴ ادامه

```
      end;
    end;
    if CallInherited then inherited DefaultHandler(Msg);
  end;
  procedure TMainForm.PostMessButtonClick(Sender: TObject);
  { posts message to form }
  begin
    PostMessage(Handle, SX_MYMESSAGE, 1, 0);
  end;

  procedure TMainForm.SendMessButtonClick(Sender: TObject);
  { sends message to form }
  begin
    SendMessage(Handle, SX_MYMESSAGE, 0, 0); // send message to form
  end;

  procedure TMainForm.AppMsgCBClick(Sender: TObject);
  { enables/disables proper radio button for checkbox click }
  begin
    if EatMsgCB.Checked then
    begin
      with TRadioButton((Sender as TCheckBox).Tag) do
    begin
      Enabled := TCheckbox(Sender).Checked;
      if not Enabled then Checked := False;
    end;
  end;
end;

procedure TMainForm.EatMsgCBClick(Sender: TObject);
{ enables/disables radio buttons as appropriate }
var
  i: Integer;
  DoEnable, EatEnabled: Boolean;
begin
  // get enable/disable flag
  EatEnabled := EatMsgCB.Checked;
  // iterate over child controls of GroupBox in order to
  // enable/disable and check/uncheck radio buttons
  for i := 0 to EatMsgGB.ControlCount - 1 do
    with EatMsgGB.Controls[i] as TRadioButton do
    begin
```

لیست ۲-۴ ادامه

```
      DoEnable := EatEnabled;
      if DoEnable then DoEnable := TCheckbox(Tag).Checked;
      if not DoEnable then Checked := False;
      Enabled := DoEnable;
    end;
end;

end.
```

روابط بین پیام‌ها و رویدادها

حال که تقریباً همه چیز را در خصوص ورود و خروج پیام‌ها آموخته‌اید، لازم است تا مطلبی را در خصوص رویدادها ارائه نمائیم. رویدادهایی که برای اشیاء موجود در دلفی (اجزاء سازندهٔ VCL) تعریف می‌شوند تناظری یک به یک با پیام‌های موجود در ویندوز دارند. یعنی هرکدام از این رویدادها باعث فعال شدن یکی از پیام‌های موجود در سیستم عامل ویندوز می‌شوند. تناظر یک به یک این رویدادها و پیام‌ها را در جدول ۳-۴ مشاهده می‌کنید.

جدول ۳-۴ تناظر بین رویدادهای اجزاء سازندهٔ VCL و پیام‌های ویندوز

رویداد VCL	پیام ویندوز
OnActivate	WM_Activate
OnClick	WM_XButtonDown
OnCreate	WM_Create
OnDblClick	WM_XButtonDbclick
OnKeyPress	WM_KeyDown
OnKeyPress	WM_Char
OnKeyUp	WM_KeyUp
OnPaint	WM_PAINT
OnResize	WM_Size
OnTimer	WM_Timer

خلاصه

در این فصل مفاهیم اولیه و همچنین اصول کار با پیام‌ها را ارائه نمودیم. پس از پایان این فصل انتظار می‌رود که با درک مفاهیم پیام‌ها در سیستم عامل ویندوز و روابط آن با رویدادهای تعریف شده در دلفی، از پیام‌ها نیز به عنوان یکی از اجزای ضروری برنامه‌های کاربردیتان استفاده نمائید.

فصل پنجم

معماری بانک‌های اطلاعاتی در دلفی

در این فصل می‌خوانید

- **انواع بانک‌های اطلاعاتی در دلفی**
- **معماری بانک اطلاعاتی دلفی**
- **برقراری ارتباط با سرویس‌دهندهٔ بانک اطلاعاتی**
- **به کارگیری Dataset ها**
- **کار با فیلدهای بانک‌های اطلاعاتی**

در این فصل شما را با ساختار بانک‌های اطلاعاتی دلفی و روشهای دسترسی به آنها آشنا خواهیم ساخت. دلفی ابزار قدرتمندی برای کار با بانک‌های اطلاعاتی دارد. پس از به پایان رساندن این فصل خواهید دید که ابزارهای موجود در دلفی تا چه اندازه انجام کارهای مدیریتی بانک‌های اطلاعاتی را آسان می‌نمایانند.

انواع بانک‌های اطلاعاتی

هنگامی که از دلفی برای ایجاد بانک‌های اطلاعاتی استفاده می‌کنید، تعیین نوع یا مدل بانک اطلاعاتی از ابتدایی‌ترین و ضروری‌ترین اقدامات خواهد بود. در این قسمت به تشریح بانک‌های اطلاعاتی که در دلفی قابل پشتیبانی می‌باشند خواهیم پرداخت.

- BDE : این مدل از اجزاء سازنده‌ای استفاده می‌کند که براساس ساختارهای "موتور بانک اطلاعاتی بورلند" پیاده‌سازی گردیده‌اند. انجام اعمال مدیریتی بانک اطلاعاتی در این مدل عموماً به وسیله توابع API صورت می‌پذیرد.
- ADO : در این مدل از اجزاء سازندهٔ ActiveX استفاده می‌شود. شیءهای کنترلی ActiveX را می‌توان برای تولید انیمیشن و جلوه‌های چند رسانه‌ای، شیءهای محاوره‌ای و برنامه‌های کاربردی پیچیده به کار برد.
- dbExpress : در این مدل از اجزاء سازندهٔ موجود در تب dbExperss برای دستیابی به بانک‌های اطلاعاتی استفاده می‌شود. dbExpress مجموعه‌ای از راه‌اندازهایی است که ایجاد بانک‌های اطلاعاتی و انجام اعمال پردازشی آنها را سریع‌تر می‌نمایاند.

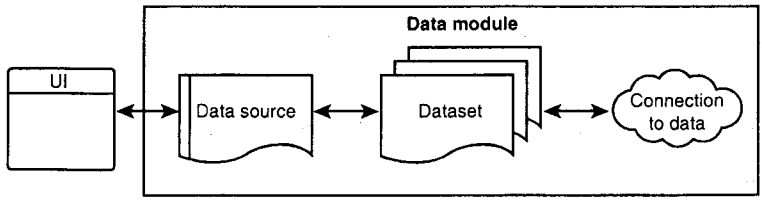
● InterBase : از این مدل جهت دسترسی به بانک‌های اطلاعاتی مبتنی بر InterBase استفاده می‌شود.

معماری بانک‌های اطلاعاتی دلفی

معماری بانک اطلاعاتی دلفی مشتمل بر اجزاء سازنده‌ای است که وظیفهٔ ذخیره‌سازی، پردازش و نهایتاً مدیریت داده‌های موجود در بانک‌ها را به عهده خواهند داشت.

شکل ۱-۵ نحوهٔ قرارگیری این اجزاء سازنده در معماری بانک اطلاعاتی دلفی را نمایش می‌دهد.

همانطور که ملاحظه کنید هستهٔ بانک اطلاعاتی دلفی از دو جزء سازندهٔ Datasource و Dataset تشکیل شده است. Datasource جزء سازنده‌ای است که وضعیت منبع داده‌ها و نحوهٔ اعمال تغییرات در آن را مشخص می‌سازد و از طریق اجزاء سازندهٔ Dataset می‌توانید به اطلاعات بانک اطلاعاتی دسترسی داشته باشید. به بیانی دیگر Datasource همان منبع داده‌ها یا بانک اطلاعاتی است که خود آن می‌تواند حاوی یک سری فایل‌های داده یا جداول باشد که این فایل‌های داده و جداول توسط اجزاء سازندهٔ Dataset قابل دسترسی می‌باشند.



شکل ۱-۵ معماری بانک اطلاعاتی دلفی

برقراری ارتباط با بانک‌های اطلاعاتی

دستیابی به یک بانک اطلاعاتی و برقراری ارتباط با آن منوط به ساختار و نوع بانک اطلاعاتی می‌باشد. بدین معنی که برای دستیابی به هر یک از انواع بانک‌های اطلاعاتی (که در ابتدای فصل به آنها اشاره نمودیم) از اجزاء سازندهٔ مربوط به خودشان استفاده خواهد شد. تعدادی از این اجزاء سازنده را در زیر معرفی می‌کنیم.

● TDataBase : از این جزء سازنده جهت برقراری ارتباط بانک‌های اطلاعاتی مبتنی بر موتور بانک اطلاعاتی دلفی (BDE) استفاده می‌شود. Dataset های تعریف شده برای این جزء سازنده عبارتند از TTable ، TQuery و TStoredProc .

● TADOConnection : از این جزء سازنده جهت برقراری ارتباط با بانک‌های اطلاعاتی مبتنی بر ADO استفاده می‌شود (همانند Microsoft Access و Microsoft SQL). Dataset های تعریف شده برای این جزء سازنده عبارتند از TADODataset ، TADOTable ، TADOQuery و TADOStoredProc .

- TSQLConnection : از این جزء سازنده برای برقراری ارتباط با بانک‌های اطلاعاتی مبتنی بر dbExpress استفاده می‌شود. Dataset های تعریف شده برای این جزء سازنده عبارتند از TSQLDataset ، TSQLTable ، TSQLQuery و TSQLStoredProc .
- TIBDatabase : از این جزء سازنده برای برقراری ارتباط با بانک‌های اطلاعاتی مبتنی بر InterBase استفاده می‌شود. Dataset های تعریف شده برای این جزء سازنده عبارتند از TIBDataset ، TIBTable ، TIBQuery و TIBStoredProc .

کار با Datasetها

هر Dataset مجموعه‌ای از سطرها و ستون‌های داده‌ای است که از این سطرها به عنوان "رکورد" و از این ستون‌ها به عنوان "فیلد" یاد می‌شود. Dataset می‌تواند به صورت یک Table و یا Query تعریف گردد. Table ها همان جداول اطلاعات بانک‌های اطلاعاتی (رکوردها و فیلدها) می‌باشند و Query ها حاوی مجموعه‌ای از دستورات می‌باشند که بر روی بانک‌های اطلاعاتی اعمال خواهند شد.

باز و بسته نمودن Dataset ها

قبل از انجام دادن هر کاری بر روی Dataset ها، لازم است تا Dataset (Table یا Query) مورد نظرتان را باز نمائید. این عمل به سادگی و به وسیلهٔ فراخوانی متد ()open انجام می‌گیرد.

```
Table.open;
```

روش دیگری که برای باز نمودن Dataset ها پیشنهاد می‌گردد، تخصیص دادن مقدار True به خصوصیت Active آنها است.

```
Table1.Active:= True;
```

به خاطر داشته باشید هنگامی که از متد ()open برای باز نمودن یک Dataset استفاده می‌کنید، پس از فراخوانی متد ()open ، مقدار خصوصیت Active آن به صورت خودکار برابر True می‌گردد.

دوباره یادآور می‌شویم که قبل از انجام اعمال پردازشی داده‌ها (نظیر درج، حذف، تصحیح، جستجو و ...) می‌بایست Dataset را در حالت باز (open) قرار دهید.

پس از خاتمه کار، باید Dataset در حالت بسته قرار گیرد. این عبارت نیز به دو روش امکان‌پذیر است. یکی استفاده از متد ()Colse و دیگری تخصیص مقدار False به خصوصیت Active جزء سازندهٔ Dataset می‌باشد.

```
Table1. Close;        یا        Table1.Active : = False;
```

برای آشنایی بیشتر با طریقه باز و بسته نمودن انواع مختلف بانکهای اطلاعاتی (ADO, BDE و ...) به برنامهٔ ارائه شده در لیست ۱-۵ مراجعه نمایید.

لیست ۱-۵ باز و بسته نمودن Datasetها

```
unit MainFrm;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, FMTBcd, DBXpress, IBDatabase, ADODB, DBTables, DB, SqlExpr,
  IBCustomDataSet, IBQuery, IBTable, StdCtrls;

type
  TForm1 = class(TForm)
    SQLDataSet1: TSQLDataSet;
    SQLTable1: TSQLTable;
    SQLQuery1: TSQLQuery;

    ADOTable1: TADOTable;
    ADODataSet1: TADODataSet;
    ADOQuery1: TADOQuery;

    IBTable1: TIBTable;
    IBQuery1: TIBQuery;
    IBDataSet1: TIBDataSet;

    Table1: TTable;
    Query1: TQuery;

    SQLConnection1: TSQLConnection;
    Database1: TDatabase;
    ADOConnection1: TADOConnection;
    IBDatabase1: TIBDatabase;
    Button1: TButton;
    Label1: TLabel;
    Button2: TButton;
    IBTransaction1: TIBTransaction;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
    procedure OpenDatasets;
    procedure CloseDatasets;
  public
    { Public declarations }
  end;
```

لیست ۱-۵

```
var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
begin
  IBDatabase1.Connected    := True;
  ADOConnection1.Connected := True;
  Database1.Connected      := True;
  SQLConnection1.Connected := True;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  OpenDatasets;
end;

procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  CloseDatasets;
  IBDatabase1.Connected    := false;
  ADOConnection1.Connected := false;
  Database1.Connected      := false;
  SQLConnection1.Connected := false;
end;
procedure TForm1.CloseDatasets;
begin

  // Disconnect from dbExpress datasets
  SQLDataSet1.Close;  // or .Active := false;
  SQLTable1.Close;    // or .Active := false;
  SQLQuery1.Close;    // or .Active := false;

  // Disconnect from ADO datasets
  ADOTable1.Close;    // or .Active := false;
  ADODataSet1.Close;  // or .Active := false;
  ADOQuery1.Close;    // or .Active := false;

  // Disconnect from Interbase Express datasets
  IBTable1.Close;    // or .Active := false;
  IBQuery1.Close;    // or .Active := false;
  IBDataSet1.Close;  // or .Active := false;
```

لیست ۱-۵

```
  // Disconnect from BDE datasets
  Table1.Close;     // or .Active := false;
  Query1.Close;     // or .Active := false;

  Label1.Caption := 'Datasets are closed.'
end;

procedure TForm1.OpenDatasets;
begin

  // Connect to dbExpress datasets
  SQLDataSet1.Open;  // or .Active := true;
  SQLTable1.Open;    // or .Active := true;
  SQLQuery1.Open;    // or .Active := true;

  // Connect to ADO datasets
  ADOTable1.Open;    // or .Active := true;
  ADODataSet1.Open;  // or .Active := true;
  ADOQuery1.Open;    // or .Active := true;

  // Connect to Interbase Express datasets
  IBTable1.Open;    // or .Active := true;
  IBQuery1.Open;    // or .Active := true;
  IBDataSet1.Open;  // or .Active := true;

  // Connect to BDE datasets
  Table1.Open;    // or .Active := true;
  Query1.Open;    // or .Active := true;

  Label1.Caption := 'Datasets are open.';
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  CloseDatasets;
end;

end.
```

پیمایش در Dataset ها

زمانی که بانک اطلاعاتی در حال کار است، به روشی نیاز دارید که با استفاده از آن رکوردها را پیمایش کنید. برای انجام این کار در دلفی چند روش وجود دارد که یکی از آنها استفاده از متدهای تعریف شده همچون First()، Last()، Next()، Prior() و MoveBy() می‌باشد.

EOF, BOF و حلقه‌ها

BOF و EOF خصوصیتهایی از یک Dataset می‌باشند که وضعیت قرار گرفتن در ابتدا یا انتهای Dataset را مشخص می‌سازند. چنانچه قصد داشته باشید عملیات پردازشی داده‌ها را برای تمامی رکوردهای یک جدول انجام دهید علاوه بر به کارگیری EOF و BOF لازم است تا از حلقه‌ها و متدهای تعریف شده برای حرکت در جدول کمک بگیرید.

به عنوان مثال کد زیر را ملاحظه نمایید.

```
Table1.First;   //go to begining of dataset
While not Table1.EOF do   // iterate over Table
begin
 // do some stuff with current record
Table.Next;      // Move To Next record
end;
```

حتی المقدور از حلقه repeat ... until برای کار با Dataset ها استفاده نکنید، چرا که در این حلقه مواجه شدن با جداولی خالی، به بروز خطا در سیستم منجر می‌شود.

```
repeat
Do Some Stuff;
Table1.Next;
until    Table1.EOF;
```

بنابراین شکل ارائه شده در کد فوق نادرست خواهد بود.

برای آشنائی بیشتر با نحوهٔ پیمایش در انواع مختلف بانکهای اطلاعاتی به برنامهٔ ارائه شده در لیست ۲-۵ مراجعه نمائید.

لیست ۲-۵ پیمایش در Datasetها

```
unit MainFrm;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, FMTBcd, DBXpress, IBDatabase, ADODB, DBTables, DB, SqlExpr,
  IBCustomDataSet, IBQuery, IBTable, StdCtrls, Grids, DBGrids, ExtCtrls;

type
  TForm1 = class(TForm)
```

لیست ۲-۵ ادامه

```
    SQLTable1: TSQLTable;
    ADOTable1: TADOTable;
    IBTable1: TIBTable;
    Table1: TTable;

    SQLConnection1: TSQLConnection;
    Database1: TDatabase;
    ADOConnection1: TADOConnection;
    IBDatabase1: TIBDatabase;
    Button1: TButton;
    Label1: TLabel;
    Button2: TButton;
    IBTransaction1: TIBTransaction;
    DBGrid1: TDBGrid;
    DataSource1: TDataSource;
    RadioGroup1: TRadioGroup;
    btnFirst: TButton;
    btnLast: TButton;
    btnNext: TButton;
    btnPrior: TButton;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure Button2Click(Sender: TObject);
    procedure RadioGroup1Click(Sender: TObject);
    procedure btnFirstClick(Sender: TObject);
    procedure btnLastClick(Sender: TObject);
    procedure btnNextClick(Sender: TObject);
    procedure btnPriorClick(Sender: TObject);
    procedure DataSource1DataChange(Sender: TObject; Field: TField);
  private
    { Private declarations }
    procedure OpenDatasets;
    procedure CloseDatasets;
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}
```

لیست ۲-۵ ادامه

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  IBDatabase1.Connected    := True;
  ADOConnection1.Connected := True;
  Database1.Connected      := True;
  SQLConnection1.Connected := True;

  Datasource1.DataSet := IBTable1;
  OpenDatasets;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  OpenDatasets;
end;
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  CloseDatasets;
  IBDatabase1.Connected    := false;
  ADOConnection1.Connected := false;
  Database1.Connected      := false;
  SQLConnection1.Connected := false;
end;

procedure TForm1.CloseDatasets;
begin

  // Disconnect from dbExpress dataset
  SQLTable1.Close;    // or .Active := false;

  // Disconnect from ADO dataset
  ADOTable1.Close;    // or .Active := false;

  // Disconnect from Interbase Express dataset
  IBTable1.Close;    // or .Active := false;

  // Disconnect from BDE datasets
  Table1.Close;    // or .Active := false;

  Label1.Caption := 'Datasets are closed.'
end;

procedure TForm1.OpenDatasets;
begin
```

لیست ۲-۵ ادامه

```
  // Connect to dbExpress dataset
  SQLTable1.Open;    // or .Active := true;

  // Connect to ADO dataset
  ADOTable1.Open;    // or .Active := true;

  // Connect to Interbase Express dataset
  IBTable1.Open;    // or .Active := true;

  // Connect to BDE dataset
  Table1.Open;    // or .Active := true;

  Label1.Caption := 'Datasets are open.';
end;
procedure TForm1.Button2Click(Sender: TObject);
begin
  CloseDatasets;
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
begin
  case RadioGroup1.ItemIndex of
    0: Datasource1.DataSet := IBTable1;
    1: Datasource1.DataSet := Table1;
    2: Datasource1.DataSet := ADOTable1;
  end; // case
end;

procedure TForm1.btnFirstClick(Sender: TObject);
begin
  DataSource1.DataSet.First;
end;

procedure TForm1.btnLastClick(Sender: TObject);
begin
  DataSource1.DataSet.Last;
end;

procedure TForm1.btnNextClick(Sender: TObject);
begin
  DataSource1.DataSet.Next;
end;

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

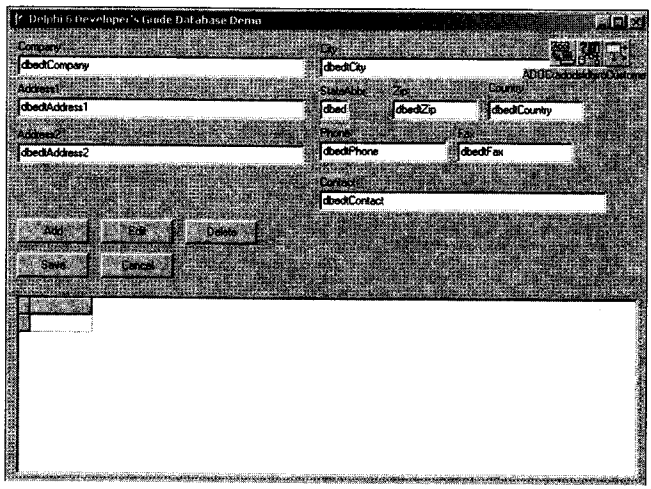
لیست ۲-۵ ادامه

```
begin
  DataSource1.DataSet.Prior;
end;

procedure TForm1.DataSource1DataChange(Sender: TObject; Field: TField);
begin
  btnLast.Enabled := not DataSource1.DataSet.Eof;
  btnNext.Enabled := not DataSource1.DataSet.Eof;
  btnFirst.Enabled := not DataSource1.DataSet.Bof;
  btnPrior.Enabled := not DataSource1.DataSet.Bof;
end;

end.
```

دستکاری داده‌ها

یکی از ویژگی‌های موتور بانک اطلاعاتی دلفی آن است که به سادگی امکان اعمال تغییرات در جداول نظیر درج، حذف و تصحیح رکوردهای جدید را فراهم می‌کند. متدهای Insert()، Delete() و Edit() در دلفی وظیفهٔ درج، حذف و تصحیح رکوردهای جداول را به عهده دارند.

در اینجا برای آشنایی بیشتر تنها مثالی را مطرح می‌نمائیم. به فرم ارائه شده در شکل ۲-۵ توجه کنید. این فرم یک برنامهٔ کاربردی سادهٔ بانک اطلاعاتی را نمایش می‌دهد که در آن امکان درج، حذف و تصحیح رکوردهای یک جدول گنجانده شده است.



شکل ۲-۵ فرم اصلی یک برنامهٔ کاربردی بانک اطلاعاتی

کد مربوط به این فرم را در لیست ۳-۵ ملاحظه کنید.

لیست ۳-۵ MainFormPas برنامه‌ای برای انجام اعمال دستکاری داده‌ها

```
unit MainFrm;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Mask, DBCtrls, DB, Grids, DBGrids, ADODB;

type
  TMainForm = class(TForm)
    ADOConnection1: TADOConnection;
    adodsCustomer: TADODataSet;
    dtsrcCustomer: TDataSource;
    DBGrid1: TDBGrid;
    adodsCustomerCustNo: TAutoIncField;
    adodsCustomerCompany: TWideStringField;
    adodsCustomerAddress1: TWideStringField;
    adodsCustomerAddress2: TWideStringField;
    adodsCustomerCity: TWideStringField;
    adodsCustomerStateAbbr: TWideStringField;
    adodsCustomerZip: TWideStringField;
    adodsCustomerCountry: TWideStringField;
    adodsCustomerPhone: TWideStringField;
    adodsCustomerFax: TWideStringField;
    adodsCustomerContact: TWideStringField;
    Label1: TLabel;
    dbedtCompany: TDBEdit;
    Label2: TLabel;
    dbedtAddress1: TDBEdit;
    Label3: TLabel;
    dbedtAddress2: TDBEdit;
    Label4: TLabel;
    dbedtCity: TDBEdit;
    Label5: TLabel;
    dbedtState: TDBEdit;
    Label6: TLabel;
    dbedtZip: TDBEdit;
    Label7: TLabel;
    dbedtPhone: TDBEdit;
    Label8: TLabel;
    dbedtFax: TDBEdit;
    Label9: TLabel;
    dbedtContact: TDBEdit;
```

لیست ۳-۵ ادامه

```
  btnAdd: TButton;
  btnEdit: TButton;
  btnSave: TButton;
  btnCancel: TButton;
  Label10: TLabel;
  dbedtCountry: TDBEdit;
  btnDelete: TButton;
  procedure btnAddClick(Sender: TObject);
  procedure btnEditClick(Sender: TObject);
  procedure btnSaveClick(Sender: TObject);
  procedure btnCancelClick(Sender: TObject);
  procedure FormCreate(Sender: TObject);
  procedure FormClose(Sender: TObject; var Action: TCloseAction);
  procedure btnDeleteClick(Sender: TObject);
private
  { Private declarations }
  procedure SetButtons;
public
    { Public declarations }
  end;

var
  MainForm: TMainForm;

implementation

{$R *.dfm}

procedure TMainForm.btnAddClick(Sender: TObject);
begin
  adodsCustomer.Insert;
  SetButtons;
end;

procedure TMainForm.btnEditClick(Sender: TObject);
begin
  adodsCustomer.Edit;
  SetButtons;
end;

procedure TMainForm.btnSaveClick(Sender: TObject);
begin
  adodsCustomer.Post;
  SetButtons;
end;

procedure TMainForm.btnCancelClick(Sender: TObject);
begin
```

```
  adodsCustomer.Cancel;
  SetButtons;
end;

procedure TMainForm.SetButtons;
begin
  btnAdd.Enabled  := adodsCustomer.State = dsBrowse;
  btnEdit.Enabled := adodsCustomer.State = dsBrowse;
  btnSave.Enabled := (adodsCustomer.State = dsInsert) or
    (adodsCustomer.State = dsEdit);
  btnCancel.Enabled := (adodsCustomer.State = dsInsert) or
    (adodsCustomer.State = dsEdit);
  btnDelete.Enabled := adodsCustomer.State = dsBrowse;
end;
procedure TMainForm.FormCreate(Sender: TObject);
begin
  adodsCustomer.Open;
  SetButtons;

end;

procedure TMainForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  adodsCustomer.Close;
  ADOConnection1.Connected := False;
end;

procedure TMainForm.btnDeleteClick(Sender: TObject);
begin
  adodsCustomer.Delete;
end;

end.
```

متدهایی که در این برنامه برای اعمال دستکاری داده‌ها وجود دارند به شرح زیر می‌باشند.

- Post() : وظیفهٔ ثبت تغییرات (درج، حذف، تصحیح) در یک جدول را به عهده دارد.
- Cancel() : از این متد جهت لغو اعمال انجام شده (درج، حذف، تصحیح) بر روی بانک اطلاعاتی استفاده می‌شود.
- Delete() : وظیفه حذف رکورد جاری یک جدول را به عهده دارد.

وضعیت Dataset

هر Dataset می‌تواند در چند وضعیت مختلف قرار گیرد. این وضعیت‌ها اطلاعاتی دربارهٔ ارتباط برنامهٔ کاربردی با Dataset فراهم می‌کنند. بررسی وضعیت یک Dataset را تنها در زمان اجرا و از طریق بررسی خاصیت state می‌توان انجام داد. به عنوان مثال برای مشاهدهٔ رکوردهای یک Dataset باید آن

را در وضعیت (مُد) dsBrowse قرار دهید. جدول ۱-۵ تعدادی از وضعیت‌های موجود برای یک Dataset را نمایش می‌دهد. برای اطلاع از سایر مقادیر وضعیت‌ها به راهنمای دلفی مراجعه نمائید.

جدول ۱-۵ مقادیر معتبر برای خصوصیت Stateی جزء سازندهٔ TDataset

مقدار	شرح
dsBrowse	dataset را در وضعیت (مُد) Browse قرار می‌دهد.
dsEdit	dataset را در مُد Edit قرار می‌دهد.
dsInactive	dataset را در حالت بسته (close) قرار می‌دهد.
dsInsert	dataset را در حالت Insert قرار می‌دهد. این وضعیت معادل فراخوانی متد Insert() بدون ثبت تغییرات (post()) می‌باشد.
dsSetKey	dataset را در مُد setKey (جستجوی یک رکورد خاص) قرار می‌دهد.
dsCalcFields	dataset را در مُد استفاده از فیلدهای محاسباتی قرار می‌دهد. فیلدهای محاسباتی در ادامهٔ این فصل ارائه شده‌اند.

کار با فیلدها

دستیابی و ویرایش فیلدهای جداول از اساسی‌ترین اعمال مدیریتی بانک‌های اطلاعاتی است. هر شیء TTable آرایه‌ای از فیلدها دارد که می‌توان آن را با استفاده از ویراستار فیلدها و یا تعریف آنها اضافه، تصحیح و حذف نمود.

مقادیر فیلدها

در دلفی می‌توان فیلدها را به طور مستقیم و بدون تعیین آنچه که نشان می‌دهند (یا حتی نوع آنها) مورد دستیابی قرار داد. به عنوان مثال در قطعه کد زیر مقدار فیلد CustName از جدول Table1 به متغیر S اختصاص داده خواهد شد.

```
S := Table1 ['CustName'];
```

CustName یک فیلد از نوع رشته‌ای است و همانطور که می‌بینید به سادگی در دسترس قرار گرفته است. روال فوق برای تمامی انواع فیلدها اعم از رشته‌ای، عددی صحیح، منطقی و ... عمومیت دارد. به عنوان مثال در قطعه کد زیر مقدار فیلد CustNo از جدول Table1 که حاوی یک مقدار عددی صحیح می‌باشد به متغیر I تخصیص داده می‌شود.

```
I := Table1 ['CustNo'];
```

اما قدرت دلفی، آنجا نمایان می‌شود که شما را قادر می‌سازد تا مقادیر چندین فیلد مختلف را در آرایه‌ای از نوع Variant ذخیره نمائید.

کد زیر شما را با این قابلیت آشنا می‌سازد.

```
const
  AStr = 'The %s is of the %s category and its length is %f in.';
var
  VarArr: Variant;
  F: Double;
begin
  VarArr := VarArrayCreate([0, 2], varVariant);
  { Assume Table1 is attached to Biolife table }
  VarArr := Table1['Common_Name;Category;Length_In'];
  F := VarArr[2];
  ShowMessage(Format(AStr, [VarArr[0], VarArr[1], F]));
end;
```

هر dataset خاصیتی به نام Fields دارد که این خاصیت می‌تواند نام فیلدها، نوع داده‌های آن، اندازهٔ داده‌های آن و مقادیر رکورد جاری را مشخص سازد. هنگامی که با خصوصیت Fields کار می‌کنید دانستن نوع داده برای دستیابی صحیح به آن لازم است. تعیین نوع داده برای خصوصیت Fields به شرح جدول ۲-۵ انجام می‌پذیرد.

جدول ۲-۵ مقادیر تعیین کننده نوع داده‌ها در شی TField

مقدار	خروجی
ASBoolean	الگوی خروجی به صورت یک مقدار Boolean خواهد بود.
ASFloat	الگوی خروجی به صورت یک مقدار Double خواهد بود.
ASInteger	الگوی خروجی به صورت یک مقدار LongInt خواهد بود.
ASString	الگوی خروجی به صورت یک مقدار String خواهد بود.
ASDateTime	الگوی خروجی به صورت یک مقدار DateTime خواهد بود.
Value	الگوی خروجی به صورت یک مقدار Variant خواهد بود.

در رابطه با خصوصیت Fields و آرایه‌ای که در این خصوص به وجود می‌آید، توجه داشته باشید که آرایه موردنظر با آوردن صفر شروع می‌شود. به عنوان مثال برای دستیابی به فیلد شمارهٔ صفر از جدول Table1 که حاوی یک مقدار رشته‌ای می‌باشد از دستور زیر استفاده می‌شود.

```
S := Table1.Fields[0]. Asstring;
```

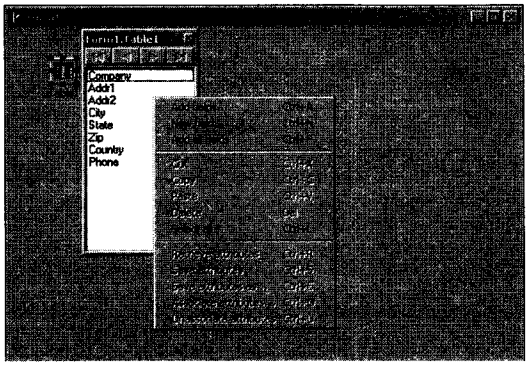
دلفی همچنین به شما امکان می‌دهد که برای دستیابی به یک فیلد از نام آن استفاده کنید. برای این کار از متد FieldByName استفاده می‌شود.

```
I := Table1.FieldsByName('orderNo').AsInteger;
```

ویراستار فیلدها

ویراستار فیلدها یکی دیگر از ابزارهای کارآمد دلفی جهت کار با بانک‌های اطلاعاتی است. برای فعال نمودن این ویراستار کافی است که دکمهٔ ماوس را بر روی شیء TTable (یا TQuery و یا TStoredproc) برده و آن را برای دوبار متوالی بفشارید.

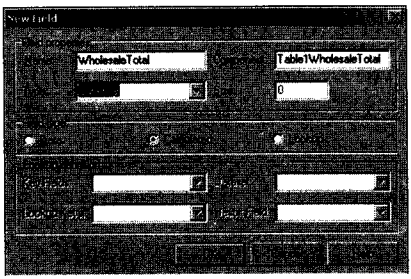
ویراستار فیلدها و منوی محلی آن را در شکل ۳-۵ ملاحظه می‌کنید.



شکل ۳-۵

فیلدهای محاسباتی

دلفی روش ویژه‌ای برای تعریف فیلدهای اضافی جداول بانک‌های اطلاعاتی دارد. این فیلدها که از آنها تحت عنوان فیلدهای محاسباتی یاد می‌شود براساس فیلدهای دیگر جدول به دست می‌آیند. ویژگی یک فیلد محاسباتی آن است که مقدار محاسبه شده در بانک اطلاعاتی ذخیره نخواهد شد. برای افزودن فیلدهای محاسباتی از ویراستار فیلدها استفاده می‌شود.



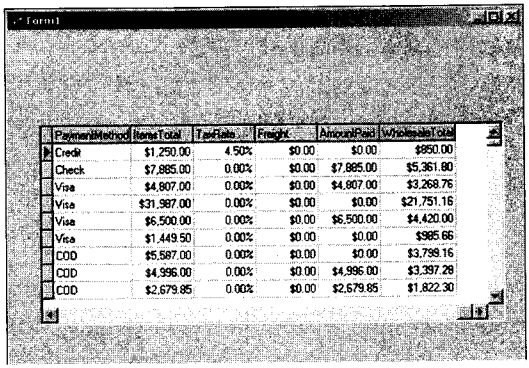
شکل ۴-۵ افزودن یک فیلد محاسباتی از طریق ویراستار فیلدها در دلفی

پس از قرار دادن یک جدول کاربردی برای نمایش محتویات جدول بر روی یک فرم، عنوان ORDERS (از جداول ارائه شدهٔ مثال‌های نصب شده در دلفی) را برای این جدول برگزیده سپس با فراخوانی ویراستار فیلدها، گزینه، Add New Fields را انتخاب نمائید. با انجام این کار شکلی همانند شکل ۴-۵ به نمایش در می‌آید. همانطور که در این شکل ملاحظه می‌کنید نام فیلد برابر با wholesaleTotal و نوع آن از نوع محاسباتی (Calculated) در نظر گرفته شده است.

برای کار با فیلدهای محاسباتی از رویداد OncalcFields و به صورتی مشابه با کد زیر استفاده می‌شود.

```
Procedure TForm1.Table1 CalcFields (Dataset : TDataset);
begin
Dataset['wholesaleTotal'] := Dataset['ItemsTotal'] *0.68;
end;
```

اکنون یک فیلد محاسباتی تحت عنوان wholesaleTotal به جدول ORDERS اضافه شده است. (به شکل ۵-۵ مراجعه نمائید)

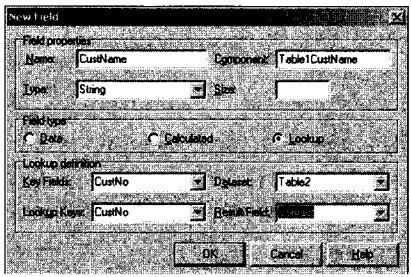


شکل ۵-۵ اضافه شدن یک فیلد محاسباتی به یک جدول

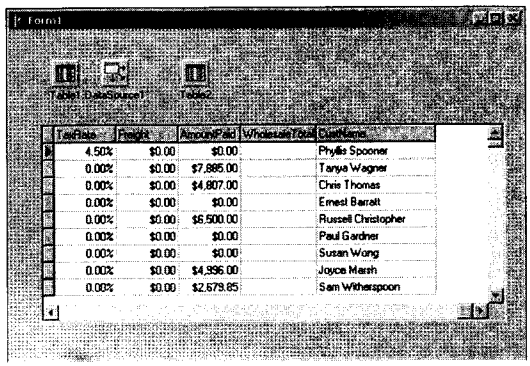
فیلدهای Lookup

یکی دیگر از انواع فیلدهای قابل تعریف در دلفی، فیلدهای Lookup می‌باشند (به قسمت نوع فیلدها در شکل ۴-۵ مراجعه کنید) تعریف یک فیلد از نوع Lookup به شما امکان می‌دهد تا از یک جدول یا پرس‌وجوی Lookup استفاده کرده و نهایتاً فیلد را به طور خودکار مقداردهی کنید. برای انجام این کار وجود دو dataset ضروری خواهد بود.

برای مثال به شکلهای ۶-۵ و ۷-۵ مراجعه نمائید. در این دو شکل نحوهٔ اضافه‌کردن یک فیلد از نوع Lookup و همچنین طریقه برقراری ارتباط بین جداول ORDERS و CUSTOMER نشان داده شده است. (این جداول با تخصیص مقدار DBDEMOS به خصوصیت DatasetName قابل دسترسی می‌باشند).



شکل ۶-۵ افزودن یک فیلد Lookup از طریق ویراستار فیلدها در دلفی



شکل ۷-۵ نمایش جدول حاوی فیلد Lookup

کار با فیلدهای BLOB

BLOB سر نام عبارت Binary Large Object است و اصولاً به فیلدهایی اتلاق می‌شود که مقدار آنها به طور پویا و در زمان اجرا مشخص می‌شوند. به بیان دیگر می‌توان گفت که مقادیر فیلدها BLOB قابل پیش‌بینی نیست. به عنوان مثال در بعضی مواقع تنها سه بایت از حافظه برای ذخیره نمودن یک فیلد BLOB کافی است و در بعضی اوقات این مقدار تا حدود ۳KB افزایش خواهد یافت. عموماً فیلدهای BLOB برای ذخیره نمودن متن‌ها، تصاویر و اشیاء OLE به کار می‌روند.

مقادیر معتبر برای شیء TBLobField را در جدول ۴-۵ ملاحظه خواهید فرمود.

جدول ۴-۵ مقادیر موجود برای TBlobField

نوع	شرح
ftBlob	فیلد حاوی دادهٔ بدون نوع و یا داده‌های تعریف شده توسط کاربر.
ftMeMo	فیلد حاوی متن (Text).
ftGraphic	فیلد حاوی تصاویر گرافیکی Bitmap.
ftFmtMemo	فیلد حاوی MeMo هایی با قالب ارائه شده در Paradox.
ftParadoxole	فیلد حاوی اشیاء OLE با قالب ارائه شده در Paradox.
ftDBaseOLE	فیلد حاوی اشیاء OLE با قالب ارائه شده در dBASE.
ftraBlob	فیلد BloB در جدول بانک‌های اطلاعاتی oracle (از نسخهٔ oracle8 به بعد).
ftoraClob	فیلد CloB در جداول بانک‌های اطلاعاتی oracle (از نسخه oracle8 به بعد).

مثالی از فیلدهای BLOB

در این بخش برنامه‌ای را مطرح می‌سازیم که شما را قادر می‌سازد تا یک فایل صوتی از نوع WAV را به عنوان فیلدی از یک جدول ذخیره و پخش نمائید. فرم اصلی این برنامه در شکل ۸-۵ نمایش داده شده است. (این برنامه را wavez می‌نامیم).

شکل ۸-۵ فرم اصلی برنامهٔ wavez

ساختار جدول این برنامه به صورت زیر در نظر گرفته شده است.

نام فیلد	نوع فیلد	اندازه (سایز)
waveTitle	کاراکتری	۲۵
Filename	کاراکتری	۲۵
wave	BLOB	-

برای درج کردن یک فایل WAV در جدول موردنظر از دکمهٔ Add (+) موجود بر روی فرم (شکل ۸-۵)

استفاده خواهد شد. بنابراین لازم است تا رویداد onclick این دکمه را به صورت زیر تعریف نمائیم.

```
procedure TMainForm.sbAddClick(Sender: TObject);
begin
  if OpenDialog.Execute then
  begin
    tblSounds.Append;
    tblSounds['FileName'] := ExtractFileName(OpenDialog.FileName);
    tblSoundsWave.LoadFromFile(OpenDialog.FileName);
    edTitle.SetFocus;
  end;
end;
```

برای ذخیره کردن فایل WAV در یک فایل خارجی به طور مشابه از دکمهٔ save موجود بر روی فرم استفاده خواهد شد. رویداد onclick این دکمه به صورت زیر خواهد بود.

```
procedure TMainForm.sbSaveClick(Sender: TObject);
begin
  with SaveDialog do
  begin
    FileName := tblSounds['FileName'];     // initialize file name
    if Execute then                        // execute dialog
      tblSoundsWave.SaveToFile(FileName); // save blob to file
  end;
end;
```

با فشردن دکمهٔ Play موجود بر روی فرم، فایل WAV موردنظر از جدول مربوطه فراخوانی و برای پخش به تابع Playsound() (از توابع API ویندوز) فرستاده می‌شود. رویداد onclick این دکمه حاوی کد زیر می‌باشد.

```
procedure TMainForm.sbPlayClick(Sender: TObject);
var
  B: TBlobStream;
  M: TMemoryStream;
begin
  B := TBlobStream.Create(tblSoundsWave, bmRead); // create blob stream
  Screen.Cursor := crHourGlass;                   // wait hourglass
  try
    M := TMemoryStream.Create;                    // create memory stream
    try
      M.CopyFrom(B, B.Size);              // copy from blob to memory stream
      // Attempt to play sound. Raise exception if something goes wrong
      Win32Check(PlaySound(M.Memory, 0, SND_SYNC or SND_MEMORY));
      finally
        M.Free;
      end;
```

```
  finally
    Screen.Cursor := crDefault;
    B.Free;                                    // clean up
  end;
end;
```

■ نکته

توجه داشته باشید که در این برنامه، لازم است تا dataset خود را در حالتهای Edit ، Insert و Append قرار دهید.

یونیت اصلی برنامهٔ wavez در لیست ۴-۵ ارائه گردیده است.

لیست ۴-۵ یونیت اصلی پروژهٔ wavez

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ExtCtrls, DBCtrls, DB, DBTables, StdCtrls, Mask, Buttons, ComCtrls;

type
  TMainForm = class(TForm)
    tblSounds: TTable;
    dsSounds: TDataSource;
    tblSoundsWaveTitle: TStringField;
    tblSoundsWave: TBlobField;
    edTitle: TDBEdit;
    edFileName: TDBEdit;
    Label1: TLabel;
    Label2: TLabel;
    OpenDialog: TOpenDialog;
    tblSoundsFileName: TStringField;
    SaveDialog: TSaveDialog;
    pnlToobar: TPanel;
    sbPlay: TSpeedButton;
    sbAdd: TSpeedButton;
    sbSave: TSpeedButton;
    sbExit: TSpeedButton;
    Bevel1: TBevel;
    dbnNavigator: TDBNavigator;
    stbStatus: TStatusBar;
    procedure sbPlayClick(Sender: TObject);
    procedure sbAddClick(Sender: TObject);
    procedure sbSaveClick(Sender: TObject);
    procedure sbExitClick(Sender: TObject);
```

```
    procedure FormCreate(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private
    procedure OnAppHint(Sender: TObject);
  end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

uses MMSystem;

procedure TMainForm.sbPlayClick(Sender: TObject);
var
  B: TBlobStream;
  M: TMemoryStream;
begin
  B := TBlobStream.Create(tblSoundsWave, bmRead); // create blob stream
  Screen.Cursor := crHourGlass;                   // wait hourglass
  try
  M := TMemoryStream.Create;                     // create memory stream
  try
    M.CopyFrom(B, B.Size);              // copy from blob to memory stream
    // Attempt to play sound. Raise exception if something goes wrong
    Win32Check(PlaySound(M.Memory, 0, SND_SYNC or SND_MEMORY));
  finally
    M.Free;
  end;
finally
  Screen.Cursor := crDefault;
  B.Free;                                        // clean up
end;
end;

procedure TMainForm.sbAddClick(Sender: TObject);
begin
  if OpenDialog.Execute then
  begin
    tblSounds.Append;
    tblSounds['FileName'] := ExtractFileName(OpenDialog.FileName);
    tblSoundsWave.LoadFromFile(OpenDialog.FileName);
    edTitle.SetFocus;
  end;
end;
```

```
procedure TMainForm.sbSaveClick(Sender: TObject);
begin
  with SaveDialog do
  begin
    FileName := tblSounds['FileName'];     // initialize file name
    if Execute then                        // execute dialog
      tblSoundsWave.SaveToFile(FileName); // save blob to file
  end;
end;

procedure TMainForm.sbExitClick(Sender: TObject);
begin
  Close;
end;

procedure TMainForm.FormCreate(Sender: TObject);
begin
  Application.OnHint := OnAppHint;
  tblSounds.Open;
end;
procedure TMainForm.OnAppHint(Sender: TObject);
begin
  stbStatus.SimpleText := Application.Hint;
end;

procedure TMainForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  tblSounds.Close;
end;

end.
```

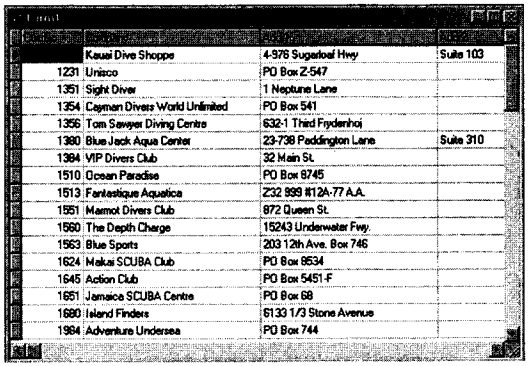
فیلترگذاری بر روی داده‌ها

فیلترگذاری به مفهوم انتخاب بخش معینی از dataset براساس یک سری شرایط خاص (تعریف شده) می‌باشد. مهمترین مزیت فیلترگذاری، عدم نیاز به داشتن فایل شاخص (ایندکس) است. البته عموماً روشهای فیلترگذاری سرعت کمتری از فایل‌های شاخص خواهند داشت. برای تعریف فیلتر در دلفی دو گام به شرح زیر وجود دارد:

۱- تعریف رویه‌ای برای رویداد onFilterRecord یک dataset براساس شرایط و محدودیتهای موردنظر.

۲- تخصیص مقدار True به خصوصیت Filtered همان dataset.

به عنوان مثال شکل ۹-۵ را ملاحظه کنید. در این شکل از یک شیء TDBGrid برای نمایش محتویات جدول CUSTOMER استفاده شده است. حال برای رویداد onFilterRecord آن، قطعه کد



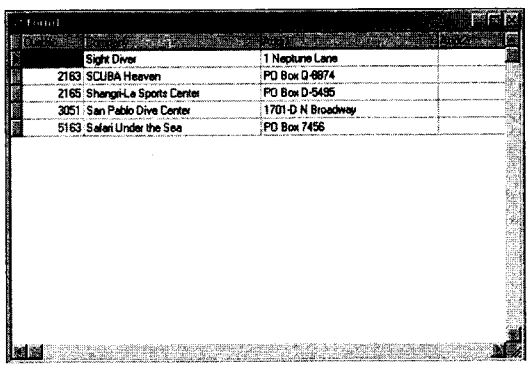
	Kauai Dive Shoppe	4-976 Sugarloaf Hwy	Suite 103
1231	Unisco	PO Box Z-547	
1351	Sight Diver	1 Neptune Lane	
1354	Cayman Divers World Unlimited	PO Box 541	
1356	Tom Sawyer Diving Centre	632-1 Third Frydenhoj	
1380	Blue Jack Aqua Center	23-738 Paddington Lane	Suite 310
1384	VIP Divers Club	32 Main St.	
1510	Ocean Paradise	PO Box 8745	
1513	Fantastique Aquatica	Z32 999 #12A-77 A.A.	
1551	Marmot Divers Club	872 Queen St.	
1560	The Depth Charge	15243 Underwater Fwy.	
1563	Blue Sports	203 12th Ave. Box 746	
1624	Makai SCUBA Club	PO Box 8534	
1645	Action Club	PO Box 5451-F	
1651	Jamaica SCUBA Centre	PO Box 68	
1680	Island Finders	6133 1/3 Stone Avenue	
1984	Adventure Undersea	PO Box 744	

شکل ۹-۵ محتویات جدول CUSTOMER قبل از فیلترگذاری

زیر را تعریف نمائید.

```
procedure TForm1.Table1FilterRecord(DataSet: TDataSet;
  var Accept: Boolean);
var
  FieldVal: String;
begin
  FieldVal := DataSet['Company'];  // Get the value of the Company field
  Accept := FieldVal[1] = 'S';     // Accept record if field starts with 'S'
end;
```

تعریف این کد به منزلهٔ اجرای گام اول از روش فیلترگذاری است. برای گام دوم تنها می‌بایست خصوصیت filtered این جدول را برابر True نمائید. پس از این کار شکلی همانند شکل ۱۰-۵ برای شما به نمایش در می‌آید.



	Sight Diver	1 Neptune Lane
2163	SCUBA Heaven	PO Box Q-6874
2165	Shangri-La Sports Center	PO Box D-5495
3051	San Pablo Dive Center	1701-D N Broadway
5163	Safari Under the Sea	PO Box 7456

شکل ۱۰-۵ محتویات جدول CUSTOMER پس از فیلترگذاری

جستجو در Datasetها

تا به اینجا مثال‌هایی را برای پیمایش و ویرایش رکوردها مشاهده کرده‌اید. حال چنانچه خواسته باشید به رکورد خاصی از یک جدول دستیابید به چه صورت عمل خواهید کرد. با استفاده از ابزارهایی که از طریق جزء سازنده TTable یا متدهای دلفی در اختیار شما قرار می‌گیرد. به سادگی می‌توانید عملیات جستجوی رکوردها را به انجام برسانید. در اینجا قصد داریم تا شما را با این متدها آشنا سازیم.

FindNext() و FindFirst()

FindFirst() ، FindNext() ، FindPrior() و FindLast() از ابتدایی‌ترین متدهای دلفی برای جستجوی یک رکورد خاص در یک جدول می‌باشند. این متدها هیچ پارامتر ورودی ندارند و تنها با باز گرداندن یک مقدار منطقی (False یا True) نتیجهٔ جستجو را مشخص خواهند ساخت.

متد Locate()

تکنیک دیگری که برای جستجوی رکوردهای خاص وجود دارد استفاده از تابع Locate() است. از این تابع جهت جستجوی رکوردها براساس مقادیر فیلدهای آنها استفاده می‌شود. الگوی تابع Locate() به صورت زیر می‌باشد.

```
Function Locate(Const KeyFields: String; Const KeyValues: Variant; Options:
TLocateoptions): Boolean;
```

پارامتر KeyFields نمایانگر نام فیلدی است که جستجو براساس آن انجام می‌پذیرد. پارامتر KeyValues مقدار فیلد و پارامتر Options نوع جستجو را مشخص خواهند ساخت. همانطور که ملاحظه می‌کنید پارامتر options از نوع TLocateOptions تعریف شده است. TLocateOptions در یونیت DB و به صورت زیر تعریف گردیده است.

```
Type
TLocateOption = (LocateInsensitive, LoPartialKey);
TLocateOptions = Set of TLocateOption;
```

به عنوان مثال برای جستجوی رکوردی در جدول Table1 که مقدار فیلد CustNo آن برابر ۱۳۵۶ می‌باشد، از تابع Locate() به صورت زیر استفاده می‌شود.

```
Table1.Locate('CustNo', 1356, []);
```

جستجو براساس کلید

متدهای FindKey() و SetKey() ابزارهای دیگری برای جستجوی رکوردها می‌باشند. برای جستجوی رکوردهای یک جدول، ابتدا باید جدول را در وضعیت قابل جستجو قرار دهید. برای این کار از متد

SetKey() استفاده می‌شود. برای انتقال به رکوردی که جستجو را برای آن انجام می‌دهید، از متد GotoKey() استفاده خواهد شد. در صورت پیدا نشدن کلید، رکورد جاری به همان شکل باقی مانده و متد GotoKey() مقدار False را برمی‌گرداند. این روش در صورت داشتن مقدار دقیق فیلد کار خواهد کرد، اما اگر تنها بخشی از فیلد مشخص باشد از متد GotoNearest() استفاده می‌شود. به عنوان مثال نمونه‌ای از به کارگیری متد SetKey() و GotoKey() در زیر ارائه گردیده است.

```
While Table1 do begin
 SetKey;
 Fields[0].AsInteger := 123;
 Fields[1].AsString := 'Hello';
 if not GotoKey then MessageBeep(0);
end;
```

روش به کارگیری متد GotoNearest() را در مثال ارائه شده زیر ملاحظه می‌کنید.

```
While Table1 do begin
 SetKey;
 Fields[0].AsInteger := 123;
 GotNearest;
end;
```

FindKey() نیز از دیگر متدهایی است که در عملیات جستجوی رکوردها کاربرد دارد. در متد FindKey() جستجو براساس یک یا چند فیلد کلیدی به صورت همزمان صورت می‌پذیرد. نمونه‌ای از به کارگیری متد FindKey() را در زیر مشاهده می‌کنید.

```
if not Table1.FindKey([123, 'Hello'] then MessageBeep(0);
```

■ نکته

اگر عملیات جستجو را برای جداول شاخص‌دار انجام می‌دهید بهتر است از متدهای FindKey() و FindNearest() استفاده کنید.

به کارگیری شاخص‌ها[1]

هر شاخص مُدل مرتب شده‌ای از یک جدول می‌باشد. عموماً در بانک‌های اطلاعاتی رابطه‌ای چندین شاخص برای یک جدول وجود دارد. ایجاد شاخص در هنگام عملیات جستجو و مرتب‌سازی جداول

1- Indexes

کمک زیادی در افزایش کارآیی برنامه‌ها خواهد داشت. توجه داشته باشید که ایجاد و استفاده از شاخص‌ها موجب کندی اعمالی نظیر درج و بروز رسانی بانک‌های اطلاعاتی می‌شود.

برای این که عملیات جستجوی موردنظرتان را با شاخصی غیر از شاخص اصلی (مثلاً فیلد کلیدی بانک اطلاعاتی) انجام دهید، لازم است تا خاصیت IndexName را به صورت دقیق مقداردهی نمائید.

به عنوان مثال چنانچه جدول Table1 دارای یک شاخص بر روی فیلد Company به نام Bycompany باشد، آنگاه قطعه کد زیر در یافتن رکوردی که نام فیلد company آن برابر 'Unisco' است، بسیار مناسب عمل خواهد نمود.

```
While Table1 do begin
 IndexName := 'ByCompany';
 SetKey;
 FieldValues['Company'] := 'Unisco';
end;
```

گاهی اوقات لازم است رکوردهای جداول خود را با استفاده از معیارهای ویژه‌ای محدود نمائید. این عملیات به شما امکان می‌دهد تا با زیر مجموعهٔ کوچکتری از داده‌ها کار کنید. یکی از روشهای تعیین محدوده، استفاده از متد SetRange می‌باشد.

به عنوان مثال کد زیر باعث محدودسازی رکوردهایی از table1 خواهد شد که اولین فیلد آنها، مقداری عددی بین ۱۰ تا ۱۵ را دارند.

```
Table1.SetRange([10], [15]);
```

روش دیگر، استفاده از متدهای ApplyRange() ، SetRangeStart() و SetRangeEnd() می‌باشد. متد SetRangeStart() مشخص کنندهٔ ابتدای بازه محدودیت‌گذاری و متد SetRangeEnd() تعیین کننده انتهای بازهٔ محدودیت‌گذاری بر روی جدول خواهد بود. از متد ApplyRange() نیز جهت تعریف و پیاده‌سازی محدودیت‌های جدید استفاده می‌شود.

به مثال زیر در این رابطه توجه کنید.

```
While Table1 do begin
 SetRangeStart;
 Fields[0].AsInteger := 10;     // range starts at 10
 SetRangeEnd;
 Fields[0].AsInteger := 15;      // range ends at 15
 ApplyRange;
end;
```

جهت حذف محدودیت اعمال شده نیز از متد CancelRange() استفاده خواهد شد.

```
Table1.CancelRange;
```

یک مثال کاربردی

در این بخش به ارائه یک مثال کاربردی متشکل از اعمال جستجو، فیلترگذاری و ... در خصوص بانک‌های اطلاعاتی خواهیم پرداخت. این مثال را تحت پروژه SRF نامگذاری می‌نمائیم.

طراحی ماژول داده‌ای پروژه SRF

فرم اصلی این پروژه (MainForm) را در شکل ۱۱-۵ مشاهده خواهید نمود. یونیت اصلی این پروژه را تحت نام Main ذخیره خواهیم نمود. همانطور که ملاحظه می‌کنید، این فرم حاوی یک جزء سازندهٔ TDBGrid (به نام DBGrid1) برای نمایش داده‌ها و دو دکمهٔ رادیویی (Radio Button) جهت انتخاب شاخص‌های جدول Customers می‌باشد. توجه داشته باشید که این دکمه‌های رادیویی در یک Radio Group به نام RGKeyField قرار دارند.

کد مربوط به رویداد Onclick این Radio Group را در زیر مشاهده می‌نمائید.

```
procedure TMainForm.RGKeyFieldclick(Sender: TObject);
begin
 case RGKeyField.ItemIndex of
 0: DM.Table1.IndexName := '';     // primary index
1: DM.Table1.IndexName := 'ByCompany': // secondary, by company
 end:
end;
```

جزء سازندهٔ دیگری که در فرم اصلی این برنامه قرار دارد TMainMenu می‌باشد (که از آن جهت باز و بسته کردن سایر فرم‌ها استفاده می‌شود). جزء سازنده تحت عنوان MainMenu1 نامگذاری شده و عبارتهای KeySearch ، Range و Filter جزو عناوین درج شده در آن خواهند بود.

یونیت اصلی این برنامه در لیست ۵-۵ ارائه شده است.

لیست ۵-۵ یونیت Main.Pas

```
unit Main;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, ExtCtrls, Grids, DBGrids, DB, DBTables,
  Buttons, Mask, DBCtrls, Menus, KeySrch, Rng, Fltr;

type
  TMainForm = class(TForm)
    DBGrid1: TDBGrid;
    RGKeyField: TRadioGroup;
```

لیست ۵-۵ ادامه

```
  MainMenu1: TMainMenu;
  Forms1: TMenuItem;
  KeySearch1: TMenuItem;
  Range1: TMenuItem;
  Filter1: TMenuItem;
  N1: TMenuItem;
  Exit1: TMenuItem;
  procedure RGKeyFieldClick(Sender: TObject);
  procedure KeySearch1Click(Sender: TObject);
  procedure Range1Click(Sender: TObject);
  procedure Filter1Click(Sender: TObject);
  procedure Exit1Click(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;
var
  MainForm: TMainForm;

implementation

uses DataMod;

{$R *.DFM}

procedure TMainForm.RGKeyFieldClick(Sender: TObject);
begin
  case RGKeyField.ItemIndex of
    0: DM.Table1.IndexName := '';           // primary index
    1: DM.Table1.IndexName := 'ByCompany';  // secondary, by company
  end;
end;

procedure TMainForm.KeySearch1Click(Sender: TObject);
begin
  KeySearch1.Checked := not KeySearch1.Checked;
  KeySearchForm.Visible := KeySearch1.Checked;
end;

procedure TMainForm.Range1Click(Sender: TObject);
begin
  Range1.Checked := not Range1.Checked;
  RangeForm.Visible := Range1.Checked;
end;

procedure TMainForm.Filter1Click(Sender: TObject);
begin
  Filter1.Checked := not Filter1.Checked;
```

لیست ۵-۵ ادامه

```
  FilterForm.Visible := Filter1.Checked;
end;

procedure TMainForm.Exit1Click(Sender: TObject);
begin
  Close;
end;

end.
```

یونیت مربوط به عنوان KeySearch نیز تحت عنوان KeySrch نامگذاری و در لیست ۶-۵ ارائه گردیده است.

لیست ۶-۵ یونیت KeySrch.Pas

```
unit KeySrch;

interface

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

type
  TKeySearchForm = class(TForm)
    Panel1: TPanel;
    Label3: TLabel;
    SearchEdit: TEdit;
    RBNormal: TRadioButton;
    Incremental: TRadioButton;
    Label6: TLabel;
    ExactButton: TButton;
    NearestButton: TButton;
    procedure ExactButtonClick(Sender: TObject);
    procedure NearestButtonClick(Sender: TObject);
    procedure RBNormalClick(Sender: TObject);
    procedure IncrementalClick(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private
    procedure NewSearch(Sender: TObject);
  end;

var
  KeySearchForm: TKeySearchForm;

implementation

uses DataMod, Main;
```

لیست ۶-۵ ادامه

```
{$R *.DFM}

procedure TKeySearchForm.ExactButtonClick(Sender: TObject);
begin
  { Try to find record where key field matches SearchEdit's Text value. }
  { Notice that Delphi handles the type conversion from the string      }
  { edit control to the numeric key field value.                       }
  if not DM.Table1.FindKey([SearchEdit.Text]) then
    MessageDlg(Format('Match for "%s" not found.', [SearchEdit.Text]),
               mtInformation, [mbOk], 0);
end;

procedure TKeySearchForm.NearestButtonClick(Sender: TObject);
begin
  { Find closest match to SearchEdit's Text value. Note again the }
  { implicit type conversion.                                     }
  DM.Table1.FindNearest([SearchEdit.Text]);
end;

procedure TKeySearchForm.NewSearch(Sender: TObject);
{ This is the method which is wired to the SearchEdit's OnChange }
{ event whenever the Incremental radio is selected. }
begin
  DM.Table1.FindNearest([SearchEdit.Text]); // search for text
end;

procedure TKeySearchForm.RBNormalClick(Sender: TObject);
begin
  ExactButton.Enabled := True;   // enable search buttons
  NearestButton.Enabled := True;
  SearchEdit.OnChange := Nil;    // unhook the OnChange event
end;

procedure TKeySearchForm.IncrementalClick(Sender: TObject);
begin
  ExactButton.Enabled := False;      // disable search buttons
  NearestButton.Enabled := False;
  SearchEdit.OnChange := NewSearch;  // hook the OnChange event
  NewSearch(Sender);                 // search current text
end;

procedure TKeySearchForm.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
  Action := caHide;
  MainForm.KeySearch1.Checked := False;
end;

end.
```

یونیت Filter نیز تحت عنوان Fltr نامگذاری شده است. لیست ۷-۵ کد مربوط به این یونیت را نشان می‌دهد.

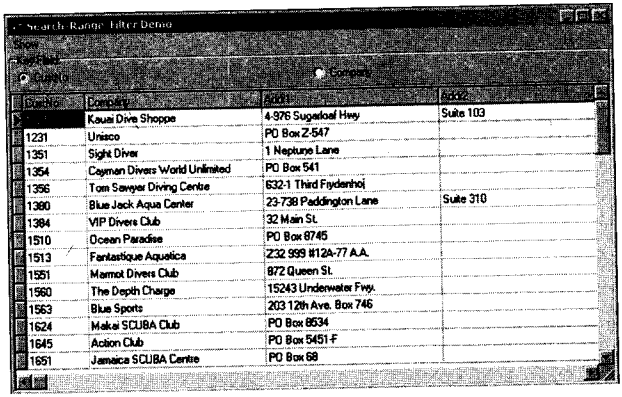
لیست ۷-۵ یونیت Fltr.Pas

```
unit Fltr;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons, Mask, DBCtrls, ExtCtrls;

type
  TFilterForm = class(TForm)
    Panel1: TPanel;
    Label4: TLabel;
    DBEdit1: TDBEdit;
    cbFiltered: TCheckBox;
    Label5: TLabel;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    Panel2: TPanel;
    EValue: TEdit;
    LocateBtn: TButton;
    Label1: TLabel;
    Label2: TLabel;
    CBField: TComboBox;
    MatchGB: TGroupBox;
    RBExact: TRadioButton;
    RBClosest: TRadioButton;
    CBCaseSens: TCheckBox;
    procedure cbFilteredClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure LocateBtnClick(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  end;

var
  FilterForm: TFilterForm;
implementation

uses DB, DataMod, Main;

{$R *.DFM}
```

لیست ۷-۵ ادامه

```
procedure TFilterForm.cbFilteredClick(Sender: TObject);
begin
  { Filter table if checkbox is checked }
  DM.Table1.Filtered := cbFiltered.Checked;
end;

procedure TFilterForm.FormCreate(Sender: TObject);
var
  i: integer;
begin
  with DM.Table1 do begin
    for i := 0 to FieldCount - 1 do
      CBField.Items.Add(Fields[i].FieldName);
  end;
end;

procedure TFilterForm.LocateBtnClick(Sender: TObject);
var
  LO: TLocateOptions;
begin
  LO := [];
  if not CBCaseSens.Checked then Include(LO, loCaseInsensitive);
  if RBClosest.Checked then Include(LO, loPartialKey);
  if not DM.Table1.Locate(CBField.Text, EValue.Text, LO) then
    MessageDlg('Unable to locate match', mtInformation, [mbOk], 0);
end;

procedure TFilterForm.SpeedButton1Click(Sender: TObject);
begin
  DM.Table1.FindFirst;
end;

procedure TFilterForm.SpeedButton2Click(Sender: TObject);
begin
  DM.Table1.FindNext;
end;

procedure TFilterForm.SpeedButton3Click(Sender: TObject);
begin
  DM.Table1.FindPrior;
end;

procedure TFilterForm.SpeedButton4Click(Sender: TObject);
begin
  DM.Table1.FindLast;
end;

procedure TFilterForm.FormClose(Sender: TObject; var Action: TCloseAction);
```

```
begin
  Action := caHide;
  MainForm.Filter1.Checked := False;
end;

end.
```



شکل ۱۱-۵ فرم اصلی پروژه SRF

خلاصه

در این فصل مفاهیم اساسی بانک‌های اطلاعاتی در دلفی و روشهای پردازش و مدیریت آنها را فرا گرفتید. توجه داشته باشید که جزء سازندهٔ TDatasource به صورت یک واسط بین TTable و سایر اجزاء دیگر عمل می‌کند. عناوین مربوط به کارآیی بانک‌های اطلاعاتی نظیر تعریف شاخص‌ها نیز از دیگر موارد ذکر شده در این فصل بودند که با یادگیری آنها این فصل را با ارائه یک مثال کاربردی (پروژه SRF) خاتمه دادیم.

فصل ششم

dbExpress

ابزاری برای طراحی بانک‌های اطلاعاتی

در این فصل می‌خوانید

- **کاربرد dbExpress**
- **اجزاء سازنده در dbExpress**
- **طراحی برنامه‌های کاربردی به وسیلهٔ dbExpress**

دلفی ۷ دربرگیرندهٔ ابزاری به نام dbExpress جهت ایجاد و طراحی بانک‌های اطلاعاتی است. dbExpress شرایط بهینه‌تری را نسبت به BDE (موتور بانک اطلاعاتی دلفی) برای مدیریت و پردازش بانک‌های اطلاعاتی داراست و از این رو مورد توجه بسیاری از برنامه‌سازان حرفه‌ای قرار گرفته است.

از طرف دیگر قابلیت سازگاری با سیستم عامل Linux در آن گنجانده شده و مهم‌تر از همه آنکه ابزاری قابل انعطاف و توسعه‌پذیر است.

ساختار داخلی dbExpress نیز بسیار ساده است. این ساختار حاوی تعدادی راه‌انداز جهت کار با بانک‌های اطلاعاتی است. تعدادی از این راه‌اندازها نیز وظیفه اتصال به منبع داده‌ها را به عهده دارند. به طور مثال شما با افزودن جزء سازنده DataCLX به برنامه خود امکان ارتباط با منبع داده‌ها را فراهم می‌کنید (همانند جزء سازنده TDatabase در BDE).

کار با dbExpress

همانطور که اشاره شد، dbExpress ابزاری بسیار کارآمد جهت دسترسی و مدیریت بانک‌های اطلاعاتی است. باید توجه داشت که دلیل عمدهٔ این کارآمدی، استفاده از مجموعه داده‌های یک سویه[1] می‌باشد.

۱- unidirectional datasets

مجموعه داده‌های یک سویه، فاقد عملیاتهای درج، حذف و پردازش در حافظه موقت[۱] هستند و اصولاً هیچ عملیاتی را در حافظه موقت انجام نمی‌دهند. در مقابل مجموعه داده‌های دو سویه[۲] که در BDE کاربرد دارند حاوی عملیاتهای پردازشی در حافظه موقت سیستم می‌باشند. با اینکه بکارگیری مزایای مجموعه داده‌های یک سویه در این dbExpress عمده‌ای را در برداشته است اما باید اذعان داشت که مجموعه‌ها دارای محدودیت‌هایی به شرح ذیل می‌باشد:

- این مجموعه داده‌ها تنها قادرند اعمالی نظیر First() و Next() را جهت حرکت در منبع داده‌ها انجام دهند و انجام اعمالی نظیر Prior() و Last() باعث بروز استثناء خواهد شد.
- این مجموعه‌ها قابل ویرایش نیستند. حتی اگر از اجزائی همچون TClientDataset و یا TSQLClientDataset استفاده نمایید باز هم امکان ویرایش وجود نخواهد داشت.
- عملیات فیلترگذاری بر روی این مجموعه از داده‌ها انجام‌پذیر نمی‌باشد.

کدام یک را بکار ببریم dbExpress یا BDE ؟

توجه داشته باشید که BDE بخش عمده‌ای از منابع موجود بر روی سرور بانک اطلاعاتی را در اختیار خود می‌گیرد. از طرفی بر روی دستگاه‌های سرویس‌گیرنده نیز به همین منوال خواهد بود و این در حالی است که dbExpress هیچ منبعی را بر روی سرور اشغال نمی‌کند و بر روی دستگاه‌های سرویس‌گیرنده نیز، جز در مواقعی نیاز به منابع خاص ندارد. حتماً درخواهید یافت که این ویژگی به خاطر استفاده از مجموعه داده‌های تک سویه و عدم بکارگیری حافظه موقت است. ضمناً یادآور می‌شویم که پرس‌وجوهای طراحی شده در dbExpress و همچنین کدهای SQL در زمان اجرا، سرعت اجرای بهتری خواهند داشت. اکنون خودتان انتخاب نمائید، به عنوان یک برنامه‌نویس مایلید که از کدام ابزار استفاده کنید، dbExpress یا BDE ؟

طراحی تحت سیستم‌عامل Linux

همانطور که در اول فصل اشاره کردیم dbExpress قابلیت طراحی و پیاده‌سازی تحت Linux را نیز داراست و به سادگی اضافه کردن یک جزء سازنده مانند CLX قادر خواهید بود تا برنامه‌تان را تحت سیستم عامل Linux اجرا نمائید.

اجزاء سازنده در dbExpress

جزء سازنده TSQLConnection

از این جزء همانند جزء TDatabase در BDE برای ارجاع به نام مستعار منبع داده‌ها یا بانک اطلاعاتی فیزیکی استفاده می‌شود. با استفاده از TSQLConnection شما قادر خواهید بود dbExpress را به منبع

۱- buffer ۲- bi-directional datasets

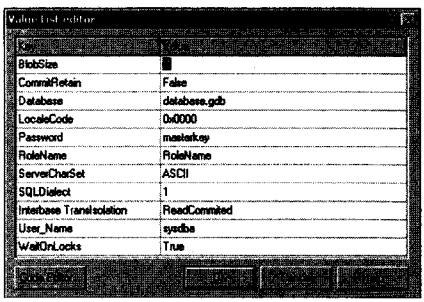
داده‌های خود مرتبط کنید.

راه‌اندازهای مربوط به جزء سازندهٔ TSQLConnection در دو فایل به نامهای dbxdrivers.ini و dbxconnections.ini قرار دارند. در صورت لزوم به دانستن محتویات آنها می‌توانید به فهرست "Program Files/Common Files/Borland Shares/DBExpress/" مراجعه نمایید (منظور محل نصب دلفی است).

dbxdrivers.ini وظیفه تنظیمات اولیه dbExpress و سایر راه‌اندازها را به عهده داشته و dbxconnetions.ini حاوی لیستی از نامهای مستعار جهت ارتباط با بانک‌های اطلاعاتی است. برای فعال ساختن dbxconnection.ini می‌بایست مقدار Ture را به خصوصیت TSQLConnection.LoadParamsOnConnect تخصیص داد. برای آشنایی با شرح کامل TSQLConnection به راهنمای دلفی رجوع کنید.

فراهم ساختن ارتباط با بانک اطلاعاتی

برای برقراری ارتباط با یک بانک اطلاعاتی که از قبل ایجاد شده است، می‌بایست دکمه ماوس را بر روی TSQLConnection برده و فشار دهید. اکنون کافی است خصوصیت ConnectionName را که بر روی Object Inspector قرار دارد با یکی ازمقادیر داده شده درلیست مربوط به‌آن مقداردهی نمائید. در این لیست چهار نوع ارتباط نمایش داده می‌شوند که عبارتند از DB2Connection,TBLcoal ، MSConnection و Oracle . با مقداردهی ConnectionName ، خواهید دید که خصوصیتهای GetDriverFunc ، LibraryName و VendorLib به صورت خودکار مقداردهی می‌شوند. مقادیر پیش‌فرضی که برای این پارامترها مشاهده می‌کنید همان مقدار مشخص شده در فایل dbxdriver.ini می‌باشد. ضمناً می‌توانید سایر پارامترها را از طریق خصوصیت TSQLConnection.Params مقداردهی نمائید (شکل ۱-۶).



شکل ۱-۶ ویراستار مربوط به خصوصیت TSQLConnection.Params

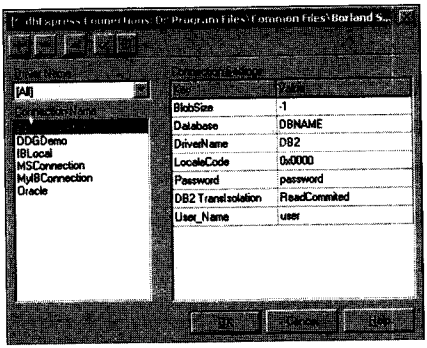
یادداشت

همانطور که در شکل ۱-۶ مشاهده می‌کنید، کلید "Database" برابر با database.gdb می‌باشد. لازم به ذکر است که این بانک اطلاعاتی بر روی سیستم شما وجود ندارد. در صورت نیاز می‌توانید آن را با "Employee.gbd" مقداردهی نمایید. این بانک اطلاعاتی در فهرست ذیل وجود دارد.

"...\ProgramFiles\Borland\Interbase6\examples\Database\Employee.gdb".

▲

گاهی اوقات لازم است برنامه کاربردی شما به چند منبع مجزا از داده‌ها دسترسی داشته باشد. به عبارت دیگر ممکن است بانک‌های اطلاعاتی شما به صورت مجزا و متفاوت از هم در نظر گرفته شده باشند. برای این ارتباط کافی است دکمه ماوس را بر روی جزء سازنده TSQLConnection دوبار بفشارید. در این حالت پنجره مربوط به ویراستار ارتباط با بانک اطلاعاتی فعال و بر روی صفحه ظاهر می‌گردد (شکل ۲-۶). ضمناً با فشردن دکمه سمت راست ماوس بر روی TSQLConnection و انتخاب "Edit Conection Properies" نیز می‌توانید این ویراستار را فراخوانی کنید.



شکل ۲-۶ ویراستار ارتباطی مرتبط با جزء سازنده TSQLConnection

تنظیمات مربوط به این ویراستار بسیار ساده است. در قسمت بالای این ویراستار پنج دکمه را مشاهده خواهید کرد.

برای برقراری ارتباط با ویراستار ارتباطی مرتبط با جزء سازنده TSQLConnection دکمه Add را بفشارید. سپس مقادیر لازم را به Driver Name و Connection Name تخصیص دهید. فراموش نکنید که علاوه بر تعیین کد کاربر و کلمه رمز برای فعال ساختن این ارتباط می‌بایست خصوصیت Connected را با Ture مقداردهی نمایید.

درخواست Login

دلفی ۷ شما را قادر می‌سازد تا از کاربران برای کار با بانک اطلاعاتی، کد و کلمه رمز درخواست نمایید. در صورتی که تمایلی به نمایش پنجره Login ندارید، می‌توانید خصوصیت LoginPrompt را برابر با False نمایید. در غیر این صورت علاوه بر تخصیص مقدار Ture به خصوصیت LoginPrompt لازم است قطعه کدی را در جهت دریافت و تست کد کاربر و کلمه رمز به برنامه خود اضافه نمایید. کد ذیل چگونگی این کار را نشان می‌دهد.

```
procedure TMainForm.SQLConnection1Login(Database: TSQLConnection;
  LoginParams: TStrings);
var
  UserName: String;
  Password: String;
begin
  if InputQuery('Get UserName', 'Enter UserName', UserName) then
  if InputQuery('Get Password', 'Enter Password', Password) then
  begin
```

```
    LoginParams.Values['UserName'] := UserName;
    LoginParams.Values['Password'] := Password;
  end;
end;
```

برقراری ارتباط در زمان اجرا

نحوه تنظیم ارتباطهایی که در بالا به آنها اشاره شد، همگی در فایل dbxconnections.ini و در زمان طراحی فرمها و برنامه‌ها فراخوانی می‌شوند. امکان تنظیم این ارتباطات در زمان اجرا نیز وجود دارد. برای این کار می‌بایست مقدار Ture را به خصوصیت LoadParamsOnConnect اختصاص دهید. در این صورت هنگامی که برنامه شما اجرا می‌شود، جزء سازنده TSQLConnection ارتباط لازم را برقرار نموده و تنظیمات را براساس فایل dbxconnection.ini به انجام می‌رساند.

جزء سازنده TSQLDataset

از این جزء جهت دستیابی به عناصر بانک اطلاعاتی استفاده می‌شود. این بانک‌ها می‌بایست سازگاری لازم را با dbExpress را داشته باشند. اعمالی نظیر نمایش اطلاعات موجود در بانک‌های اطلاعاتی و یا اجرای یک پرس‌وجوی خاص در منابع داده‌ها را می‌توان با کمک این جزء سازنده اجرا نمود.

دو خصوصیت مهم TSQLDataset ، CommandType و CommandText می‌باشند. این دو پارامتر ارتباط تنگاتنگی با هم دارند، بدین ترتیب که نحوه کاربری CommandText ، توسط مقادیر مشخص شده در CommandType تعیین و مشخص می‌گردد. جدول ۱-۶ نحوه این ارتباط را بهتر نمایش می‌دهد.

جدول ۱-۶ مقادیر معتبر برای CommandType

CommandType	مقدار متناظر در CommandText
ctQuery	کد SQL که بر روی منبع داده‌ها اجرا خواهد شد.
ctStoredProc	نام روالی که از قبل تهیه و ذخیره‌سازی شده است.
ctTable	نام جدولی که در یک سرویس‌دهنده بانک اطلاعاتی وجود دارد. SQL نوشته شده بر روی داده‌های موجود در این جدول اعمال خواهد شد.

همانطور که در جدول بالا اشاره شده است، اگر خصوصیت CommandType با مقدار ctQuery تنظیم گردد نمایانگر وجود یک کد SQL نظیر "SELECT * FROM CUSTOMER" در خصوصیت CommandText است.

اگر مقدار ctStoredProc را به خصوصیت CommandType تخصیص دهیم، آنگاه می‌توانیم یک روال از پیش تعیین شده را اجراء نماییم. برای اجرای این روال دو راه وجود دارد. یکی راه استفاده از متد TSQLDataset.ExecSQL() جهت اجرای روال موردنظر است. راه دیگر تخصیص دادن مقدار Ture

به خصوصیت Active می‌باشد.

دستیابی به داده‌ها

برای دسترسی به داده‌های موجود در یک جدول، تنها کافی است که TSQLDataset.CommandType را با مقدار ctTable تنظیم نمائیم.

نمایش پرس‌وجوها

همانطور که قبلاً نیز اشاره کردیم، برای اجرای یک پرس‌وجوی خاص می‌بایست TSQLDataset.CommandType را با مقدار ctQuery مقداردهی نمائیم. در این حالت باید پرس‌وجوی لازم را در خصوصیت CommandText تایپ کنیم.

اجرای زیر برنامه‌ها و روال‌ها

در اینجا زیربرنامه‌ای ارائه شده است که عملیات ساده‌ای نظیر درج رکورد جدید در یک جدول را انجام می‌دهد.

```
CREATE PROCEDURE ADD_COUNTRY (
  ICOUNTRY VARCHAR(15),
  ICURRENCY VARCHAR(10)
)  AS
BEGIN
  INSERT INTO COUNTRY(COUNTRY, CURRENCY)
  VALUES (:iCOUNTRY, :iCURRENCY);
  SUSPEND;
END
```

برای اجرای این روال باید روتین زیر را در برنامه خود اضافه نمائید.

```
procedure TForm1.btnAddCurrencyClick(Sender: TObject);
begin
  sqlDSAddCountry.ParamByName('ICountry').AsString := edtCountry.Text;
  sqlDSAddCountry.ParamByName('ICURRENCY').AsString := edtCurrency.Text;
  sqlDSAddCountry.ExecSQL(False);
end;
```

توجه داشته باشید که تمامی پارامترهای مورد نیاز جهت روتین ExecSQL() از نوع منطقی می‌باشند و به صورت پیش‌فرض برابر با Ture خواهند بود.

اطلاعات و ساختار بانک‌های اطلاعاتی

یکی از متداول‌ترین کارهای یک مدیر بانک اطلاعاتی، دانستن اطلاعات مربوط به ساختار جداول و داده‌های موجود است. با استفاده از روال TSQLDataset.SetSchemaInfo() به اطلاعاتی نظیر تعداد

سطر و ستون‌های جداول، شاخص‌ها و فیلترهای موجود بر روی جداول، دست می‌یابید.
الگوی استفاده از SetSchemaInfo به صورت زیر است:

```
Procedure SetSchemaInfo (SchemaType: TSchemaType;
SchemaObjectName, SchemaPattern: string):
```

در الگوی فوق، پارامتر SchemaObjectName معرف نام جدول موردنظر است.
پارامتر SchemaObjectName نیز می‌تواند حاوی یک کد SQL کوچک برای فیلترگذاری بر روی جداول باشد.
پارامتر SchemaType مشخص کننده نوع اطلاعات در مورد جداول است. مقادیر مربوط به این پارامتر در جدول ۲-۶ ارائه شده است.

جدول ۲-۶ مقادیر مربوط به پارامتر SchmaType

مقدار	شرح
StNoSchema	با انتخاب این پارامتر هیچگونه اطلاعاتی در مورد جداول در برنامه در دسترس نخواهد بود، منظور از این اطلاعات، اطلاعات مربوط به ساختار جداول، ایندکس‌ها، صفات، مشخصه و فیلدهای بانک اطلاعاتی است.
StsysTables	اطلاعات مربوط به جداول سیستمی موجود در سرویس‌دهنده بانک اطلاعاتی.
stProcedures	اطلاعات مربوط به زیربرنامه‌ها و روتین‌های موجود در سرویس‌دهنده بانک اطلاعاتی.
stColumns	اطلاعات مربوط به ستونهای جداول بانک اطلاعاتی.
stProcedureParams	اطلاعات مربوط به پارامترهای تخصیص داده شده به روتین‌ها و زیربرنامه.
stIndexes	اطلاعات مربوط به شاخصهای تعریف شده برای جداول بانک اطلاعاتی.

در لیست ۱-۶ با استفاده از دکمه‌های رادیویی، روتینی ارائه شده است که با استفاده از آن می‌توان نوع اطلاعات درخواستی را تعریف و تعیین نمود.

لیست ۱-۶ مثالی از TSQLDataset.SetSchemaInfo

```
procedure TMainForm.Button1Click(Sender: TObject);
begin
  sqldsSchemaInfo.Close;
  cdsSchemaInfo.Close;

  case RadioGroup1.ItemIndex of
  0: sqldsSchemaInfo.SetSchemaInfo(stSysTables, '', '');
  1: sqldsSchemaInfo.SetSchemaInfo(stTables, '', '');
  2: sqldsSchemaInfo.SetSchemaInfo(stProcedures, '', '');
```

لیست ۱-۶ ادامه

```
  3: sqldsSchemaInfo.SetSchemaInfo(stColumns, 'COUNTRY', '');
  4: sqldsSchemaInfo.SetSchemaInfo(stProcedureParams, 'ADD_COUNTRY', '');
  5: sqldsSchemaInfo.SetSchemaInfo(stIndexes, 'COUNTRY', '');
  end; // case

  sqldsSchemaInfo.Open;
  cdsSchemaInfo.Open;
end;
```

اجزایی با قابلیت‌های BDE

با بررسی تب dbExpress سه جزء سازنده دیگر به نامهای TSQLTable ، TSQLQuery ، TSQLStoredProc مشاهده می‌کنید. این اجزاء در BDE هم وجود داشتند، با این تفاوت که در dbExpress تنها قادرید بر روی داده‌های تک‌سویه تأثیرگذار باشید.

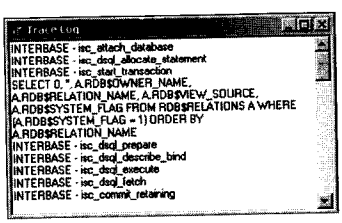
جزء سازنده TSQLMonitor

یکی از کارهای متداول برنامه‌نویسان، خطایابی و نحوه تست برنامه‌هایشان است.

TSQLMonitor جزء سازنده‌ای است که با استفاده از آن می‌توان عملیات خطایابی کدهای SQL را انجام داد. برای این کار لازم است تا TSQLMonitor.SQLConnection را با یک جزء سازنده TSQLConnection ارتباط داد. مراحل مربوط به پردازش SQL و خطایابی آن در خصوصیت TSQLMonitor.TraceList گنجانده می‌شود. بهتر است این لیست را در یک جزء سازنده مانند Memo ها قرار داده و به برنامه خود اضافه نمایید.

یادداشت

با استفاده از خصوصیت AutoSave, FileName می‌توانید لیست مربوط به خطاها را به صورت خودکار در یک فایل ذخیره نمایید.



نتیجه استفاده از جزء سازندهٔ TSQLMonitor برای یک SQL فرضی در شکل ۳-۶ نمایش داده شده است.

شکل ۳-۶ نتیجهٔ اجرای جزء سازندهٔ TSQLMonitor

قابلیت ویرایش در dbExpress

آنچه تابحال دیدیم مربوط به کارآیی ابزار dbExpress بر روی داده‌های تک سویه و فقط خواندنی بود.

در این میان به معرفی جزء سازنده TSQLDataset پرداختیم که با برخی از خصوصیتهای آن می‌توانستیم یک سطر را در یک جدول از داده‌ها درج نمائیم. در رابطه با مجموعه داده‌های دو سویه، جهت انجام اعمال ویرایش می‌توان از جزء سازنده TSQLClientDataset بهره گرفت.

جزء سازنده TSQLClientDataset

TSQLClientDataset دربرگیرنده دو جزء سازنده دیگر به نام TSQLDataset و TSQLProvider است.

TSQLDataset امکان دستیابی به داده‌ها رابرای TSQLClientDataset فراهم‌می‌آورد و TSQLProvider امکان ویرایش داده‌ها را محیا می‌سازد. توجه داشته باشید که در اینجا منظور از داده‌ها همان مجموعه داده‌های دو سویه است.

اما چگونه از TSQLClientDataset استفاده کنیم؟ برای اینکار باید دو جزء سازنده TDataSource و TSQLConnection و همچنین خود TSQLClientDataset را در یک فرم قراردهیم. سپس لازم است خصوصیت TSQLClientDataset.DBConnection را برابربا TSQLConnection نمائیم. خصوصیتهای CommandType و CommandText نیز همانند آنچه که قبل از این اشاره کردیم، کارآیی خواهند داشت.

اکنون قادرید کلیه عملیاتهای ویرایشی نظیر درج، حذف و تصحیح داده‌ها را در جداول بانک اطلاعاتی انجام دهید.

توجه داشته باشید که کلیه این تغییرات و ویرایشها در سطح حافظه موقت انجام می‌گیرد و برای ذخیره نمودن آن نیاز به فراخوانی متد TSQLClientDataset.ApplyUpdates() دارید.

ارائه برنامه‌های کاربردی با dbExpress

جالب خواهد بود اگر بدانید که با dbExpress قادرید برنامه‌های اجرائی را نیز ارائه نمایید. منظورمان از برنامه‌های اجرائی برنامه‌هایی است که بدون نیاز به کامپایلر دلفی و به صورت مستقل بر روی سیستم‌ها قابل اجرا می‌باشند. قابلیت ساخت فایل‌های DLL نیز در dbExpress گنجانده شده است. اما توجه داشته باشید که برای ایجاد یک برنامه اجرایی به صورت مستقل می‌بایست یونیتهای ذکر شده در جدول ۳-۶ را به برنامه خود اضافه کنید. در صورت نیاز به فایل‌های DLL به جدول ۴-۶ مراجعه نمایید.

جدول ۳-۶ یونیتهای مورد نیاز برای ساخت برنامه‌های Standalone

DLL بانک اطلاعاتی	شرح
dbExpInt	برنامه‌های کاربردی تحت InterBase
dbExpOra	برنامه‌های کاربردی تحت Oracle
dbExpDb2	برنامه‌های کاربردی تحت DB2
dbExpMy	برنامه‌های کاربردی تحت MySQL
Crtl, MidasLib	برنامه‌های کاربردی سرویس‌دهنده، سرویس‌گیرنده

جدول ۴-۶ فایل‌های DLL برای طراحی یک برنامه کاربردی با dbExpress

DLL بانک اطلاعاتی	شرح
dbexpint.dll	مرتبط ساختن یک برنامه با بانک‌های اطلاعاتی تحت InterBase
dbexpora.dll	مرتبط ساختن یک برنامه با بانک‌های اطلاعاتی تحت Oracle
dbexpdbz.dll	مرتبط ساختن یک برنامه با بانک‌های اطلاعاتی تحت BD2
dbexpmy.dll	مرتبط ساختن یک برنامه با بانک‌های اطلاعاتی تحت MySQL
Midas.dll	مرتبط ساختن یک برنامه با بانک‌های اطلاعاتی سرویس‌دهنده، سرویس‌گیرنده

خلاصه

در این فصل آموختید که dbExpress یک ابزار قدرتمند برای طراحی بانک‌های اطلاعاتی است. dbExpress ابزاری قابل انعطاف و توسعه‌پذیر است که شرایط بهینه‌تری را نسبت به BDE برای مدیریت بانک‌های اطلاعاتی فراهم می‌آورد.

فصل هفتم

طراحی بانک‌های اطلاعاتی ADO به کمک dbGO

در این فصل می‌خوانید

- مقدمه‌ای بر dbGO
- مروری بر استراتژی جامع دسترسی داده‌ای مایکروسافت
- مروری بر ODBC, ADO, OLE DB
- استفاده از dbGO در ADO
- dbGO برای اجزاء ADO
- پردازش تراکنش‌ها

مقدمه‌ای بر dbGO

در این فصل به بررسی برنامه‌نویسی با استفاده از اشیاء ADO مایکروسافت که در dbGO قرار داده شده‌اند خواهیم پرداخت.

dbGO مربوط به ADO توسط جزءهایی ارائه می‌شود که در برگه ADO در Component Palette قرار گرفته‌اند و دسترسی داده‌ای را از طریق چارچوب ADO عرضه می‌نمایند.

مروری بر استراتژی جامع دسترسی داده‌ای[1] مایکروسافت

استراتژی مایکروسافت در دسترسی داده‌ای عمومی به منظور میسر ساختن دسترسی به محدوده گسترده‌ای از داده‌ها از طریق یک مدل دسترسی ساده عرضه شده است. این داده‌ها ممکن است رابطه‌ای یا غیررابطه‌ای باشند. مایکروسافت این کار را به کمک اجزاء دسترسی داده‌ای[2] (*MDAC*) انجام می‌دهد که در تمامی سیستم‌های Windows 2000 نصب می‌شود و یا می‌توان آنها را از آدرس **http://ww.microsoft.com/data** برداشت.

۱- Universal Data Access

۲- Microsoft Data Access Component

MDAC شامل سه جزء است: OLE DB ، ADO و ODBC .

مروری بر OLE DB , ADO و ODBC

OLE DB یک رابط سیستمی است که برای دسترسی به انواع مختلف داده‌ای از جمله قالب‌های رابطه‌ای و غیر رابطه‌ای از COM استفاده می‌کند. می‌توان کدی نوشت که مستقیماً با لایه OLE DB ارتباط برقرار می‌کند. در حالی که با ADO این کار پیچیده‌تر و در اکثر حالات غیرضروری است.

بسیاری از عرضه کنندگان OLE DB پیاده‌سازی‌هایی از رابط‌های OLE DB هستند تا امکان دسترسی به داده‌های خاص را فراهم نمایند. به عنوان نمونه برخی عرضه کنندگان امکان دسترسی به داده‌ها را از طریق Paradox ، Oracle ، Microsoft SQL Server ، Microsoft Jet Engine و ODBC فراهم می‌نمایند.

ADO یک رابط سطح برنامه است که کاربران برای دسترسی داده‌ها از آن استفاده می‌نمایند. در حالی که OLE DB از بسیاری از رابط‌های مختلف (بیشتر از ۶۰ رابط) تشکیل شده است، ADO فقط از تعداد اندکی رابط تشکیل شده است. ADO در حقیقت از OLE DB به عنوان تکنولوژی دسترسی داده‌ای استفاده می‌کند.

ODBC پیشگامی برای OLE DB به شمار رفته و امروزه نیز مکانیزمی بسیار مفید است که توسط آن کاربران قادرند به داده‌های رابطه‌ای و برخی داده‌های غیررابطه‌ای دسترسی پیدا کنند.

استفاده از dbGO برای ADO

dbGO برای ADO تشکیل شده است از مجموعه‌ای از اجزاء Delphi که رابط ADO را پوشانده و آنها را به روشی مجرد برای توسعه بانک اطلاعاتی که در Delphi متداول است تطبیق می‌دهد. در بخش‌های بعد نحوه استفاده از این اجزاء را نشان خواهیم داد. در این فصل از یک بانک اطلاعاتی Microsoft Access از طریق یک عرضه کننده ODBC استفاده خواهیم نمود.

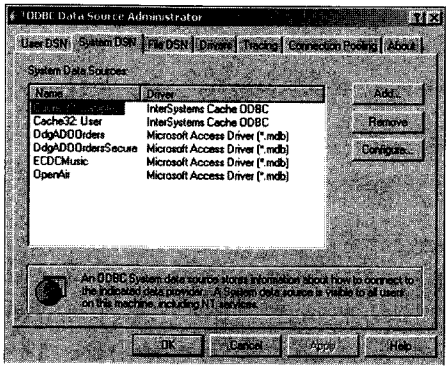
ایجاد یک عرضه کننده OLE DB برای ADO

برای ایجاد اتصال با بانک اطلاعاتی بایستی یک *ODBC Data Source Name* یا DSN ایجاد شود. DSN ها مشابه با نام‌های مستعار BDE هستند که به شما اجازه می‌دهند نقاط اتصالی سطح سیستمی را به همراه اطلاعات اتصالی برای بانک‌های اطلاعاتی که قابل دسترسی توسط کامپیوتر شما هستند ارائه نمایید.

برای ایجاد DSN ها بایستی از مدیر ODBC که در Windows قرار دارد استفاده شود. در Windows 2000 این بخش از داخل Control Panel و در زیرشاخه Administrative Tools قابل دسترسی است. هنگامی که این برنامه راه‌اندازی شود کادر گفتگوی شکل ۱-۷ ظاهر خواهد شد.

سه نوع DSN وجود دارد:

- DSN کاربر - Data Source های کاربر محدود به یک کامپیوتر بوده و فقط هنگامی قابل دسترسی



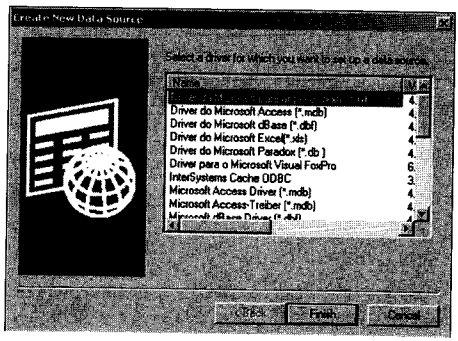
شکل ۱-۷ مدیر ODBC

هستند که به عنوان کاربر جاری وارد سیستم شویم.

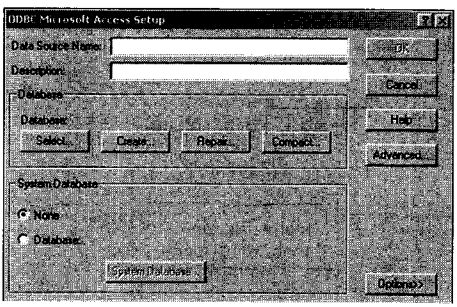
- DSN سیستمی- Data Source های سیستمی محدود به یک کامپیوتر هستند و برای همه کاربران قابل دسترسی می‌باشند. این Data Source ها برای تمامی کاربران مجاز قابل دسترسی هستند.
- DSN های فایل- Data Source های فایل برای تمامی کاربرانی که درایوهای مناسب فایل را نصب کرده‌اند قابل دسترسی می‌باشند.

در مثال مورد بحث یک DSN سیستمی ایجاد می‌شود. ابتدا مدیر ODBC را راه‌اندازی کنید. سپس برگه System DSN را انتخاب کرده و روی دکمه Add کلیک نمایید. با این کار کادر گفتگوی Create New Source شکل ۲-۷ باز می‌شود.

در این کادر گفتگو لیستی از درایوهای قابل دسترسی برای شما به نمایش در می‌آید. درایوی که به آن نیاز داریم Microsoft Access Driver (*.mdb) است. سپس روی Finish کلیک کنید. کادر گفتگوی ODBC Microsoft Access Setup (شکل ۳-۷) نمایان خواهد شد.



شکل ۲-۷ کادر گفتگوی
Create New Data Source



شکل ۳-۷ کادر گفتگوی
ODBC Microsoft Access Setup

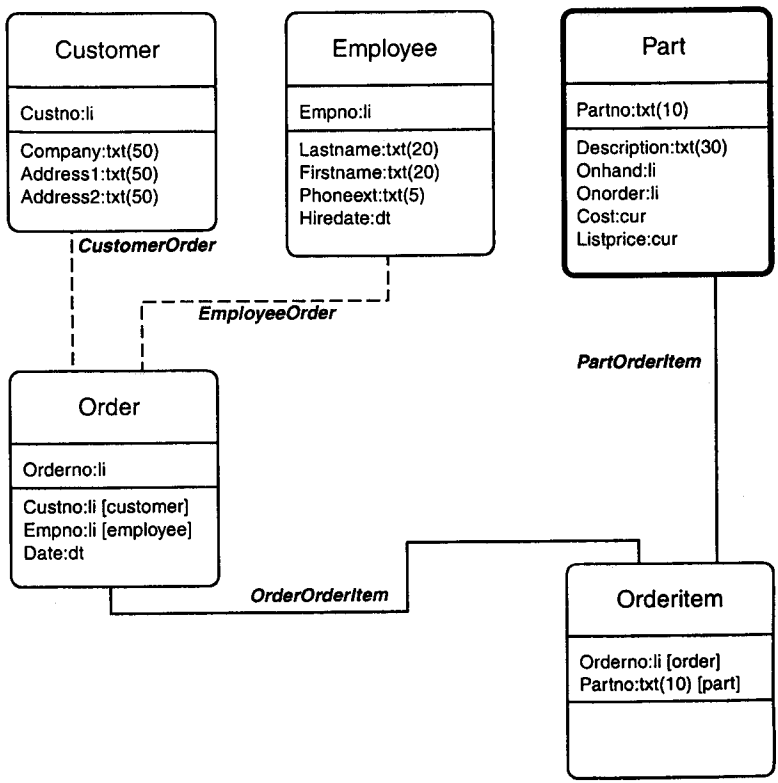
اکنون بایستی با کلیک کردن روی Select یک بانک اطلاعاتی را انتخاب نمایید. با انجام این کار یک کادرگفتگوی File Open باز شده و خواهید توانست یک فایل mdb.* را از درون آن انتخاب نمایید. فایلی که به کار خواهید برد ddgADO.mdb بوده و بایستی در شاخه‌ای به نام Delphi Developer's Guide\.. محلی که فایل‌های خود را در آن نصب کرده‌اید نصب شود. هنگامی که روی OK کلیک کنید DSN شما در لیست System Data Sourceهای قابل دسترسی ظاهر خواهد شد. اکنون می‌توانید روی OK کلیک کنید تا کار با مدیر ODBC خاتمه یابد.

بانک اطلاعاتی Access

بانک اطلاعاتی که برای آن یک DSN ایجاد کرده‌اید در شکل ۴-۷ نشان داده شده است.

این بانک اطلاعاتی یک بانک سفارش ساده است که برای تمرینات این فصل به کار می‌رود. هیچ نکته پیچیده‌ای در این بانک وجود نداشته و در حقیقت این بانک یک بانک کامل نیست.

جداولی چند را در کنار یکدیگر قرار داده و رابطه‌ای با معنی در آنها برقرار نموده‌ایم تا نحوه استفاده از dbGO برای جزءهای سازنده ADO مشخص شود.



شکل ۴-۷ یک بانک اطلاعاتی نمونه

dbGO برای جزءهای سازنده ADO

تمامی dbGOهای مربوط به اجزاء سازنده ADO دربرگه ADO بخش Component Palette ظاهر می‌شوند.

TADOConnection

TADOConnection شی اتصال ADO را عرضه می‌کند. از این جزء سازنده برای اتصال به ADO استفاده شده و از طریق آن دیگر جزءهای سازنده توسط data sourceهای ADO به کار گرفته می‌شوند. این جزء سازنده مشابه با TDatabase مربوط به اتصالات بانک اطلاعاتی BDE است. مشابه با TDatabase این جزء سازنده نیز اعمالی نظیر ورود و تراکنش را به کار می‌گیرد.

ایجاد یک اتصال بانک اطلاعاتی

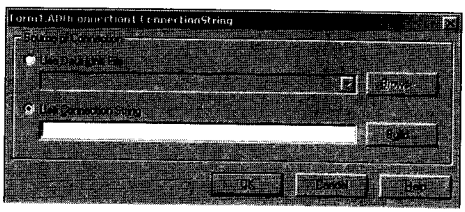
می‌توانید در صورت تمایل یک برنامه جدید ایجاد کنید و نحوه اتصال به یک بانک اطلاعاتی ADO را فراگیرید. کار با یک فرم حاوی جزء سازنده **TADOConnection** آغاز می‌شود. بایستی با کلیک کردن بر روی دکمه‌ای که روی خصوصیت **TADOConnection.ConnectionString** قرار دارد این صفت را تغیر دهید که با انجام این کار ConnectionString راه‌اندازی می‌شود. (شکل ۵-۷ را ببینید).

TConnectionString حاوی یک یا چند آرگومان است که ADO برای ایجاد یک اتصال با بانک اطلاعاتی به آن نیاز دارد. آرگومان‌های مورد نیاز بستگی به نوع OLE DB Provider مورد استفاده شما دارد.

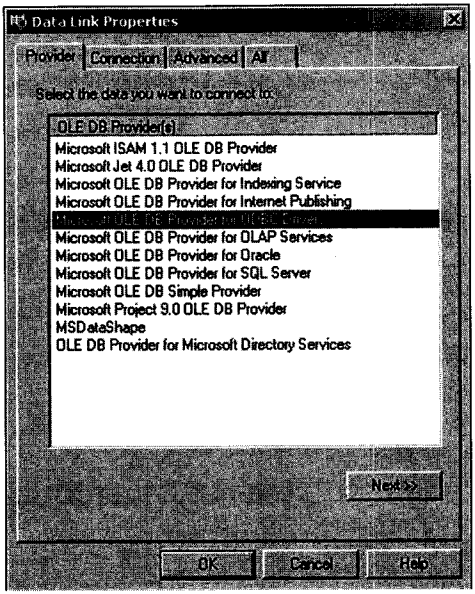
ویرایشگر **ConnectionString** با ایجاد یک Data Link File (فایلی که حاوی رشته اتصال می‌باشد) و یا ایجاد رشته اتصال که می‌توان بعداً آن را در فایل ذخیره کرد منبع اتصال را دریافت می‌نماید. قبلاً یک DSN ایجاد کرده‌اید. بنابراین رشته اتصالی ایجاد خواهید نمود که DSN شما را مرجع قرار می‌دهد. روی دکمه Bulid کلیک کنید تا کادر گفتگوی Data Link Properties راه‌اندازی شود (شکل ۶-۷ را ببینید).

اولین صفحه این کادر گفتگو به شما امکان می‌دهد که یک عرضه کننده OLE DB را انتخاب نمایید. در این حالت همانگونه که در شکل ۶-۷ دیده می‌شود گزینه Microsoft OLE DB For ODBC Drivers را انتخاب کنید. دکمه Next صفحه Connection Page را ظاهر می‌کند که از طریق آن می‌توانید DSN موردنظر را در لیست Data Source Name انتخاب نمایید (شکل ۷-۷).

هیچ امنیتی برای بانک اطلاعاتی برقرار نشده است بنابراین قادر خواهید بود که روی گزینه



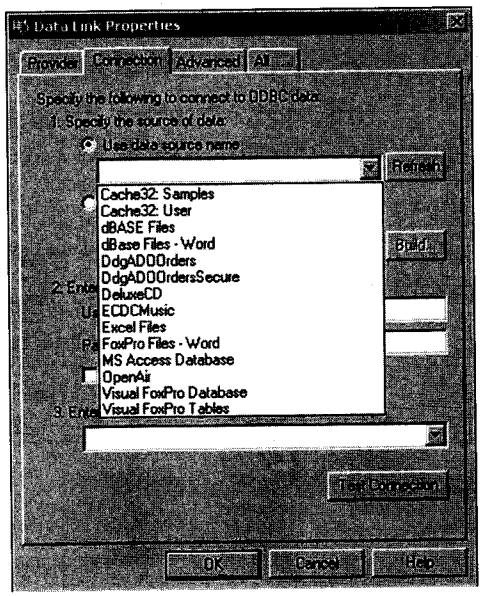
شکل ۵-۷ ویرایشگر خصوصیت TADOConnection.ConnectionString



شکل ٦-٧ کادر گفتگوی Data Link Properties

Text Connection کلیک کنید تا ارتباطی موفق با بانک اطلاعاتی برقرار نمایید. دوبار روی OK کلیک کنید تا به فرم اصلی باز گردید. رشته اتصالی که حاصل می‌شود در اینجا نشان داده شده است.

```
Provider=MSDASQL.1;Persist Security Info=False;Data Source=DdgADOOrder
```



شکل ۷-۷ انتخاب یک نام data source

اگر عرضه کننده OLE DB متفاوتی را به کار برده باشید رشته اتصال کاملاً متفاوت خواهد بود. بـه

عنوان نمونه اگر از عرضه کننده Microsoft Jet 4.0 OLE DB استفاده کرده باشید رشته اتصالی شما به شکل زیر خواهد بود:

```
Provider=Microsoft.Jet.OLEDB.4.0;Data Source="C:\Program Files\Delphi
➥Developer's Guide\Data\ddgADO.mdb";Persist Security Info=False
```

در این لحظه بایستی قادر باشید که با تنظیم **Connection.Connected** از TADO با مقدار **True** به بانک اطلاعاتی مورد بحث متصل شوید. یک مسیر Login ظاهر خواهد شد. روی OK کلیک کنید تا بدون نیاز به وارد کردن اطلاعات ورود اتصال را برقرار کنید. در بخش بعد نحوه بی‌اثر کردن Login و یا تغییر آن بررسی خواهد شد.

تغییر/بی‌اثر کردن Login Prompt

برای بی‌اثر کردن Login Prompt بایستی خصوصیت TADOConnection.LoginPrompt را به **False** تغییر دهید. اگر تنظیمی برای Login اعمال نشده است هیچ کار دیگری برای انجام لازم نیست. البته، اگر نام کاربر و کلمه عبور مورد نیاز است بایستی اعمال دیگری نیز انجام دهید.

■ نکته

می‌توانید با افزودن یک کلمه عبور به بانک اطلاعاتی این مسأله را بررسی کنید. برای انجام این کار می‌توان از Microsoft Access بهره گرفت. بایستی بانک اطلاعاتی را به صورت انحصاری باز کنید که این تنظیم در Tools ، Options ، Advancde Page در Access قرار دارد.

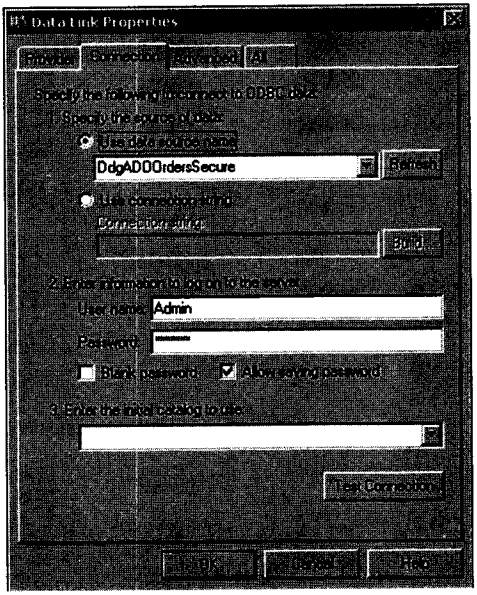
برای این تمرین یک DSN جدید ایجاد کرده‌ایم **(DdgADOOrdersSecture)** که به بانک اطلاعاتی مورد بحث **(ddgADOPW.mdb)** مراجعه می‌کند. اگر تمایل به انجام این تمرین دارید بایستی این DSN را ایجاد نمایید.

برای بی‌اثر کردن یک Login Prompt در یک بانک اطلاعاتی امن بایستی یک نام کاربر و کلمه عبور معتبر را در **ConnectionString** آماده کنید. این کار را می‌توان به صورت دستی و یا با بکارگیری ویرایشگر **ConnectionString** ، اضافه کردن نام کاربر و کلمه عبور صحیح، و علامتدار کردن Allow Saving Password انجام داد (شکل ۸-۷). اکنون **ConnectionString** به شکل زیر ظاهر می‌شود:

```
Provider=MSDASQL.1;Password=ddg;Persist Security Info=True;
➥User ID=Admin;Data Source=DdgADOOrdersSecure
```

به ظاهر شدن کلمه عبور و نام کاربر (Password و ID) توجه کنید. اکنون بایستی قادر باشید که خصوصیت **Connected** را به **True** تغییر دهید، در حالی که خصوصیت **LoginPrompt** مقدار **False** دارد.

تصور کنید که قصد دارید یک Login dialog دیگر آماده نمایید. در این حالت بایستی کلمه عبور را از



شکل ۸-۷ اضافه کردن نام کاربر و کلمه عبور بر ConnectionString

خصوصیت **ConnectionString** حذف نموده و یک رسیدگی کننده رویداد برای رویداد **TADOConnection.OnWillConnect** همانند لیست ۱-۷ ایجاد نمایید.

این مبادله ساده رد و بدل کردن نام و کلمه عبور را نشان می‌دهد.

لیست ۱-۷ رویداد Onwillconnect

```
procedure TForm1.ADOConnection1WillConnect(Connection: TADOConnection;
  var ConnectionString, UserID, Password: WideString;
  var ConnectOptions: TConnectOption; var EventStatus: TEventStatus);
var
  vUserID,
  vPassword: String;
begin
  if InputQuery('Provide User name', 'Enter User name', vUserID) then
    if InputQuery('Provide Password', 'Enter Password', vPassword) then
    begin
      UserID := vUserID;
      Password := vPassword;
    end;
end;
```

تذکر

ممکن است به نظر برسد که رویداد TADOComponent.OnLogin محلی است که بایستی یک نام و کلمه عبور در آن آماده شود تا منطبق با TDatasbase باقی بماند. البته رویداد TDOComponent.OnWillConnect برای این منظور رویداد استاندارد ADO را در خود دارد. کلمه OnLogin برای استفاده توسط کلاس TDispatchConnection عرضه شده است.

TADOCommand

جزء سازنده **TADOCommand** شی ADO Command را در خود جای داده است. این جزء سازنده برای اجرای دستوراتی به کار می‌رود که مجموعه نتایجی نظیر دستورات DDL و SQL را ارجاع نمی‌دهد. از این جزء سازنده برای اجرای دستورات SQL نظیر **INSERT** ، **DELETE** و **UPDATE** استفاده می‌شود. به عنوان نمونه مثالی را مطرح می‌نمائیم. این مثال مثالی ساده است که نحوه درج یا حذف یک رکورد را از جدول employee با کمک عبارات INSERT و DELETE در SQL شرح می‌دهد. در مثال مورد بحث **TADOCommand.CommandText** جزئی که در یک رکورد درج می‌شود حاوی عبارت SQL زیر است:

```
INSERT INTO EMPLOYEE (
LastName,
FirstName,
PhoneExt,
HireDate)

VALUES
(
'Smith',
 'Rob',
  '123',
  '12/28/1998')
```

برای اجرای دستور SQL بایستی متد **TADOCommand.Execute()** را به کار گیرید.

TADODataset

جزء سازنده **TADODataset** داده‌ها را از یک یا چند جدول بانک اطلاعاتی دریافت می‌کند. این جزء سازنده همچنین قادر است دستورات SQL و روال‌های تعریف شده توسط کاربر را اجراء نماید.

همانند جزء سازنده **TADOCommand** ، **TADODataset** نیز قادر است دستوراتی نظیر **INSERT** ، **DELETE** و **UPDATE** را انجام دهد. البته **TADODataset** قادر است با استفاده از دستور SELECT نیز نتایجی حاصل کند.

در مثال مورد بحث دستور SELECT زیر به بانک اطلاعاتی اعمال می‌شود:

SELECE *FROM Customer این دستور نتیجه را از جدول Customer ارجاع می‌دهد. همچنین می‌توانید از الگوهای فیلترسازی SQL نظیر **WHERE** نیز استفاده کنید.

در مثال مورد بحث جزء سازنده **TDBNavigator** به جزء سازنده **TADODataset** متصل شده است تا کارایی ویرایش و ناوبری جزء سازنده مشخص شود.

در ادامه فصل در مورد نحوه استفاده از جزء سازنده **TADODataSet** در یک برنامه نمونه توضیحات بیشتری ارائه خواهد شد.

اجزاء سازنده Dataset شبیه به BDE

برگه ADO در Component Palette سه جزء سازنده دارد که به منظور تبدیل ساده‌تر برنامه‌های BDE به ADO ضمیمه شده‌اند. این جزءهای سازنده عبارتند از **TADOTable** ، **TADOQuery** و **TADOStoredProc** . هنگام توسعه برنامه‌های ADO هیچ دلیلی مبنی بر استفاده محض از جزء سازنده **TADODataSet** وجود ندارد. البته در صورتی که این کار باعث ساده‌تر شدن عملیات می‌شود می‌توانید از این اجزاء که بسیار شبیه به اجزاء سازنده BDE یعنی **TTable** ، **TQuery** و **TStoredProc** هستند استفاده کنید.

TADOTable

TADOTable یک وارث مستقیم از **TCustomADODataSet** است. **TADOTable** به شما امکان می‌دهد که روی یک جدول بانک اطلاعاتی کار کنید. این جزء سازنده عملی بسیار شبیه به **TTable** در BDE دارد. در واقع **TADOTable** یک صفت **TableName** را اضافه می‌کند. یک مزیت آن نسبت به نوع جدول dataset آن است که اندیس را نیز پشتیبانی می‌نماید. اندیس‌ها امکان می‌دهند که عمل جستجو و ذخیره‌سازی با سرعت بیشتری انجام شود. البته هنگام استفاده از نوع SQL بانک اطلاعاتی بهتر است که ذخیره‌سازی، فیلتر کردن، و اعمال دیگر از طریق زبان SQL انجام شود. برای کسب اطلاعات بیشتر در مورد datasetها راهنمای Delphi را بررسی کنید.

تذکر

طبق بیان راهنمای Delphi یکی از مزایای استفاده از datasetهای نوع جدول سهولت خالی کردن جداول است. در مثالی که ارائه شده است از متد TCustomADODataSet.DeleteRecords() بهره گرفته شده است. البته مشکلی که در شیء ADORecordSet وجود دارد آن است که از انجام این کار جلوگیری می‌نماید.

```
TCustomADODataSet.Supports([coDelete])
```

این شیء مقدار True را ارجاع خواهد داد و DeleteRecords() هنوز ممکن است در حالت

استثنائی دچار مشکل شود. بنابراین برای خالی کردن یک جدول بایستی از عبارت DELETE FROM TableName استفاده کنید و یا با استفاده از حلقه، تک تک رکوردهای جدول را حذف کنید.

در مثالی که در زیر آمده است، استفاده از جزء سازنده **TADOTable** با یک اندیس شرح داده شده است. به علاوه در این مثال نحوه اجرای یک جستجو در جدول با استفاده از تابع **TADOTable.Locate()** بیان شده است. لیست ۲-۷ بخشی از کد را نشان می‌دهد.

رسیدگی کننده رویداد **FormCreate()** جدول را باز نموده و یک کنترل **TListBox** را به همراه تمامی نام‌های فیلد جدول در آن قرار می‌دهید. سپس در رسیدگی کننده رویداد **TListBox.OnClick** خصوصیت **TADOTable.IndexFieldName** را با نام فیلدی که می‌خواهید جدول را روی آن تنظیم کنید مقداردهی می‌نماید.

در پایان، یک رویداد **ButtonClick()** اجرای یک جستجو روی جدول را با استفاده از متد **Locate()** مشخص می‌نماید.

TADOTable برای اشخاصی که با جزء سازنده **TTable** خو گرفته‌اند مناسب است. البته هنگام استفاده از بانک‌های اطلاعاتی SQL بهتر است که از اجزاء سازنده **TADODataSet** یا **TADOQuery** استفاده شود.

لیست ۲-۷ استفاده از جزء سازنده TADOTable

```
procedure TForm1.FormCreate(Sender: TObject);
var
  i: integer;
begin
  adotblCustomer.Open;
  for i := 0 to adotblCustomer.FieldCount - 1 do
    ListBox1.Items.Add(adotblCustomer.Fields[i].FieldName);
end;

procedure TForm1.ListBox1Click(Sender: TObject);
begin
  adotblCustomer.IndexFieldNames := ListBox1.Items[ListBox1.ItemIndex];
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  adotblCustomer.Locate('Company', Edit1.Text, [loPartialKey]);
end;
```

TADOQuery

TADOQuery نیز وارثی از **TCustomADODataSet** بوده و شباهت بسیاری بـه **TADODataSet** دارد. **TADOQuery** دارای یک صفت SQL است که در آن دستور SQL خود را قرار مـی‌دهیم. جـزء سازنده **TADODataSet** هنگامی که **TADODataSet.Command** با **cmdText** تنظیم شده است به خصوصیت **CommandText** منتهی می‌شود.

TADOStoredProc

جزء سازنده **TADOStredProc** به شما امکان می‌دهد که از روال ذخیره شده در یک بانک اطلاعاتی روی سرور استفاده نمایید. استفاده از این جزء تفاوتی با استفاده از جزء سازنده **TADOCommand** که خصوصیت **CommandType** آن با **cmdStoredProc** تنظیم شده است ندارد.

پردازش تراکنش‌ها

ADO پردازش تراکنش‌ها را پشتیبانی می‌نماید و این مسأله از طریق جزء سازنده **TADOConnection** به کار گرفته می‌شود. به عنوان مثال، کد درون لیست ۳-۷ از برنامه ورود سفارشات که قبلاً ارائه شده برگرفته شده است.

متد درون لیست ۳-۷ عمل ایجاد یک سفارش مشتری را انجام می‌دهد. دو بخش در ایـن بـرنامه وجود دارند. ابتدا رکورد سفارشی بایستی در جدول Order ایجاد شود. سپس آیـتم‌های سـفارش در جدول OrderItem اضافه می‌شوند. از آنجا که دو بهنگام‌سازی در جدول داریم بهتر است آن را در یک تراکنش قرار دهیم.

لیست ۳-۷ برنامهٔ ورود سفارشات

```
procedure TMainForm.Button1Click(Sender: TObject);
begin
  if TNewOrderForm.Execute then
  begin
    ADOConnection1.BeginTrans;
    try
      // First Create an Orders Record
      adodsOrders.Insert;
      adodsOrders.FieldByName('CustNo').Value :=
        adodsCustomer.FieldByName('CustNo').Value;
      adodsOrders.FieldByName('EmpNo').Value :=
        adodsEmployee.FieldByName('EmpNo').Value;
      adodsOrders.FieldByName('Date').Value := Date;
      ShowMessage(IntToStr(adodsOrders.FieldByName('OrderNo').AsInteger));
      adodsOrders.Post;
```

لیست ۳-۷ ادامه

```
    // Now create the Order Line Items.

    cdsPartList.First;
    while not cdsPartList.Eof do
    begin
      adocmdInsertOrderItem.Parameters.ParamByName('iOrderNo').Value :=
        adodsOrders.FieldByName('OrderNo').Value;
      adocmdInsertOrderItem.Parameters.ParamByName('iPartNo').Value :=
        cdsPartListPartNo.Value;
      adocmdInsertOrderItem.Execute;
      cdsPartList.Next;
    end;
    adodsOrderItemList.Requery([]);
    ADOConnection1.CommitTrans;
    cdsPartList.EmptyDataSet;
  except
    ADOConnection1.RollbackTrans;
    raise;
  end;
 end;
end;
```

اسکلت تراکنش در اینجا ارائه شده است:

```
begin
  ADOConnection1.BeginTrans;
  try
    // First Create an Orders Record
    // Now create the Order Line Items.
    ADOConnection1.CommitTrans;
  except
    ADOConnection1.RollbackTrans;
    raise;
  end;
 end;
end;
```

خواهید دید که تراکنش خود را در بلوک **try ... except** قرار می‌دهیم. متد **Connection1.BeginTrans()** از ADO تراکنش را آغاز می‌نماید. متد **ADOConnection1.CommitTrans()** عمل تراکنش را به انجام می‌رساند. اگر اشکالی رخ دهد یک استثناء رخ داده و متد **ADOConnection.RollbackTrans()** تغییراتی که در جداول رخ داده‌اند را باز می‌گرداند.

خلاصه

در این فصل کار با اجزاء **dbGO** را برای ADO آغاز نمودید. این اجزاء سازنده به شما امکان استفاده از تکنولوژی ADO مایکروسافت را برای دسترسی به داده‌های رابطه‌ای و غیررابطه‌ای عرضه می‌نمایند.

فصل هشتم

ساخت اجزاء سازنده VCL

در این فصل می‌خوانید

- **اصول ساخت اجزاء سازنده**
- **طراحی و پیاده‌سازی اجزاء سازنده**
- **الحاق اجزاء سازنده**

همچنان که نرم‌افزارها پیچیده و گسترده‌تر می‌شوند، کاربرد اجزاء سازنده در تولید برنامه‌های کاربردی نیز جلوه‌ای مؤثرتر پیدامی‌کند. دلفی ابزار تولید فوق‌العاده‌ای برای ایجاد اجزاء سازنده است که می‌توان آنها را در نرم‌افزارهای مختلف بکار برد. اجزاء سازنده را می‌توان بدون نگرانی ساخت و آنها را در محیطهایی همچون دلفی، ویژوال بیسیک، صفحات وب و هر محصول دیگری که از میزبانی اجزاء سازنده پشتیبانی می‌کند، نصب نمود.

در این فصل، کار خود با بررسی اصول و مفاهیم اجزاء سازنده و برخی از دلائلی که ممکن است اقدام به نوشتن اجزاء سازنده کنید آغاز نموده و سپس مراحل ایجاد اجزاء سازنده ویژوال را شرح خواهیم داد.

مفاهیم و اصول اولیه طراحی اجزاء سازنده

در این بخش دلائل طراحی اجزاء سازنده و نحوهٔ پیاده‌سازی و مراحل ایجاد آنها را بررسی خواهیم نمود.

چرا باید اجزاء سازنده نوشت

تمام برنامه‌نویسانی که از اجزاء سازنده استفاده می‌کنند، به قدرت آنها در طراحی برنامه‌ها واقفند. توجه داشته باشید که اجزاء سازنده جزئیات پیاده‌سازی را از شما پنهان می‌سازند. به طور مثال تولید کننده یک برنامهٔ اینترنتی برای بکارگیری یک جزء سازنده نیازی به دانستن شیوهٔ عملکرد آن ندارد، بلکه تنها لازم است تا نحوهٔ برقراری ارتباط با آن را آموخته باشد. چند دلیل عمده برای نوشتن اجزاء سازنده

وجود دارد که در زیر به آنها اشاره خواهیم نمود.

- اگر در برنامه‌هایتان به طور مرتب از یک شیء استفاده می‌کنید، به راحتی می‌توانید آن را به جزء سازنده تبدیل نموده و به جعبهٔ ابزار برنامه‌تان اضافه کنید. اینکار به کارگیری جزء سازنده را آسانتر نموده و جزئیات پیاده‌سازی آن را در کتابخانه پنهان می‌سازد.
- با به کارگیری اجزاء سازنده در طراحی برنامه‌هایتان قادرید تا قابلیت کارآیی در شرایط بحرانی را به آنها بیفزائید.
- چنانچه مجبور به افزودن قابلیت‌های کاربردی خاص به برنامه‌های خود هستید، می‌توانید یک جزء سازنده که از قبل طراحی شده است را تهیه نمائید. این یکی از مهمترین مزایای سیستم‌های مبتنی بر اجزاء سازنده است.
- از آنجاکه امروزه اکثر زبانهای برنامه‌نویسی شیءگرا می‌باشند و هر جزء سازنده نیز یک شیء است، به راحتی می‌توانید یک جزء سازنده ایجاد کنید (که می‌توانید به عنوان زیرکلاسی از شیءهای موجود باشد) و آن را به عنوان کلاس‌های والد برای سایر اشیاء بکار ببرید.

نوشتن اجزاء سازنده

پیش از آنکه وارد مرحلهٔ ایجاد اجزاء سازنده شوید، ابتدا نگاهی مختصر به مراحل ساخت این اجزاء خواهیم داشت. این مراحل به شرح زیر خواهند بود.

۱- ضرورت ایجاد یک جزء سازنده و تعیین روش عملکرد آن در برنامه‌ها.
۲- ارائه یک الگوریتم کارآمد برای پیاده‌سازی آن.
۳- تعیین و تعریف نیازمندیها و تست و خطایابی دستی آن.
۴- پیاده‌سازی جزء سازنده.
۵- تست و خطایابی آن توسط سیستم.
۶- افزودن آن به جدول اجزاء سازنده.

این مراحل در دلفی ۷ به شرح زیر خواهند بود:

۱- استفاده از یک کلاس والد برای جزء سازنده.
۲- ایجاد یونیت برای جزء سازنده.
۳- پیاده‌سازی متدها، خصوصیت‌ها و رویدادهای مربوط به جزء سازنده.
۴- تست و خطایابی جزء سازنده.
۵- ثبت جزء سازنده در محیط دلفی.
۶- تولید یک فایل راهنما برای جزء سازنده.

در زیر به تشریح این مراحل خواهیم پرداخت.

استفاده از یک کلاس والد برای جزء سازنده

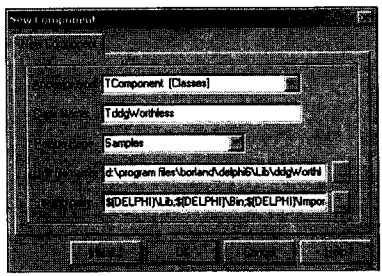
برای راحتی کار جدول مربوط به کلاس‌های والد در دلفی را در این قسمت ارائه کرده‌ایم.

جدول ۱-۸ کلاس‌های VCL که به عنوان کلاس‌های والد برای اجزاء سازنده مطرح می‌باشند

نام کلاس VCL	شرح
TObject	چنانچه کلاس والدی مشخص نشود، تمام کلاس‌های دلفی زیرکلاسی از TObject خواهند شد.
TComponent	این کلاس به عنوان کلاس والد برای اجزاء سازندهٔ غیربصری مطرح می‌باشد.
TGraphicControl	این کلاس به عنوان کلاس والد برای اجزاء سازنده‌ای است که از پنجره‌ها برای عملکرد خود استفاده نمی‌کنند.
TWinControl	کلاس والد برای اجزاء سازنده‌ای که از پنجره‌ها برای عملکرد خود استفاده می‌کنند.
TCustomControl	این کلاس از کلاس TWinControl مشتق شده و به عنوان کلاس والد برای متدهایی همچون Paint() مطرح می‌باشد.
TComponentName	این کلاس مشابه با کلاس‌های TEdit ، TPanel و TScrollBox بوده و کارآیی متناظر با آنها دارد.

ایجاد یونیت برای جزء سازنده

بهترین روش برای ایجاد یک یونیت جهت اجزاء سازنده استفاده از Component Expert است.



شکل ۱-۸ Component Export

این کار با انتخاب گزینهٔ New Component از منوی Component امکان‌پذیر می‌باشد. اولین گزینه، همانطور که در شکل مشخص است مربوط به تعیین نوع کلاس والد است. نام یونیت نیز یکی از گزینه‌های موجود بر روی این فرم می‌باشد. پس از فشردن دکمهٔ OK ، دلفی یونیت موردنظر را به صورت خودکار تولید می‌نماید که این کد در لیست ۱-۸ ارائه شده است.

لیست ۱-۸ مثالی برای یک جزء سازنده

```
unit Worthless;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs;
type
  TddgWorthless = class(TCustomControl)
  private
    { Private declarations }
  protected
    { Protected declarations }
  public
    { Public declarations }
  published
    { Published declarations }
  end;
procedure Register;
implementation
procedure Register;
begin
  RegisterComponents('DDG', [TddgWorthless]);
end;
end.
```

تعیین خصوصیت اجزاء سازنده

در این قسمت نحوهٔ تعریف خصوصیتها و تخصیص آنها را به اجزاء سازنده تشریح خواهیم نمود.

انواع خصوصیتها

بدون توضیح اضافی به تعریف چند خصوصیت جهت کلاس فرضی TddgWorthless (شکل ۱-۸) می‌پردازیم.

تعریف خصوصیتها در قالب داده‌های اولیه

منظوراز خصوصیتهای ساده، خصوصیتهایی است که در قالب داده‌های اولیه نظیر متغیرهای عددی، کاراکتری و رشته‌ای تعریف می‌گردند.

خصوصیتهای ساده همچون متغیرها می‌توانند در انواع کاراکتری، عددی و یا رشته‌ای تعریف گردند. این خصوصیتها از طریق Object Inspector دلفی مستقیماً قابل ویرایش خواهند بود. لیست ۲-۸ تعریف سه خصوصیت را برای کلاس TddgWorthless نمایش می‌دهد.

لیست ۲-۸ تعریف خصوصیتهای ساده

```
TddgWorthless = class(TCustomControl)
  private
    // Internal Data Storage
    FIntegerProp: Integer;
    FStringProp: String;
    FCharProp: Char;
published
    // Simple property types
    property IntegerProp: Integer read FIntegerProp write FIntegerProp;
    property StringProp: String read FStringProp write FStringProp;
    property CharProp: Char read FCharProp write FCharProp;
  end;
```

یادداشت

همانطور که در کد فوق ملاحظه می‌کنید، نام فیلدهای موجود در بخش Private با حرف F شروع شده‌اند. این تعریف برای اجزاء سازنده و انواع عمومی داده‌ها با حرف T نمایان خواهد شد. بهتر خواهد بود که شما هم جهت خواناتر بودن کدهایتان از این قانون پیروی نمائید. ▲

تعریف خصوصیتها در قالب داده‌های شمارشی

تعریف و تخصیص این نوع از خصوصیتها را با ارائه مثالی برای کلاس TddgWorthless نشان خواهیم داد. برای کلاس TddgWorthless ابتدا یک نوع داده شمارشی به شکل زیر تعریف کنید:

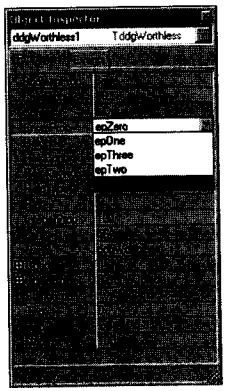
```
TEnumProp = (epZero, epOne, epTwo, epThree);
```

و سپس کد ارائه شده در لیست ۳-۸ را به این کلاس اختصاص دهید:

اکنون پس از کامپایل و نصب این جزء سازنده، شکلی همانند شکل ۲-۸ را برای Object Inspector این جزء سازنده مشاهده خواهید کرد.

لیست ۳-۸ تعریف خصوصیتهای شمارشی

```
TddgWorthless = class(TCustomControl)
  private
    // Enumerated data types
    FEnumProp: TEnumProp;
    FBooleanProp: Boolean;
published
    property EnumProp: TEnumProp read FEnumProp write FEnumProp;
    property BooleanProp: Boolean read FBooleanProp write FBooleanProp;
  end;
```



شکل ۲-۸ نحوهٔ نمایش خصوصیتهای شمارشی در Object Inspector کلاس TddgWorthless

تعریف خصوصیتها در قالب داده‌های مجموعه‌ای

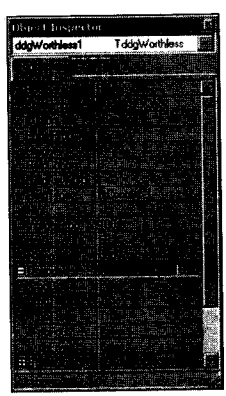
نحوهٔ تعریف این نوع از خصوصیتها نیز همانند تعریف مجموعه‌ها در پاسکال است. برای افزودن این خصوصیتها به کلاس TddgWorthless ابتدا یک نوع داده در قالب مجموعه‌ها به شکل (به عنوان مثال) زیر تعریف می‌گردد:

```
TsetPropOption = (poOne, poTwo, poThree, poFour, poFive);
TsetPropOptions = set of TsetPropOption;
```

اکنون کد زیر را به کلاس TddgWorthless اضافه نموده و سپس آن را کامپایل و نصب کنید.

```
TddgWorthless = class(TCustomControl) s
 private
  FOptions: TSetPropOptions;
published
  property Options: TSetPropOptions read FOptions write FOptions;
end;
```

Object Inspector مربوط به این کلاس در شکل ۳-۸ نشان داده شده است.



شکل ۳-۸ نحوهٔ نمایش خصوصیتهای مجموعه‌ای در Object Inspector کلاس TddgWorthless

تعریف خصوصیتها در قالب اشیاء

یک شیء می‌تواند به عنوان یک خاصیت نیز برای سایر اشیاء معرفی گردد. عموماً اشیائی که به عنوان خصوصیت تعریف می‌شوند از کلاس Tpersistent مشتق خواهند شد. در این قسمت با ارائه یک مثال، نحوهٔ تعریف این نوع از خصوصیتها را تشریح می‌نمائیم.

قبل از هر چیز می‌بایست یک شیء تعریف نمائیم. این تعریف برای نمونه در لیست ۴-۸ ارائه گردیده است.

لیست ۴-۸ تعریف شیء TSomeObject

```
TSomeObject = class(TPersistent)
  private
    FProp1: Integer;
    FProp2: String;
  public
    procedure Assign(Source: TPersistent);
  published
    property Prop1: Integer read FProp1 write FProp1;
    property Prop2: String read FProp2 write FProp2;
  end;
```

توجه داشته باشید که این کد تنها مثالی برای تعریف یک شیء می‌باشد. همانطور که در کد فوق ملاحظه می‌کنید شیء TSomeObject از کلاس TPersistent اشتقاق یافته است. نحوهٔ تعریف این شیء به عنوان یک خصوصیت از کلاس TddgWorthless در لیست ۵-۸ ارائه گردیده است.

لیست ۵-۸ تعریف خصوصیتها در قالب اشیاء

```
TddgWorthless = class(TCustomControl)
private
  FSomeObject: TSomeObject;
  procedure SetSomeObject(Value: TSomeObject);
public
  constructor Create(AOwner: TComponent); override;
  destructor Destroy; override;
published
  property SomeObject: TSomeObject read FSomeObject write SetSomeObject;
end;
```

همانطور که ملاحظه می‌کنید در کد فوق از دو متد به نامهای ()Create و ()Destroy نیز استفاده کرده‌ایم. تعریف این متدها در زیر ارائه شده است.

```
constructor TddgWorthless.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
```

```
  FSomeObject := TSomeObject.Create;
end;
destructor TddgWorthless.Destroy;
begin
  FSomeObject.Free;
  inherited Destroy;
end;
```

نحوهٔ مراجعه به این خصوصیت نیز به شکل زیر خواهد بود:

```
procedure TddgWorthLess.SetSomeObject(Value: TSomeObject);
begin
  if Assigned(Value) then
    FSomeObject.Assign(Value);
end;
```

متد TSomeObject.Assign() هم به صورت زیر پیاده‌سازی می‌گردد.

```
procedure TSomeObject.Assign(Source: TPersistent);
begin
  if Source is TSomeObject then
  begin
    FProp1 := TSomeObject(Source).Prop1;
    FProp2 := TSomeObject(Source).Prop2;
    inherited Assign(Source);
  end;
end;
```

تعریف خصوصیتها در قالب آرایه‌ها

نحوهٔ تعریف این نوع از خصوصیتها را نیز با ارائهٔ یک مثال نشان خواهیم داد. کلاسی تحت عنوان TddgPlanets را با دو خصوصیت به نامهای PlanetName و PlanetPosition.PlanetName در نظر می‌گیریم. این کلاس نمایانگر نام سیارات و شمارهٔ (که به صورت آرایه تعریف خواهد شد) مربوط به آنها می‌باشد. می‌خواهیم این کلاس و خصوصیتهای آن را طوری طراحی کنیم که هر سیاره با یک شماره منحصر به فرد شناسایی شود. به عنوان مثال با اجرای کد زیر پیامی تحت عنوان "Neptune" نمایان گردد.

```
ShowMessage (ddgPlanets.PlanetName [8]);
```

یا رشته From the sun, Neptune is planet number: 8 به وسیله اجرای کد زیر به نمایش درآید.

```
ShowMessage('From the sun, Neptune is planet number: '+
  IntToStr(ddgPlanets.PlanetPosition['Neptune']));
```

پیاده‌سازی جزء سازندهٔ TddgPlanets به طور کامل در لیست ۶-۸ آورده شده است.

لیست ٦-٨ تعریف خصوصیتها در قالب آرایه‌ها برای کلاس TddgPlantes

```
unit planets;

interface

uses
```

لیست ۶-۸ ادامه

```
  Classes, SysUtils;

type

  TddgPlanets = class(TComponent)
  private
    // Array property access methods
    function GetPlanetName(const AIndex: Integer): String;
    function GetPlanetPosition(const APlanetName: String): Integer;
  public
    { Array property indexed by an integer value. This will be the default
      array property.  }
    property PlanetName[const AIndex: Integer]: String
        read GetPlanetName; default;
    // Array property index by a string value
    property PlanetPosition[const APlantetName: String]: Integer
        read GetPlanetPosition;
  end;

implementation

const
  // Declare a constant array containing planet names
  PlanetNames: array[1..9] of String[7] =
   ('Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn',
    'Uranus', 'Neptune', 'Pluto');

function TddgPlanets.GetPlanetName(const AIndex: Integer): String;
begin
  { Return the name of the planet specified by Index. If Index is
    out of the range, then raise an exception }
  if (AIndex < 0) or (AIndex > 9) then
    raise Exception.Create('Wrong Planet number, enter a number 1-9')
  else
    Result := PlanetNames[AIndex];
end;

function TddgPlanets.GetPlanetPosition(const APlanetName: String): Integer;
var
  i: integer;
begin
  Result := 0;
  i := 0;
  { Compare PName to each planet name and return the index of the
    appropriate position where PName appears in the constant array.
    Otherwise return zero. }
  repeat
    inc(i);
  until (i = 10) or (CompareStr(UpperCase(APlanetName),
        UpperCase(PlanetNames[i])) = 0);
```

```
  if i <> 10 then // A Planet name was found
    Result := i;
end;

end.
```

تعریف مقادیر اولیه (پیش‌فرض)

دلفی از تعیین مقادیر اولیه (پیش‌فرض) برای خصوصیتهای کلاس‌ها نیز پشتیبانی می‌کند. این تعریف در قسمت سازندهٔ کلاسها (Constructor) انجام خواهد پذیرفت. به عنوان مثال می‌توان خصوصیت FIntegerProp را به شکل زیر در سازندهٔ کلاس TddgWorthless وارد نمود.

```
FIntegerProp := 100;
```

این تعریف می‌تواند به نحوهٔ دیگری نیز انجام پذیرد. در زیر نمونهٔ دیگری از تعریف مقادیر پیش‌فرض با استفاده از کلمهٔ کلیدی default ارائه شده است.

```
Property Tag: Longint read FTag write FTag default 0;
```

اما چنانچه خواسته باشید تا مقدار پیش‌فرض معرفی نکنید می‌بایست از کلمهٔ کلیدی NoDefault استفاده کنید. به نمونهٔ زیر در این رابطه توجه کنید:

```
TSample = class(TComponent)
published
  property Tag NoDefault;
```

تعریف مقادیر اولیه برای آرایه‌ها

در دلفی، تعریف مقادیر اولیه برای آرایه‌ها هم امکان‌پذیر است. نحوهٔ این تعاریف همچون تعاریف مقادیر اولیه برای متغیرهاست.

ایجاد و ساخت رویدادها

تا اینجا نحوهٔ تعریف خصوصیتها و چگونگی افزودن آنها به اجزاء سازنده را آموخته‌اید. مرحلهٔ بعدی فراگیری روشهای تعریف و ایجاد رویدادها برای اجزاء سازنده است. در این قسمت به بررسی و ارائه راه‌کارهای لازم جهت این کار خواهیم پرداخت.

رویدادها

رویدادها به کاربران یا تولیدکنندگان نرم‌افزار امکان می‌دهند که هنگام وقوع یک امر، جزء سازنده را فعال سازند. به عنوان مثال، رویداد OnClick به معنی انجام عملیات در هنگام انتخاب جزء سازنده به وسیله ماوس می‌باشد. رویداد OnClick (یا متد ()Click) به صورت زیر در دلفی پیاده‌سازی گردیده است.

```
TControl = class(TComponent)
private
  FOnClick: TNotifyEvent;
protected
```

```
  procedure Click; dynamic;
  property OnClick: TNotifyEvent read FOnClick write FOnClick;
end;
```

متد TControl.click() نیز به صورت زیر پیاده‌سازی شده است.

```
procedure TControl.Click;
begin
  if Assigned(FOnClick) then FOnClick(Self);
end;
```

تعریف و ایجاد رویدادها

در این قسمت رویدادی را برای یک جزء سازنده ایجاد خواهیم کرد که در هر نیم دقیقه (برحسب ساعت سیستم) یکبار فراخوانی شود. از آنجا که نحوهٔ عملیات این جزء سازنده ارتباط مستقیم با ساعت سیستم دارد از جزء سازندهٔ TTimer برای برقراری این ارتباط استفاده خواهیم کرد. این جزء سازنده را تحت عنوان TddgHalfMinute و مطابق با لیست ۷-۸ پیاده‌سازی می‌نمائیم.

لیست ۷-۸ تعریف رویدادها برای کلاس TddgHalfMinute

```
unit halfmin;

interface

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

type
  { Define a procedure for the event handler. The event property will
    be of this procedure type. This type will take two parameters, the
    object that invoked the event and a TDateTime value to represent
    the time that the event occurred. For our component this will be
    every half-minute. }
  TTimeEvent = procedure(Sender: TObject; TheTime: TDateTime) of object;

  TddgHalfMinute = class(TComponent)
  private
    FTimer: TTimer;
    { Define a storage field to point to the user's event handler.
      The user's event handler must be of the procedural type
      TTimeEvent. }
    FOnHalfMinute: TTimeEvent;
    FOldSecond, FSecond: Word; // Variables used in the code
    { Define a procedure, FTimerTimer that will be assigned to
      FTimer.OnClick. This procedure must be of the type TNotifyEvent
      which is the type of TTimer.OnClick. }
      procedure FTimerTimer(Sender: TObject);
  protected
    { Define the dispatching method for the OnHalfMinute event. }
    procedure DoHalfMinute(TheTime: TDateTime); dynamic;
  public
```

لیست ۷-۸ ادامه

```
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
  published
    // Define the actual property that will show in the Object Inspector
    property OnHalfMinute: TTimeEvent read FOnHalfMinute write FOnHalfMinute;
  end;

implementation

constructor TddgHalfMinute.Create(AOwner: TComponent);
{ The Create constructor, creates the TTimer instanced for FTimer. It
  then sets up the various properties of FTimer, including its OnTimer
  event handler which is TddgHalfMinute's FTimerTimer() method. Notice
  that FTimer.Enabled is set to true only if the component is running
  and not while the component is in design mode. }
begin
  inherited Create(AOwner);
  // If the component is in design mode, do not enable FTimer.
  if not (csDesigning in ComponentState) then
  begin
    FTimer := TTimer.Create(self);
    FTimer.Enabled := True;
    // Set up the other properties, including the FTimer.OnTimer event handler
    FTimer.Interval := 500;
    FTimer.OnTimer := FTimerTimer;
   end;
end;

destructor TddgHalfMinute.Destroy;
begin
  FTimer.Free;
  inherited Destroy;
end;

procedure TddgHalfMinute.FTimerTimer(Sender: TObject);
{ This method serves as the FTimer.OnTimer event handler and is assigned
  to FTimer.OnTimer at run-time in TddgHalfMinute's constructor.
  This method gets the system time, and then determines whether or not
  the time is on the minute, or on the half-minute. If either of these
  conditions are true, it calls the OnHalfMinute dispatching method,
  DoHalfMinute. }
var
  DT: TDateTime;
  Temp: Word;
begin
  DT := Now; // Get the system time.
  FOldSecond := FSecond; // Save the old second.
```

لیست ۷-۸ ادامه

```
  // Get the time values, needed is the second value
  DecodeTime(DT, Temp, Temp, FSecond, Temp);

  { If not the same second when this method was last called, and if
    it is a half minute, call DoOnHalfMinute. }
  if FSecond <> FOldSecond then
    if ((FSecond = 30) or (FSecond = 0)) then
      DoHalfMinute(DT)
end;

procedure TddgHalfMinute.DoHalfMinute(TheTime: TDateTime);
{ This method is the dispatching method for the OnHalfMinute event.
  it checks to see if the user of the component has attached an
  event handler to OnHalfMinute and if so, calls that code. }
begin
  if Assigned(FOnHalfMinute) then
    FOnHalfMinute(Self, TheTime);
end;

end.
```

متدها

یک متد، تابعی است که توسط یک کلاس تعریف شده و در اختیار سایر اشیاء و کلاس‌ها قرار می‌گیرد. متدها می‌توانند چندین پارامتر را بپذیرند و داده‌هایی را از طریق متغیرها برگردانند.

پس از اینکه جزء سازنده‌ای را ایجاد می‌کنید، به راحتی می‌توانید آن را با مقداردهی خواص و فراخوانی متدها و روتین‌های پاسخگویی به رویدادها پردازش و کنترل کنید. برای این کار نیازی به دانستن نحوهٔ عملکرد خواص و متدها نخواهید داشت. جزء سازنده به محض پیش آمدن یک رویداد خاص، روتین پاسخگویی به آن را فرا می‌خواند. زمانی که جزء سازنده‌ای را طراحی می‌کنید، لازم است تا خواص و متدهای آن را نیز طراحی و پیاده‌سازی نمائید و در صورت پیش آمدن رویدادها نیز، روتین‌های پاسخگویی به آنها را فرا خوانید. روش انجام این کار تعریف کلاسی است که در برگیرندهٔ جزء سازنده می‌باشد.

نحوهٔ دستیابی به متدها

چهارگونه روش دستیابی برای متدها، متغیرها و توابع در دلفی وجود دارد که در جدول ۲-۸ به آنها اشاره شده است.

جدول ۲-۸ روشهای دستیابی خصوصی، محافظت شده، عمومی و توزیع یافته

روش	شرح
خصوصی	دستیابی تنها به توابع و متدهایی که در تعریف کلاس موردنظر گنجانیده شده‌اند، امکان‌پذیر می‌باشد.
محافظت شده	دستیابی تنها به توابع و متدهایی که در تعریف کلاس گنجانیده شده‌اند و همچنین به متدهایی که در ساختار سلسله مراتبی کلاس‌ها در مراحل پایین‌تر قرار دارند، امکان دارد.
عمومی	تمامی توابع و متدها قابل دسترسی می‌باشند.
توزیع یافته	همانند دستیابی عمومی، تمامی توابع و متدها قابل دسترسی می‌باشند. در این نحوهٔ تعریف، ارتباطی نیز با IDE دلفی برقرار خواهد شد تا اطلاعات در صفحات رویدادها و خواص نشان داده شوند.

سازنده

یک سازنده، زمان ایجاد یک کلاس فراخوانی خواهد شد. سازنده‌ها اغلب مسئول تخصیص پویای حافظه و گردآوری منابع موردنیاز برای کلاس‌ها می‌باشند. تعیین مقادیر پیش‌فرض برای کلاس‌ها نیز توسط سازنده‌ها صورت می‌پذیرد. برای معرفی یک کلاس، لازم است تا آن را در قسمت Public کلاس مربوطه اضافه نمائید. در زیر مثالی از یک سازنده تحت عنوان Create ارائه شده است.

```
TSomeComopnent = class(TComponent)
private
  { Private declarations }
protected
  { Protected declarations }
public
  constructor Create(AOwner: TComponent); override;
published
  { Published declarations }
end;
```

بعد از تعریف بالا، لازم است تا دستورات زیر را در قسمت Implementation یونیت درج نمائید.

```
constructor TSomeComponent.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  // Place your code here.
end;
```

مخرب

همانطور که در زیر نشان داده شده است، پس از پایان کار یک جزء سازنده، می‌بایست به عملیات آن خاتمه داد. این کار به وسیلهٔ متدهای مخرب انجام می‌گیرد.

```
destructor TMyComponent.Destroy;
begin
```

```
  FTimer.Free;
  MyStrings.Free;
  inherited Destroy;
end;
```

عملیات ثبت جزء سازنده

عملیات مربوط به طراحی یک جزء سازنده با ثبت کردن آن به پایان می‌رسد. همانطورکه در لیست ۸-۸ نشان داده شده است، جزء سازنده می‌بایست در صفحهٔ کتابخانه اجزاء سازنده ویژوال دلفی جای داده شود.

لیست ۸-۸ عملیات ثبت جزء سازنده

```
Unit MyComp;
interface
type
  TMyComp = class(TComponent)
  ...
  end;
  TOtherComp = class(TComponent)
  ...
  end;
procedure Register;
implementation
{ TMyComp methods }
{ TOtherCompMethods }
procedure Register;
begin
  RegisterComponents('DDG', [TMyComp, TOtherComp]);
end;
end.
```

آزمایش جزء سازنده

پس از نصب یک جزء سازنده، بهتر است که نحوهٔ عملکرد آن را نیز آزمایش نمائید. این کار با ایجاد یک برنامهٔ کاربردی برای آزمایش جزء سازنده صورت خواهد پذیرفت. برنامهٔ ارائه شده در لیست ۹-۸ عملیات مربوط به آزمایش جزء سازندهٔ TddgExtendedMemo را انجام می‌دهد. این برنامه فقط آزمایش جزء سازندهٔ TddgExtendedMemo را انجام می‌دهد و پیاده‌سازی این جزء سازنده را در قسمت‌های بعدی این فصل ارائه خواهیم نمود.

لیست ۹-۸ آزمایش جزء سازندهٔ TddgExtendedMemo

```
unit MainFrm;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
```

لیست ۹-۸ ادامه

```
  Forms, Dialogs, StdCtrls, exmemo, ExtCtrls;

type

  TMainForm = class(TForm)
    btnCreateMemo: TButton;
    btnGetRowCol: TButton;
    btnSetRowCol: TButton;
    edtColumn: TEdit;
    edtRow: TEdit;
    Panel1: TPanel;
    procedure btnCreateMemoClick(Sender: TObject);
    procedure btnGetRowColClick(Sender: TObject);
    procedure btnSetRowColClick(Sender: TObject);
  public
    EMemo: TddgExtendedMemo;  // Declare the component.
    procedure OnScroll(Sender: TObject);
  end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}
procedure TMainForm.btnCreateMemoClick(Sender: TObject);
begin
  { Dynamically create the component. Make sure to make the appropriate
    property assignments so that the component can be used normally.
    These assignments depend on the component being tested }
  if not Assigned(EMemo) then
  begin
    EMemo := TddgExtendedMemo.Create(self);
    EMemo.Parent := Panel1;
    EMemo.ScrollBars := ssBoth;
    EMemo.WordWrap := True;
    EMemo.Align := alClient;
    // Assign event handlers to untested events.
    EMemo.OnVScroll := OnScroll;
    EMemo.OnHScroll := OnScroll;
  end;
end;

{ Write whatever methods are required to test the run-time behavior
  of the component. This includes methods to access each of the
  new properties and methods belonging to the component.

  Also, create event handlers for user-defined events so that you can
  test them. Since you're creating the comoponent at run-time, you
```

لیست ۹-۸ ادامه

```
  have to manually assign the event handlers as was done in the
  above Create() constructor.
}
procedure TMainForm.btnGetRowColClick(Sender: TObject);
begin
  if Assigned(EMemo) then
    ShowMessage(Format('Row: %d  Column: %d', [EMemo.Row, EMemo.Column]));
  EMemo.SetFocus;
end;

procedure TMainForm.btnSetRowColClick(Sender: TObject);
begin
  if Assigned(EMemo) then
  begin
    EMemo.Row := StrToInt(edtRow.Text);
    EMemo.Column := StrToInt(edtColumn.Text);
    EMemo.SetFocus;
  end;
end;
procedure TMainForm.OnScroll(Sender: TObject);
begin
  MessageBeep(0);
end;

end.
```

ساخت Icon برای اجزاء سازنده

در بدو امر هیچ نشانه‌ای (icon) برای اجزاء سازنده‌ای که تولید خواهید نمود وجود ندارد، این بدان معنی است که باید یک نشانه خاص را برای اجزاء سازنده تهیه کنید. برای انجام این کار می‌توانید از Image Editor دلفی و یا ویراستارهای تصاویر bitmap استفاده کنید. در صورت استفاده از Image Editor یک فایل با پسوند DCR تولید خواهد شد و باید توجه داشته باشید که در محیط دلفی تنها از این نوع فایل‌ها برای تعیین نشانه‌ها استفاده می‌شود.

■ نکته

سعی کنید تصاویر مربوط به نشانه‌ها را در حالت ۱۶ رنگ ذخیره نمائید. حتی اگر کارت گرافیک شما قابلیت نمایش تصاویر بالاتر از ۲۵۶ رنگ را هم دارا است باز هم ذخیره‌سازی تصاویر در حالت ۱۶ رنگ کمک شایانی به قابلیت حمل برنامه‌تان می‌نماید.

افزایش قابلیتهای اجزاء سازنده

گاهی اوقات اجزاء سازنده را با استفاده از اجزاء سازنده موجود خواهید ساخت و بدین ترتیب با تغییر

قابلیت آنها و جایگزینی برخی خصوصیات دیگر، جزء سازندهٔ جدیدی را ایجاد خواهید نمود.

یکی از مزایای زبان‌های برنامه‌سازی شیءگرا آن است که یک شیء را می‌توان از یک کلاس والد مشتق نمود. با وجود مسأله وراثت، به راحتی می‌توانید کلاس جدیدی را با تمام قابلیت‌های یک کلاس والد، همراه با ویژگی‌های شخصی بدست آورید. در این قسمت اجزاء سازنده‌ای را ایجاد خواهیم کرد که از اجزاء TMemo و TListBox مشتق شده‌اند.

طراحی جزء سازندهٔ TddgExtendedMeme

در این قسمت با افزودن چند خصوصیت به TMemo، جزء سازندهٔ جدیدی تحت عنوان TddgExtendedMemo تولید خواهیم کرد. این خصوصیات جدید مربوط به قرار دادن یک نشانه برای حرکت بین سطرها و ستونهای محیط ویرایشی TMemo است. پیاده‌سازی این جزء سازنده در لیست ۱۰-۸ ارائه گردیده است.

لیست ۱۰-۸ پیاده‌سازی جزء سازندهٔ TddgExtendedMemo

```
unit ExtMemo;

interface

uses
  Windows, Messages, Classes, StdCtrls;

type

  TddgExtendedMemo = class(TMemo)
  private
    FRow: Longint;
    FColumn: Longint;
    FOnHScroll: TNotifyEvent;
    FOnVScroll: TNotifyEvent;
    procedure WMHScroll(var Msg: TWMHScroll); message WM_HSCROLL;
    procedure WMVScroll(var Msg: TWMVScroll); message WM_VSCROLL;
    procedure SetRow(Value: Longint);
    procedure SetColumn(Value: Longint);
    function GetRow: Longint;
    function GetColumn: Longint;
  protected
    // Event dispatching methods
    procedure HScroll; dynamic;
    procedure VScroll; dynamic;
  public
    property Row: Longint read GetRow write SetRow;
    property Column: Longint read GetColumn write SetColumn;
  published
    property OnHScroll: TNotifyEvent read FOnHScroll write FOnHScroll;
```

لیست ۱۰-۸ ادامه

```
    property OnVScroll: TNotifyEvent read FOnVScroll write FOnVScroll;
  end;


implementation

procedure TddgExtendedMemo.WMHScroll(var Msg: TWMHScroll);
begin
  inherited;
  HScroll;
end;

procedure TddgExtendedMemo.WMVScroll(var Msg: TWMVScroll);
begin
  inherited;
  VScroll;
end;

procedure TddgExtendedMemo.HScroll;
{ This is the OnHScroll event dispatch method. It checks to see
  if OnHScroll points to an event handler and calls it if it does. }
begin
  if Assigned(FOnHScroll) then
    FOnHScroll(self);
end;
procedure TddgExtendedMemo.VScroll;
{ This is the OnVScroll event dispatch method. It checks to see
  if OnVScroll points to an event handler and calls it if it does. }
begin
  if Assigned(FOnVScroll) then
    FOnVScroll(self);
end;

procedure TddgExtendedMemo.SetRow(Value: Longint);
{ The EM_LINEINDEX returns the character position of the first
  character in the line specified by wParam. The Value is used for
  wParam in this instance. Setting SelStart to this return value
  positions the caret on the line specified by Value. }
begin
  SelStart := Perform(EM_LINEINDEX, Value, 0);
  FRow := SelStart;
end;

function TddgExtendedMemo.GetRow: Longint;
{ The EM_LINEFROMCHAR returns the line in which the character specified
  by wParam sits. If -1 is passed as wParam, the line number at which
  the caret sits is returned. }
begin
```

لیست ۱۰-۸ ادامه

```
  Result := Perform(EM_LINEFROMCHAR, -1, 0);
end;

procedure TddgExtendedMemo.SetColumn(Value: Longint);
begin
  { Get the length of the current line using the EM_LINELENGTH
    message. This message takes a character position as WParam.
    The length of the line in which that character sits is returned. }
  FColumn := Perform(EM_LINELENGTH, Perform(EM_LINEINDEX, GetRow, 0), 0);
  { If the FColumn is greater than the value passed in, then set
    FColumn to the value passed in }
  if FColumn > Value then
    FColumn := Value;
  // Now set SelStart to the newly specified position
  SelStart := Perform(EM_LINEINDEX, GetRow, 0) + FColumn;
end;

function TddgExtendedMemo.GetColumn: Longint;
begin
  { The EM_LINEINDEX message returns the line index of a specified
    character passed in as wParam. When wParam is -1 then it
    returns the index of the current line. Subtracting SelStart from this
    value returns the column position }
  Result := SelStart - Perform(EM_LINEINDEX, -1, 0);
end;

end.
```

همانطور که ملاحظه کردید در این قطعه کد یکسری خصوصیات در رابطه با نمایش اطلاعات سطرها و ستون‌ها بر جزء سازنده TMemo اضافه گردیده است.

مقادیر سطرها و ستون‌ها در متغیرهای FRow و FColumn که به صورت خصوصی تعریف شده‌اند، ذخیره می‌گردد. از Row و Colunm نیز به عنوان متدهای خواننده و نویسنده استفاده خواهد شد.

در این جزء سازنده دو رویداد جدید به نامهای OnHScroll و OnVScroll اضافه گردیده است. OnHScroll رویدادی است که در هنگام فشردن میله افقی جعبهٔ ویرایش فعال می‌شود و OnVScroll رویدادی است که در زمان فشردن میله عمودی جعبهٔ ویرایش به کار می‌افتد.

طراحی جزء سازندهٔ TddgTabbedListBox

در این بخش نیز با افزودن چند خصوصیت به TListBox ، جزء سازندهٔ جدیدی تحت عنوان TddgTabbedListBox را ایجاد خواهیم کرد. TListBox کادر فهرست استاندارد ویندوز است که امکان ایجاد فهرستهای قابل انتخاب را برایتان فراهم می‌کند. در این جزء سازنده خصوصیتی جدیدی جهت نمایش عناوین لیست‌هایی که طولانی‌تر از اندازه TListBox می‌باشند، گنجانیده شده است. بدین معنی

که با قرار دادن یک Scrollbar در جزء سازندهٔ جدید، این قابلیت فراهم گردیده است. این Scrollbar دارای قابلیت ویژه‌ای است که آن را در هنگام اجرای آن مشاهده خواهید نمود. برای این جزء سازنده یک جعبهٔ مکالمه نیز ارائه گردیده است که کد آن را در لیست ۱۱-۸ مشاهده خواهید کرد.

لیست ۱۱-۸ پیاده‌سازی جعبهٔ مکالمه مربوط به جزء سازندهٔ TddgTabbedListBox

```
unit Pixdlg;

interface

function DialogUnitsToPixelsX(DlgUnits: word): word;
function DialogUnitsToPixelsY(DlgUnits: word): word;
function PixelsToDialogUnitsX(PixUnits: word): word;
function PixelsToDialogUnitsY(PixUnits: word): word;

implementation
uses WinProcs;

function DialogUnitsToPixelsX(DlgUnits: word): word;
begin
  Result := (DlgUnits * LoWord(GetDialogBaseUnits)) div 4;
end;
function DialogUnitsToPixelsY(DlgUnits: word): word;
begin
  Result := (DlgUnits * HiWord(GetDialogBaseUnits)) div 8;
end;

function PixelsToDialogUnitsX(PixUnits: word): word;
begin
  Result := PixUnits * 4 div LoWord(GetDialogBaseUnits);
end;

function PixelsToDialogUnitsY(PixUnits: word): word;
begin
  Result := PixUnits * 8 div HiWord(GetDialogBaseUnits);
end;

end.
```

پیاده‌سازی جزء سازندهٔ TddgTabbedListBox را در لیست ۱۲-۸ مشاهده می‌کنید.

لیست ۱۲-۸ پیاده‌سازی جزء سازندهٔ TddgTabbedListBox

```
unit Lbtab;

interface
```

لیست ۱۲-۸ ادامه

```
uses
  SysUtils, Windows, Messages, Classes, Controls, StdCtrls;

type

  EddgTabListboxError = class(Exception);
  TddgTabListBox = class(TListBox)
  private
    FLongestString: Word;
    FNumTabStops: Word;
    FTabStops: PWord;
    FSizeAfterDel: Boolean;
    function GetLBStringLength(S: String): word;
    procedure FindLongestString;
    procedure SetScrollLength(S: String);
    procedure LBAddString(var Msg: TMessage); message lb_AddString;
    procedure LBInsertString(var Msg: TMessage); message lb_InsertString;
    procedure LBDeleteString(var Msg: TMessage); message lb_DeleteString;
  protected
    procedure CreateParams(var Params: TCreateParams); override;
  public
    constructor Create(AOwner: TComponent); override;
    procedure SetTabStops(A: array of word);
  published
    property SizeAfterDel: Boolean read FSizeAfterDel
➥     write FSizeAfterDel default True;
  end;

implementation

uses PixDlg;

constructor TddgTabListBox.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  FSizeAfterDel := True;
  { set tab stops to Windows defaults... }
  FNumTabStops := 1;
  GetMem(FTabStops, SizeOf(Word) * FNumTabStops);
  FTabStops^ := DialogUnitsToPixelsX(32);
end;

procedure TddgTabListBox.SetTabStops(A: array of word);
{ This procedure sets the listbox's tabstops to those specified
  in the open array of word, A.  New tabstops are in pixels, and must
  be in ascending order.  An exception will be raised if new tabs
  fail to set. }
var
```

لیست ۱۲-۸ ادامه

```
  i: word;
  TempTab: word;
  TempBuf: PWord;
begin
  { Store new values in temps in case exception occurs in setting tabs }
  TempTab := High(A) + 1;       // Figure number of tabstops
  GetMem(TempBuf, SizeOf(A));  // Allocate new tabstops
  Move(A, TempBuf^, SizeOf(A));// copy new tabstops }
  { convert from pixels to dialog units, and... }
  for i := 0 to TempTab - 1 do
    A[i] := PixelsToDialogUnitsX(A[i]);
  { Send new tabstops to listbox.  Note that we must use dialog units. }
  if Perform(lb_SetTabStops, TempTab, Longint(@A)) = 0 then
  begin
    { if zero, then failed to set new tabstops, free temp
      tabstop buffer and raise an exception }
    FreeMem(TempBuf, SizeOf(Word) * TempTab);
    raise EddgTabListboxError.Create('Failed to set tabs.')
  end
  else begin
    { if nonzero, then new tabstops set okay, so
      Free previous tabstops }
    FreeMem(FTabStops, SizeOf(Word) * FNumTabStops);
    { copy values from temps... }
    FNumTabStops := TempTab;  // set number of tabstops
    FTabStops := TempBuf;     // set tabstop buffer
    FindLongestString;        // reset scrollbar
    Invalidate;               // repaint
  end;
end;

procedure TddgTabListBox.CreateParams(var Params: TCreateParams);
{ We must OR in the styles necessary for tabs and horizontal scrolling
  These styles will be used by the API CreateWindowEx() function. }
begin
  inherited CreateParams(Params);
  { lbs_UseTabStops style allows tabs in listbox
    ws_HScroll style allows horizontal scrollbar in listbox }
  Params.Style := Params.Style or lbs_UseTabStops or ws_HScroll;
end;

function TddgTabListBox.GetLBStringLength(S: String): word;
{ This function returns the length of the listbox string S in pixels }
var
  Size: Integer;
begin
```

لیست ۱۲-۸ ادامه

```
  // Get the length of the text string
   Canvas.Font := Font;
  Result := LoWord(GetTabbedTextExtent(Canvas.Handle, PChar(S),
         StrLen(PChar(S)), FNumTabStops, FTabStops^));
  // Add a little bit of space to the end of the scrollbar extent for looks
  Size := Canvas.TextWidth('X');
  Inc(Result, Size);
end;

procedure TddgTabListBox.SetScrollLength(S: String);
{ This procedure resets the scrollbar extent if S is longer than the }
{ previous longest string                                             }
var
  Extent: Word;
begin
  Extent := GetLBStringLength(S);
  // If this turns out to be the longest string...
  if Extent > FLongestString then
  begin
    // reset longest string
    FLongestString := Extent;
    //reset scrollbar extent
    Perform(lb_SetHorizontalExtent, Extent, 0);
  end;
end;

procedure TddgTabListBox.LBInsertString(var Msg: TMessage);
{ This procedure is called in response to a lb_InsertString message.
  This message is sent to the listbox every time a string is inserted.
  Msg.lParam holds a pointer to the null-terminated string being
  inserted.  This will cause the scrollbar length to be adjusted if
  the new string is longer than any of the existing strings. }
begin
  inherited;
  SetScrollLength(PChar(Msg.lParam));
end;

procedure TddgTabListBox.LBAddString(var Msg: TMessage);
{ This procedure is called in response to a lb_AddString message.
  This message is sent to the listbox every time a string is added.
  Msg.lParam holds a pointer to the null-terminated string being
  added.  This Will cause the scrollbar length to be ajdusted if the
  new string is longer than any of the existing strings.}
```

لیست ۱۲-۸ ادامه

```
begin
  inherited;
  SetScrollLength(PChar(Msg.lParam));
end;

procedure TddgTabListBox.FindLongestString;
var
  i: word;
  Strg: String;
begin
  FLongestString := 0;
  { iterate through strings and look for new longest string }
  for i := 0 to Items.Count - 1 do
  begin
    Strg := Items[i];
    SetScrollLength(Strg);
  end;
end;

procedure TddgTabListBox.LBDeleteString(var Msg: TMessage);
{ This procedure is called in response to a lb_DeleteString message.
  This message is sent to the listbox everytime a string is deleted.
  Msg.wParam holds the index of the item being deleted.  Note that
  by setting the SizeAfterDel property to False, you can cause the
  scrollbar update to not occur.  This will improve performance
  if you're deleting often. }
var
  Str: String;
begin
  if FSizeAfterDel then
  begin
    Str := Items[Msg.wParam]; // Get string to be deleted
    inherited;                // Delete string
    { Is deleted string the longest? }
    if GetLBStringLength(Str) = FLongestString then
      FindLongestString;
  end
  else
    inherited;
end;

end.
```

در ادامهٔ این فصل، نحوهٔ پیاده‌سازی اجزاء سازندهٔ جدید را ارائه خواهیم نمود.

ساخت جزء سازندهٔ TddgRunButton

نحوهٔ کارکرد این جزء سازنده تشابهی تقریبی با TSpeedButton دارد. SpeedButton دکمهٔ ویژه‌ای است که برای کار با پانل طراحی شده است. SpeedButton برای ایجاد جعبه ابزارها و دکمه‌های ویژه، از جمله دکمه‌هایی که در حالت فشرده باقی می‌مانند، استفاده می‌شود. پیاده‌سازی جزء سازنده TddgRunButton در لیست ۱۳-۸ ارائه شده است.

لیست ۱۳-۸ پیاده‌سازی جزء سازندهٔ TddgRunButton

```
unit RunBtn;

interface

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

type

  TCommandLine = type string;

  TddgRunButton = class(TSpeedButton)
  private
    FCommandLine: TCommandLine;
    // Hiding Properties from the Object Inspector
    FCaption: TCaption;
    FAllowAllUp: Boolean;
    FFont: TFont;
    FGroupIndex: Integer;
    FLayOut: TButtonLayout;
    procedure SetCommandLine(Value: TCommandLine);
  public
    constructor Create(AOwner: TComponent); override;
    procedure Click; override;
  published
    property CommandLine: TCommandLine read FCommandLine write SetCommandLine;
    // Read only properties are hidden
    property Caption: TCaption read FCaption;
    property AllowAllUp: Boolean read FAllowAllUp;
    property Font: TFont read FFont;
    property GroupIndex: Integer read FGroupIndex;
    property LayOut: TButtonLayOut read FLayOut;
  end;

implementation
```

لیست ۱۳-۸ ادامه

```
uses ShellAPI;

const
  EXEExtension = '.EXE';
function ProcessExecute(CommandLine: TCommandLine; cShow: Word): Integer;
{ This method encapsulates the call to CreateProcess() which creates
  a new process and its primary thread. This is the method used in
  Win32 to execute another application, This method requires the use
  of the TStartInfo and TProcessInformation structures. These structures
  are not documented as part of the Delphi 6 online help but rather
  the Win32 help as STARTUPINFO and PROCESS_INFORMATION.

  The CommandLine parameter specifies the pathname of the file to
  execute.

  The cShow parameter specifies one of the SW_XXXX constants which
  specifies how to display the window. This value is assigned to the
  sShowWindow field of the TStartupInfo structure. }
var
  Rslt: LongBool;
  StartUpInfo: TStartUpInfo;  // documented as STARTUPINFO
  ProcessInfo: TProcessInformation; // documented as PROCESS_INFORMATION
begin
  { Clear the StartupInfo structure }
  FillChar(StartupInfo, SizeOf(TStartupInfo), 0);
  { Initialize the StartupInfo structure with required data.
    Here, we assign the SW_XXXX constant to the wShowWindow field
    of StartupInfo. When specifying a value to this field the
    STARTF_USESSHOWWINDOW flag must be set in the dwFlags field.
    Additional information on the TStartupInfo is provided in the Win32
    online help under STARTUPINFO. }
  with StartupInfo do
  begin
    cb := SizeOf(TStartupInfo); // Specify size of structure
    dwFlags := STARTF_USESHOWWINDOW or STARTF_FORCEONFEEDBACK;
    wShowWindow := cShow
  end;

  { Create the process by calling CreateProcess(). This function
    fills the ProcessInfo structure with information about the new
    process and its primary thread. Detailed information is provided
    in the Win32 online help for the TProcessInfo structure under
    PROCESS_INFORMATION. }
  Rslt := CreateProcess(PChar(CommandLine), nil, nil, nil, False,
    NORMAL_PRIORITY_CLASS, nil, nil, StartupInfo, ProcessInfo);
  { If Rslt is true, then the CreateProcess call was successful.
    Otherwise, GetLastError will return an error code representing the
    error which occurred. }
```

لیست ۱۳-۸ ادامه

```
  if Rslt then
    with ProcessInfo do
    begin
      { Wait until the process is in idle. }
      WaitForInputIdle(hProcess, INFINITE);
      CloseHandle(hThread); // Free the hThread  handle
      CloseHandle(hProcess);// Free the hProcess handle
      Result := 0;          // Set Result to 0, meaning successful
    end
  else Result := GetLastError; // Set result to the error code.
end;

function IsExecutableFile(Value: TCommandLine): Boolean;
{ This method returns whether or not the Value represents a valid
  executable file by ensuring that its file extension is 'EXE' }
var
  Ext: String[4];
begin
  Ext := ExtractFileExt(Value);
  Result := (UpperCase(Ext) = EXEExtension);
end;

constructor TddgRunButton.Create(AOwner: TComponent);
{ The constructor sets the default height and width properties
  to 45x45 }
begin
  inherited Create(AOwner);
  Height := 45;
  Width  := 45;
end;

procedure TddgRunButton.SetCommandLine(Value: TCommandLine);
{ This write access method sets the FCommandLine field to Value, but
  only if Value represents a valid executable file name. It also
  set the icon for the TddgRunButton to the application icon of the
  file specified by Value. }
var
  Icon: TIcon;
begin
  { First check to see that Value *is* an executable file and that
    it actually exists where specified. }
  if not IsExecutableFile(Value) then
    Raise Exception.Create(Value+' is not an executable file.');
```

لیست ۱۳-۸ ادامه

```
  if not FileExists(Value) then
    Raise Exception.Create('The file: '+Value+' cannot be found.');

  FCommandLine := Value;  // Store the Value in FCommandLine

  { Now draw the application icon for the file specified by Value
    on the TddgRunButton icon. This requires us to create a TIcon
    instance to which to load the icon. It is then copied from this
    TIcon instance to the TddgRunButton's Canvas.

    We must use the Win32 API function ExtractIcon() to retrieve the
    icon for the application. }
  Icon := TIcon.Create; // Create the TIcon instance
  try
    { Retrieve the icon from the application's file }
    Icon.Handle := ExtractIcon(hInstance, PChar(FCommandLine), 0);
    with Glyph do
    begin
      { Set the TddgRunButton properties so that the icon held by Icon
        can be copied onto it. }
      { First, clear the canvas. This is required in case another
        icon was previously drawn on the canvas }
      Canvas.Brush.Style := bsSolid;
      Canvas.FillRect(Canvas.ClipRect);
      { Set the Icon's width and height }
      Width := Icon.Width;
      Height := Icon.Height;
      Canvas.Draw(0, 0, Icon); // Draw the icon to TddgRunButton's Canvas
    end;
  finally
    Icon.Free; // Free the TIcon instance.
  end;
end;

procedure TddgRunButton.Click;
var
  WERetVal: Word;
begin
  inherited Click; // Call the inherited Click method
  { Execute the ProcessExecute method and check it's return value.
    if the return value is <> 0 then raise an exception because
    an error occurred. The error code is shown in the exception }
  WERetVal := ProcessExecute(FCommandLine, sw_ShowNormal);
  if WERetVal <> 0 then begin
```

لیست ۱۳-۸ ادامه

```
    raise Exception.Create('Error executing program. Error Code:; '+
          IntToStr(WERetVal));
  end;
end;

end.
```

اجرای یک فرآیند

منظور از فرآیند در این قسمت، برنامه‌های کاربردی است که قصد دارید آنها را در برنامه‌هایتان اجرا نمائید. این برنامه‌های کاربردی از قبل توسط خود شما و یا دیگران تهیه شده‌اند و اینک مکانیسمی لازم است تا آنها را اجرا نمائید. این عملیات توسط تابع ()ProcessExecute انجام می‌پذیرد. برای این تابع دو پارامتر تعریف می‌گردد. یکی از این پارامترها نام فایل اجرایی (برنامه) و دیگری نحوه نمایش برنامه را در پنجره مربوط به آن مشخص می‌سازد. پارامتر دوم عموماً به صورت *SW_XXXX* تعریف شده که مقادیر معتبر برای این پارامتر در جدول ۳-۸ ارائه گردیده است.

جدول ۳-۸ مقادیر معتبر برای پارامتر SW_XXXX

پارامتر	شرح
SW_HIDE	پنجرهٔ مربوط به برنامهٔ فراخوانی شده، مخفی خواهد بود. (پنجره‌ای نمایش داده نمی‌شود).
SW_MAXIMIZE	پنجرهٔ مربوط به برنامهٔ فراخوانی شده، در حالت بیشینهٔ خود نمایش داده می‌شود.
SW_MINIMIZE	پنجرهٔ مربوط به برنامهٔ فراخوانی شده، در حالت کمینهٔ خود نمایش داده می‌شود.
SW_RESTORE	پنجرهٔ مربوط به برنامهٔ فراخوانی شده، در حالت قبل از اعمال تغییرات (بیشینه یا کمینه) نمایش داده می‌شود.
SW_SHOW	پنجرهٔ مربوط به برنامهٔ فراخوانی شده، در حالت نرمال خود نمایش داده می‌شود.
SW_SHOWDEFAULT	پنجرهٔ مربوط به برنامهٔ فراخوانی شده، طبق مشخصات تعریف شده در ساختمان TStratupInfo ، نمایش داده می‌شود.

متدهای جزء سازندهٔ TddgRunButton

متد ()TddgRunButton.Create به عنوان یک سازنده عمل می‌کند و برای انجام این کار از خصوصیات وراثتی متد ()Create استفاده می‌نماید. ()SetCommandLine متد نویسندهٔ پارامتر CommandLine می‌باشد، بدین معنی که از آن برای تنظیم پارامتر CommandLine استفاده می‌شود.

متد ()TddgRunButton.Click نیز وظیفهٔ توزیع رویدادها را به عهده دارد.

ساخت جزء سازندهٔ TddgButtonEdit

گاهی اوقات مایلید ترکیبی از چند جزء سازنده را در قالب یک جزء سازنده طراحی و پیاده‌سازی نمائید.

به عنوان مثال TDBNavigator یکی از اجزاء سازنده است که متشکل از چند جزء به نام‌های TPanel و TSpeedButton می‌باشد. این کار به مفهوم الحاق خصوصیتهای چندین جزء سازنده و ایجاد اجزاء سازنده جدید می‌باشد.

در این قسمت با ترکیب اجزاء سازندهٔ TEdit و TSpeedButton ، جزء سازندهٔ جدیدی تحت عنوان TddgButtonEdit را ایجاد خواهیم نمود. تابع سازندهٔ TddgButtonEdit به صورت زیر پیاده‌سازی شده است.

```
constructor TddgButtonEdit.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  FEdit        := TEdit.Create(Self);
  FEdit.Parent := self;
  FEdit.Height := 21;

  FSpeedButton := TSpeedButton.Create(Self);
  FSpeedButton.Left := FEdit.Width;
  FSpeedButton.Height := 19; // two less then TEdit's Height
  FSpeedButton.Width  := 19;
  FSpeedButton.Caption := '...';
  FSpeedButton.Parent := Self;

  Width  := FEdit.Width+FSpeedButton.Width;
  Height := FEdit.Height;
end;
```

در طراحی این جزء سازنده از کلاس TWinControl استفاده شده است. اولین قدم برای پیاده‌سازی این جزء سازنده تخصیص یک خصوصیت Text به صورت زیر خواهد بود:

```
TddgButtonEdit = class(TWinControl)
private
  FEdit: TEdit;
  protected
  procedure SetText(Value: String);
    function  GetText: String;
published
    property Text: String read GetText write SetText;
end;
```

The `SetText()` and `GetText()` methods directly access the `Text` property of the contained `TEdit` control, as shown in the following:

```
function TddgButtonEdit.GetText: String;
begin
  Result := FEdit.Text;
```

```
end;

procedure TddgButtonEdit.SetText(Value: String);
begin
  FEdit.Text := Value;
end;
```

پیاده‌سازی جزء سازندهٔ TddgButtonEdit در لیست ۱۴-۸ ارائه شده است.

لیست ۱۴-۸ پیاده‌سازی جزء سازندهٔ TddgButtonEdit

```
unit ButtonEdit;

interface

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

type
  TddgButtonEdit = class(TWinControl)
  private
    FSpeedButton: TSpeedButton;
    FEdit: TEdit;
  protected
    procedure WMSize(var Message: TWMSize); message WM_SIZE;
    procedure SetText(Value: String);
    function  GetText: String;
    function GetFont: TFont;
    procedure SetFont(Value: TFont);
    function GetOnButtonClick: TNotifyEvent;
    procedure SetOnButtonClick(Value: TNotifyEvent);
  public
    constructor Create(AOwner: TComponent); override;
    destructor  Destroy; override;
  published
    property Text: String read GetText write SetText;
    property Font: TFont read GetFont write SetFont;
    property OnButtonClick: TNotifyEvent read GetOnButtonClick
        write SetOnButtonClick;
  end;

implementation

procedure TddgButtonEdit.WMSize(var Message: TWMSize);
begin
  inherited;
  FEdit.Width := Message.Width-FSpeedButton.Width;
  FSpeedButton.Left := FEdit.Width;
end;
```

لیست ۱۴-۸ ادامه

```
constructor TddgButtonEdit.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  FEdit        := TEdit.Create(Self);
  FEdit.Parent := self;
  FEdit.Height := 21;

  FSpeedButton := TSpeedButton.Create(Self);
  FSpeedButton.Left := FEdit.Width;
  FSpeedButton.Height := 19; // two less than TEdit's Height
  FSpeedButton.Width  := 19;
  FSpeedButton.Caption := '...';
  FSpeedButton.Parent := Self;

  Width  := FEdit.Width+FSpeedButton.Width;
  Height := FEdit.Height;
end;

destructor  TddgButtonEdit.Destroy;
begin
  FSpeedButton.Free;
  FEdit.Free;
  inherited Destroy;
end;

function TddgButtonEdit.GetText: String;
begin
  Result := FEdit.Text;
end;

procedure TddgButtonEdit.SetText(Value: String);
begin
  FEdit.Text := Value;
end;

function TddgButtonEdit.GetFont: TFont;
begin
  Result := FEdit.Font;
end;

procedure TddgButtonEdit.SetFont(Value: TFont);
begin
  if Assigned(FEdit.Font) then
    FEdit.Font.Assign(Value);
end;

function TddgButtonEdit.GetOnButtonClick: TNotifyEvent;
```

لیست ۱۴-۸ ادامه

```
begin
  Result := FSpeedButton.OnClick;
end;

procedure TddgButtonEdit.SetOnButtonClick(Value: TNotifyEvent);
begin
  FSpeedButton.OnClick := Value;
end;

end.
```

ساخت جزء سازندهٔ TddgDigitalClock

در این قسمت جزء سازندهٔ دیگری تحت عنوان TddgDigitalClock را پیاده‌سازی خواهیم نمود. این جزء سازنده به عنوان یک ساعت دیجیتال عمل خواهد کرد. کلاس مربوط به این جزء سازنده از کلاس TPanel اشتقاق یافته است. از کلاس TPanel برای ویژوال ساختن یک جزء سازنده استفاده شده است، به طوری که زمان سیستم در خصوصیت TPanel.Caption نمایش داده شود. رویدادهای زیر برای TddgDigitalClock تعریف شده است:

OnHour	این رویداد هر ساعت یکبار فعال می‌شود.
OnHalfPast	این رویداد هر نیم ساعت یکبار فعال می‌شود.
OnMinute	این رویداد هر دقیقه یکبار فعال می‌شود.
OnHalfMinute	این رویداد هر نیم دقیقه یکبار فعال می‌شود.
OnSecond	این رویداد هر ثانیه یکبار فعال می‌شود.

توجه داشته باشید که جزء سازندهٔ TTimer نیز نقشی کلیدی در طراحی این جزء سازنده دارد.
پیاده‌سازی جزء سازندهٔ TddDigitalClock در لیست ۱۵-۸ ارائه گردیده است.

لیست ۱۵-۸ پیاده‌سازی جزء سازندهٔ TddgDigitalClock

```
{$IFDEF VER110}
{$OBJEXPORTALL ON}
{$ENDIF}

unit DDGClock;
interface

uses
  Windows, Messages, Controls, Forms, SysUtils, Classes, ExtCtrls;

type
```

لیست ۱۵-۸ ادامه

```
{ Declare an event type which takes the sender of the event, and
  a TDateTime variable as parameters }
TTimeEvent = procedure(Sender: TObject; DDGTime: TDateTime) of object;

TddgDigitalClock = class(TPanel)
private
  { Data fields }
  FHour,
  FMinute,
  FSecond: Word;
  FDateTime: TDateTime;
  FOldMinute,
  FOldSecond: Word;
  FTimer: TTimer;
  { Event handlers }
  FOnHour: TTimeEvent;       // Occurs on the hour
  FOnHalfPast: TTimeEvent;   // Occurs every half-hour
  FOnMinute: TTimeEvent;     // Occurs on the minute
  FOnSecond: TTimeEvent;     // Occurs every second
  FOnHalfMinute: TTimeEvent; // Occurs every 30 seconds
  { Define OnTimer event handler for internal TTimer, FTimer }
  procedure TimerProc(Sender: TObject);
protected
  { Override the Paint methods }
  procedure Paint; override;

  { Define the various event dispatching methods }
  procedure DoHour(Tm: TDateTime); dynamic;
  procedure DoHalfPast(Tm: TDateTime); dynamic;
  procedure DoMinute(Tm: TDateTime); dynamic;
  procedure DoHalfMinute(Tm: TDateTime); dynamic;
  procedure DoSecond(Tm: TDateTime); dynamic;

public
  { Override the Create constructor and Destroy destructor }
  constructor Create(AOwner: TComponent); override;
  destructor Destroy; override;
 published
   { Define event properties }
   property OnHour: TTimeEvent read FOnHour write FOnHour;
   property OnHalfPast: TTimeEvent read FOnHalfPast write FOnHalfPast;
   property OnMinute: TTimeEvent read FOnMinute write FOnMinute;
   property OnHalfMinute: TTimeEvent read FOnHalfMinute
            write FOnHalfMinute;
   property OnSecond: TTimeEvent read FOnSecond write FOnSecond;
 end;

implementation
```

لیست ۱۵-۸ ادامه

```
constructor TddgDigitalClock.Create(AOwner: TComponent);
begin
  inherited Create(AOwner); // Call the inherited constructor
  Height := 25; // Set default width and height properties
  Width := 120;
  BevelInner := bvLowered; // Set Default bevel properties
  BevelOuter := bvLowered;
  { Set the inherited Caption property to an empty string }
  inherited Caption := '';
  { Create the TTimer instance and set both its Interval property and
    OnTime event handler. }
  FTimer:= TTimer.Create(self);
  FTimer.interval:= 200;
  FTimer.OnTimer:= TimerProc;
end;


destructor TddgDigitalClock.Destroy;
begin
  FTimer.Free; // Free the TTimer instance.
  inherited Destroy; // Call inherited Destroy method
end;

procedure TddgDigitalClock.Paint;
begin
  inherited Paint; // Call the inherited Paint method
  { Now set the inherited Caption property to current time. }
  inherited Caption := TimeToStr(FDateTime);
end;

procedure TddgDigitalClock.TimerProc(Sender: TObject);
var
  HSec: Word;
begin
  { Save the old minute and second for later use }
  FOldMinute := FMinute;
  FOldSecond := FSecond;
  FDateTime := Now; // Get the current time.
  { Extract the individual time elements }
  DecodeTime(FDateTime, FHour, FMinute, FSecond, Hsec);

  refresh; // Redraw the component so that the new time is displayed.

  { Now call the event handlers depending on the time }
  if FMinute = 0 then
    DoHour(FDateTime);
  if FMinute = 30 then
    DoHalfPast(FDateTime);
  if (FMinute <> FOldMinute) then
```

لیست ۱۵-۸ ادامه

```
    DoMinute(FDateTime);
  if FSecond <> FOldSecond then
    if ((FSecond = 30) or (FSecond = 0)) then
      DoHalfMinute(FDateTime)
    else
       DoSecond(FDateTime);
end;

{ The event dispatching methods below determine if component user has
  attached event handlers to the various clock events and calls them
  if they exist }
procedure TddgDigitalClock.DoHour(Tm: TDateTime);
begin
  if Assigned(FOnHour) then
   TTimeEvent(FOnHour)(Self, Tm);
end;

procedure TddgDigitalClock.DoHalfPast(Tm: TDateTime);
begin
  if Assigned(FOnHalfPast) then
   TTimeEvent(FOnHalfPast)(Self, Tm);
end;

procedure TddgDigitalClock.DoMinute(Tm: TDateTime);
begin
  if Assigned(FOnMinute) then
   TTimeEvent(FOnMinute)(Self, Tm);
end;
procedure TddgDigitalClock.DoHalfMinute(Tm: TDateTime);
begin
  if Assigned(FOnHalfMinute) then
   TTimeEvent(FOnHalfMinute)(Self, Tm);
end;

procedure TddgDigitalClock.DoSecond(Tm: TDateTime);
begin
  if Assigned(FOnSecond) then
   TTimeEvent(FOnSecond)(Self, Tm);
end;

end.
```

خلاصه

دلفی یکی از قدیمی‌ترین نرم‌افزارها برای ایجاد اجزاء سازنده‌ای است که آنها را می‌توان هم در دلفی و هم در سایر برنامه‌های کاربردی به کار برد. در این فصل شیوهٔ عملکرد اجزای سازنده و نیز آنچه که برای ایجاد، نصب و کاربرد آنها لازم است را بررسی کردیم. دلفی از مدل شیءگرا استفاده می‌کند و این بدان

معناست که هر یک از اجزای موجود را می‌توانید بهبود بخشید. سعی کنید مطالب این فصل را در روشهای برنامه‌نویسی خود تعمیم دهید. در این صورت چنانچه روتین‌های خود را به صورت اجزای سازندهٔ قابل استفاده مجدد بنویسید، در مراحل بعدی که نیاز به انجام کارهای مشابه دارید، جزء سازندهٔ موردنظرتان موجود و قابل استفاده خواهد بود.

فصل نهم

ساخت اجزاء سازندهٔ VCL پیشرفته

در این فصل می‌خوانید

- **اجزاء Visual**
- **نوشتن ویرایشگرهای صفت.**
- **ویرایشگرهای اجزاء.**
- **دسته‌بندی‌های صفات.**
- **لیست‌های اجزاء: TCollection و TCollectionItem .**

در فصل قبل به نوشتن اجزاء سازندهٔ VCL پرداختیم و مقدمات کار برای شما مشخص شد. در این فصل، با به کارگیری تکنیک‌های پیشرفته طراحی در اجزاء سازندهٔ Delphi نحوه به کارگیری عمل نوشتن اجزاء سازنده در سطح متن را خواهید آموخت. در این فصل مثالهایی از تکنیک‌های پیشرفته نظیر اجزاء سازندهٔ شبه Visual[1] ، ویرایشگرهای صفات، ویرایشگرهای مؤلفه و گرادیه‌ها به شما ارائه خواهند شد.

اجزاء سازندهٔ شبه Visual

شما تاکنون در مورد اجزاء سازندهٔ Visual نظیر **TButton** و **TEdit** و همچنین اجزاء سازندهٔ NonVisual مانند **TTable** و **TTimer** مطالبی آموخته‌اید. در این بخش در مورد نوعی از اجزاء سازنده که بین این دو گروه قرار دارند، مطالبی ارائه خواهد شد. این اجزاء سازنده را اجزاء سازندهٔ شبه Visual خواهیم نامید.

۱- Pseudo– Visual

بسط تذکر (Hint) ها

به طور خاص، جزء سازندهٔ نشان داده شده در این بخش بسطی است یک پنجره hint است. این جزء سازنده را شبه Visual می‌نامیم زیرا جزء سازنده‌ای نیست که در زمان طراحی به صورت بصری به کار رود، ولی در زمان اجراء در بدنه hint ها خود را به صورت بصری نمایان می‌کند.

برای تعویض قالب پنجره hint در یک برنامه Delphi نیاز دارید که چهارگام زیر را به انجام برسانید:

۱- یک توارث از **THintWindow** ایجاد کنید.

۲- کلاس پنجره hint قدیمی را تخریب کنید.

۳- کلاس پنجره hint جدید را نسبت دهید.

۴- یک کلاس پنجره hint جدید ایجاد کنید.

ایجاد یک توارث THintWindow

قبل از اینکه کدی برای توارث **THintWindow** ایجاد کنیم بایستی روی چگونگی تفاوت کلاس پنجره hint خود با گونه پیش‌فرض آن تصمیم‌گیری کنید. در این حالت، یک پنجره hint مستتر به جای چهارگوش پیش‌فرض آن ایجاد خواهید نمود. لیست ۱-۹ یونیت **RndHint.pas** را نشان می‌دهد که توارث **TDDGHintWindow** از **THintWindow** است.

لیست ۱-۹ یونیت RndHint.Pas

```
unit RndHint;

interface

uses Windows, Classes, Controls, Forms, Messages, Graphics;
type
  TDDGHintWindow = class(THintWindow)
  private
    FRegion: THandle;
    procedure FreeCurrentRegion;
  public
    destructor Destroy; override;
    procedure ActivateHint(Rect: TRect; const AHint: string); override;
    procedure Paint; override;
    procedure CreateParams(var Params: TCreateParams); override;
  end;

implementation

destructor TDDGHintWindow.Destroy;
begin
  FreeCurrentRegion;
  inherited Destroy;
end;
```

لیست ۱-۹ ادامه

```
procedure TDDGHintWindow.FreeCurrentRegion;
{ Regions, like other API objects, should be freed when you are  }
{ through using them.  Note, however, that you cannot delete a   }
{ region which is currently set in a window, so this method sets }
{ the window region to 0 before deleting the region object.      }
begin
  if FRegion <> 0 then begin              // if Region is alive...
    SetWindowRgn(Handle, 0, True);        // set win region to 0
    DeleteObject(FRegion);                // kill the region
    FRegion := 0;                         // zero out field
  end;
end;

procedure TDDGHintWindow.ActivateHint(Rect: TRect; const AHint: string);
{ Called when the hint is activated by putting the mouse pointer }
{ above a control. }
begin
  with Rect do
    Right := Right + Canvas.TextWidth('WWWW');  // add some slop
  BoundsRect := Rect;
  FreeCurrentRegion;
  with BoundsRect do
    { Create a round rectangular region to display the hint window }
    FRegion := CreateRoundRectRgn(0, 0, Width, Height, Width, Height);
  if FRegion <> 0 then
    SetWindowRgn(Handle, FRegion, True);          // set win region
  inherited ActivateHint(Rect, AHint);            // call inherited
end;

procedure TDDGHintWindow.CreateParams(var Params: TCreateParams);
{ We need to remove the border created on the Windows API-level }
{ when the window is created. }
begin
  inherited CreateParams(Params);
  Params.Style := Params.Style and not ws_Border;  // remove border
end;

procedure TDDGHintWindow.Paint;
{ This method gets called by the WM_PAINT handler.  It is }
{ responsible for painting the hint window. }
var
  R: TRect;
begin
  R := ClientRect;                    // get bounding rectangle
  Inc(R.Left, 1);                     // move left side slightly
  Canvas.Font.Color := clInfoText;    // set to proper color
  { paint string in the center of the round rect }
  DrawText(Canvas.Handle, PChar(Caption), Length(Caption), R,
```

لیست ۱-۹ ادامه

```
            DT_NOPREFIX or DT_WORDBREAK or DT_CENTER or DT_VCENTER);
end;

initialization
  Application.ShowHint := False;      // destroy old hint window
  HintWindowClass := TDDGHintWindow; // assign new hint window
  Application.ShowHint := True;       // create new hint window
end.
```

متدهای **CreateParams()** و **Paint()** به اندازه کافی صریح هستند. **CreateParams()** موقعیتی برای تنظیم ساختار الگوی پنجره قبل از ایجاد پنجره hint روی یک سطح API فراهم می‌کند. در این متد، الگوی **WS_BORDER** از کلاس پنجره حذف شده است تا از قرارگرفتن حاشیه چهارگوش در اطراف پنجره جلوگیری کند. رنگ متن با **clInfoText** تنظیم می‌شود که رنگ متن hint تعریف شده توسط سیستم است.

یک پنجره مستتر

متد **ActivateHint()** حاوی ابزاری برای ایجاد یک پنجره hint غیرچهارگوش است. دو فراخوانی API انجام این کار را ممکن می‌سازند: **CreateRoundRectRgn()** و **SetWindowRgn()**.

CreateRoundRectRgn() یک ناحیه کامل چهارگوش در پنجره‌ای خاص تعریف می‌کند. یک ناحیه *(region)*، یک شیء خاص API است که به شما اجازه می‌دهد طراحی‌های خاص پرکردن، و درج تصاویر را در آن انجام دهید. علاوه بر CreateRoundRectRgn() تعداد دیگری از توابع WIN32 API وجود دارند که انواع دیگری از نواحی را ایجاد می‌کنند:

- `CreateEllipticRgn()`
- `CreateEllipticRgnIndirect()`
- `CreatePolygonRgn()`
- `CreatePolyPolygonRgn()`
- `CreateRectRgn()`
- `CreateRectRgnIndirect()`
- `CreateRoundRectRgn()`
- `ExtCreateRegion()`

به علاوه، تابع CombineReg() را می‌توان برای ترکیب نواحی چندگانه در یک ناحیه پیچیده به کار برد. تمامی این توابع به صورت مفصل در راهنمای Win32 API شرح داده شده‌اند.

سپس ()SetWindowRgn فراخوانی شده و دستگیره ناحیه‌ای که اخیراً ایجاد شده است را به عنوان پارامتر رد می‌کند. این تابع سیستم عامل را وادار می‌کند که مالکیتی روی ناحیه اعمال کند و تمامی طراحی‌های درون پنجره فقط در داخل ناحیه ظاهر می‌شوند. بنابراین اگر ناحیه تعریف شده یک چهار ضلعی کامل باشد، تنها طراحی آن ناحیه چهار ضلعی ظاهر خواهد شد.

یادداشت

شما بایستی هنگام استفاده از ()SetWindowRgn مراقب دو تأثیر جانبی باشید. اولاً، از آنجا که فقط بخشی از پنجره در درون ناحیه نقاشی شده است، پنجره شما احتمالاً دارای قاب و یا نوار عنوان نخواهد بود. شما بایستی روشی را برای کاربران فراهم کنید که روشی انتخابی برای حرکت دادن، تغییر اندازه و بستن پنجره داشته باشد. دوماً، سیستم عامل مالکیت ناحیه مشخص شده در ()SetWindowRgn را در اختیار می‌گیرد، شما بایستی دقت کنید که هنگامی که این ناحیه در حال استفاده است آن را دستکاری و یا حذف نکنید. مؤلفه TDDGHintWindow این کار را با فراخوانی روش ()FreeCurrentRegion قبل از نابودی پنجره و یا ایجاد پنجره جدید انجام می‌دهد. ▲

کد مقداردهی اولیه برای یونیت **RndHint** عمل تبدیل **TDDGHintWindow** به یک پنجره hint کاربردی را انجام می‌دهد. تنظیم مقدار **Application.ShowHint** با **False** باعث می‌شود که پنجره hint قدیمی تخریب شود. در این لحظه بایستی کلاس وراثت **THintWindow** را به متغیر سراسری **HintWindowClass** نسبت دهید. سپس تنظیم مجدد **Application.ShowHint** به **True** باعث می‌شود که یک پنجره hint ایجاد شود. این مثالی از کلاس وراثت شما است.

فعال کردن TDDGHintWindow

آماده کردن این جزء سازندهٔ شبه بصری با آماده کردن اجزاء سازندهٔ Visual و NonVisual تفاوت دارد. از آنجا که کل کار معرفی اجزاء سازنده در بخش **initialization** لازم است تا عنوان یونیت مربوطه در عبارت **uses** یکی از فایل‌های منبع پروژه شما افزوده شود.

باز هم اجزاء سازنده

در گذشته در هنگام نوشتن برنامه‌های دلفی تصور می‌کردیم که "این برنامه برنامه‌ای معمولی است و بایستی چیزی به آن بیافزائیم که جذابیت آن بیشتر شود".

جزء سازنده marquee

اجازه بدهید نحوه کار marquee را تحلیل کنیم. **TCustomPanel** به عنوان مبنای کلاس برای مؤلفه **TddgMarquee** عمل می‌کند زیرا اعمال پایه‌ای مورد نیاز شما را در خود جای داده است و علاوه بر آن

لبه برجسته سه بعدی را نیز در خود دارد.

TddgMarquee رشته‌هایی متنی را در یک نگاشت بیتی مقیم حافظه رسم کرده و سپس بخش‌های نگاشت بیتی حافظه را در صفحه ترسیم خود کپی می‌کند تا با این عمل لغزاندن را شبیه‌سازی کند. جزء سازندهٔ مورد بحث با کمک تابع API **BitBlt()** کارش را انجام می‌دهد که بخشی از صفحه ترسیم را با اندازه‌ای که جزء سازنده مشخص می‌کند در آن کپی می‌کند. سپس بخشی از صفحه ترسیم حافظه را به پایین رانده و آن تصویر را در کنترل کپی می‌کند. مجدداً راندن و کپی کردن را اجراء نموده و این پروسه را مکرراً انجام می‌دهد به گونه‌ای که اجزاء صفحه ترسیم حافظه در درون جزء سازنده می‌لغزند.

اکنون زمان شناسایی کلاس‌های اضافی است که نیاز دارید آنها را به منظور فعال کردن در TddgMarquee گردآوری کنید. در حقیقت دو کلاس با این ویژگی وجود دارد.

ابتدا نیاز به کلاس TStringList دارید تا تمامی رشته‌هایی که قصد لغزاندن آنها را دارید را نگهداری کند. سپس بایستی یک نگاشت بیتی حافظه داشته باشید که بتوانید روی آن تمامی رشته‌های متنی را ارائه نمایید. مؤلفه TBitmap از VCL این کار را به خوبی انجام می‌دهد.

نوشتن اجزاء سازنده

همانند اجزاء دیگر که در این فصل معرفی شدند کد مربوطه به TddMarquee بایستی به یک طرح منطقی نزدیک شود. در این حالت کار کد را به بخش‌هایی معقول تقسیم می‌کنیم. مؤلفه TddMarquee را می‌توان به پنج بخش اصلی تقسیم کرد.

- مکانیزمی که متن را به صحفه ترسیمی حافظه منتقل می‌کند.
- مکانیزمی که متن را از صفحه ترسیمی حافظه در پنجره marquee کپی می‌نماید.
- زمان‌سنجی که بر نحوه و زمان لغزاندن پنجره به منظور اجرای متحرک‌سازی نظارت می‌کند.
- سازنده و مخرب کلاس و متدهای مرتبط با آن.
- کارهای پایانی نظیر صفات و متدهای راهنما.

ترسیم روی یک نگاشت بیتی خارج از صفحه

هنگام ایجاد نمونه‌ای از TBitmap بایستی بدانید که چه اندازه‌ای داشته باشد تا لیست موردنظر از رشته‌ها را در حافظه نگهداری کند. این کار را با سنجیدن طول هر سطر متن و ضرب کردن آن در تعداد سطرها انجام خواهید داد. برای یافتن طول و فضای هر سطر متن در با قلمی خاص از تابع **API**، GetTextMetrics() استفاده کرده و آن را به دستگیره صفحه ترسیم بفرستید یک رکورد TTextMetric توسط تابع زیر پر می‌شود:

```
var
  Metrics: TTextMetric;
begin
  GetTextMetrics(Canvas.Handle, Metrics);
```

تذکر

این تابع علاوه بر اطلاعاتی در مورد عرض و پهنای قلم، سیاه، ایتالیک بودن آن و حتی مواردی نظیر نام مجموعه قلم را ارائه می‌کند.

متد ()TextHeight از Tcanvas در اینجا کار نمی‌کند. این متد فقط ارتفاع یک سطر خاص متن را مشخص می‌کند.

فیلد tmHeight از رکورد Metrics ارتفاع یک سلول کاراکتر را در قلم جاری صفحهٔ ترسیم مشخص می‌کند. اگر به آن مقدار فیلد tmInternalLeading را بیافزایید تا فضایی بین سطرها ایجاد شود ارتفاع هر سطر متن که در صفحه ترسیم رسم می‌شود را خواهید داشت:

```
LineHi : = Metrics,tmHeight + Metrics.tmInternalLeading;
```

اندازه لازم برای صفحه ترسیم حافظه را اکنون می‌توان با ضرب LineHi در تعداد سطرهای متن و افزودن حاصل به ارتفاع کنترل TddgMarquee (برای ایجاد یک فضای خالی در آغاز و پایان Marquee) بدست آورد. تصور کنید که TstringList که در آن تمامی رشته‌ها قرار دارند **FItems** نامیده شود؛ اکنون ابعاد صفحه ترسیم حافظه را در یک ساختمان TRect قرار دهید:

```
var
  VRect: TRect;
begin
  { VRect rectangle represents entire memory bitmap }
  VRect := Rect(0, 0, Width, LineHi * FItems.Count + Height * 2);
end;
```

پس از معرفی و تعیین اندازه، نگاشت بیتی حافظه با تنظیم قلم به منظور انطباق با صفت Font مربوط به TddgMarquee ، پر کردن پس‌زمینه با رنگی که توسط صفت Color مربوط به TddgMarquee مشخص می‌شود، و تنظیم صفت Style مربوط به Brush با bsClear مقداردهی اولیه کامل خود را به دست می‌آورد.

■ نکته

هنگامی که متن را روی TCanvas منتقل می‌کنید، پس‌زمینه متن با رنگ جاری TCavas.Brush پر می‌شود. برای اینکه پس‌زمینه غیرقابل رؤیت شود مقدار TCanvas.Brush.Style را به bsClear تنظیم کنید.

مقدمات کار اکنون آماده است، بنابراین وقت آن است که متن را به حافظه نگاشت بیتی تحویل دهیم. ساده‌ترین راه برای دادن متن به یک صفحه ترسیم استفاده از متد ()TextOut از TCanvas است؛ البته

هنگامی که از تابع پیچیده‌تر API ، ()DrawText استفاده کنید کنترل بیشتری روی قالب‌بندی متن خواهید داشت. به دلیل نیاز به کنترل روی تقدم‌ها، TddgMarquee از تابع ()DrawText استفاده می‌کند.

```
type
   Tyustification = (tjCenter, tjLeft, tjRight);
```

کد زیر متد ()OainLine را برای TddgMarquee نشان می‌دهد که باعث استفاده از ()DrawText برای ارائه متن به نگاشت بیتی می‌شود. در این متد، FJust یک متغیر از نوع TJustification ارائه می‌کند.

```
procedure TddgMarquee.PaintLine(R: TRect; LineNum: Integer);
{ this method is called to paint each line of text onto MemBitmap }
const
  Flags: array[TJustification] of DWORD = (DT_CENTER, DT_LEFT, DT_RIGHT);
var
  S: string;
begin
  { Copy next line to local variable for clarity }
  S := FItems.Strings[LineNum];
  { Draw line of text onto memory bitmap }
  DrawText(MemBitmap.Canvas.Handle, PChar(S), Length(S), R,
    Flags[FJust] or DT_SINGLELINE or DT_TOP);
end;
```

ترسیم اجزاء سازنده

اکنون که نحوه ایجاد نگاشت بیتی و ترسیم متن را در آن می‌دانید گام بعدی فراگیری نحوه کپی آن متن در صفحه رسم TddgMarquee است.

متد ()Paint از یک جزء سازنده در پاسخ به یک پیغام WM_PAINT از طرف windows به کار گرفته می‌شود. متد ()Paint به جزء سازنده شما حیات می‌دهد؛ شما از متد ()Paint برای ترسیم، نقاشی، و تشخیص نحوه نمایش گرافیکی اجزاء سازندهٔ خود استفاده می‌کنید. وظیفه ()TddMarquee.Paint کپی رشته‌ها از صفحه ترسیم حافظه به صفحه ترسیم TddgMarquee است. این کار با کمک تابع API ، ()BitBLt انجام می‌شود که بیت‌ها را از یک متن ابزار به دیگری کپی می‌کند.

برای تشخیص اینکه TddgMarquee در حال اجرا شدن است، از یک متغیر بولی به نام FActive استفاده می‌شود که فعال شدن قابلیت لغزاندن marquee را مشخص می‌کند. بنابراین متد ()Paint بسته به اینکه جزء سازنده فعال باشد ترسیمات متفاوتی انجام می‌دهد.

```
procedure TddgMarquee.Paint;
{ this virtual method is called in response to a }
{ Windows paint message }
begin
  if FActive then
```

```
    { Copy from memory bitmap to screen }
    BitBlt(Canvas.Handle, 0, 0, InsideRect.Right, InsideRect.Bottom,
      MemBitmap.Canvas.Handle, 0, CurrLine, srcCopy)
  else
    inherited Paint;
end;
```

اگر marquee فعال باشد، جزء سازنده از تابع BitBlt() برای ترسیم بخشی از صفحه رسم حافظه در صفحه رسم TddgMarquee استفاده می‌کند. به متغیر CurrLine دقت کنید که به عنوان پارامتر BitBlt() فرستاده می‌شود. مقدار این پارمتر مشخص می‌کند که کدام بخش از صفحه ترسیم حافظه به صفحه فرستاده می‌شود. با افزایش یا کاهش پی‌درپی مقدار CurrLine می‌توانید به TddgMarquee جلوه‌ای بدهید که گویی متن به پایین یا بالا لغزانده می‌شود.

متحرک‌سازی Marquee

جنبه‌های بصری TddgMarquee اکنون آماده هستند. بقیه کاری که برای به انجام رساندن عمل این جزء سازنده انجام می‌شود انجام برخی محاسبات است. اکنون TddgMarquee نیاز به مکانیزمی برای تغییر مقدار CurrLine دارد تا جزء سازنده را مجدداً ترسیم نماید. این کار را می‌توان به سادگی و با استفاده از جزء سازندهٔ TTimer در دلفی انجام داد.

قبل از آنکه بتوانید از TTimer استفاده کنید بایستی کلاس را ایجاد و مقداردهی اولیه کنید. TddgMarquee دارای TTimer‌ای به نام FTimer خواهد بود و آن را در روالی به نام DoTimer مقداردهی اولیه خواهید نمود.

```
procedure DoTimer;
{ procedure sets up TddgMarquee's timer }
begin
  FTimer := TTimer.Create(Self);
  with FTimer do
  begin
    Enabled := False;
    Interval := TimerInterval;
    OnTimer := DoTimerOnTimer;
  end;
end;
```

در این روال، FTmier ایجاد شده و در آغاز غیرفعال شده است. صفت Interval این جزء به مقدار ثابتی به نام TimerInterval تخصیص مقدار شده است. در پایان، رویداد OnTimer برای FTimer به متدی از TddgMarquee تخصیص مقدار شده است. این متدی است که هنگام وقوع رویداد OnTimer فراخوانی خواهد شد.

تذکر

هنگام تخصیص مقادیر در کد خود بایستی دو قانون زیر را در نظر داشته باشید:

- **روالی که به رویداد نسبت می‌دهید بایستی متدی از یک شی باشد. این روال می‌تواند یک تابع یا روال مستقل باشد.**
- **متدی که به رویداد نسبت می‌دهید بایستی لیست پارامتری مشابه با نوع رویداد را بپذیرد. به عنوان مثال، رویداد *OnTimer* برای TTimer از نوع TNotifyEvent است از آنجا که TNotifyEvent یک پارامتر، Sender ، از نوع TObject را می‌پذیرد هر متدی که به OnTimer تخصیص می‌دهید بایستی یک پارامتر و از نوع TObject داشته باشد.**

متد ()DoTimerOnTimer به شکل زیر تعریف می‌شود.

```
procedure TddgMarquee.DoTimerOnTimer(Sender:  TObject);
{ This method is executed in response to a timer event }
begin
  IncLine;
  { only repaint within borders }
  InvalidateRect(Handle, @InsideRect, False);
end;
```

در این متد، یک روال به نام ()IncLine فراخوانی شده است، این روال مقدار CurrLine را در صورت لزوم افزایش یا کاهش می‌دهد. سپس تابع API ()InvalidateRect برای خنثی کرن (یا ترسیم مجدد) بخش درونی مؤلفه فراخوانی می‌شود. به جای انتخاب و استفاده از متد ()Invalidate مربوط به TCanvas متد ()InvalidRect را برمی‌گزینیم زیرا ()Invalidate باعث می‌شود که کل صفحه رسم تغییر کند و نه بخش درون چهارگوش انتخاب شده. این متد، از آنجا که جزء سازندهٔ مورد نظر را مکرراً رنگ‌کاری مجدد نمی‌کند بخش زیادی از پرش تصویر را برطرف می‌سازد. در نظر داشته باشید که پرش تصویر نامطلوب است.

متد ()IncLine که مقدار CurrLine را بهنگام‌سازی کرده و کامل شدن عمل لغزش را تشخیص می‌دهد به شکل زیر تعریف می‌شود.

```
procedure TddgMarquee.IncLine;
{ this method is called to increment a line }
begin
  if not FScrollDown then        // if Marquee is scrolling upward
  begin
    { Check to see if marquee has scrolled to end yet }
    if FItems.Count * LineHi + ClientRect.Bottom -
      ScrollPixels  >= CurrLine then
      { not at end, so increment current line }
      Inc(CurrLine, ScrollPixels)
```

```
    else SetActive(False);
  end
  else begin                    // if Marquee is scrolling downward
    { Check to see if marquee has scrolled to end yet }
    if CurrLine >= ScrollPixels then
      { not at end, so decrement current line }
      Dec(CurrLine, ScrollPixels)
    else SetActive(False);
  end;
end;
```

سازنده TddgMarquee به اندازه کافی ساده است. این سازنده متد وراثتی ()Create را فراخوانی کرده، یک وهله TStringList ایجاد نموده، FTimer را تنظیم کرده و سپس تمامی مقادیر پیش‌فرض را برای متغیرها تنظیم می‌نماید. مجدداً، بایستی به خاطر داشته باشید که متد ()Create را در جزء سازنده خود فراخوانی کنید. نقصان در انجام این کار باعث می‌شود که در کارائی متد خلل حاصل شود و اعمالی نظیر ایجاد دستگیره و صفحه ترسیم، جریان، و پاسخهای Windows دچار اشکال شوند. کد زیر سازنده TddgMarquee یعنی ()Create را نشان می‌دهد.

```
constructor TddgMarquee.Create(AOwner: TComponent);
{ constructor for TddgMarquee class }

  procedure DoTimer;
  { procedure sets up TddgMarquee's timer }
  begin
    FTimer := TTimer.Create(Self);
    with FTimer do
    begin
      Enabled := False;
      Interval := TimerInterval;
      OnTimer := DoTimerOnTimer;
    end;
  end;

begin
  inherited Create(AOwner);
  FItems := TStringList.Create;  { instantiate string list }
  DoTimer;                       { set up timer }
  { set instance variable default values }
  Width := 100;
  Height := 75;
  FActive := False;
  FScrollDown := False;
  FJust := tjCenter;
  BevelWidth := 3;
end;
```

مخرب TddgMarquee از این هم ساده‌تر است: متد با ارسال False به متد ()SetActive مؤلفه را بی‌اثر کرده، زمان‌سنج و لیست رشته را آزاد نموده، و سپس متد موروثی ()Destroy را فراخوانی می‌کند.

```
destructor TddgMarquee.Destroy;
{ destructor for TddgMarquee class }
begin
  SetActive(False);
  FTimer.Free;              // free allocated objects
  FItems.Free;
  inherited Destroy;
end;
```

یادداشت

هنگامی که سازنده‌ها را باطل می‌کنید، اغلب ابتدا inherited را فراخوانی می‌نمایید، و هنگامی که مخرب‌ها را باطل می‌کنید اغلب inherited را در انتهای کار فراخوانی می‌کنید. به خاطر سپردن "اولین ورودی، آخرین خروجی" عملی مفید است. با این کار این اطمینان حاصل می‌شود که کلاس قبل از تغییر توسط شما تنظیم شده است و تمام منابع مرتبط قبل از آزادسازی کلاس پاک شده‌اند.

برای این قانون استثناءهایی وجود دارند؛ البته، بایستی در حالت کلی به این قانون استناد کنید مگر آنکه دلیل قانع کننده‌ای برای عدم انجام آن وجود داشته باشد. ▲

متد ()SetActive که توسط متد ()IncLine و مخرب فراخوانی می‌شود (علاوه بر ایفای نقش به عنوان نویسنده‌ای برای صفت Active) در نقش حاملی عمل می‌کند که باعث شروع و توقف لغزاندن صفحه ترسیم توسط marquee می‌شود.

```
procedure TddgMarquee.SetActive(Value: Boolean);
{ called to activate/deactivate the marquee }
begin
  if Value and (not FActive) and (FItems.Count > 0) then
  begin
    FActive := True;                  // set active flag
    MemBitmap := TBitmap.Create;
    FillBitmap;                       // Paint Image on bitmap
    FTimer.Enabled := True;           // start timer
  end
  else if (not Value) and FActive then
  begin
    FTimer.Enabled := False;    // disable timer,
    if Assigned(FOnDone)        // fire OnDone event,
      then FOnDone(Self);
    FActive := False;           // set FActive to False
    MemBitmap.Free;             // free memory bitmap
    Invalidate;                 // clear control window
  end;
end;
```

یک جنبه مهم از TddgMarquee که قبلاً وجود نداشت آن است که کاربر را از اتمام عمل لغزاندن مطلع می‌سازد. اولین گام برای افزودن یک رویداد به جزء سازنده، اعلان یک متغیر از یک نوع رویداد در بخش Private تعریف کلاس است. برای رویداد FonDone از نوع TNotifyEvent استفاده می‌شود.

```
FOnDone : TNotifyEvent;
```

رویداد سپس بایستی در بخش Published کلاس به عنوان یک صفت تعریف شود.

```
Property OnDone: TNotifyEvient tead FOnDone write FOnDone;
```

به خاطر بیاورید که دستورات read و write مشخص می‌کنند که یک صفت بایستی مقدار خود را از کدام تابع یا متغیر دریافت کند.

با انجام این دو کار ساده باعث می‌شویم که یک ورودی برای OnDone در صفحه Events مربوط به Object Inspector در زمان طراحی نمایش داده شود. تنها کار دیگری که بایستی انجام شود فراخوانی نگهدارنده OnDone به شکلی است که توسط TddgMarquee در خط کد زیر در متد Deactivate() نشان داده شده است.

```
if Assigned (FOnDone) then FOnDone (self); // fire OnDone event
```

این سطر بدین گونه تفسیر می‌شود: "اگر کاربر جزء سازنده، متدی را به رویداد OnDone نسبت داده است، آن متد فراخوانی شده و کلاس TddgMarquee (self) به عنوان پارامتر ارسال شود".

لیست برنامه ۲-۹ کد مربوط به یونیت Marquee را نشان می‌دهد. توجه داشته باشید که از آنجا که جزء سازنده از یک کلاس TCustonXXX ارث می‌برد بایستی بسیاری از صفات فراهم شده توسط TCustomPanel را انتشار دهید.

■ نکته

به دستور default و مقدار به کار رفته به همراه صفت Justify از TddgMarquee توجه کنید. با این استفاده از default جزء سازنده بهینه می‌شود و کارائی زمان طراحی جزء سازنده بالا می‌رود. می‌توانید مقادیر پیش‌فرض را به صفات هر یک از نوع‌های اصلی بدهید (Longint ، Word، Integer) و همچنین نوع‌های تعریف شده)، ولی این کار را نمی‌توان با نوع صفات غیرترتیبی نظیر رشته‌ها، اعداد ممیز شناور، رکوردها، و کلاس‌ها انجام داد.

همچنین بایستی مقادیر پیش‌فرض صفات را در سازنده خود مقداردهی اولیه کنید، بروز خطا در این عمل، باعث بروز مشکلات در جریان جزء سازنده می‌شود.

لیست ۲-۹ یونیت Marquee.Pas

```
unit Marquee;

interface

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

const
  ScrollPixels = 3;      // num of pixels for each scroll
  TimerInterval = 50;    // time between scrolls in ms

type
  TJustification = (tjCenter, tjLeft, tjRight);

  EMarqueeError = class(Exception);

  TddgMarquee = class(TCustomPanel)
  private
    MemBitmap: TBitmap;
    InsideRect: TRect;
    FItems: TStringList;
    FJust: TJustification;
    FScrollDown: Boolean;
    LineHi : Integer;
    CurrLine : Integer;
    VRect: TRect;
    FTimer: TTimer;
    FActive: Boolean;
    FOnDone: TNotifyEvent;
    procedure SetItems(Value: TStringList);
    procedure DoTimerOnTimer(Sender:  TObject);
    procedure PaintLine(R: TRect; LineNum: Integer);
    procedure SetLineHeight;
    procedure SetStartLine;
    procedure IncLine;
    procedure SetActive(Value: Boolean);
  protected
    procedure Paint; override;
    procedure FillBitmap; virtual;
  public
```

لیست ۲-۹ ادامه

```
    property Active: Boolean read FActive write SetActive;
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
  published
    property ScrollDown: Boolean read FScrollDown write FScrollDown;
    property Justify: TJustification read FJust write FJust default tjCenter;
    property Items: TStringList read FItems write SetItems;
    property OnDone: TNotifyEvent read FOnDone write FOnDone;
    { Publish inherited properties: }
    property Align;
    property Alignment;
    property BevelInner;
    property BevelOuter;
    property BevelWidth;
    property BorderWidth;
    property BorderStyle;
    property Color;
    property Ctl3D;
    property Font;
    property Locked;
    property ParentColor;
    property ParentCtl3D;
    property ParentFont;
    property Visible;
    property OnClick;
    property OnDblClick;
    property OnMouseDown;
    property OnMouseMove;
    property OnMouseUp;
    property OnResize;
  end;

implementation

constructor TddgMarquee.Create(AOwner: TComponent);
{ constructor for TddgMarquee class }

  procedure DoTimer;
  { procedure sets up TddgMarquee's timer }
  begin
    FTimer := TTimer.Create(Self);
    with FTimer do
    begin
```

لیست ۲-۹ ادامه

```
      Enabled := False;
      Interval := TimerInterval;
      OnTimer := DoTimerOnTimer;
    end;
  end;

begin
  inherited Create(AOwner);
  FItems := TStringList.Create;  { instantiate string list }
  DoTimer;                       { set up timer }
  { set instance variable default values }
  Width := 100;
  Height := 75;
  FActive := False;
  FScrollDown := False;
  FJust := tjCenter;
  BevelWidth := 3;
end;

destructor TddgMarquee.Destroy;
{ destructor for TddgMarquee class }
begin
  SetActive(False);
  FTimer.Free;              // free allocated objects
  FItems.Free;
  inherited Destroy;
end;
procedure TddgMarquee.DoTimerOnTimer(Sender: TObject);
{ This method is executed in response to a timer event }
begin
  IncLine;
  { only repaint within borders }
  InvalidateRect(Handle, @InsideRect, False);
end;

procedure TddgMarquee.IncLine;
{ this method is called to increment a line }
begin
  if not FScrollDown then       // if Marquee is scrolling upward
  begin
    { Check to see if marquee has scrolled to end yet }
    if FItems.Count * LineHi + ClientRect.Bottom -
      ScrollPixels  >= CurrLine then
```

لیست ۲-۹ ادامه

```
      { not at end, so increment current line }
      Inc(CurrLine, ScrollPixels)
    else SetActive(False);
  end
  else begin                    // if Marquee is scrolling downward
    { Check to see if marquee has scrolled to end yet }
    if CurrLine >= ScrollPixels then
      { not at end, so decrement current line }
      Dec(CurrLine, ScrollPixels)
    else SetActive(False);
  end;
end;

procedure TddgMarquee.SetItems(Value: TStringList);
begin
  if FItems <> Value then
    FItems.Assign(Value);
end;

procedure TddgMarquee.SetLineHeight;
{ this virtual method sets the LineHi instance variable }
var
  Metrics : TTextMetric;
begin
  { get metric info for font }
  GetTextMetrics(Canvas.Handle, Metrics);
  { adjust line height }
  LineHi := Metrics.tmHeight + Metrics.tmInternalLeading;
end;
procedure TddgMarquee.SetStartLine;
{ this virtual method initializes the CurrLine instance variable }
begin
  // initialize current line to top if scrolling up, or...
  if not FScrollDown then CurrLine := 0
  // bottom if scrolling down
  else CurrLine := VRect.Bottom - Height;
end;

procedure TddgMarquee.PaintLine(R: TRect; LineNum: Integer);
{ this method is called to paint each line of text onto MemBitmap }
const
```

لیست ۲-۹ ادامه

```
  Flags: array[TJustification] of DWORD = (DT_CENTER, DT_LEFT, DT_RIGHT);
var
  S: string;
begin
  { Copy next line to local variable for clarity }
  S := FItems.Strings[LineNum];
  { Draw line of text onto memory bitmap }
  DrawText(MemBitmap.Canvas.Handle, PChar(S), Length(S), R,
    Flags[FJust] or DT_SINGLELINE or DT_TOP);
end;

procedure TddgMarquee.FillBitmap;
var
  y, i : Integer;
  R: TRect;
begin
  SetLineHeight;                  // set height of each line
  { VRect rectangle represents entire memory bitmap }
  VRect := Rect(0, 0, Width, LineHi * FItems.Count + Height * 2);
  { InsideRect rectangle represents interior of beveled border }
  InsideRect := Rect(BevelWidth, BevelWidth, Width - (2 * BevelWidth),
    Height - (2 * BevelWidth));
  R := Rect(InsideRect.Left, 0, InsideRect.Right, VRect.Bottom);
  SetStartLine;
  MemBitmap.Width := Width;       // initialize memory bitmap
  with MemBitmap do
  begin
    Height := VRect.Bottom;
    with Canvas do
    begin
      Font := Self.Font;
    Brush.Color := Color;
    FillRect(VRect);
    Brush.Style := bsClear;
  end;
end;
y := Height;
i := 0;
repeat
```

لیست ۲-۹ ادامه

```
      R.Top := y;
      PaintLine(R, i);
      { increment y by the height (in pixels) of a line }
      inc(y, LineHi);
      inc(i);
    until i >= FItems.Count;       // repeat for all lines
  end;

  procedure TddgMarquee.Paint;
  { this virtual method is called in response to a }
  { Windows paint message }
  begin
    if FActive then
      { Copy from memory bitmap to screen }
      BitBlt(Canvas.Handle, 0, 0, InsideRect.Right, InsideRect.Bottom,
        MemBitmap.Canvas.Handle, 0, CurrLine, srcCopy)
    else
      inherited Paint;
  end;

  procedure TddgMarquee.SetActive(Value: Boolean);
  { called to activate/deactivate the marquee }
  begin
    if Value and (not FActive) and (FItems.Count > 0) then
    begin
      FActive := True;                 // set active flag
      MemBitmap := TBitmap.Create;
      FillBitmap;                      // Paint Image on bitmap
      FTimer.Enabled := True;          // start timer
    end
    else if (not Value) and FActive then
    begin
      FTimer.Enabled := False;    // disable timer,
      if Assigned(FOnDone)        // fire OnDone event,
        then FOnDone(Self);
      FActive := False;           // set FActive to False

     MemBitmap.Free;             // free memory bitmap
     Invalidate;                 // clear control window
   end;
end;

end.
```

تست کردن TddgMarquee

علیرغم اینکه خاتمه ایجاد این جزء سازنده و قرارگیری آن در مرحله تست بسیار هیجان‌انگیز است، هنوز زمان افزودن این جزء سازنده به Component Palette نیست و ابتدا بایستی آن را اشکال‌زدایی نماییم. بایستی تمامی تست‌های مقدماتی را با ایجاد پروژه‌ای که یک نمونهٔ دینامیک را ایجاد کرده و به کار می‌برد انجام دهید. لیست ۳-۹ این یونیت را برای پروژه‌ای به نام TestMarq شرح می‌دهد که برای تست TddgMarquee به کار می‌رود. این پروژه ساده از فرمی تشکیل شده است که دو دکمه دارد.

■ نکته

همواره برای اجزاء سازندهٔ جدید خود یک پروژه تست ایجاد کنید. هرگز با اضافه کردن یک جزء سازنده به Component Platette سعی در انجام تست‌های آغازین نکنید. با سعی در اشکال‌زدایی اجزایی که در یک پالت مستقر است نه تنها با انجام عمل بیهوده ساخت مجدد بسته، زمان خود را هدر می‌دهید، بلکه امکان آن وجود دارد که به عنوان نتیجه ایرادی در جزء سازندهٔ شما، در IDE ایرادی حاصل شود.

لیست ۳-۹ TestU.pas -جزء سازندهٔ TddgMarquee را تست می‌کند

```
unit Testu;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, Marquee, StdCtrls, ExtCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
```

لیست ۳-۹ ادامه

```
  private
    Marquee1: TddgMarquee;
    procedure MDone(Sender: TObject);
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.MDone(Sender: TObject);
begin
  Beep;
end;

procedure TForm1.FormCreate(Sender:  TObject);
begin
  Marquee1 := TddgMarquee.Create(Self);
  with Marquee1 do
  begin
    Parent := Self;
    Top := 10;
    Left := 10;
    Height := 200;
    Width := 150;
    OnDone := MDone;
    Show;
    with Items do
    begin
      Add('Greg');
      Add('Peter');
      Add('Bobby');
      Add('Marsha');
      Add('Jan');
      Add('Cindy');
    end;
  end;
end;
```

لیست ۳-۹ ادامه

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Marquee1.Active := True;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Marquee1.Active :=  False;
end;

end.
```

پس از آنکه تمامی ایرادهای یافت شده در برنامه از بین رفتند زمان افزودن برنامه به Component Palette است. همانگونه که به خاطر دارید انجام این کار بسیار آسان است: جزء سازنده‌ای را انتخاب کرده، آن را از منوی اصلی نصب کرده و سپس نام فایل یونیت و نام بسته را در جعبه مکالمه Install Component وارد کنید. روی OK کلیک نمایید و نتیجتاً Delphi بسته‌ای که جزء سازنده در آن اضافه می‌شود را بازسازی مجدد نموده و Component Palette را بهنگام‌سازی می‌نماید. البته، جزء سازندهٔ شما نیاز دارد که یک روال ()Register را به منظور قرارگیری در Component Palette آماده کند.

نوشتن ویرایشگرهای صفت

در فصل ۸ نشان دادیم که صفات چگونه برای اکثر نوع‌های صفات رایج در Object Inspector ویرایش می‌شوند. هدفی که طبق آن یک صفت ویرایش می‌شود به کمک ویرایشگر صفت آن مشخص می‌شود. برخی ویرایشگرهای صفت از پیش تعریف شده برای صفات موجود به کار رفته‌اند. البته ممکن است حالتی وجود داشته باشد که در آن هیچ یک از ویرایشگرهای از پیش تعریف شده نیازهای شما را ارضاء نکنند. با بروز چنین وضعیتی بایستی ویرایشگر خود را برای آن صفت ایجاد کنید.

می‌توانید صفات را به دو طریق در Object Inspector ویرایش کنید. یک راه دادن امکان ویرایش مقادیر به صورت متنی به کاربر است. راه دیگر استفاده از مکالمه‌ای است که عمل ویرایش صفت را به انجام می‌رساند. در برخی از حالات بایستی هر دو امکان را برای یک صفت مهیا سازید. در اینجا مراحل مورد نیاز برای نوشتن یک ویرایشگر صفت ذکر شده‌اند:

۱- ایجاد یک شیء ویرایشگر صفت موروثی.
۲- ویرایش متنی صفت.
۳- ویرایش صفت به صورت کلی و توسط یک مکالمه (اختیاری).
۴- تخصیص ویژگیهای ویرایشگر صفت.
۵- ثبت ویرایشگر صفت.

ایجاد یک شیء ویرایشگر صفت موروثی

دلفی ویرایشگرهای صفت متعددی در DesignEditors.pas تعریف می‌کند که همه آنها از کلاس مبنای TPropertyEditor ارث می‌برند. هنگامی که یک ویرایشگر صفت را ایجاد می‌کنید، ویرایشگر صفت شما بایستی از TPropertyEditor و یا یکی از نسل‌های آن ارث ببرد. جدول ۱-۹ نسل‌های TPropertyEditor که با صفات موجود به کار می‌روند را نشان می‌دهد.

جدول ۱-۹ ویرایشگرهای صفت که در DesignEditors.pas تعریف شده‌اند

ویرایشگر صفت	توصیف
TordinalProperty	کلاس مبنا برای تمامی ویرایشگرهای صفت ترتیبی، نظیر TIntegerPropery ، TEnumProperty ، TCharProperty ، و ...
TIntegerProperty	ویرایشگر صفت پیش‌فرض برای صفات عددی صحیح با اندازه‌های مختلف
TCharProperty	ویرایشگر صفت برای صفاتی که از یک نوع char و زیرنوعی از char هستند؛ "A" "Z" ... .
TEnumProperty	صفت پیش‌فرض برای تمامی نوع‌های تعریف شده توسط کاربر
TFloatProperty	ویرایشگر صفت پیش‌فرض برای صفات عددی ممیز شناور
TStringProperty	ویرایشگر صفت پیش‌فرض برای صفات رشته‌ای
TSetElementProperty	ویرایشگر صفت پیش‌فرض برای عناصر set . هر عنصر مجموعه به عنوان یک گزینه بولی نمایش داده می‌شود.
TsetProperty	ویرایشگر صفت پیش‌فرض برای صفات set . مجموعه برای هر عنصر به عناصر مجموعه‌ای مجزا تفکیک می‌شود.
TClassProperty	ویرایشگر صفت پیش‌فرض برای صفاتی که خود شیء هستند.
TMethodProperty	ویرایشگر صفت پیش‌فرض برای صفاتی که اشاره‌گر به متد هستند، یعنی رویدادها.
TComponentProperty	ویرایشگر صفت پیش‌فرض برای صفاتی که به اجزاء سازنده اشاره می‌کنند. این مشابه با ویرایشگر TClassProperty نیست. در عوض، این ویرایشگر به کاربران اجازه می‌دهد که جزء سازنده‌ای که صفت به آن مراجعه می‌کند را مشخص نماید یعنی ActiveControl .
TCloroProperty	ویرایشگر صفت پیش‌فرض برای صفات با نوع TColor .
TFontNameProperty	ویرایشگر صفت پیش‌فرض برای صفاتی از نوع TFont ، که امکان ویرایش زیرصفات را فراهم می‌کند.
TFontProperty	امکان ویرایش زیر صفات را می‌دهد زیرا از TClassProperty اشتقاق یافته است.
TInt64Property	ویرایشگر صفت پیش‌فرض برای Int64 و اشتقاق‌های آن.
TNestedProperty	این ویرایشگر صفت از ویرایشگر صفت پدر خود استفاده می‌کند.
TClassProperty	ویرایشگر صفت پیش‌فرض برای اشیاء.
TMethodProperty	ویرایشگر صفت پیش‌فرض برای متدها.
TInterfaceProperty	ویرایشگر صفت پیش‌فرض برای Interface ها.
TDateProperty	ویرایشگر صفت پیش‌فرض برای بخش تاریخ یک نوع صفت TDateTime .
TTimeProperty	ویرایشگر صفت برای بخش زمان صفت TDaceTime .

جدول ۱-۹ ادامه

ویرایشگر صفت	توصیف
TComponentNameProperty	ویرایشگر صفت برای نام جزء سازنده. این ویرایشگر هنگامی که بیشتر از یک جزء سازنده انتخاب شده باشد صفت Name را از نمایش داده‌شده منع می‌کند.
TDateTimeProperty	ویرایشگر صفت مربوط به یک نوع صفت TDateTime .
TVariantProperty	ویرایشگر صفت برای نوع‌های دیگر.

ویرایشگر صفتی که ویرایشگر صفت شما بایستی از آن ارث برد بستگی به رفتار صفت در هنگام ویرایش دارد. در برخی حالات صفت شما ممکن است نیاز به عملکردی مشابه با TIntegerProperty داشته باشد. ولی ممکن است منطق اضافی در پروسه ویراش نیز داشته باشد. بنابراین منطقی به نظر می‌رسد که ویرایشگر صفت شما از TIntegerProperty ارث ببرد.

■ نکته

در نظر داشته باشید که حالاتی پیش می‌آیند که نیاز به ایجاد ویرایشگر صفتی که وابسته به نوع صفت شما است ندارید. به عنوان مثال نوع‌های subrange به صورت اتوماتیک کنترل می‌شوند (به عنوان مثال 1..10 توسط TIntegerProperty کنترل می‌شوند)، نوع‌های تعریف شده لیستی در اختیار شما قرار می‌دهند و ... بایستی به جای ویرایشگرهای صفت سعی در استفاده از تعاریف نوع کنید.

ویرایشگر صفت به صورت متنی

ویرایشگر صفت دو هدف پایه‌ای دارد: یکی اینکه امکان ویرایش صفت را به کاربر می‌دهد، این موضوع مشخص است. هدف دیگر که چندان واضح نیست فراهم کردن نمایش رشته‌ای مقدار صفت برای Object Insperctor است به گونه‌ای که بتواند به درستی نمایش داده شود.

هنگامی که یک کلاس ویرایشگر موروثی صفت را ایجاد می‌کنید بایستی متدهای ()GetValue و ()SetValue را باطل نماید. ()GetValue نمایش رشته‌ای مقدار صفت را به منظور نمایش به Object Inspector ارجاع می‌دهد. ()SetValue مقدار مبتنی بر نمایش رشته‌ای را به همان شکلی که وارد Object Inspector شده ارجاع می‌دهد.

به عنوان یک مثال تعریف نوع کلاس TintegerProerty را به شکلی که DSGINTF.PAS تعریف شده است در نظر بگیرید.

```
TIntegerProperty = class(TOrdinalProperty)
public
  function GetValue: string; override;
  procedure SetValue(const Value: string); override;
end;
```

Here, you see that the `GetValue()` and `SetValue()` methods have been overridden. The `GetValue()` implementation is as follows:

```
function TIntegerProperty.GetValue: string;
begin
  Result :=  IntToStr(GetOrdValue);
end;
```

Here's the `SetValue()` implementation:

```
procedure TIntegerProperty.SetValue(const Value: String);
var
  L: Longint;
begin
  L := StrToInt(Value);
  with GetTypeData(GetPropType)^ do
    if (L < MinValue) or (L > MaxValue) then
      raise EPropertyError.CreateResFmt(SOutOfRange, [MinValue, MaxValue]);
  SetOrdValue(L);
end;
```

GetValue() نمایش رشته‌ای یک صفت عددی صحیح را برمی‌گرداند. Object Inspector از این مقدار برای نمایش مقدار صفت استفاده می‌کند. GetOrdValue() متدی است که توسط TProperyEditor تعریف شده و برای بازیابی مقدار صفتی که توسط ویرایشگر صفت به آن مراجعه می‌شود به کار می‌رود.

SetValue() مقدار رشته‌ای که توسط کاربر وارد شده است راگرفته و آن را در قالبی صحیح به صفت نسبت می‌دهد. SetValue() همچنین عمل کنترل خطا را به منظور اطمینان از اینکه مقدار در محدوده مشخص شده است انجام می‌دهد. با این کار نحوه اجرای کنترل خطا با ویرایشگرهای موروثی صفت شما مشخص می‌شود. متد SetOrdValue() را برای حصول نمایش رشته‌ای نوع‌های مختلف تعریف می‌کند. به علاوه TPropertyEditor حاوی متدهای "set" متناظر برای تنظیم مقادیر در قالب‌های مرتبط است. نسل‌های TPropertyEditor این صفات را به ارث می‌برند. این متدها برای حصول و تنظیم مقادیر صفاتی که مرجع‌های ویرایشگر صفت هستند به کار می‌روند. در جدول ۲-۹ این متدها دیده می‌شوند. برای شرح ایجاد یک ویرایشگر صفت جدید مثال منظومه شمسی را به کار می‌گیریم. در این لحظه یک جزء

جدول ۲-۹ متدهای صفت Read/Write برای TPropertyEditor

Property Type	*"Get" Method*	*"Set" Method*
Floating point	`GetFloatValue()`	`SetFloatValue()`
Event	`GetMethodValue()`	`SetMethodValue()`
Ordinal	`GetOrdValue()`	`SetOrdValue()`
String	`GetStrValue()`	`SetStrValue()`
Variant	`GetVarValue()`	`SetVarValue()`, `SetVarValueAt()`

سازنده ساده، TPlanet را به منظور نمایش یک سیاره تعریف‌کرده‌ایم. TPlanet دارای صفت PlanetName است. حافظه داخلی مربوط به PlanetName نوع عددی صحیح داشته و وضعیت سیاره در منظومه شمسی را ذخیره می‌نماید. البته، این مقدار در Object Inspector با نام سیاره نمایش داده می‌شود.

می‌خواهیم که کاربران قادر به وارد کردن دو مقدار به منظومه نمایش سیاره باشند. کاربر بایستی قادر به تایپ کردن نام سیاره به صورت رشته‌ای باشد، مثلاً Venus ، VENUS ، یا VeNuS . همچنین کاربر بایستی قادر به تایپ محل سیاره در منظومه شمسی باشد. بنابراین برای سیاره زهره کاربر بایستی مقدار عددی 2 را وارد نماید.

جزء سازندهٔ TPlanet به صورت زیر است.

```
type
  TPlanetName = type Integer;

  TPlanet = class(TComponent)
  private
    FPlanetName: TPlanetName;
  published
    property PlanetName: TPlanetName read FPlanetName write FPlanetName;
  end;
```

همانگونه که می‌بینید چیز زیادی در این جزء سازنده وجود ندارد. این جزء سازنده فقط یک صفت دارد: PlanetName با نوع TPlanetName . در اینجا تعریف ویژه TPlanetName به گونه‌ای مورد استفاده قرار گرفته است که اطلاعات نوع زمان اجرای خود را دریافت کرده است و هنوز با آن به صورت یک نوع عددی رفتار می‌شود.

این وظیفه‌مندی از جزء سازندهٔ TPlanet نمی‌آید بلکه در عوض از ویرایشگر صفت مربوط به نوع صفت TPlanetName می‌آید. این ویرایشگر صفت در لیست ۴-۹ نشان داده شده است.

لیست ۴-۹ PlanetE.PAS، کد منبع برای TPlanetNameProperty

```
unit PlanetPE;

interface

uses
  Windows, SysUtils, DsgnIntF;

type
  TPlanetNameProperty = class(TIntegerProperty)
  public
    function GetValue: string; override;
    procedure SetValue(const Value: string); override;
  end;

implementation
```

لیست ۴-۹ ادامه

```
const
  { Declare a constant array containing planet names }
  PlanetNames: array[1..9] of String[7] =
   ('Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn',
    'Uranus', 'Neptune', 'Pluto');


function TPlanetNameProperty.GetValue: string;
begin
  Result := PlanetNames[GetOrdValue];
end;

procedure TPlanetNameProperty.SetValue(const Value: String);
var
  PName: string[7];
  i, ValErr: Integer;
begin
  PName := UpperCase(Value);
  i := 1;
  { Compare the Value with each of the planet names in the PlanetNames
    array. If a match is found, the variable i will be less than 10 }
  while (PName <> UpperCase(PlanetNames[i])) and (i < 10) do
    inc(i);
  { If i is less than 10, a valid planet name was entered. Set the value
    and exit this procedure. }
  if i < 10 then  // A valid planet name was entered.
  begin
    SetOrdValue(i);
    Exit;
  end
  { If i was greater than 10, the user might have typed in a planet number, or
    an invalid planet name. Use the Val function to test if the user typed in
    a number, if an ValErr is non-zero, an invalid name was entered,
    otherwise, test the range of the number entered for (0 < i < 10). }
  else begin
    Val(Value, i, ValErr);
    if ValErr <> 0 then
      raise Exception.Create(Format('Sorry, Never heard of the planet %s.',
        [Value]));
    if (i <= 0) or (i >= 10) then
       raise Exception.Create('Sorry, that planet is not in OUR solar
system.');
    SetOrdValue(i);
  end;
end;

end.
```

```
procedure RegisterPropertyEditor(PropertyType: PTypeInfo;
  ComponentClass: TClass; const PropertyName: string;
  EditorClass: TPropertyEditorClass);
```

اولین پارامتر، PropertyType اشاره‌گری به Runtime Type Information صفت در حال ویرایش است. این اطلاعات با استفاده از تابع ()TypeInfo حاصل می‌شوند. ComponentClass برای مشخص کردن اینکه ویرایشگر صفت در چه کلاسی به کار می‌رود مورد استفاده قرار می‌گیرد.

PropertyName نام صفت روی جزء سازنده را مشخص کرده و پارامتر EditorClass نوع ویرایشگر صفت را برای استفاده مشخص می‌کند. برای صفت Tplanet.PlanetName تابع به شکل زیر است.

```
RegisterPropertyEditor(TypeInfo(TPlanetName), TPlanet, 'PlanetName ,
  TPlanetNameProperty);
```

■ نکته

به منظور ارائه مثال، این صفت خاص برای استفاده با جزء سازندهٔ TPlanet و نام صفت PlanetName ثبت شده است، ممکن است خواسته باشید در ثبت ویرایشگرهای صفت خود محدودیت کمتری را اعمال کنید. با تنظیم پارامتر ComponentClass به nil و پارامتر PropertyName با ، ویرایشگر صفت شما برای هر صفت با نوع TPlanetName کار خواهد کرد.

می‌توانید ویرایشگر صفت را به همراه ثبت جزء سازنده در یونیت جزء سازنده به شکل لیست برنامه ۵-۹ به ثبت برسانید.

لیست ۵-۹ یونیت Planet.pas

```
unit Planet;

interface

uses
  Classes, SysUtils;

type
  TPlanetName = type Integer;

  TddgPlanet = class(TComponent)
  private
    FPlanetName: TPlanetName;
  published
    property PlanetName: TPlanetName read FPlanetName write FPlanetName;
  end;

implementation

end.
```

نکته

قرار دادن عمل ثبت ویرایشگر صفت در روال Register() یونیت جزء سازنده باعث می‌شود که تمامی کد ویرایشگر صفت هنگام قرارگیری جزء سازنده شما در بسته به همراه آن الحاق شود. برای اجزاء سازندهٔ پیچیده، ابزار زمان طراحی ممکن است فضای کد بیشتری نسبت به خود اجزاء اشغال نمایند.

ابتدا ویرایشگر صفت خود TPlanetNameProperty که از TInteegerProperty ارث می‌برد را ایجاد می‌کنیم. لازم است یونیت‌های DesignEditors و DssignIntf را در قسمت uses یونیت ضمیمه کنیم.

آرایه‌ای از ثابت‌های رشته‌ای را برای نمایش سیارات در منظومه شمسی با کمک وضعیت آنها نسبت به خورشید تعریف کرده‌ایم. این رشته‌ها برای نمایش شکل رشته‌ای سیاره در Object Inspector به کار خواهند رفت.

همانگونه که قبلاً گفتیم بایستی متدهای GetValue() و SetValue() را باطل کنیم. در متد GetValue() رشته را از آرایه PlanetNames ارجاع داده‌ایم که توسط مقدار صفت اندیس‌گذاری شده است. البته این مقدار بایستی در محدوده ۹-۱ باشد. کنترل این شرط با عدم اجازه ورود عددی خارج از محدوده در متد SetValue() به انجام می‌رسد. SetValue() یک رشته را همچنانکه از Object Inspector وارد می‌شود دریافت می‌کند. این رشته می‌تواند یک شماره سیاره و یا عددی مشخص کننده وضعیت سیاره باشد. اگر یک نام سیاره یا شماره سیاره معتبر وارد شود مقدار نسبت داده شده به صفت توسط متد SetOrdValue() مشخص می‌شود. اگر کاربر یک نام یا محل غیرمعتبر وارد کند که استثناء مناسبی را فراخوانی می‌کند.

نکات مربوط به تعریف یک ویرایشگر صفت ارائه شدند ولی هنوز بایستی قبل از شناخته به کارگیری و اتصال به یک صفت، آن را ثبت نماییم.

ثبت یک ویرایشگر صفت جدید

ثبت یک ویرایشگر صفت با استفاده از روالی انجام می‌شود که عمل آن با توجه به نام آن مشخص می‌شود: RegisterPropertyEditor() ، این متد به شکل زیر اعلان می‌شود: علیرغم اینکه اندازه کد برای اجزاء کوچکی نظیر این اهمیت چندانی ندارد این مطلب را به خاطر داشته باشید.

هر چیزی که در بخش interface یونیت جزء سازندهٔ شما لیست شده است (نظیر روال Register) هنگام عمل کمپایل به همراه جزء سازندهٔ شما برچسب‌گذاری خواهد شد. برای این منظور، ممکن است بخواهید ثبت ویرایشگر صفت خود را در یونیتی جدا انجام دهید. علاوه بر این، برخی از طراحان، هر دو بسته‌های زمان اجراء و زمان طراحی را برای ایجاد کردن انتخاب می‌نمایند در حالی که ویرایشگرهای صفت و دیگر ابزارهای زمان طراحی فقط در بستر زمان طراحی قرار می‌گیرند.

ویرایش یکپارچه صفت به کمک مکالمه

برخی اوقات لازم است که قابلیت ویرایشی بیشتری نسبت به ویرایش در "محل" Object Inspector فراهم نماییم. این زمانی است که استفاده از مکالمه به عنوان یک ویرایشگر صفت لازم می‌شود. به عنوان مثالی از این مسأله می‌توان صفت Font مربوط به اکثر اجزاء سازندهٔ دلفی را نام برد. اصولاً مارک کننده‌های دلفی می‌توانند کاربر را وارد کنند که نام قلم و اطلاعات دیگر مرتبط با آن را لیست نمایند. البته انتظار از کاربر مبنی بر درک این اطلاعات ممکن است غیرمنطقی باشد. فراهم کردن مکالمه‌ای برای کاربر به این منظور که بتواند این صفات مختلف را برای قلم تنظیم کرده و قبل از انتخاب مثالی از آن را ببیند آسان‌تر است.

برای شرح استفاده از یک مکامله به منظور ویرایش یک صفت قصد داریم که دامنه عملکرد TddgRunButton که در فصل ۸ ایجاد شد را توسعه دهیم. اکنون کاربر قادر خواهد بود که روی دکمه‌ای مربوط به صفت commondLine که یک مکالمه Open File را به منظور انتخاب فایلی برای نمایش توسط TddgRunButton به کار می‌گیرد کلیک نماید.

ویرایشگر صفت نمونه: توسعه TddgRunButton

جزء سازندهٔ TddgRunButton در لیست ۳-۸ در فصل ۸ نشان داده شد. این جزء سازنده را مجدداً در اینجا نشان نخواهیم داد، ولی نکاتی هستند که باید به آنها اشاره شود. صفت TddgRunButton.CommandLine از نوع TCommandLine است که به شکل زیر تعریف می‌شود.

```
TCommandLine = type string;
```

این اعلان خاصی است که اطلاعات نوع زمان اجراء را به این نوع خاص پیوند می‌زند. این کار شما را قادر می‌سازد که یک ویرایشگر صفت خاص برای نوع TCommandLine تعریف کنید. به علاوه از آنجا که با TCommandLine به عنوان رشته برخوردمی‌شود ویرایشگر صفت برای ویرایش صفات رشته هنوز در نوع TCommandLine نیز به خوبی به کار گرفته می‌شود.

همانگونه که ویرایشگر صفت مربوط به نوع TCommandLine را شرح دادیم، به خاطر داشته باشید که TddgRunButton دارای کنترل خطای لازم روی تخصیص صفات متدهای دسترسی صفات است. بنابراین تکرار این کنترل خطا در منطق ویرایشگر صفت لازم نیست. لیست ۶-۹ تعریف ویرایشگر صفت TCommandLineProperty را نشان می‌دهد.

بررسی TCommandLineProperty نشان می‌دهد که خود ویرایشگر صفت بسیار ساده است. ابتدا توجه داشته باشید که شیء موردنظر از TStringProperty ارث می‌برد به گونه‌ای که قابلیت‌های ویرایش پشتیبانی شده‌اند. بنابراین در Object Inspector به کارگیری مکالمه لازم نیست. کاربر می‌تواند خط فرمان را مستقیماً تایپ کنید. همچنین متدهای ()SetValue و ()GetValue را باطل سازی نکردیم زیرا TStringProperty قبلاً این کار را به درستی انجام داده است. البته برای دانستن آنکه صفت قابل ویرایش

لیست ۶-۹ RunBtnPE.pas، یونیت حاوی TcommandLineProperty

```
unit runbtnpe;

interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, Buttons, DsgnIntF, TypInfo;

type

  { Descend from the TStringProperty class so that this editor
    inherits the string property editing capabilities }
  TCommandLineProperty = class(TStringProperty)
    function GetAttributes: TPropertyAttributes; override;
    procedure Edit; override;
  end;

implementation

function TCommandLineProperty.GetAttributes: TPropertyAttributes;
begin
  Result := [paDialog]; // Display a dialog in the Edit method
end;

procedure TCommandLineProperty.Edit;
{ The Edit method displays a TOpenDialog from which the user obtains
  an executable file name that gets assigned to the property }
var
  OpenDialog: TOpenDialog;
begin
  { Create the TOpenDialog }
  OpenDialog := TOpenDialog.Create(Application);
  try
    { Show only executable files }
    OpenDialog.Filter := 'Executable Files|*.EXE';
    { If the user selects a file, then assign it to the property.  }
    if OpenDialog.Execute then
      SetStrValue(OpenDialog.FileName);
  finally
    OpenDialog.Free // Free the TOpenDialog instance.
  end;
end;


end.
```

شدن توسط مکالمه است لازم بود که متد ()GetAttributes باطل شده باشد. ()GetAttributes شایسته بحث بیشتری است.

تعیین ویژگیهای ویرایشگر صفت

هر ویرایشگر صفت بایستی نحوه ویرایش یک صفت و ویژگیهای خاصی که در هنگام ویرایش صفت بایستی به کار روند را برای Object Inspector مشخص نماید. اکثر اوقات، ویژگیهای موروثی از یک ویرایشگر صفت تسلسلی کفایت می‌کند. در وضعیت‌های حقیقی بایستی متد ()GetAttributess از TPropertyEditor که مجموعه‌ای از پرچم‌های ویژگی صفت را برمی‌گرداند را باطل نماییم. پرچم‌های ویژگی صفت (پرچم‌های TPropertyAttribute) ویژگیهای خاص ویرایش صفت را مشخص می‌کنند. پرچم‌های مختلف TPropertyAttribute در جدول ۳-۹ نشان داده شده‌اند.

جدول ۳-۹ پرچم‌های TPropertyAttribute

صفت	**نحوه کار ویرایشگر با** Object Inspector
paValueList	لیستی تعریف شده از مقادیر مربوط به صفت را برمی‌گرداند. متد ()GetValue لیست را مقداردهی می‌کند. یک دکمه با علامت پیکان رو به پایین در سمت راست مقدار صفت ظاهر می‌شود. عمل موردنظر در صفات تعریف شده نظیر TForm.BorderStyle به کار گرفته می‌شود.
paSubProperties	زیر صفات با تورفتگی مشخص بعد از صفت جاری نمایش داده می‌شوند. paValueList نیز بایستی تنظیم شده باشد. این ویژگی برای تنظیم صفات و صفات کلاس نظیر TOpenDialog.Options و TForm.Font به کار گرفته می‌شود.
paDialog	یک دکمه در سمت راست صفت در Object Inspector ظاهر می‌شود که پس از کلیک کردن روی آن متد Edit ویرایشگر صفت مکالمه‌ای را به کار می‌گیرد. این ویژگی در صفاتی نظیر TForm.Font به کار گرفته می‌شود.
paMultiSelect	صفات هنگامی نمایش داده می‌شوند که بیشتر از یک جزء سازنده روی FormDesigner انتخاب شده باشد، با این کار به کاربر اجازه داده می‌شود که مقادیر صفت چندین جزء سازنده را به صورت یکجا تغییر دهند. برخی صفات برای این قابلیت مناسب نیستند، نظیر صفت Name .
paAutoUpdate	()SetValue روی هر تغییر اعمال شده در صفت فراخوانی می‌شود. اگر این پرچم تنظیم نشده باشد، ()SetValue هنگامی فراخوانی می‌شود که پروسه کاربر، صفت را وارد Object Inspector کند و یا از آن خارج نماید. این ویژگی در صفاتی نظیر TForm.Caption به کار گرفته می‌شود. به Object Inspector می‌گوید که مقادیر لازم نیست که تحویل داده شوند و نام بایستی پهنای کامل Inspector را ارائه دهد.
paFullWidthname	به Object Insperctor می‌گوید که مقادیر لازم نیست که تحویل داده شوند و نام بایستی پهنای کامل Inspector را ارائه دهند.

paSortList	Object Inspector لیست ارجاع شده توسط ()GetValues را مرتب می‌کند.
paReadOnly	مقدار صفت غیرقابل تغییر است.
paRevertable	صفت را می‌توان به مقدار اصلی آن برگرداند. برخی صفات نظیر صفات تودرتو را نباید برگرداند. TFont مثالی از این صفات است.

تذکر

بایستی دقت بر DesignEditors.pas داشته و بررسی کنید که کدام یک از پرچم‌های TPropertyAttribute برای ویرایشگرهای مختلف تنظیم شده‌اند.

تنظیم ویژگی paDialog برای TCommandLineProperty

از آنجا که TCommandLineProperty مرتبط با نمایش مکالمه است بایستی به Object Inspector بگویید که با تنظیم صفت paDialog در متد ()TCmmandLineProperty.GetAttributes از این قابلیت استفاده نماید. با این کار دکمه‌ای در سمت راست مقدار صفت CommandLine در Object Inspector ظاهر می‌شود. هنگامی که کاربر روی این دکمه کلیک می‌کند متد ()TcommandLinProperty.Edit فراخوانی خواهد شد.

ثبت TCmmandLineProperty

آخرین گام مورد نیاز برای پیاده‌سازی ویرایشگر صفت TCmmandLineProperty ثبت آن با استفاده از روال ()RegisterPropertyEditor است که قبلاً شرح دادیم. این روال به روال ()Register در DDGReg.pas در بسته DDGDsgn اضافه شده است:

```
RegisterComponents('DDG', [TddgRunButton]);
  RegisterPropertyEditor(TypeInfo(TCommandLine), TddgRunButton,
       , TCommandLineProperty);
```

توجه داشته باشید که یونیت‌های DsnIntf و RunBnPE بایستی در قسمت uses برنامه اضافه شده باشند.

ویرایشگرهای اجزاء سازنده

ویرایشگرهای اجزاء سازنده با دادن امکان افزودن آیتم‌هایی به منوی محلی متناظر با یک جزء سازنده خاص و تغییر عمل پیش‌فرض هنگام دوبارکلیک کردن روی جزء سازنده در Form Designer رفتار زمان طراحی اجزاء سازنده شما را تغییر می‌دهند. در صورتی که از ویرایشگرهای فیلد که در اجزاء سازنده TTable ، TQuery و TStoredProc آماده شده‌اند استفاده کرده باشید با ویرایشگرهای اجزاء سازنده آشنائی دارید.

TComponentEditor

ممکن است تابحال این مطلب را لمس نکرده باشید، ولی برای هر جزء سازنده که در Form Designey انتخاب شده است ویرایشگر متفاوتی ایجاد می‌شود. نوع ویرایشگر ایجاد شده بستگی به نوع جزء سازنده دارد، علیرغم اینکه تمامی ویرایشگرهای اجزاء سازنده از TComponentEditor ارث می‌برند. این کلاس به شکل زیر در DesignEditors به شکل زیر تعریف شده است.

```
  TComponentEditor = class(TBaseComponentEditor, IComponentEditor)
  private
    FComponent: TComponent;
    FDesigner: IDesigner;
  public
    constructor Create(AComponent: TComponent; ADesigner: IDesigner); override;
    procedure Edit; virtual;
    procedure ExecuteVerb(Index: Integer); virtual;
    function GetComponent: TComponent;
    function GetDesigner: IDesigner;
    function GetVerb(Index: Integer): string; virtual;
    function GetVerbCount: Integer; virtual;
    function IsInInlined: Boolean;
procedure Copy; virtual;
    procedure PrepareItem(Index: Integer; const AItem: IMenuItem); virtual;
    property Component: TComponent read FComponent;
    property Designer: IDesigner read GetDesigner;
  end;
```

صفات

صفت Component از TComponentEditor نمونه‌ای از جزء سازنده‌ای است که شما در پروسه ویرایش آن هستید. از آنجا که این صفت از نوع کلی TComponentEditor است بایستی به منظور دسترسی به فیلدهای فراهم شده توسط کلاس‌های وراثت، صفت را مشخص نمایید.

صفت Designer وهله‌ای از IDesigner است که در حال حاضر برنامه را در زمان طراحی میزبانی می‌نماید. تعریف کامل این کلاس را در یونیت DesignEditors.pas خواهید یافت.

متدها

متد ()Edit هنگامی فراخوانی می‌شود که کاربر در زمان طراحی روی جزء سازنده دوبار کلیک کند. اغلب این متد گونه‌ای از محاوره طراحی را به کار خواهد گرفت. رفتار پیش‌فرض برای این متد فراخوانی ExecuteVerb(**0**) است اگر ()GetVerbCount مقدار 1 یا بزرگتر از آن را برگرداند. بایستی در صورتی که جزء سازنده را از قید ()Designer.Modified تغییر داده باشید آن را فراخوانی کنید.

واژه *verb* همانگونه که در متدها به کار می‌رود روی اعمالی که یک شیء می‌تواند انجام دهد نیز به کار گرفته می‌شود. دلفی در آغاز کار اطلاعاتی در مورد اشیاء جدید ندارد و بایستی هنگام افزودن آنها از این مطلب مطلع شود. با در نظر گرفتن این موضوع طراحی به گونه‌ای است که بتوان متدهای متعددی را

برای تشخیص عمل شیء به کار برد. متدهای GetVerbCount ، GetVerb و ExecteVerb متدهای عمومی هستند که برای گونه‌های متعددی از اجزاء سازنده مفهوم داشته و فراخوانی‌هایی هستند که برای معرفی اجزاء خود به دلفی از آنها استفاده می‌کنید.

متد GetVerbCount() به منظور بازیابی تعدادی از آیتم‌ها که بایستی به منوی محلی اضافه شوند فراخوانی می‌شود.

GetVerb یک عدد، Index را پذیرفته و رشته‌ای را برمی‌گرداند که شامل رشته‌ای است که بایستی در منوی محلی و در وضعیت مرتبط با Index ظاهر شود.

هنگامی که یک آیتم از منوی محلی انتخاب شود متد ExecuteVerb() فراخوانی می‌گردد. این متد یک **اندیس** مبتنی بر صفر از آیتم انتخاب شده در منوی محلی را در پارامتر Index دریافت می‌کند. شما بایستی با انجام دادن عمل لازم بسته به کلمه انتخاب شده در منوی محلی پاسخ مناسبی ارائه کنید.

متد Paste() هنگامی فراخوانی می‌شود که مؤلفه در حافظه Clipboard قرار داده شود. دلفی تصویر فیلدهای اجزاء سازنده را در Clipboard قرار می‌دهد ولی شما می‌توانید از این متد برای قالب‌های داده‌ای دیگر نیز استفاده کنید.

TDefaultEditor

اگر یک ویرایشگر خاص ثبت نشده باشد، جزء سازند از ویرایشگر پیش‌فرض، TDefaultEditor استفاده خواهد نمود. TDefaultEditor رفتار متد Edit() را باطل‌سازی می‌کند به گونه‌ای که این متد صفات جزء سازنده را جستجو نموده و رویدادهای onCrate ، onChanged یا onClick را ایجاد (یا ناوبری) می‌کند (اولین رویدادید که یافت شود). اگر هیچ کدام از این رویدادها برای اجزاء سازندهٔ موردنظر وجود نداشته باشند اولین رویداد تعریف شده انتخاب خواهد شد.

یک جزء سازندهٔ ساده

جزء سازندهٔ ساده زیر را در نظر بگیرید:

```
type
  TComponentEditorSample = class(TComponent)
  protected
    procedure SayHello; virtual;
    procedure SayGoodbye; virtual;
  end;

procedure TComponentEditorSample.SayHello;
begin
  MessageDlg('Hello, there!', mtInformation, [mbOk], 0);
end;

procedure TComponentEditorSample.SayGoodbye;
```

```
begin
  MessageDlg('See ya!', mtInformation, [mbOk], 0);
end;
```

همانگونه که می‌توانید ببینید این بخش کوچک، کار زیادی انجام نمی‌دهد: جزء سازندهٔ NonVisual است که مستقیماً از TComponent ارث برده و دو متد دارد، ()SayHello و ()SayGoodbye که مکالمات پیغام‌گونه را نمایش می‌دهند.

یک ویرایشگر جزء سازندهٔ ساده

در اینجا مؤلفه‌ای ایجاد خواهیم کرد که متدهای خود را در زمان طراحی اجراء می‌نماید. متدهایی از TComponentEditor که بایستی باطل‌سازی شوند حداقل عبارتند از: ()ExecuteVerb ، ()GetVerb و ()GetVerbCount. کد مربوط به این ویرایشگر به صورت زیر است.

```
type
  TSampleEditor = class(TComponentEditor)
  private
    procedure ExecuteVerb(Index: Integer); override;
    function GetVerb(Index: Integer): string; override;
    function GetVerbCount: Integer; override;
  end;

procedure TSampleEditor.ExecuteVerb(Index: Integer);
begin
  case Index of
    0: TComponentEditorSample(Component).SayHello;    // call function
    1: TComponentEditorSample(Component).SayGoodbye;  // call function
  end;
end;

function TSampleEditor.GetVerb(Index: Integer): string;
begin
  case Index of
    0: Result := 'Hello';       // return hello string
    1: Result := 'Goodbye';     // return goodbye string
  end;
end;

function TSampleEditor.GetVerbCount: Integer;
begin
  Result := 2;      // two possible verbs
end;
```

متد ()GetVerbCount مقدار 2 را برمی‌گرداند که مشخص می‌کند دو کلمه متفاوت توسط ویرایشگر مؤلفه برای اجراء آماده شده‌اند. ()GetVerb برای هر یک از این کلمات رشته‌ای برمی‌گرداند تا روی

منوی محلی ظاهر شوند. متد ExecuteVerb() بسته به اندیس کلمه‌ای که به عنوان پارامتر دریافت می‌کند متد مناسبی را از درون مؤلفه فراخوانی می‌نماید.

ثبت یک ویرایشگر برای اجزاء سازنده

همانند اجزاء سازنده و ویرایشگرهای صفت، ویرایشگرهای اجزاء سازنده نیز بایستی به همراه IDE در درون متد Register() یونیت ثبت شود. برای ثبت یک ویرایشگر روال RegisterComponentEditor() که به صورت زیر تعریف شده است را فراخوانی کنید.

```
procedure RegisterComponentEditor(ComponentClass: TComponentClass;
  ComponentEditor: TComponentEditorClass);
```

اولین پارامتر این تابع نوع جزء سازنده‌ای است که یک ویرایشگر را برای آن ثبت می‌کنید و دومین پارامتر خود ویرایشگر است.

در لیست ۷-۹ یونیت CompEdit.pas که جزء سازنده، ویرایشگر جزء سازنده و فراخوانی‌های ثبت را دربردارد نشان داده شده است.

لیست ۷-۹ CompEdit.pas، یک ویرایشگر جزء سازنده را شرح می‌دهد

```
unit CompEdit;

interface

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

type
  TComponentEditorSample = class(TComponent)
  protected
    procedure SayHello; virtual;
    procedure SayGoodbye; virtual;
  end;

  TSampleEditor = class(TComponentEditor)
  private
    procedure ExecuteVerb(Index: Integer); override;
    function GetVerb(Index: Integer): string; override;
    function GetVerbCount: Integer; override;
  end;

implementation

{ TComponentEditorSample }
```

```
procedure TComponentEditorSample.SayHello;
begin
  MessageDlg('Hello, there!', mtInformation, [mbOk], 0);
end;
procedure TComponentEditorSample.SayGoodbye;
begin
  MessageDlg('See ya!', mtInformation, [mbOk],  0);
end;

{ TSampleEditor }

const
  vHello = 'Hello';
  vGoodbye = 'Goodbye';

procedure TSampleEditor.ExecuteVerb(Index: Integer);
begin
  case Index of
    0: TComponentEditorSample(Component).SayHello;    // call function
    1: TComponentEditorSample(Component).SayGoodbye;  // call function
  end;
end;

function TSampleEditor.GetVerb(Index: Integer): string;
begin
  case Index of
    0: Result := vHello;       // return hello string
    1: Result := vGoodbye;      // return goodbye string
  end;
end;

function TSampleEditor.GetVerbCount: Integer;
begin
  Result := 2;      // two possible verbs
end;

end.
```

جریان دادن داده‌های جزء سازندهٔ منتشر نشده

در فصل ۸ دیدیم که IDE در Delphi به صورت خودکار نحوه جریان دادن صفات منتشر شده یک جزء سازنده از / به یک فایل DFM را می‌داند. هنگامی که داده‌های منتشر نشده‌ای دارید که می‌خواهید با نگهداری آنها در فاصل DFM ماندگار بمانند چه اتفاقی رخ می‌دهد؟ خوشبختانه اجزاء سازندهٔ Delphi مکانیزمی برای نوشتن و خواندن داده‌های تعریف شده توسط برنامه از / به فایل DFM ارائه می‌کنند.

تعریف صفات

اولین گام در تعریف "صفات" منتشر نشده مقیم، باطل‌سازی متد ()DefineProperties است این متد از

TPresistent ارث برده و به شکل زیر تعریف می‌شود.

```
procedure DefineProperties(Filer: TFiler); virtual;
```

این متد به صورت پیش‌فرض اعمال خواندن و نوشتن صفات منتشر شده از / به فایل DFM را انجام می‌دهد می‌توانید این متد را باطل‌سازی کنید و سپس از فراخوانی inherited می‌توانید متد TFilter، DefineProperty() یا DefineBinaryProperty() را یکبار برای هر بخش از داده‌ای که می‌خواهید بخشی از فایل DFM باشد فراخوانی کنید. این متدها به صورت زیر تعریف می‌شوند.

```
procedure DefineProperty(const Name: string; ReadData: TReaderProc;
    WriteData: TWriterProc; HasData: Boolean); virtual;

procedure DefineBinaryProperty(const Name: string; ReadData,
    WriteData: TStreamProc; HasData: Boolean); virtual;
```

DefineProperty() برای ایجاد نوع‌های داده‌ای استاندارد نظیر string، integer، Boolean، char، float و انواع تعریف شده به کار می‌رود. DefineBinaryProperty() برای فراهم کردن دسترسی به انواع داده دودویی نظیر صوت و تصویر به کار می‌رود.

برای هر دو این توابع پارامتر Name نام صفتی که بایستی در فیلد DFM نوشته شود را شناسایی می‌کند. این نام لازم نیست که مشابه با نام داخلی فیلد داده‌ای باشد که به آن دسترسی پیدا می‌کنید. پارامترهای ReadData و WriteData از نظر نوع در DefineProperty() و DefineBinaryProperty() تفاوت دارند ولی هدف مشابهی را دنبال می‌کنند: این متدها به منظور نوشتن یا خواندن داده‌ها به / از فایل DFM فراخوانی می‌شوند. پارامتر HasData مشخص می‌کند که آیا "صفت" داده‌ای را که برای ذخیره‌سازی مورد نیاز است در بردارد یا خیر. پارامترهای ReadData و WriteData از DefineProperty() به ترتیب از نوع TReaderProc و TWriterProc هستند. این نوع‌ها به صورت زیر تعریف می‌شوند.

```
type
  TReaderProc = procedure(Reader: TReader) of object;
  TWriterProc = procedure(Writer: TWriter) of object;
```

TReader و TWriter نسل‌های ویژه TFilter هستند که متدهایی اضافی برای خواندن و نوشتن نوع‌های بومی دارند. متدهای این نوع‌ها اتصالی میان داده مؤلفه منتشر شده و فایل DFM برقرار می‌کنند.

پارامترهای ReadData و WriteData از DefineBinaryProperty() از نوع TStreamProc هستند که به شکل زیر تعریف می‌شود:

```
type
  TStreamProc = procedure(Stream: TStream) of object;
```

از آنجا که متدهای نوع TStreamProc فقط TStream را به عنوان پارامتر دریافت می‌کنند این کار به شما امکان می‌دهدکه داده‌ها را به سادگی خوانده و یا بنویسید. همانند دیگر انواع متد که قبلاً شرح داده شدند متدهای دارای این نوع رابطی میان داده‌های غیراستاندارد و فایل DFM برقرار می‌کنند.

مثالی از DefineProperty()

به منظور جمع‌آوری اطلاعات تکنیکی، لیست ۸-۹ یونیت DefProp.pas را نشان می‌دهد. این یونیت با فراهم کردن حافظه‌ای برای دو فیلد داده‌ای اختصاصی: یک رشته و یک عدد صحیح کاربرد DefineProperty() را شرح می‌دهد.

لیست ۸-۹ DefProp.pas با استفاده از تابع DefineProperty() شرح داده شده است

```
unit DefProp;

interface

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

type
  TDefinePropTest = class(TComponent)
  private
    FString: String;
    FInteger: Integer;
    procedure ReadStrData(Reader: TReader);
    procedure WriteStrData(Writer: TWriter);
    procedure ReadIntData(Reader: TReader);
    procedure WriteIntData(Writer: TWriter);
  protected
    procedure DefineProperties(Filer: TFiler); override;
  public
    constructor Create(AOwner: TComponent); override;
  end;

implementation

constructor TDefinePropTest.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  { Put data in private fields }
  FString := 'The following number is the answer...';
  FInteger := 42;
end;
```

لیست ۸-۹ ادامه

```
procedure TDefinePropTest.DefineProperties(Filer: TFiler);
begin
  inherited DefineProperties(Filer);
  { Define new properties and reader/writer methods }
  Filer.DefineProperty('StringProp', ReadStrData, WriteStrData,
    FString <> '');
  Filer.DefineProperty('IntProp', ReadIntData, WriteIntData, True);
end;

procedure TDefinePropTest.ReadStrData(Reader: TReader);
begin
  FString := Reader.ReadString;
end;

procedure TDefinePropTest.WriteStrData(Writer: TWriter);
begin
  Writer.WriteString(FString);
end;

procedure TDefinePropTest.ReadIntData(Reader: TReader);
begin
  FInteger := Reader.ReadInteger;
end;

procedure TDefinePropTest.WriteIntData(Writer: TWriter);
begin
  Writer.WriteInteger(FInteger);
end;

end.
```

یادداشت

همواره از متدهای ReadString() و WriteString() مربوط به TReader و TWriter برای خواندن و نوشتن داده‌های رشته‌ای استفاده کنید. هرگز از متدهای ReadStr() و WriteStr() استفاده نکنید زیرا فایل DFM شما را خراب می‌کنند. ▲

TddgWaveFile: مثالی از DefineBinaryProperty()

قبلاً گفتیم که زمان مناسب برای استفاده از DefineBinaryProperty() زمانی است که نیاز به ذخیره‌سازی اطلاعات گرافیکی یا صوتی دارید. در واقع، VCL از این تکنیک برای ذخیره‌سازی تصاویر وابسته به اجزاء سازنده -به عنوان مثال Glyph مربوط به یک TBinBtn و یا Icon مربوط به یک TFrom - استفاده می‌کند. در این بخش نحوه استفاده از این تکنیک‌ها را هنگام ذخیره‌سازی صوت مربوط به جزء سازندهٔ TddgWaveFile خواهید آموخت.

نکته

TddgWaveFile جزء سازنده‌ای است که جنبه‌های مختلف را دربردارد، با استفاده از ویرایشگرهای اجزاء سازنده می‌توانید امکان پخش صوت در زمان طراحی را داشته باشید. شما قادر خواهید بود که در آینده‌ای نزدیک کدهای موردنظر خود را بنویسید ولی اکنون روی مکانیزم ذخیره‌سازی صفات دودویی تمرکز می‌کنیم.

متد ()DefineProperty برای TddgWaveFile به شکل زیر است.

```
procedure TddgWaveFile.DefineProperties(Filer: TFiler);
{ Defines binary property called "Data" for FData field. }
{ This allows FData to be read from and written to DFM file. }

  function DoWrite: Boolean;
  begin
    if Filer.Ancestor <> nil then
      Result := not (Filer.Ancestor is TddgWaveFile) or
        not Equal(TddgWaveFile(Filer.Ancestor))
    else
      Result := not Empty;
  end;

begin
  inherited DefineProperties(Filer);
  Filer.DefineBinaryProperty('Data', ReadData, WriteData, DoWrite);
end;
```

این متد یک صفت دودویی به نام Data را تعریف می‌کند که با استفاده از متدهای اجزاء سازندهٔ ()ReadData و ()WriteData خوانده و نوشته می‌شود. علاوه بر این، داده فقط هنگامی نوشته می‌شود که مقدار ارجاع شده ()DoWrite برابر True باشد.

متدهای ()ReadData و ()WriteData به شکل زیر تعریف می‌شوند.

```
procedure TddgWaveFile.ReadData(Stream: TStream);
{ Reads WAV data from DFM stream. }
begin
  LoadFromStream(Stream);
end;

procedure TddgWaveFile.WriteData(Stream: TStream);
{ Writes WAV data to DFM stream }

begin
  SaveToStream(Stream);
end;
```

همانگونه که می‌بینید این متدها آنچنان پیچیده نیستند فقط متدهای LeadFromStream() و SaveTostream() را فراخوانی می‌کنند که این متدها توسط جزء سازندهٔ TddgWaveFile تعریف شده‌اند. متد LeadFromStream() به شکل زیر است.

```
procedure TddgWaveFile.LoadFromStream(S: TStream);
{ Loads WAV data from stream S.  This procedure will free }
{ any memory previously allocated for FData. }
begin
  if not Empty then
    FreeMem(FData, FDataSize);
  FDataSize := 0;
  FData := AllocMem(S.Size);
  FDataSize := S.Size;
  S.Read(FData^, FDataSize);
end;
```

این متد ابتدا کنترل می‌کند که حافظه قبلاً با تست مقدار فیلد FDataSize تخصیص مقدار داده شده باشد. اگر این مقدار بزرگتر از صفر باشد حافظه‌ای که توسط فیلد FData به آن اشاره می‌شود آزاد شده است. در این لحظه یک بلوک جدید از حافظه برای FData تخصیص داده شده و FDataSize با اندازه جریان داده‌ای که در حال وارد شدن است تنظیم می‌شود سپس محتویات متد SaveToStream() ساده‌تر از این بوده و به صورت زیر تعریف می‌شود.

```
procedure TddgWaveFile.SaveToStream(S: TStream);
{ Saves WAV data to stream S. }
begin
  if FDataSize > 0 then
    S.Write(FData^, FDataSize);
end;
```

این متد داده‌ای که توسط اشاره‌گر FData به آن اشاره می‌شود را در TStreamS می‌نویسد. تابع محلی DoWrite داخل متد DefineProperties() مشخص می‌کند که آیا صفت Data نیاز به جریان یافتن دارد یا خیر. البته، اگر FData تهی باشد نیازی به جریان دادن داده نیست. به علاوه، بایستی اندازه‌گیری‌هایی اضافی را انجام دهید تا مطمئن شوید که اجزاء سازندهٔ شما با توارثی که از فرم گرفته است به درستی کار می‌کند. بایستی مشخص کنیدکه صفت Ancestor مربوط به Filer، غیر از n است. اگر چنین بود و به **پدری** از جزء سازندهٔ جاری اشاره می‌کرد بایستی مشخص کنیدکه داده‌ای که قصد نوشتن آن را دارید با **پدر** تفاوت دارد. اگر این تست‌های اضافی را اجراء نکنید یک کپی از داده (در این مثال فایل صوتی) در هر یک از فرم‌های اولاد نوشته شده و به فایل صوتی تغییر پیدا می‌کند که در فرم‌های اولاد کپی نخواهد شد.

لیست ۹-۹ برنامه Wavez.pas را نشان می‌دهد که کد منبع کامل جزء سازنده را دربردارد.

لیست ۹-۹ یونیت WaveZ_pas

```
unit Wavez;

interface

uses
  SysUtils, Classes;

type
  { Special string "descendant" used to make a property editor. }
  TWaveFileString = type string;

  EWaveError = class(Exception);

  TWavePause = (wpAsync, wpsSync);
  TWaveLoop = (wlNoLoop, wlLoop);

  TddgWaveFile = class(TComponent)
  private
    FData: Pointer;
    FDataSize: Integer;
    FWaveName: TWaveFileString;
    FWavePause: TWavePause;
    FWaveLoop: TWaveLoop;
    FOnPlay: TNotifyEvent;
    FOnStop: TNotifyEvent;
    procedure SetWaveName(const Value: TWaveFileString);
    procedure WriteData(Stream: TStream);
    procedure ReadData(Stream: TStream);
  protected
    procedure DefineProperties(Filer: TFiler); override;
  public
    destructor Destroy; override;
    function Empty: Boolean;
    function Equal(Wav: TddgWaveFile): Boolean;
    procedure LoadFromFile(const FileName: String);
    procedure LoadFromStream(S:  TStream);
    procedure Play;
    procedure SaveToFile(const FileName: String);
    procedure SaveToStream(S: TStream);
    procedure Stop;
  published
    property WaveLoop: TWaveLoop read FWaveLoop write FWaveLoop;
```

لیست ۹-۹ ادامه

```
    property WaveName: TWaveFileString read FWaveName write SetWaveName;
    property WavePause: TWavePause read FWavePause write FWavePause;
    property OnPlay: TNotifyEvent read FOnPlay write FOnPlay;
    property OnStop: TNotifyEvent read FOnStop write FOnStop;
  end;

implementation

uses MMSystem, Windows;

{ TddgWaveFile }

destructor TddgWaveFile.Destroy;
{ Ensures that any allocated memory is freed }
begin
  if not Empty then
    FreeMem(FData, FDataSize);
  inherited Destroy;
end;

function StreamsEqual(S1, S2: TMemoryStream): Boolean;
begin
  Result := (S1.Size = S2.Size) and CompareMem(S1.Memory, S2.Memory, S1.Size);
end;

procedure TddgWaveFile.DefineProperties(Filer: TFiler);
{ Defines binary property called "Data" for FData field. }
{ This allows FData to be read from and written to DFM file. }

  function DoWrite: Boolean;
  begin
    if Filer.Ancestor <> nil then
      Result := not (Filer.Ancestor is TddgWaveFile) or
        not Equal(TddgWaveFile(Filer.Ancestor))
    else
      Result := not Empty;
  end;

begin
  inherited DefineProperties(Filer);
  Filer.DefineBinaryProperty('Data', ReadData, WriteData, DoWrite);
end;

function TddgWaveFile.Empty: Boolean;
```

لیست ۹-۹ ادامه

```
begin
  Result := FDataSize = 0;
end;

function TddgWaveFile.Equal(Wav: TddgWaveFile): Boolean;
var
  MyImage, WavImage: TMemoryStream;
begin
  Result := (Wav <> nil) and (ClassType = Wav.ClassType);
  if Empty or Wav.Empty then
  begin
    Result := Empty and Wav.Empty;
    Exit;
  end;
  if Result then
  begin
    MyImage := TMemoryStream.Create;
    try
      SaveToStream(MyImage);
      WavImage := TMemoryStream.Create;
      try
        Wav.SaveToStream(WavImage);
        Result := StreamsEqual(MyImage, WavImage);
      finally
        WavImage.Free;
      end;
    finally
      MyImage.Free;
    end;
  end;
end;

procedure TddgWaveFile.LoadFromFile(const FileName: String);
{ Loads WAV data from FileName. Note that this procedure does }
{ not set the WaveName property. }
var
  F: TFileStream;
begin
  F := TFileStream.Create(FileName, fmOpenRead);
  try
    LoadFromStream(F);
  finally
    F.Free;
  end;
end;
```

لیست ۹-۹ ادامه

```
procedure TddgWaveFile.LoadFromStream(S: TStream);
{ Loads WAV data from stream S.  This procedure will free }
{ any memory previously allocated for FData. }
begin
  if not Empty then
    FreeMem(FData, FDataSize);
  FDataSize := 0;
  FData :=  AllocMem(S.Size);
  FDataSize := S.Size;
  S.Read(FData^, FDataSize);
end;

procedure TddgWaveFile.Play;
{ Plays the WAV sound in FData using the parameters found in }
{ FWaveLoop and FWavePause. }
const
  LoopArray: array[TWaveLoop] of DWORD = (0, SND_LOOP);
  PauseArray: array[TWavePause] of DWORD = (SND_ASYNC, SND_SYNC);
begin
  { Make sure component contains data }
  if Empty then
    raise EWaveError.Create('No wave data');
  if Assigned(FOnPlay) then FOnPlay(Self);    // fire event
  { attempt to play wave sound }
  if not PlaySound(FData, 0, SND_MEMORY or PauseArray[FWavePause] or
                   LoopArray[FWaveLoop]) then
    raise EWaveError.Create('Error playing sound');
end;

procedure TddgWaveFile.ReadData(Stream: TStream);
{ Reads WAV data from DFM stream. }
begin
  LoadFromStream(Stream);
end;

procedure TddgWaveFile.SaveToFile(const FileName: String);
{ Saves WAV data to file FileName. }
var
  F: TFileStream;
begin
  F := TFileStream.Create(FileName,  fmCreate);
  try
    SaveToStream(F);
  finally
```

لیست ۹-۹ ادامه

```
    F.Free;
  end;
end;

procedure TddgWaveFile.SaveToStream(S: TStream);
{ Saves WAV data to stream S. }
begin
  if not Empty then
    S.Write(FData^, FDataSize);
end;

procedure TddgWaveFile.SetWaveName(const Value: TWaveFileString);
{ Write method for WaveName property. This method is in charge of }
{ setting WaveName property and loading WAV data from file Value. }
begin
  if Value <> '' then begin
    FWaveName := ExtractFileName(Value);
    { don't load from file when loading from DFM stream }
    { because DFM stream will already contain data. }
    if (not (csLoading in ComponentState)) and FileExists(Value) then
      LoadFromFile(Value);
  end
  else begin
    { if Value is an empty string, that is the signal to free }
    { memory allocated for WAV data. }
    FWaveName := '';
    if not Empty then
      FreeMem(FData, FDataSize);
    FDataSize := 0;
  end;
end;

procedure TddgWaveFile.Stop;
{ Stops currently playing WAV sound }
begin
  if Assigned(FOnStop) then FOnStop(Self);  // fire event
  PlaySound(Nil, 0, SND_PURGE);
end;

procedure TddgWaveFile.WriteData(Stream: TStream);
{ Writes WAV data to DFM stream }
begin
  SaveToStream(Stream);
end;

end.
```

طبقه‌بندی‌های صفات

طبقه‌بندی‌های صفات امکانی برای صفات VCL فراهم می‌کند که به گونه‌ای مشخص شوند که متعلق به طبقات خاص باشند و همچنین برای Object Inspector امکانی فراهم می‌کند که با این طبقات ذخیره‌سازی شود. با استفاده از توابع ()RegisterPropertyInCategory و ()RegisterPropertiesInCategory که در یونیت DesignIntf اعلان شده‌اند می‌توانید صفات را به گونه‌ای ثبت کنید که متعلق به طبقه خاصی باشند.

()RegisterPropertyInCategory چهار نگارش مختلف دارد که نیازهای شما را برآورده می‌کنند. تمامی نگارش‌های این تابع TPropertyCategoryClass را به عنوان اولین پارامتر دریافت می‌کنند. از این پس هر یک از این نگارش‌ها ترکیب متفاوتی از نام صفت، نوع صفت، و کلاس جزء سازنده را در برمی‌گیرند تا امکان انتخاب بهترین متد برای ثبت صفات را به شما بدهند. نگارش‌های مختلف ()RegisterPropertyInCategory در اینجا نشان داده شده‌اند.

```
function RegisterPropertyInCategory(ACategoryClass: TPropertyCategoryClass;
  const APropertyName: string): TPropertyFilter; overload;
function RegisterPropertyInCategory(ACategoryClass: TPropertyCategoryClass;
  AComponentClass: TClass; const APropertyName: string): TPropertyFilter
  overload;
function RegisterPropertyInCategory(ACategoryClass: TPropertyCategoryClass;
  APropertyType: PTypeInfo; const APropertyName: string): TPropertyFilter;
  overload;
function RegisterPropertyInCategory(ACategoryClass: TPropertyCategoryClass;
  APropertyType: PTypeInfo): TPropertyFilter; overload;
```

این توابع سمبل‌های خاص را می‌شناسند و شما به عنوان مثال می‌توانید تمامی صفاتی که با "Data*" انطباق می‌یابند را بیافزایید. برای مشاهده لیست کامل کاراکترهای خاصی که پشتیبانی شده‌اند و همچنین بررسی رفتار آنها به راهنمای کلاس TMask مراجعه کنید.

()RegisterPropertyInCategory در سه گونه مختلف ظاهر می‌شود.

```
function RegisterPropertiesInCategory(ACategoryClass: TPropertyCategoryClass;
  const AFilters: array of const): TPropertyCategory; overload;
function RegisterPropertiesInCategory(ACategoryClass: TPropertyCategoryClass;
  AComponentClass: TClass; const AFilters: array of string): TPropertyCategory;
  overload;
function RegisterPropertiesInCategory(ACategoryClass: TPropertyCategoryClass;
  APropertyType: PTypeInfo; const AFilters: array of string):
TPropertyCategory;
  overload;
```

کلاس‌های طبقه

نوع ()TPropertyCategoryClass مرجع‌کلاس برای یک TPropertyCategory است. TPropertyCategory کلاس مبنا برای تمامی طبقات صفات استاندارد در VCL است. ۱۱ طبقه صفت مختلف داریم و

کلاس‌ها در جدول ۴-۹ شرح داده شده‌اند.

جدول ۴-۹ کلاس‌های طبقه صفت استاندارد

نام کلاس	توصیف
TAtionCategory	صفات مرتبط به اعمال زمان اجراء صفات Enabled و Hint از TControl در این طبقه هستند.
TDatabaseCategory	صفات مرتبط به عملیات بانک اطلاعاتی. صفات SQL و DatabaseName از TQuery در این طبقه هستند.
TDragNDropCategory	صفات مرتبط به اعمال کشیدن و رهاکردن (Drag and Drop) و Docking. صفات DragCursor و Dragkind از TControl در این طبقه هستند.
THelpCategory	صفات مرتبط به استفاده از راهنمای فعال و توضیحات. صفات Hint و HelpContext از TWinControl در این طبقه هستند.
TLayoutategory	صفات مرتبط با نمایش بصری یک کنترل در زمان طراحی. صفات Top و Left از TControl در این طبقه هستند.
TLegacyCategory	صفات مرتبط با اعمال منسوخ. صفات CtL3D و ParentCtL3D از TWinControl در این طبقه هستند.
TLinkageCategory	صفات مرتبط با همبند کردن و یا پیوند زدن یک جزء سازنده با جزء سازندهٔ دیگر. صفت DataSet از TDataSource در این طبقه قرار دارد.
TLocaleCategory	صفات مرتبط با متغیرهای محلی عمومی. صفات BiDiMode و ParentBiDiMode از TContorl در این طبقه قرار دارند.
TLocalizableCategory	صفات مرتبط با اعمال بانک اطلاعاتی. صفات DatabaseName و SQL از TQuery در این طبقه قرار دارند.
TVisualCategory	صفات مرتبط به نمایش بصری کنترل در زمان اجراء؛ صفات Align و Visible از TControl در این طبقه قرار دارند.
TInputCategory	صفات مرتبط با ورود داده (نیازی نیست که با اعمال بانک اطلاعاتی مرتبط باشند). صفات Enable و ReadOnly از TEdit در این طبقه قرار دارند.

به عنوان مثال فرض می‌کنیم که جزء سازنده‌ای به نام TNaeto با صفتی به نام Keen نوشته‌ایم و می‌خواهیم صفت Keen را به عنوان عضوی از طبقه Action که توسط TActionCategoly معرفی می‌شود ثبت نماییم. این کار را با افزودن فراخوانی RegisterPropertyInCategory() در روال Register() کنترل خود به صورت زیر می‌توان انجام داد:

```
RegisterPropertyInCategory (TActionCategory,TNeato, 'Keenl);
```

طبقات سفارشی

همانگونه که آموختید یک طبقه صفت به صورت کلاسی نمایش داده می‌شود که از

TPropertyCategory ارث می‌برد. ایجاد طبقات صفات خاص به این طریق چه دشواری دارد؟ در حقیقت انجام این کار بسیار آسان است. در بیشتر حالات تنها کاری که بایستی انجام دهید باطل‌سازی توابع مجازی کلاس ()Name و ()Description از TPropertyCategory است تا اطلاعات خاص طبقه شما ارجاع داده شوند.

به عنوان یک مثال عملی، در اینجا یک طبقه جدید Sound ایجاد خواهید کرد که برای طبقه‌بندی برخی از صفات TddgWaveFile به کار خواهد رفت. این کلاس طبقه جدید که TSoundCategory نامیده می‌شود در لیست ۱۰-۹ نشان داده شده است. این لیست حاوی فایل WavezEd.pas است که فایلی است که حاوی طبقه جزء سازنده، ویرایشگر صفت، و ویرایشگر جزء سازنده می‌باشد.

لیست ۱۰-۹ wavezEd.pas

```
unit WavezEd;

interface

uses DsgnIntf;

type
  { Category for some of TddgWaveFile's properties }
  TSoundCategory = class(TPropertyCategory)
public
  class function Name: string; override;
  class function Description: string; override;
end;

{ Property editor for TddgWaveFile's WaveName property }
TWaveFileStringProperty = class(TStringProperty)
public
  procedure Edit; override;
  function GetAttributes: TPropertyAttributes; override;
end;

{ Component editor for TddgWaveFile.  Allows user to play and stop }
{ WAV sounds from local menu in IDE. }
TWaveEditor = class(TComponentEditor)
private
  procedure EditProp(PropertyEditor: TPropertyEditor);
public
  procedure Edit; override;
  procedure ExecuteVerb(Index: Integer); override;
  function GetVerb(Index: Integer): string; override;
  function GetVerbCount: Integer; override;
end;
```

لیست ۱۰-۹

```
implementation

uses TypInfo, Wavez, Classes, Controls, Dialogs;

{ TSoundCategory }

class function TSoundCategory.Name: string;
begin
  Result := 'Sound';
end;

class function TSoundCategory.Description: string;
begin
  Result := 'Properties dealing with the playing of sounds'
end;

{ TWaveFileStringProperty }

procedure TWaveFileStringProperty.Edit;
{ Executed when user clicks the ellipses button on the WavName   }
{ property in the Object Inspector.  This method allows the user }
{ to pick a file from an OpenDialog and sets the property value. }
begin
  with TOpenDialog.Create(nil) do
    try
      { Set up properties for dialog }
      Filter := 'Wav files|*.wav|All files|*.*';
      DefaultExt := '*.wav';
      { Put current value in the FileName property of dialog }
      FileName := GetStrValue;
      { Execute dialog and set property value if dialog is OK }
      if Execute then
        SetStrValue(FileName);
    finally
      Free;
    end;
end;

function TWaveFileStringProperty.GetAttributes: TPropertyAttributes;
{ Indicates the property editor will invoke a dialog. }
begin
  Result := [paDialog];
end;

{ TWaveEditor }

const
```

لیست ۱۰-۹

```
    VerbCount = 2;
    VerbArray: array[0..VerbCount - 1] of string[7] = ('Play', 'Stop');

  procedure TWaveEditor.Edit;
  { Called when user double-clicks on the component at design time. }
  { This method calls the GetComponentProperties method in order to }
  { invoke the Edit method of the WaveName property editor. }
  var
    Components: TDesignerSelectionList;
  begin
    Components := TDesignerSelectionList.Create;
    try
      Components.Add(Component);
      GetComponentProperties(Components, tkAny, Designer, EditProp);
    finally
      Components.Free;
    end;
  end;
procedure TWaveEditor.EditProp(PropertyEditor: TPropertyEditor);
{ Called once per property in response to GetComponentProperties }
{ call.  This method looks for the WaveName property editor and  }
{ calls its Edit method. }
begin
  if PropertyEditor is TWaveFileStringProperty then begin
    TWaveFileStringProperty(PropertyEditor).Edit;
    Designer.Modified;     // alert Designer to modification
  end;
end;

procedure TWaveEditor.ExecuteVerb(Index: Integer);
begin
  case Index of
    0: TddgWaveFile(Component).Play;
    1: TddgWaveFile(Component).Stop;
  end;
end;

function TWaveEditor.GetVerb(Index: Integer): string;
begin
  Result := VerbArray[Index];
end;

function TWaveEditor.GetVerbCount: Integer;
begin
  Result := VerbCount;
end;

end.
```

هنگامی که کلاس طبقه تعریف شده است تنها کاری که بایستی انجام شود ثبت صفات طبقه با استفاده از یکی از توابع ثبت است. این کار در روال ()Register مربوط به TddgWaveFile و با استفاده از کد زیر انجام می‌شود.

```
RegisterPropertiesInCategory(TSoundCategory, TddgWaveFile,
  ['WaveLoop', 'WaveName', 'WavePause']);
```

لیست‌های اجزاء سازنده: TCollection و TCollectionItem

در اجزاء سازنده متداول است که لیستی از آیتم‌ها نظیر نوع‌های داده، رکوردها، اشیاء و یا حتی دیگر اجزاء پشتیبانی شده و یا در اختیار گرفته شوند. در برخی حالات، فشرده‌سازی این لیست در درون شیء و سپس در نظر گرفتن شیء موردنظر به عنوان صفتی از جزء سازندهٔ مالک مناسب است. مثالی از این کار را می‌توانید در صفت Lines از TMemo مشاهده کنید. Lines یک نوع شیء TStrings است که لیستی از رشته‌ها را مجتمع‌سازی می‌کند. با استفاده از این روش TString مسئول مکانیزم جریان داده به کار رفته برای ذخیره‌سازی خطوط در هنگامی است که کاربر فرم را ذخیره می‌کند.

هنگامی که قصد ذخیره‌سازی لیستی از آیتم‌ها نظیر اشیاء را داشته باشید که قبلاً توسط کلاسی مانند TStrings مجتمع‌سازی نشده باشند چه باید کرد؟ البته می‌توانید کلاسی را ایجاد کنید که عمل جاری ساختن آیتم‌های لیست شده را انجام دهد و سپس این کلاس را به عنوان صفتی از جزء سازندهٔ مالک قرار دهید. نتیجتاً، می‌توانید مکانیزم جاری‌سازی پیش‌فرض جزء سازندهٔ مالک را به گونه‌ای باطل‌سازی کنید که نحوه جاری‌سازی لیست آیتم‌های خود را بداند.

کلاس TCollection شیء است که برای ذخیره‌سازی لیستی از اشیاء TCollectionItem به کار رفته است. TCollection خود یک جزء سازنده نیست بلکه نسلی از TPersistent است.

برای استفاده از TCollection به منظور ذخیره‌سازی لیستی از آیتم‌ها بایستی یک کلاس وراثتی از TCollection ایجاد کنید که می‌توان آن را TNewCollection نامید. TNewCollection به عنوان نوع صفت یک جزء سازنده عمل می‌کند. سپس بایستی کلاسی از کلاس TCollectionItem اشتقاق دهید که می‌توان آن را TNewCollectionItem نامید. TNewCollection لیستی از اشیاء TNewCollectionItem را پشتیبانی خواهد نمود. حسن این کار در آنجاست که داده‌های مربوط به TNewCollectionItem که باید جریان یابند فقط بایستی توسط TNewCollectionItem منتشر شده باشند. دلفی می‌داند که چگونه صفات منتشر شده را جریان دهد.

مثالی از به کارگیری TCollection را می‌توان در جزء سازندهٔ TStatusBar یافت. StatusBar یک نسل از TWinControl است. یکی از صفات این جزء سازنده Panels است. TStatusBar.Panels از نوع TStatusPanels است که خود نسلی از TCollection می‌باشد و به شکل زیر تعریف می‌شود.

```
type
  TStatusPanels = class(TCollection)
  private
    FStatusBar: TStatusBar;
```

```
  function GetItem(Index: Integer): TStatusPanel;
  procedure SetItem(Index: Integer; Value: TStatusPanel);
protected
  procedure Update(Item: TCollectionItem); override;
public
  constructor Create(StatusBar: TStatusBar);
  function Add: TStatusPanel;
  property Items[Index: Integer]: TStatusPanel read GetItem write SetItem;
    default;
end;
```

TStatusPanels لیستی از نسل‌های TCollectionItem ، TStatusPanel را ذخیره نموده و به صورت زیر تعریف می‌شود.

```
type
  TStatusPanel = class(TCollectionItem)
  private
    FText: string;
    FWidth: Integer;
    FAlignment: TAlignment;
    FBevel: TStatusPanelBevel;
    FStyle: TStatusPanelStyle;
    procedure SetAlignment(Value: TAlignment);
    procedure SetBevel(Value: TStatusPanelBevel);
    procedure SetStyle(Value: TStatusPanelStyle);
    procedure SetText(const Value: string);
    procedure SetWidth(Value: Integer);
  public
    constructor Create(Collection: TCollection); override;
    procedure Assign(Source: TPersistent); override;
  published
    property Alignment: TAlignment read FAlignment
      write SetAlignment default taLeftJustify;
    property Bevel: TStatusPanelBevel read FBevel
      write SetBevel default pbLowered;
    property Style: TStatusPanelStyle read FStyle write SetStyle
      default psText;
    property Text: string read FText write SetText;
    property Width: Integer read FWidth write SetWidth;
  end;
```

صفات TStatusPanel در بخش Published از اعلان کلاس به صورت خودکار توسط دلفی جریان خواهند یافت. TStatusPanel در سازنده ()Create خود یک پارامتر TCollection را گرفته و خود را با TCollection همبسته می‌کرد. همچنین، TStatusPanels جزء سازنده TStatusBar را در سازنده با خود همبسته می‌کند و TStatusPanels جزء سازندهٔ TStutsBar را در سازنده که با خود همبسته کرده است می‌گیرد. موتور TCollection نحوه برخورد با جریان اجزاء سازندهٔ TCollectionItem را دانسته

و متدها و صفاتی برای دستکاری آیتم‌های پشتیبانی شده در TCollection تعریف می‌کند.

برای شرح نحوه استفاده از این دو کلاس جدید جزء سازنده TddgLaunchPad را ایجاد می‌کنیم. TddgLaunchPad کاربر را قادر به ذخیره‌سازی لیستی از اجزاء سازنده TddgRunButton می‌کند که در فصل ۸ ایجاد شده‌اند.

TddgLaunchPad نسلی از جزء سازنده TScrollBox است. یکی از صفات TddgLaunchPad، RunButtons یک نسل از TCollection است. RunButtons لیستی از اجزاء سازنده TRunBtunItem را پشتیبانی می‌کند. TRunButton نسلی است‌که صفات‌آن برای ایجادیک جزء سازنده TddgRunButton به کار رفته‌اند که این جزء سازنده در TddgLaunchPad قرار دارد. در بخش‌های آتی، نحوه ایجاد این جزء سازنده را شرح خواهیم داد.

تعریف کلاس TRunBtnItem : TCollectionItem

اولین گام، تعریف آیتم است که از آن در یک لیست استفاده کنیم. برای TddgLaunchPad این آیتم یک جزء سازنده TddgRuButton است. بنابراین هر نمونهٔ TRunBtnItem بایستی خود را با یک جزء سازنده TddgRunButton پیوند بزند. کد زیر یک تعریف جزئی از کلاس TRunBtnItem را نشان می‌دهد.

```
type
  TRunBtnItem = class(TCollectionItem)
  private
    FCommandLine: String;    // Store the command line
    FLeft: Integer;          // Store the positional properties for the
    FTop: Integer;           //    TddgRunButton.
    FRunButton: TddgRunButton; // Reference to a TddgRunButton
    ...
  public
    constructor Create(Collection: TCollection); override;
  published
    { The published properties will be streamed }
    property CommandLine: String read FCommandLine write SetCommandLine;
    property Left: Integer read FLeft write SetLeft;
    property Top: Integer read FTop write SetTop;
  end;
```

توجه داشته باشید که TRunBtnItem مرجعی به TddgRunButton را نگه می‌دارد که هنوز فقط صفات مورد نیاز برای ایجاد یک TddgRunButton را جریان می‌دهد. در ابتدا ممکن است تصور کنید که از آنجا که TRunBtnItem خود را با یک TddgRunButton پیوند زده است می‌توانست فقط جزء سازنده را منتشر کرده و بقیه کار را به موتور جریان بسپارد. این کار برخی مشکلات را برای موتور جریان باقی می‌گذارد و مسأله چگونگی برخورد متفاوت میان کلاس‌های TComponent و TPersistent را باقی می‌گذارد. در اینجا قانون بنیادی آن است که سیستم جریان مسئول ایجاد نمونه‌های جدید برای هر نام کلاس مشتق شده از TComponent است که در یک جریان یافت می‌شود، با توجه به این مسأله که

فرض می‌کند TPersistent همواره موجود بوده و قصد معرفی نمونه‌های جدید را ندارد.

با دنبال کردن این قانون اطلاعات مورد نیاز برای TddgRunButton را جریان داده و سپس TddgRunButton را در سازنده TRunBtnItem ایجاد می‌کنیم.

تعریف کلاس TRunButtons : TCollection

گام بعدی تعریف شیی است‌که لیست TRunBtnItem را پشتیبانی خواهدکرد. قبلاً گفتیم که این شیی بایستی یک نسل از TCollection باشد. این‌کلاس را TRunButtons می‌نامیم که به شکل زیر تعریف می‌شود.

```
type
  TRunButtons = class(TCollection)
  private
    FLaunchPad: TddgLaunchPad; // Keep a reference to the TddgLaunchPad
    function GetItem(Index: Integer): TRunBtnItem;
    procedure SetItem(Index: Integer; Value: TRunBtnItem);
  protected
    procedure Update(Item: TCollectionItem); override;
  public
    constructor Create(LaunchPad: TddgLaunchPad);
    function Add: TRunBtnItem;
    procedure UpdateRunButtons;
    property Items[Index: Integer]: TRunBtnItem read GetItem
      write SetItem; default;
  end;
```

TRunButtons خود را با یک جزء سازندهٔ TddgLaunchPad پیوند می‌زند. این‌کار در سازنده ()Create انجام می‌شودکه همانگونه‌که می‌توانید ببینید یک TddgLaunchPad را به‌عنوان پارامتر خود دریافت می‌کند. به صفات و متدهای مختلفی که اضافه شده‌اند تاکاربر را قادر به دستکاری‌کلاس‌های TRunBtnItem مختلف‌کنند توجه‌کنید. درحالت خاص، صفت Items آرایه‌ای در لیست TRunBtnItem است.

استفاده از کلاس‌های TRunBtnItem و TRunButtons هنگامی که پیاده‌سازی جزء سازندهٔ TddgLaunchPad را شرح دهیم شفاف‌تر می‌شود.

پیاده‌سازی اشیاء TddgLaunchPad، TRunBtnItem و TRunButtons

TddgLaunchPad صفتی از نوع TRunButtons دارد و پیاده‌سازی آن همانند پیاده‌سازی TRunBtnItem و TRunButtons است که در لیست ۱۱-۹ نشان داده شده است.

لیست ۱۱-۹ پیاده‌سازی TddgLancuhPad

```
unit LnchPad;

interface

uses
```

لیست ۱۱-۹ ادامه

```
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, RunBtn, ExtCtrls;

type
  TddgLaunchPad = class;

  TRunBtnItem = class(TCollectionItem)
  private
    FCommandLine: string;    // Store the command line
    FLeft: Integer;          // Store the positional properties for the
    FTop: Integer;           // TddgRunButton.
    FRunButton: TddgRunButton; // Reference to a TddgRunButton
    FWidth: Integer;         // Keep track of the width and height
    FHeight: Integer;
    procedure SetCommandLine(const Value: string);
    procedure SetLeft(Value: Integer);
    procedure SetTop(Value: Integer);
  public
    constructor Create(Collection: TCollection); override;
    destructor Destroy; override;
    procedure Assign(Source: TPersistent); override;
    property Width: Integer read FWidth;
    property Height: Integer read FHeight;
  published
    { The published properties will be streamed }
    property CommandLine: String read FCommandLine
      write SetCommandLine;
    property Left: Integer read FLeft write SetLeft;
    property Top: Integer read FTop write SetTop;
  end;

  TRunButtons = class(TCollection)
  private
    FLaunchPad: TddgLaunchPad; // Keep a reference to the TddgLaunchPad
    function GetItem(Index: Integer): TRunBtnItem;
    procedure SetItem(Index: Integer; Value: TRunBtnItem);
  protected
    procedure Update(Item: TCollectionItem); override;
  public
    constructor Create(LaunchPad: TddgLaunchPad);
    function Add: TRunBtnItem;
    procedure UpdateRunButtons;
    property Items[Index: Integer]: TRunBtnItem read
      GetItem write SetItem; default;
  end;

  TddgLaunchPad = class(TScrollBox)
```

لیست ۱۱-۹ ادامه

```
  private
    FRunButtons: TRunButtons;
    TopAlign: Integer;
    LeftAlign: Integer;
    procedure SetRunButtons(Value: TRunButtons);
    procedure UpdateRunButton(Index: Integer);
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
    procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
  published
    property RunButtons: TRunButtons read FRunButtons write SetRunButtons;
  end;

implementation

{ TRunBtnItem }

constructor TRunBtnItem.Create(Collection: TCollection);
{ This constructor gets the TCollection that owns this TRunBtnItem.  }
begin
  inherited Create(Collection);
  { Create an FRunButton instance. Make the launch pad the owner
    and parent. Then initialize its various properties. }
  FRunButton := TddgRunButton.Create(TRunButtons(Collection).FLaunchPad);
  FRunButton.Parent := TRunButtons(Collection).FLaunchPad;
  FWidth := FRunButton.Width;   // Keep track of the width and the
  FHeight := FRunButton.Height; //   height.
end;

destructor TRunBtnItem.Destroy;
begin
  FRunButton.Free;   // Destroy the TddgRunButton instance.
  inherited Destroy; // Call the inherited Destroy destructor.
end;

procedure TRunBtnItem.Assign(Source: TPersistent);
{ It is necessary to override the TCollectionItem.Assign method so that
  it knows how to copy from one TRunBtnItem to another. If this is done,
  then don't call the inherited Assign(). }
begin
  if Source is TRunBtnItem then
  begin
    { Instead of assigning the command line to the FCommandLine storage
      field, make the assignment to the property so that the accessor
      method will be called. The accessor method as some side-effects
      that we want to occur. }
    CommandLine := TRunBtnItem(Source).CommandLine;
    { Copy values to the remaining fields. Then exit the procedure. }
```

لیست ۱۱-۹ ادامه

```
      FLeft := TRunBtnItem(Source).Left;
      FTop := TRunBtnItem(Source).Top;
    Exit;
  end;
  inherited Assign(Source);
end;

procedure TRunBtnItem.SetCommandLine(const Value: string);
{ This is the write accessor method for TRunBtnItem.CommandLine. It
  ensures that the private TddgRunButton instance, FRunButton, gets
  assigned the specified string from Value }
begin
  if FRunButton <> nil then
  begin
    FCommandLine := Value;
    FRunButton.CommandLine := FCommandLine;
    { This will cause the TRunButtons.Update method to be called
      for each TRunBtnItem }
    Changed(False);
  end;
end;

procedure TRunBtnItem.SetLeft(Value: Integer);
{ Access method for the TRunBtnItem.Left property. }
begin
  if FRunButton <> nil then
  begin
    FLeft := Value;
    FRunButton.Left := FLeft;
   end;
end;

procedure TRunBtnItem.SetTop(Value: Integer);
{ Access method for the TRunBtnItem.Top property }
begin
  if FRunButton <> nil then
  begin
    FTop := Value;
    FRunButton.Top := FTop;
   end;
end;

{ TRunButtons }

constructor TRunButtons.Create(LaunchPad: TddgLaunchPad);
{ The constructor points FLaunchPad to the TddgLaunchPad parameter.
  LauchPad is the owner of this collection. It is necessary to keep
```

لیست ۱۱-۹ ادامه

```
  a reference to LauchPad as it will be accessed internally. }
begin
  inherited Create(TRunBtnItem);
  FLaunchPad := LaunchPad;
end;

function TRunButtons.GetItem(Index: Integer): TRunBtnItem;
{ Access method for TRunButtons.Items which returns the TRunBtnItem
  instance. }
begin
  Result := TRunBtnItem(inherited GetItem(Index));
end;

procedure TRunButtons.SetItem(Index: Integer; Value: TRunBtnItem);
{ Access method for TddgRunButton.Items which makes the assignment to
  the specified indexed item. }
begin
  inherited SetItem(Index, Value)
end;

procedure TRunButtons.Update(Item: TCollectionItem);
{ TCollection.Update is called by TCollectionItems
  whenever a change is made to any of the collection items. This is
  initially an abstract method. It must be overridden to contain
  whatever logic is necessary when a TCollectionItem has changed.
  We use it to redraw the item by calling TddgLaunchPad.UpdateRunButton.}
begin
  if Item <> nil then
    FLaunchPad.UpdateRunButton(Item.Index);
end;

procedure TRunButtons.UpdateRunButtons;
{ UpdateRunButtons is a public procedure that we made available so that
  users of TRunButtons can force all run-buttons to be re-drawn. This
  method calls TddgLaunchPad.UpdateRunButton for each TRunBtnItem
  instance. }
var
  i: integer;
begin
  for i := 0 to Count - 1 do
    FLaunchPad.UpdateRunButton(i);
end;

function TRunButtons.Add: TRunBtnItem;
```

لیست ۱۱-۹ ادامه

```
{ This method must be overridden to return the TRunBtnItem instance when
  the inherited Add method is called. This is done by typcasting the
  original result }
begin
  Result := TRunBtnItem(inherited Add);
end;

{ TddgLaunchPad }

constructor TddgLaunchPad.Create(AOwner: TComponent);
{ Initializes the TRunButtons instance and internal variables
  used for positioning of the TRunBtnItem as they are drawn }
begin
  inherited Create(AOwner);
  FRunButtons := TRunButtons.Create(Self);
  TopAlign := 0;
  LeftAlign := 0;
end;

destructor TddgLaunchPad.Destroy;
begin
  FRunButtons.Free;  // Free the TRunButtons instance.
  inherited Destroy; // Call the inherited destroy method.
end;

procedure TddgLaunchPad.GetChildren(Proc: TGetChildProc; Root: TComponent);
{ Override GetChildren to cause TddgLaunchPad to ignore any TRunButtons
  that it owns since they do not need to be streamed in the context
  TddgLaunchPad. The information necessary for creating the TddgRunButton
  instances is already streamed as published properties of the
  TCollectionItem descendant, TRunBtnItem. This method prevents the
  TddgRunButton's from being streamed twice. }
var
  I: Integer;
begin
  for I := 0 to ControlCount - 1 do
    { Ignore the run buttons and the scrollbox }
    if not (Controls[i] is TddgRunButton) then
      Proc(TComponent(Controls[I]));
end;

procedure TddgLaunchPad.SetRunButtons(Value: TRunButtons);
{ Access method for the RunButtons property }
begin
  FRunButtons.Assign(Value);
end;
```

لیست ۱۱-۹ ادامه

```
procedure TddgLaunchPad.UpdateRunButton(Index: Integer);
{ This method is responsible for drawing the TRunBtnItem instances.
  It ensures that the TRunBtnItem's do not extend beyond the width
  of the TddgLaunchPad. If so, it creates rows. This is only in effect
  as the user is adding/removing TRunBtnItems. The user can still
  resize the TddgLaunchPad so that it is smaller than the width of a
  TRunBtnItem }
begin
  { If the first item being drawn, set both positions to zero. }
  if Index = 0 then
  begin
    TopAlign := 0;
    LeftAlign := 0;
  end;
  { If the width of the current row of TRunBtnItems is more than
    the width of the TddgLaunchPad, then start a new row of TRunBtnItems. }
  if (LeftAlign + FRunButtons[Index].Width) > Width then
  begin
    TopAlign := TopAlign + FRunButtons[Index].Height;
    LeftAlign := 0;
 end;
 FRunButtons[Index].Left := LeftAlign;
 FRunButtons[Index].Top := TopAlign;
 LeftAlign := LeftAlign + FRunButtons[Index].Width;
end;

end.
```

پیاده‌سازی TRunBtnItem

سازنده TRunBtnItem.Create() وهله‌ای از TddgRunButton را ایجاد می‌کند. دو سطر زیر را TRunBtnItem.Create() نیاز به شرح بیشتری دارند.

```
FRunButton := TddgRunButton.Create(TRunButtons(Collection).FLaunchPad);
FRunButton.Parent := TRunButtons(Collection).FLaunchPad;
```

اولین سطر TddgRunButton ، FRunButton را ایجاد می‌کند. مالک FRunButton ، FLaunchPad را ایجاد می‌کند. مالک FRunButton ، FLaunchPad است که یک جزء سازندهٔ TddgLaunchPad است و فیلدی از شیی TCollection به عنوان پارامتر ارسال شده است. استفاده از FLaunchPad به عنوان مالک FRunButton ضروری است.

TRunBtnItem و شیء TRunButton هیچ کدام نمی‌توانند مالک باشند زیرا از TPersistent

ارث می‌برند. به خاطر داشته باشید که مالک بایستی یک TComponent باشد.

مـی‌خواهـیم مسأله‌ای را مـطرح کـنیم کـه هـنگامی رخ مـی‌دهد کـه FLaunchPad را مـالک TRunButton قرار دهیم. با انجام این کار، FLaunchPad را مالک FRunButton قرار خواهـیم داد. رفتار طبیعی موتور جریان باعث می‌شود که هنگامی که کاربر فرم را ذخیره می‌کند، دلفی FRunButton را به عنوان جزء سازنده‌ای جریان دهد که توسط FLaunchPad تحت مالکیت در آمده است. این رفتار مطلوب نیست زیرا FRunButton قبلاً در سازنده TRunBtnItem و بر مبنای اطلاعاتی که در مـتن TRunBtnItem جریان یافته‌اند ایجاد شده است. این یک بخش جزئی ولی حیاتی از اطلاعات است. در ادامه، نحوه جلوگیری از جریان یافتن TddgRunButton توسط TddgLaunchPad را بـه مـنظور اصلاح این رفتار نامطلوب خواهید دید.

در دومین سطر FLaunchPad را به عنوان پدر FRunButton تخصیص می‌دهد بـه گـونه‌ای کـه FLaunchPad بتواند مراقب ترسیم FRunButton باشد.

مخرب TRunBtnItem.Destroy()، FRunButton را قبل از فراخوانی مخرب موروثی آن آزاد می‌سازد.

در حـالات اصـلی لازم است کـه مـتد TRunBtnItem.Assign() کـه فـراخـوانـی شـده است را باطل‌سازی نماییم. یکی از این نمونه‌ها هنگامی است که برنامه اجرا شده و فرم از جریان داده خوانده می‌شود. در متد Assign() به TRunBtnItem گفته می‌شود که مقادیر جریان یـافته از صـفات را بـه صفات اجزائی که آن را احاطه کرده است (در این حالت TddgRunButton) نسبت می‌دهد.

متدهای دیگر متدهای دسترس برای صفات مختلف TRunBtnItem هستند و در توضیحات کد شرح داده شده‌اند.

پیاده‌سازی TRunButtons

TRunButtons.Create()، FLaunchPad را به پارامتر TddgLaunchPad که به آن فـرستاده شـده است اشاره می‌دهد به گونه‌ای که بتوان بعداً به آن مراجعه نمود.

TRunButtons.Update() متدی است که هنگامی به کار گـرفته مـی‌شود کـه تـغییر در یکـی از نمونه‌های TRunBtnItem رخ دهد. این متدهای منطقی است که از طریق آن تغییر رخ خواهد داد. از این متد برای فراخوانی متدی از TddgLaunchPad استفاده می‌کنیم که TRunBtnItem را مجدداً ترسیم می‌نماید. همچنین یک متد عمومی، UpdateRunButteons() اضافه کرده‌ایـم تـا کـاربر را قـادر بـه تحمیل رسم مجدد نماید.

متدهای باقیمانده TRunButtons متدهای دسترسی صفت هستند که در توضیحات کد در لیست ۱۱-۹ در مورد آنها توضیح داده شده است.

پیاده‌سازی TddgLaunchPad

سازنده و مخرب TddgLaunchPad ساده هستند. TddgLaunchPad.Create() یک نمونه از شـیء

TRunButtons ایجاد کرده و خود را به عنوان یک پارامتر ارسال می‌کند. TddgLaunchPad.Destroy() نمونهٔ TRunButtons را آزاد می‌سازد.

در اینجا توجه به باطل‌سازی TddgLaunchPad.GetChildren() دارای اهمیت است.

به خاطر داشته باشید که این کار از آن جهت لازم است که نبایستی آنها را در متن شیء TddgLaunchPad و بلکه در متن TRunBtnItem ایجاد کنیم. به دلیل آن که هیچ TddgRunButton ای به روال Proc ارسال نشده است، نبایستی آنها را از جریان داده خواند و یا جریان داد.

متدهای دیگر متدهای دسترس صفات بوده و در لیست ۱۱-۹ توضیح داده شده‌اند.

در پایان ویرایشگر صفت کلاس گردایه TRunButtons را در روال Rogister() این یونیت ثبت می‌کنیم. در بخش بعد این ویرایشگر صفت شرح داده شده و نحوه ویرایش لیستی از اجزاء سازنده از یک کادر ویرایشگر صفت بیان می‌شود.

ویرایش لیست TCollectionItem به کمک یک کادر ویرایشگر صفت

اکنون که TddgLaunchPad ، کلاس TRunButtons ، و کلاس TRunBtnItem را تعریف کردیم بایستی راهی برای کاربران فراهم کنیم تا TddgRunButton را به TRunButtons بیافزایند. بهترین راه برای انجام این کار استفاده از ویرایشگر صفتی است که لیست پشتیبانی شده توسط TRunButtons را دستکاری می‌نماید.

نکته

یک قرارداد که برای ویرایشگرهای صفت صادق است، قرار دادن دکمه‌ای به منظور به کارگیری تغییرات در فرم می‌باشد. این دکمه را در اینجا نشان نداده‌ایم ولی می‌توانید افزودن چنین دکمه‌ای به ویرایشگر صفت RunButtons را به عنوان تمرین انجام دهید.

لیست ۱۲-۹ کد مربوط به ویرایشگر صفت TddgLanchPad-RunButtons و مکالمه آن را نشان می‌دهد.

لیست ۱۲-۹ LPadPE.Pas

```
unit LPadPE;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, Buttons, RunBtn, StdCtrls, LnchPad, DesignIntf, DesignEditors,
  ExtCtrls, TypInfo;

type
```

لیست ۱۲-۹ ادامه

```
{ First declare the editor dialog }
TLaunchPadEditor = class(TForm)
  PathListBox: TListBox;
  AddBtn: TButton;
  RemoveBtn: TButton;
  CancelBtn: TButton;
  OkBtn: TButton;
  Label1: TLabel;
  pnlRBtn: TPanel;
  procedure PathListBoxClick(Sender: TObject);
  procedure AddBtnClick(Sender: TObject);
  procedure RemoveBtnClick(Sender: TObject);
  procedure FormCreate(Sender: TObject);
  procedure FormDestroy(Sender: TObject);
  procedure CancelBtnClick(Sender: TObject);
private
  TestRunBtn: TddgRunButton;
  FLaunchPad: TddgLaunchPad;   // To be used as a backup
  FRunButtons: TRunButtons; // Will refer to the actual TRunButtons
  Modified: Boolean;
  procedure UpdatePathListBox;
end;

{ Now declare the TPropertyEditor descendant and override the
  required methods }
TRunButtonsProperty = class(TPropertyEditor)
  function GetAttributes: TPropertyAttributes; override;
  function GetValue: string; override;
  procedure Edit; override;
end;
{ This function will be called by the property editor. }
function EditRunButtons(RunButtons: TRunButtons): Boolean;

implementation

{$R *.DFM}

function EditRunButtons(RunButtons: TRunButtons): Boolean;
{ Instantiates the TLaunchPadEditor dialog which directly modifies
  the TRunButtons collection. }
begin
  with TLaunchPadEditor.Create(Application) do
    try
      FRunButtons := RunButtons; // Point to the actual TRunButtons
      { Copy the TRunBtnItems to the backup FLaunchPad which will be
        used as a backup in case the user cancels the operation }
      FLaunchPad.RunButtons.Assign(RunButtons);
      { Draw the listbox with the list of TRunBtnItems. }
```

لیست ۱۲-۹ ادامه

```
      UpdatePathListBox;
      ShowModal; // Display the form.
      Result := Modified;
    finally
      Free;
    end;
end;

{ TLaunchPadEditor }

procedure TLaunchPadEditor.FormCreate(Sender:  TObject);
begin
  { Created the backup instances of TLaunchPad to be used if the user
    cancels editing the TRunBtnItems }
  FLaunchPad := TddgLaunchPad.Create(Self);

  // Create the TddgRunButton instance and align it to the
  // enclosing panel.
  TestRunBtn := TddgRunButton.Create(Self);
  TestRunBtn.Parent := pnlRBtn;

  TestRunBtn.Width  := pnlRBtn.Width;
  TestRunBtn.Height := pnlRBtn.Height;
end;

procedure TLaunchPadEditor.FormDestroy(Sender: TObject);
begin
  TestRunBtn.Free;
  FLaunchPad.Free; // Free the TLaunchPad instance.
end;

procedure TLaunchPadEditor.PathListBoxClick(Sender: TObject);
{ When the user clicks on an item in the list of TRunBtnItems, make
  the test TRunButton reflect the currently selected item }
begin
  if PathListBox.ItemIndex > -1 then
    TestRunBtn.CommandLine := PathListBox.Items[PathListBox.ItemIndex];
end;

procedure TLaunchPadEditor.UpdatePathListBox;
{ Re-initializes the PathListBox so that it reflects the list of
  TRunBtnItems }
var
  i: integer;
begin
  PathListBox.Clear; // First clear the list box.
  for i := 0 to FRunButtons.Count - 1 do
    PathListBox.Items.Add(FRunButtons[i].CommandLine);
end;
```

لیست ۱۲-۹ ادامه

```
procedure TLaunchPadEditor.AddBtnClick(Sender: TObject);
{ When the add button is clicked, launch a TOpenDialog to retrieve
  an executable filename and path. Then add this file to the
  PathListBox. Also, add a new FRunBtnItem. }
var
  OpenDialog: TOpenDialog;
begin
  OpenDialog := TOpenDialog.Create(Application);
  try
    OpenDialog.Filter := 'Executable Files|*.EXE';
    if OpenDialog.Execute then
    begin
      { add to the PathListBox. }
      PathListBox.Items.Add(OpenDialog.FileName);
      FRunButtons.Add; // Create a new TRunBtnItem instance.
      { Set focus to the new item in PathListBox }
      PathListBox.ItemIndex := FRunButtons.Count - 1;
      { Set the command line for the new TRunBtnItem to that of the
        file name gotten as specified by PathListBox.ItemIndex }
      FRunButtons[PathListBox.ItemIndex].CommandLine :=
      PathListBox.Items[PathListBox.ItemIndex];
    { Invoke the PathListBoxClick event handler so that the test
      TRunButton will reflect the newly added item }
    PathListBoxClick(nil);
    Modified := True;
  end;
 finally
  OpenDialog.Free
 end;
end;

procedure TLaunchPadEditor.RemoveBtnClick(Sender: TObject);
{ Remove the selected path/filename from PathListBox as well as the
  corresponding TRunBtnItem from FRunButtons }
var
  i: integer;
begin
  i := PathListBox.ItemIndex;
  if i >= 0 then
  begin
    PathListBox.Items.Delete(i);  // Remove the item from the listbox
    FRunButtons[i].Free;          // Remove the item from the collection
    TestRunBtn.CommandLine := ''; // Erase the test run button
    Modified := True;
  end;
end;
```

لیست ۱۲-۹ ادامه

```
procedure TLaunchPadEditor.CancelBtnClick(Sender: TObject);
{ When the user cancels the operation, copy the backup LaunchPad
  TRunBtnItems back to the original TLaunchPad instance. Then,
  close the form by setting ModalResult to mrCancel. }
begin
  FRunButtons.Assign(FLaunchPad.RunButtons);
  Modified := False;
  ModalResult := mrCancel;
end;

{ TRunButtonsProperty }

function TRunButtonsProperty.GetAttributes: TPropertyAttributes;
{ Tell the Object Inspector that the property editor will use a
  dialog. This will cause the Edit method to be invoked when the user
  clicks the ellipsis button in the Object Inspector.  }
begin
  Result := [paDialog];
end;

procedure TRunButtonsProperty.Edit;
{ Invoke the EditRunButton() method and pass in the reference to the
  TRunButton's instance being edited. This reference can be obtain by
  using the GetOrdValue method. Then redraw the LaunchDialog by calling
  the TRunButtons.UpdateRunButtons method. }
begin
  if EditRunButtons(TRunButtons(GetOrdValue)) then
    Modified;
  TRunButtons(GetOrdValue).UpdateRunButtons;
end;

function TRunButtonsProperty.GetValue: string;
{ Override the GetValue method so that the class type of the property
  being edited is displayed in the Object Inspector. }
begin
  Result := Format('(%s)',  [GetPropType^.Name]);
end;

end.
```

این یونیت ابتدا مکالمه TddgLaunchPadEditor و سپس ویرایشگر صفت TRunButtonProperty را تعریف می‌کند. هدف ما آن است که ابتدا ویرایشگر صفت را شرح دهیم زیرا این ویرایشگر صفت است که مکالمه را به کار می‌برد.

ویرایشگر صفت TRunButtonProperty تفاوت زیادی با ویرایشگر صفت مکالمه که قبلاً شرح

دادیم ندارد. در اینجا، متدهای ()GetAttibutes ، ()Edit و ()GetValue را باطل‌سازی می‌کنیم.

()GetAttributes مقدار ارجاع TPropertyAttributes را طوری تنظیم می‌کند که مشخص نماید که این ویرایشگر مکالمه‌ای را به‌کار می‌گیرد. مجدداً با انجام این‌کار یک دکمه حذف در Inspector Object قرار می‌گیرد.

متد ()GetValue از تابع ()GetPropType به منظور ارجاع اشاره‌گری به Runtime Type Information صفت در حال ویرایش استفاده می‌کند. این متد نام فیلد اطلاعاتی که رشته نوع صفت را نمایش می‌دهد را ارجاع می‌دهد. رشته در Object Inspector و در درون پرانتز قرار می‌گیرد که این کار قراردادی است که در دلفی رعایت می‌شود.

در پایان، متد ()Edit تابعی که در این یونیت تعریف شده است، ()EditRunButtons را فراخوانی می‌نماید. به عنوان پارامتر، مرجع به صفت TRunButtons با استفاده از تابع GetOrdValue گذر داده می‌شود. هنگامی که تابع ارجاع می‌شود، متد ()UpdateRunButton به کار گرفته می‌شود تا باعث شود RunButtons به منظور انعکاس تغییرات مجدداً ترسیم شود.

تابع ()EditRunButtons نمونه‌ای از TddgLaunchPadEditor را ایجاد کرده و فیلد FRunButtons خود را به پارامتر TRunButtons که به آن ارسال شده است اشاره می‌دهد. این تابع از این مراجعه به طور داخلی برای انجام تغییرات در TRunButtons استفاده می‌کند. تابع سپس TRunButtons را در یک TddgLaunchPad ، FLaunchPad کپی می‌کند. از این نمونه به عنوان پشتیبان در مواقعی که کاربر عمل ویرایش را لغو کند استفاده می‌شود.

سازنده ()Create فرم یک TddgLaunchPad را ایجاد می‌کند. مخرب ()Destroy اطمینان حاصل می‌کند که هنگام تخریب فرم آزاد شده است.

()PathListBoxClick به کار گیرنده رویداد OnClick مربوط به PathListBox است. این متد باعث می‌شود که TestRunBtn آیتم انتخاب شده در PathListBox را منعکس کند که مسیری به فایل اجرائی را نشان می‌دهد. کاربر می‌تواند روی این نمونهٔ TddgRunButton کلیک کند تا برنامه را آغاز نماید.

()AddButtonClick رویداد OnClick دکمه Add را به کار می‌گیرد. این تابع برای گرفتن یک نام فایل اجرایی از کاربر مسیر File | Open را به کار گرفته و مسیر این نام فایل را به PathListBox می‌افزاید.

()RemoveBtnClick رویداد OnClick دکمه Remove را به کار می‌گیرد. این تابع آیتم انتخاب شده را از PathListBox حذف می‌کند.

خلاصه

در این فصل تکنیک‌ها و روشهای پیشرفته‌تری برای طراحی اجزاء دلفی در اختیار شما قرار گرفت. مطالبی از قبیل افزودن توضیحات و متحرک‌سازی اجزاء، ویرایشگرهای اجزاء و ویرایشگرهای صفات مورد بحث قرار گرفتند. با مجهز بودن به این اطلاعات به همراه معلوماتی که در فصول قبل کسب کرده‌اید قادر به طراحی اجزاء سازنده برای رفع نیازهای خود خواهید بود.

فصل دهم

ساخت اجزاء سازندهٔ CLX

در این فصل می‌خوانید

- **CXL چیست؟**
- **معماری CLX**
- **اجزاء نمونه**
- **ویرایشگرهای طراحی CLX**
- **بسته‌ها**

در دو فصل گذشته بر روی ایجاد اجزاء دلفی تمرکز کردیم. به طور واضح‌تر، در فصل ۸ و فصل ۹ روی اجزاء سازندهٔ VCL تمرکز داشتیم. خوشبختانه اکثر کاری که در ایجاد اجزاء VCL فرا گرفتید در ایجاد اجزاء CLX نیز به کار خواهد آمد.

CLX چیست؟

CLX مخفف Component Library for Cross-Platform بوده و در آغاز در ابزار Linux RAD و kylix معرفی شده است. البته، CLX فقط VCL تحت Linux نیست. یعنی معماری CLX در دلفی ۷ نیز قابل دسترسی بوده و بنابراین مبنای ایجاد برنامه‌های محلی با استفاده از دلفی ۷ و kylix را فراهم می‌کند.

در دلفی، VCL نوعاً با اجزائی متناظر است که در Component Palette ظاهر می‌شوند. این موضوع تعجب‌آور نیست زیرا بیشتر اجزائی که در پالت ظاهر می‌شوند کنترل‌های Visual هستند. البته، CLX چیزی فراتر از ساختار یک جزء Visual را در برمی‌گیرد. به طور خاص، CLX به چهار بخش مجزا تقسیم می‌شود: BaseCLX ، VisualCLX ، DataCLX و NetCLCX .

BaseCLX حاوی یونیت‌ها و کلاس‌های مبنایی است که بین kylix و دلفی ۷ مشترکند. به عنوان مثال، یونیت‌های System ، SysUtils و Classes بخش‌هایی از BaseCLX هستند. VisualCLX شبیه چیزی است که اکثر افراد آن را VCL می‌شناسند. البته VisualCLX مبتنی بر کتابخانهٔ Qt است و نه کنترل‌های استاندارد Windows که در User32.dll یا ComCtl32.dll تعریف شده‌اند. DataCLX

حاوی اجزاء دسترسی داده‌ای بوده و تکنولوژی جدید dbExpress را در خود جای داده است. و در پایان، NetCLX حاوی تکنولوژی WebBorker برای تغییر سخت‌افزار و سیستم عامل است.

اگر با نگارش‌های قبلی دلفی آشنایی دارید تشخیص خواهید داد که یونیت‌هایی که در BaseCLX قرار دارند از نگارش ۱ دلفی در آن قابل دسترس هستند. بنابراین می‌توان ادعا کرد که این یونیت‌ها بخشی از VCL هستند. در واقع، بورلند متوجه این قضیه شد که فراخوانی این یونیت‌های مبنا به صورت مجموعه‌ای باعث بروز اغتشاش می‌شود و در دلفی ۷ این یونیت‌های مبنا به عنوان RTL شناخته می‌شوند.

نکته در اینجاست که حتی با این تصور که این یونیت‌های مبنا در هر دو برنامه‌های VCL و CLX به کار گرفته می‌شوند، یک برنامه CLX نوعاً به عنوان سازه‌ای تعریف می‌شود که از کلاس‌های VisualCLX استفاده می‌کند.

در این فصل بر روی VisualCLX تمرکز خواهیم کرد. به طور خاص نحوه توسعه معماری VisualCLX با ایجاد اجزاء CLX خود بررسی خواهیم نمود. همانگونه که قبلاً گفتیم VisualCLX مبتنی بر کتابخانه Qt است که توسط Troll Tech تولید شده است. Qt که Cute نامیده می‌شود کتابخانه کلاس C++ مستقل از بستر سخت‌افزاری و نرم‌افزاری است. برای شفافیت بیشتر، Qt در حال حاضر سیستم‌های Windows و X Windows را پشتیبانی می‌کند و بنابراین می‌توان آن را در محیط‌های کاری Windows و Linux به کار برد. در واقع، Qt متداول‌ترین کتابخانه کلاس به کار رفته برای توسعه رابط گرافیکی کاربرد (Linux GUT) Linux است.

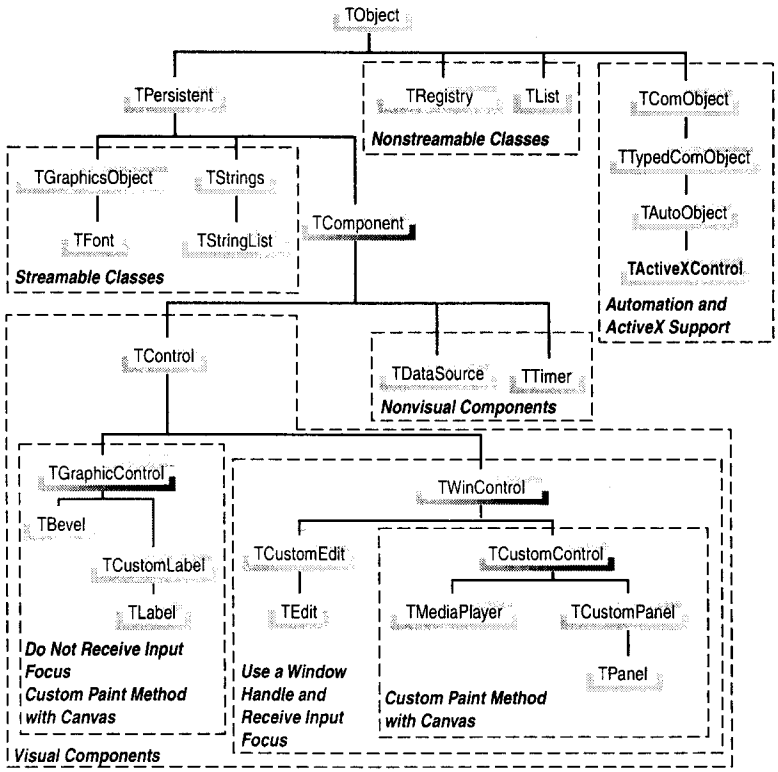
دیگر کتابخانه‌های کلاس نیز قابل دسترسی هستند ولی بورلند بنا به اهداف بسیار، VisualCLX را به منظور ساخت در بالای Qt انتخاب کرده است. اولاً کلاس‌های QT شباهت بسیاری به اجزاء VCL دارند. به عنوان مثال، صفات به صورت زوج متدهای get/set تعریف می‌شوند.

علاوه بر این، مدل گرافیکی Qt بسیار شبیه به مدل به کار رفته در VCL است. و در پایان، کتابخانه Qt گونه‌های وسیعی از کنترل‌های رابط کاربر را تعریف می‌کند.

معماری CLX

همانگونه که قبلاً بیان شد، VisualCLX تشکیل شده است از کلاس‌های ObjectPascal که قابلیت‌های موجود تعریف شده در کلاس‌های Qt آن را احاطه نموده‌اند. این مسأله شباهت زیادی به روشی دارد که در آن VCL قابلیت‌های API در Windows و کنترل‌های متداول آن را در برمی‌گیرد. یکی از اهداف طراحی در ایجاد CLX ساده شدن ارتباط میان برنامه‌های VCL با معماری CLX بوده است. به عنوان یک نتیجه، سلسله مراتب کلاس‌ها در CLX بسیار مشابه با VCL است که این موضوع در شکل‌های ۱-۱۰ و ۲-۱۰ بیان شده است. کادرهای تیره‌رنگ در شکل ۱-۱۰ کلاس‌های مبنای اصلی در VCL را مشخص می‌کنند.

البته سلسله مراتب کلاس‌ها عیناً مشابه هم نمی‌باشد. در حالت خاص، برخی کلاس‌های جدید به برخی کلاس‌ها اضافه شده و به انشعاب‌های متفاوتی با همتای VCL خود منتقل شده‌اند. این تفاوت‌ها

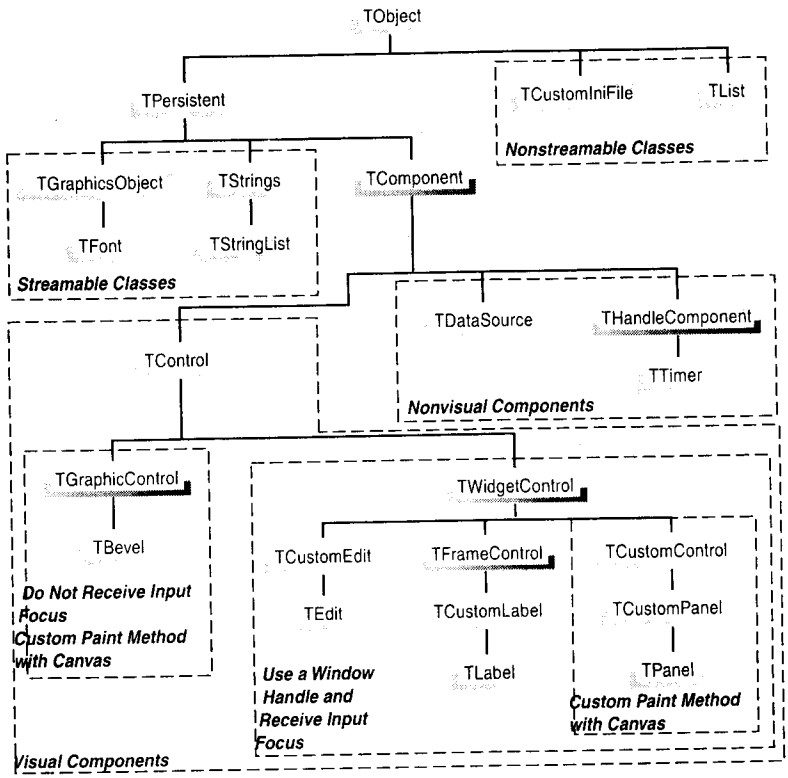


شکل ۱-۱۰ سلسله مراتب کلاس مبنای VCL

در شکل ۲-۱۰ با رنگ خاکستری روشن مشخص شده‌اند. به عنوان مثال جزء Timer مربوط به CLX مستقیماً از TComponent ارث نمی‌برد در حالی که در VCL چنین حالتی وجود داشت. در عوض این جزء از جزء جدید THandleComponent ارث می‌برد که یک کلاس مبنا است که بایستی هنگامی به کار رود که یک جزء NonVisual نیاز به دسترسی به دستگیره‌ای از یک کنترل Qt دارد. توجه کنید که جزء Label مربوط به CLX دیگر یک کنترل گرافیکی نیست و نسلی از کلاس جدید TFrameControl است. کتابخانه Qt گزینه‌های متعددی برای لبه‌های کنترل‌ها فراهم می‌کند و کلاس TFrameControl پوششی برای این کار ارائه می‌نماید.

همانگونه که قبلاً بیان شد کنترل‌های کتابخانه Qt ، widget نامیده می‌شوند. به عنوان نتیجه، کلاس TWidgetControl هم‌ارز CLX با TWinControl در VCL است چرا نام کلاس را تغییر دادیم؟ تغییر وضعیت به widget باعث می‌شود که کلاس در خطی با کلاس‌های مبنای Qt قرار گیرد و حذف Win باعث می‌شود که ارتباط میان کنترل‌های Windows در VisualCLX از بین برود.

جالب است‌که بدانید بورلندکلاس TWincontrol را در CLX نیز به‌عنوان نامی مستعاربرای TWidgetControl



شکل ۲-۱۰ سلسله مراتب کلاس مبنای CLX

تعریف کرده است. در طی توسعه Kylix و CLX ، یکی از ایده‌های اولیه که رواج یافت آن بود که یک فایل منبع یکتا را می‌توان به منظور تعریف یک جزء CLX و یک جزء VCL به کار برد.

دستورات شرطی برای مشخص کردن بند uses از VCL هنگام کمپایل تحت Windows و بند uses از CLX هنگام کمپایل تحت Linux به کار می‌روند. البته این روش فقط برای جزءهای بسیار ساده کاربرد دارد. در عمل، معمولاً تغییرات مهمی میان پیاده‌سازی‌ها وجود دارد تا ایجاد یونیت‌های مجزا را تضمین کند.

تذکر

ایجاد یک جزء VCL و یک جزء CLX در یک فایل منبع یکتا با ایجاد یک جزء CLX (در یک فایل منبع یکتا) که می‌توان آن را در هر دو Delphi 7 و Kylix به کار برد تفاوت دارد. .....

خوشبختانه، تغییرات در سلسله مراتب کلاس که در شکل ۲-۱۰ شرح داده شد تأثیر اندکی روی کار

تذکر

Object Browser در Delphi 7 و Kylix در فراگیری ساختار سلسله مراتب جدید کلاس کاربرد فراوان دارند. البته، از آنجا که TWinControl نام مستعار TWidgetControl دارد، در حقیقت دو طبقه کلاس مشابه در Object Browser خواهید دید.

توسعه‌دهندگان برنامه خواهد داشت. یعنی، اکثر اجزاء VCL که در دلفی ارائه می‌شوند متناظرهای VisualCLX نظیر TEdit ، TListBox ، TComboBox و ... را دارند. متأسفانه نویسندگان اجزاء چنین خوش‌شانس نیستند زیرا با توجه به سلسله مراتب کلاس‌ها، روی کار آنها تأثیر بیشتری گذاشته است. خوشبختانه شباهت‌های میان معماری VCL و CLX وجود دارد که در شکل‌ها مشخص نشده‌اند. به عنوان مثال، کلاس TCanvas در هر دو معماری بسیار مشابه می‌باشد. البته پیاده‌سای که توسط کلاس اعمال می‌شود هنوز هم تفاوت دارد. در VCL ، TCanvas پوششی بر متن ابزار GDI ویندوز اعمال می‌کند که از طریق صفت TCanvas.Handle قابل دسترسی است. در CLX کلاس TCanvas پوششی بر یک رسام Qt اعمال می‌کند که از طریق صفت TCanvas.Handle قابل دسترسی می‌باشد. به عنوان نتیجه، می‌توانید از صفت Handle برای دسترسی به هر یک از توابع GDI سطح پایین‌تر در یک برنامه VCL و توابع کتابخانه‌ای گرافیکی Qt در یک برنامه CLX استفاده کنید.

اجزاء در CLX به منظور سهولت بخشیدن ارتباط میان برنامه‌های VCL به یک برنامه CLX طراحی شده‌اند. به عنوان نتیجه، رابط‌های عمومی و انتشار یافته به بسیاری از اجزاء تقریباً در هر دو معماری شبیه هم می‌باشند. این بدان معنی است که رویدادهایی نظیر OnClick ، OnChange و OnKeyPress همانند متدهای ارسال مربوط به آنها- Click() ، Change() و KeyPress در هر دو VCL و CLX پیاده‌سازی شده‌اند.

ارتباط

CLX بسیاری از شباهت‌های خود با VCL را به اشتراک نگذاشته است. البته، بسیاری از تفاوت‌ها در محیط کاری را می‌توان به خصوص برای نویسندگان اجزاء پوشش داد. شما بایستی وابستگی‌های Win32 را در کد خود مشخص نمایید. به عنوان مثال، هر فراخوانی Win32 API (در Windows.pas) در صورتی که جزء مورد نظر در Kylix اجرا شود بایستی تغییر پیدا کند.

تذکر

ساختن یک جزء CLX به این موضوع دلالت دارد که شما می‌خواهید از آن جزء در kylix و تحت Linux و یا در Delphi 7 تحت Windows استفاده کند. اگر فقط نیاز به پشتیبانی Windows دارید جزء VCL ایجاد کنید.

به علاوه، برخی مطالب مربوط به کتابخانه‌های زمان اجراء (RTL) بایستی در Windows و Linux به صورتی متفاوت مورد برخورد قرار گیرند، نظیر حساس به حرف بودن نام فایل‌ها و جداکننده‌های مسیر در Linux. برای برخی اجزاء VCL، می‌توان آن را با Linux ارتباط داد. جزء VCL ای را در نظر بگیرید که پوششی در اطراف Messaging API (MPAI) قرار می‌دهد. از آنجا که MAPI تحت Linux وجود ندارد مکانیزمی متفاوت بایستی به کار رود.

علاوه بر مطالب مربوط به محیط کاری که ذکر شد، برخی مطالب مربوط به ارتباطات بایستی هنگام ادغام به CLX در نظر گرفته شود. به عنوان مثال، COM تحت Linux پشتیبانی نشده است ولی رابط‌ها بایستی چنین باشند. تکنیک‌های Owner – Draw که در بسیاری از پوشش‌های VCL در کنترل‌های Windows قابل دسترسی هستند در اجزاء CLX توصیه نشده‌اند. قابلیت‌های Owner – Draw به جای الگوهای Qt مناسب تشخیص داده نشده‌اند. بقیه جنبه‌های VCL که در CLX پشتیبانی نشده‌اند، عبارتند از bi– direction support, docking، input method editor و Asign Locale Support.

یک تغییر اضافی که باعث بروز مشکلاتی برای کاربران می‌شود آن است که نگارش‌های CLX برای اجزاء نسبت به VCL در مجموعه‌ای متفاوت از یونیت‌ها قرار دارند. به عنوان مثال، یونیت Controls.pas در VCL همان یونیت QControls.pas در CLX می‌باشد. شکل این تغییر آن است که اگر یک جزء CLX و یا برنامه‌ای تحت دلفی ۷ را پیاده‌سازی می‌کنید، یونیت‌های VCL هنوز هم قابل دسترسی هستند. به عنوان نتیجه هنوز هم می‌توانید یونیت‌های CLX و VCL را در صورت بی‌دقتی در یونیت‌های اجزاء سازندهٔ خود مخلوط کنید. در برخی حالات، جزء شما ممکن است به درستی تحت Windows اجراء شود. البته اگر جزء خود را به kylix منتقل کنید، با خطاهای کامپایلری مواجه می‌شوید زیرا یونیت‌های VCL در Linux قابل دسترسی نیستند.

تذکر

بورلند پیشنهاد می‌کند که کاربران برای جلوگیری از استفاده نادرست از یونیت‌های VCL در یک جزء CLX اجزاء خاص خود را در kilyx تحت Linux ایجاد نمایند. البته، بسیاری کاربران احتمالاً کار تحت دلفی ۷ را برمی‌گزینند که پیشرفت‌هایی در IDE داشته و کار با Windows برای آنها آسان است و پس از انجام این کار اجزاء خود را تحت Kylix تست می‌نمایند.

نکته دیگری که کاربران هنگام نوشتن اجزاء CLX با آن مقابله کنند حساس به متن بودن در Linux است. در حالت خاص به دلیل آن که نام فایل‌ها و مسیرها در Linux حساس به متن هستند، نام یونیت‌هایی که در بند uses خود مشخص می‌کنید بایستی صحیح تایپ شده باشد. این نکته بایستی در هنگام کار تحت کمپایلر kylix رعایت شود تا یونیت‌ها به درستی یافت شوند. علیرغم این که دلفی حساس به متن نیست، این اولین باری نیست که دلفی نیاز دارد که عنصری حساس به متن باشد.

عدم وجود پیغام‌های دیگر

Linux و یا به بیان مناسب‌تر X Windows معماری ارسال پیغام نظیر Windows را پیاده‌سازی نمی‌کند. به عنوان نتیجه ارسال پیغام‌های WM_LButtonDown ، WM_SetCursor و یا WM_Char در Linx وجود ندارند. هنگامی که یک جزء CLX در Linux به کار می‌رود، کلاس‌های Qt مربوط به رویدادهای مناسب سیستم را به کار گرفته و به منظور پاسخ به این رویدادها hook های لازم را فراهم می‌کند. سطر پائین آن است که رویدادهایی سیستم حتی روی Windows توسط کلاس‌های Qt مورد برخورد قرار می‌گیرند.

به عنوان یک نتیجه، مکانیزم‌های برخورد با پیغام در اجزاء VCL نظیر ()CMTextChanged با متدهای دینامیک تعویض شده‌اند. مثلاً، ()TextChanged . این موضوع در بخش بعد مورد بحث قرار می‌گیرد. این بدان معنی است که رفتارهای اصلی پیاده‌سازی که به سادگی با استفاده از پیغام‌ها در VCL پیاده‌سازی می‌شوند بایستی به صورتی کاملاً متفاوت در CLX پیاده‌سازی شوند.

اجزاء نمونه

در این بخش بررسی مفصلی روی برخی اجزاء VCL که به اجزاء CLX تغییر فرم یافته‌اند خواهیم داشت. اولین جزء یک جزء سنتی است که جنبه‌های متعددی از جمله کار با صفحه کلید، تغییر تمرکز، کار با ماوس، ترسیم و حتی رویدادها را در برمی‌گیرد.

سه جزء بعد، نسل‌های متوالی جزء فوق‌الذکر هستند. هر یک جزء قبل را تعمیم می‌دهند. اولین نسل با افزودن پشتیبانی برای به کارگیری رویدادهای ماوس در زمان طراحی و نمایش نشانگرهای انتخابی عمل تعمیم را انجام می‌دهد. دومین نسل پشتیبانی برای نمایش تصاویر از یک ImageList فراهم می‌سازد. آخرین جزء پشتیبانی برای اتصال کنترل به فیلدی در یک dataset اضافه می‌نماید.

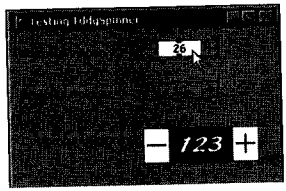
یادداشت

تمامی یونیت‌هایی که در این فصل ارائه شده‌اند را می‌توان در Delphi 7 و Kylix به کار برد. ▲

جزء TddgSpinner

شکل ۳-۱۰ سه وهله از جزء TddgSpinner که در برنامه‌های CLX استفاده می‌شوند را نشان می‌دهد. برخلاف ویرایشگرهای سنتی این جزء پیکربندی دکمه‌های افزایش و کاهش را نشان می‌دهد که مقدار spinner را در هر انتهای آن تغییر می‌دهند.

لیست ۱-۱۰ کد منبع مربوط به یونیت QddgSpin.pas را نشان می‌دهد که جزء TddgSpinner را پیاده‌سازی می‌کند. این جزء خاص، به عنوان یک کنترل پیکربندی کار می‌کند که از کلاس TCustomControl در VCL ارث می‌برد. البته کلاس TddgSpinner اکنون از کلاس



شکل ۳-۱۰ جزء TddgSpinner ،CLX را می‌توان برای مشخص کردن مقادیر عددی به کار گرفت.

تذکر

هر لیست حاوی کد خاص VCL است. توضیحاتی که با : CLX → VCL آغاز شده‌اند مطالب خاص مربوط به تغییر فرم از VCL به CLX را بیان می‌کنند.

لیست ۱-۱۰ QddgSpin.pas - کد منبع برای جزء TddgSpinner

```
unit QddgSpin;

interface

uses
  SysUtils, Classes, Types, Qt, QControls, QGraphics;
  (*
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  ImgList;
  *)

type
  TddgButtonType = ( btMinus, btPlus );
  TddgSpinnerEvent = procedure (Sender: TObject; NewValue: Integer;
                                var AllowChange: Boolean ) of object;

  TddgSpinner = class( TCustomControl )
  private
    // Instance Data for Component
    FValue: Integer;
    FIncrement: Integer;
    FButtonColor: TColor;
    FButtonWidth: Integer;
    FMinusBtnDown: Boolean;
    FPlusBtnDown: Boolean;

    // Method Pointers to Hold Custom Events
    FOnChange: TNotifyEvent;
    FOnChanging: TddgSpinnerEvent;

    (*
    // VCL->CLX:  These message handlers are not available in CLX

    // Window Message Handling Method
    procedure WMGetDlgCode( var Msg: TWMGetDlgCode );
      message wm_GetDlgCode;
```

لیست ۱-۱۰ ادامه

```
  // Component Message Handling Method
  procedure CMEnabledChanged( var Msg: TMessage );
    message cm_EnabledChanged;
  *)
protected
  procedure Paint; override;
  procedure DrawButton( Button: TddgButtonType; Down: Boolean;
                        Bounds: TRect ); virtual;

  // Support Methods
  procedure DecValue( Amount: Integer ); virtual;
  procedure IncValue( Amount: Integer ); virtual;

  function CursorPosition: TPoint;
  function MouseOverButton( Btn: TddgButtonType ): Boolean;

  // VCL->CLX:  EnabledChanged replaces cm_EnabledChanged
  //            component message handler
  procedure EnabledChanged; override;

  // New Event Dispatch Methods
  procedure Change; dynamic;
  function CanChange( NewValue: Integer ): Boolean; dynamic;

  // Overridden Event Dispatch Methods
  procedure DoEnter; override;
  procedure DoExit; override;
  procedure KeyDown(var Key: Word; Shift: TShiftState); override;

  procedure MouseDown( Button: TMouseButton; Shift: TShiftState;
                       X, Y: Integer ); override;
  procedure MouseUp( Button: TMouseButton; Shift: TShiftState;
                     X, Y: Integer ); override;

  (*
  // VCL->CLX:  These following declarations have changed in CLX

  function DoMouseWheelDown( Shift: TShiftState;
                             MousePos: TPoint ): Boolean; override;

  function DoMouseWheelUp( Shift: TShiftState;
                           MousePos: TPoint ): Boolean; override;
  *)
```

لیست ۱-۱۰ ادامه

```
  function DoMouseWheelDown( Shift: TShiftState;
                    const MousePos: TPoint ): Boolean; override;

  function DoMouseWheelUp( Shift: TShiftState;
                    const MousePos: TPoint ): Boolean; override;


  // Access Methods for Properties
  procedure SetButtonColor( Value: TColor ); virtual;
  procedure SetButtonWidth( Value: Integer ); virtual;
  procedure SetValue( Value: Integer ); virtual;
public
  // Don't forget to specify override for constructor
  constructor Create( AOwner: TComponent ); override;
published
  // New Property Declarations
  property ButtonColor: TColor
    read FButtonColor
    write SetButtonColor
    default clBtnFace;

  property ButtonWidth: Integer
    read FButtonWidth
    write SetButtonWidth
    default 18;

  property Increment: Integer
    read FIncrement
    write FIncrement
    default 1;

  property Value: Integer
    read FValue
    write SetValue;

  // New Event Declarations

  property OnChange: TNotifyEvent
    read FOnChange
    write FOnChange;

  property OnChanging: TddgSpinnerEvent
    read FOnChanging
    write FOnChanging;
```

لیست ۱-۱۰ ادامه

```
    // Inherited Properties and Events
    property Color;
    (*
    property DragCursor;      // VCL->CLX:  Property not yet in CLX
    *)
    property DragMode;
    property Enabled;
    property Font;
    property Height default 18;
    property HelpContext;
    property Hint;
    property ParentShowHint;
    property PopupMenu;
    property ShowHint;
    property TabOrder;
    property TabStop default True;
    property Visible;
    property Width default 80;

    property OnClick;
    property OnDragDrop;
    property OnDragOver;
    property OnEndDrag;
    property OnEnter;
    property OnExit;
    property OnKeyDown;
    property OnKeyPress;
    property OnKeyUp;
    property OnMouseDown;
    property OnMouseMove;
    property OnMouseUp;
    property OnStartDrag;
  end;


implementation


{=========================}
{== TddgSpinner Methods ==}
{=========================}

constructor TddgSpinner.Create( AOwner: TComponent );
begin
```

لیست ۱-۱۰ ادامه

```
  inherited Create( AOwner );

  // Initialize Instance Data
  FButtonColor := clBtnFace;
  FButtonWidth := 18;
  FValue := 0;
  FIncrement := 1;

  FMinusBtnDown := False;
  FPlusBtnDown := False;

  // Initializing inherited properties
  Width := 80;
  Height := 18;
  TabStop := True;

  // VCL->CLX:  TWidgetControl sets Color property to clNone
  Color := clWindow;

  // VCL->CLX:  InputKeys assignment replaces handling the
  //            wm_GetDlgCode message.
  InputKeys := InputKeys + [ ikArrows ];
end;


{== Property Access Methods ==}

procedure TddgSpinner.SetButtonColor( Value: TColor );
begin
  if FButtonColor <> Value then
  begin
    FButtonColor := Value;
    Invalidate;
  end;
end;

procedure TddgSpinner.SetButtonWidth( Value: Integer );
begin
  if FButtonWidth <> Value then
  begin
    FButtonWidth := Value;
    Invalidate;
  end;
end:
```

لیست ۱-۱۰ ادامه

```
procedure TddgSpinner.SetValue( Value: Integer );
begin
  if FValue <> Value then
  begin
    if CanChange( Value ) then
    begin
      FValue := Value;
      Invalidate;

      // Trigger Change event
      Change;
    end;
  end;
end;


{== Painting Related Methods ==}

procedure TddgSpinner.Paint;
var
  R: TRect;
  YOffset: Integer;
  S: string;
  XOffset: Integer;              // VCL->CLX:  Added for CLX support
begin
  inherited Paint;
  with Canvas do
  begin
    Font := Self.Font;
    Pen.Color := clBtnShadow;

    if Enabled then
      Brush.Color := Self.Color
    else
    begin
      Brush.Color := clBtnFace;
      Font.Color := clBtnShadow;
    end;

    // Display Value
    (*
    // VCL->CLX:  SetTextAlign not available in CLX
    SetTextAlign( Handle, ta_Center or ta_Top );    // GDI function
    *)
```

لیست ۱-۱۰ ادامه

```
    R := Rect( FButtonWidth - 1, 0,
               Width - FButtonWidth + 1, Height );
    Canvas.Rectangle( R.Left, R.Top, R.Right, R.Bottom );
    InflateRect( R, -1, -1 );


    S := IntToStr( FValue );
    YOffset := R.Top + ( R.Bottom - R.Top -
                         Canvas.TextHeight( S ) ) div 2;

    // VCL->CLX:  Calculate XOffset b/c no SetTextAlign function
    XOffset := R.Left + ( R.Right - R.Left -
                          Canvas.TextWidth( S ) ) div 2;

    (*
    // VCL->CLX:  Change TextRect call b/c no SetTextAlign function
    TextRect( R, Width div 2, YOffset, S );
    *)
    TextRect( R, XOffset, YOffset, S );

    DrawButton( btMinus, FMinusBtnDown,
                Rect( 0, 0, FButtonWidth, Height ) );
    DrawButton( btPlus, FPlusBtnDown,
                Rect( Width - FButtonWidth, 0, Width, Height ) );

    if Focused then
    begin
      Brush.Color := Self.Color;
      DrawFocusRect( R );
    end;
  end;
end; {= TddgSpinner.Paint =}


procedure TddgSpinner.DrawButton( Button: TddgButtonType;
                                  Down: Boolean; Bounds: TRect );
begin
  with Canvas do
  begin
    if Down then                              // Set background color
      Brush.Color := clBtnShadow
    else
      Brush.Color := FButtonColor;
    Pen.Color := clBtnShadow;
```

لیست ۱-۱۰ ادامه

```
    Rectangle( Bounds.Left, Bounds.Top,
               Bounds.Right, Bounds.Bottom );

    if Enabled then
    begin
      (*
      // VCL->CLX: clActiveCaption is set to
      //           clActiveHighlightedText in CLX.
      Pen.Color := clActiveCaption;
      Brush.Color := clActiveCaption;
      *)
      Pen.Color := clActiveBorder;
      Brush.Color := clActiveBorder;
    end
    else
    begin
      Pen.Color := clBtnShadow;
      Brush.Color := clBtnShadow;
    end;

    if Button = btMinus then                // Draw the Minus Button
    begin
      Rectangle( 4, Height div 2 - 1,
                 FButtonWidth - 4, Height div 2 + 1 );
    end
    else                                     // Draw the Plus Button
    begin
      Rectangle( Width - FButtonWidth + 4, Height div 2 - 1,
                 Width - 4, Height div 2 + 1 );
      Rectangle( Width - FButtonWidth div 2 - 1,
                 ( Height div 2 )   (FButtonWidth div 2 - 4),
                 Width - FButtonWidth div 2 + 1,
                 ( Height div 2 ) + (FButtonWidth div 2 - 4)  );
    end;
    Pen.Color := clWindowText;
    Brush.Color := clWindow;
  end;
end; {= TddgSpinner.DrawButton =}


procedure TddgSpinner.DoEnter;
begin
  inherited DoEnter;
```

لیست ۱-۱۰ ادامه

```
  // Controls gets focus--update display to show focus border
  Repaint;
end;

procedure TddgSpinner.DoExit;
begin
  inherited DoExit;
  // Control lost focus--update display to remove focus border
  Repaint;
end;


// VCL->CLX:  EnabledChanged replaces cm_EnabledChanged handler

procedure TddgSpinner.EnabledChanged;
begin
  inherited;
  // Repaint the component so that it reflects the state change
  Repaint;
end;


{== Event Dispatch Methods ==}

{=====================================================================
  TddgSpinner.CanChange

  This is the event dispatch method supporting the OnChanging
  event. Notice that this method is a function, rather than the
  common procedure variety. As a function, the Result variable is
  assigned a value before calling the user defined event handler.
=====================================================================}

function TddgSpinner.CanChange( NewValue: Integer ): Boolean;
var
  AllowChange: Boolean;
begin
  AllowChange := True;
  if Assigned( FOnChanging ) then
    FOnChanging( Self, NewValue, AllowChange );
  Result := AllowChange;
end;
```

لیست ۱-۱۰ ادامه

```
procedure TddgSpinner.Change;
begin
  if Assigned( FOnChange ) then
    FOnChange( Self );
end;


// Notice that both DecValue and IncValue assign the new value to
// the Value property (not FValue), which indirectly calls SetValue

procedure TddgSpinner.DecValue( Amount: Integer );
begin
  Value := Value - Amount;
end;

procedure TddgSpinner.IncValue( Amount: Integer );
begin
  Value := Value + Amount;
end;


{== Keyboard Processing Methods ==}

(*
// VCL->CLX:  Replaced with InputKeys assignment in constructor

procedure TddgSpinner.WMGetDlgCode( var Msg: TWMGetDlgCode );
begin
  inherited;
  Msg.Result := dlgc_WantArrows;  // Control will handle arrow keys
end;
*)


procedure TddgSpinner.KeyDown( var Key: Word; Shift: TShiftState );
begin
  inherited KeyDown( Key, Shift );

  // VCL->CLX:  Key constants changed in CLX.
  //            vk_ prefix changed to Key_

  case Key of
    Key_Left, Key_Down:
      DecValue( FIncrement );
```

لیست ۱-۱۰ ادامه

```
    Key_Up, Key_Right:
      IncValue( FIncrement );
  end;
end;


{== Mouse Processing Methods ==}

function TddgSpinner.CursorPosition: TPoint;
begin
  GetCursorPos( Result );
  Result := ScreenToClient( Result );
end;


function TddgSpinner.MouseOverButton(Btn: TddgButtonType): Boolean;
var
  R: TRect;
begin
  // Get bounds of appropriate button
  if Btn = btMinus then
    R := Rect( 0, 0, FButtonWidth, Height )
  else
    R := Rect( Width - FButtonWidth, 0, Width, Height );

  // Is cursor position within bounding rectangle?
  Result := PtInRect( R, CursorPosition );
end;


procedure TddgSpinner.MouseDown( Button: TMouseButton;
                                 Shift: TShiftState; X, Y: Integer);
begin
  inherited MouseDown( Button, Shift, X, Y );

  if not ( csDesigning in ComponentState ) then
    SetFocus;              // Move focus to Spinner only at runtime

  if ( Button = mbLeft ) and
     ( MouseOverButton(btMinus) or MouseOverButton(btPlus) ) then
  begin
    FMinusBtnDown := MouseOverButton( btMinus );
    FPlusBtnDown := MouseOverButton( btPlus );
```

لیست ۱-۱۰ ادامه

```
    Repaint;
  end;
end;


procedure TddgSpinner.MouseUp( Button: TMouseButton;
                               Shift: TShiftState; X, Y: Integer );
begin
  inherited MouseUp( Button, Shift, X, Y );

  if Button = mbLeft then
  begin
    if MouseOverButton( btPlus ) then
      IncValue( FIncrement )
    else if MouseOverButton( btMinus ) then
      DecValue( FIncrement );

    FMinusBtnDown := False;
    FPlusBtnDown := False;

    Repaint;
  end;
end;


function TddgSpinner.DoMouseWheelDown( Shift: TShiftState;
                                const MousePos: TPoint ): Boolean;
begin
  inherited DoMouseWheelDown( Shift, MousePos );
  DecValue( FIncrement );
  Result := True;
end;


function TddgSpinner.DoMouseWheelUp( Shift: TShiftState;
                                const MousePos: TPoint ): Boolean;
begin
  inherited DoMouseWheelUp( Shift, MousePos );
  IncValue( FIncrement );
  Result := True;
end;

end.
```

TCustomControl ارث برده و به عنوان نتیجه، می‌توانیم از آن در Windows و Linux استفاده کنیم.

علیرغم این که نام کلاس‌ها هنگام تبدیل به CLX تغییر می‌کنند، نام یونیت‌ها نوعاً پسوند Q دارند که وابستگی آنها به کتابخانه Qt از طریق VisualCLX را مشخص می‌نماید.

همانگونه که می‌توانید ببینید، کد منبع مربوط به نگارش CLX بسیار شبیه به نگارش VCL است. البته تفاوت‌های مهمی وجود دارند.

ابتدا به ضمیمه شدن یونیت‌های خاص Qt یعنی Qt ، QControls و QGraphics توجه کنید. Typers نیز یونیت جدیدی است که بین VCL و CLX به اشتراک گذاشته شده است. خوشبختانه بخش اعظم اعلان کلاس CLX ، TddgSpinner مشابه با گونه متناظر آن در VCL دیده می‌شود. متدهای برخورد با پیغام ()CMEnableChanged و ()WMGetDlgCode اولین تغییر پیاده‌سازی که بایستی در تبدیل به CLX در نظر گرفته شود را مشخص می‌کنند. به طور خاص، پیغام‌های متناظر cm–EnableChanged و WM–GetDlgCode در CLX وجود ندارند. بنابراین عملکرد پیاده‌سازی شده در این مکانیزم‌های برخورد با پیغام بایستی به جاهای دیگر منتقل شوند.

همانگونه که قبلاً گفتیم، در CLX ، پیغام‌های اجزاء نظیر cm–EnabledChanged با متدهای دینامیک متناظر آنها جایگزین شده‌اند. بنابراین به جای پیغام cm–EnabledChanged هنگام تغییر صفت Enabled ، کلاس TControl در CLX متد ()EnabledChangde را فراخوانی می‌کند. بنابراین کد از متد ()CMEnabledChanged به سادگی به متد باطل‌سازی شده، ()EnabledChanged منتقل می‌شود.

یک عمل متداول در نوشتن اجزاء به کارگیری کلیدهای پیکانی روی صفحه کلید است. برای جزء TddgSpinner کلیدهای پیکانی را می‌توان به منظور افزایش و یا کاهش تعداد به کار برد. در یک جزء VCL ، این رفتار با به کارگیری پیغام WM–GetDlgCode و مشخص کردن این که کدام کلیدها به کار گرفته خواهند شد به انجام می‌رسد. همانگونه که قبلاً گفتیم پیغام WM–GetDlgCode برای جزء CLX وجود ندارد. بنابراین روش متفاوت بایستی اتخاذ شود. خوشبختانه، کلاس TWidgetControl صفت InputKyes را تعریف می‌کند که این صفت ما را قادر به مشخص کردن کلیدهای مورد استفاده در سازنده و یا جزء می‌کند.

کد سازنده همچنین تغییر دیگری میان VCL و CLX را مشخص می‌کند. یعنی، کلاس TWidgetControl صفت Color را تنظیم می‌کند که در TControl با مقدار clNone اعلان شده است. در VCL ، کلاس TWinControl از مقدار موروثی Color مربوط به clWindow استفاده می‌کند.

به عنوان نتیجه، بایستی صفت Color را در سازنده، با clWindow تنظیم کنیم به گونه‌ای که spinner با رنگی صحیح ظاهر شود.

پس از انجام این تغییرات در سازنده، تغییر زیاد دیگری نباید انجام شود همانگونه که می‌توانید ببینید، اکثر متدهای ارسال رویداد در CLX نیز قابل دسترسی هستند. به عنوان نتیجه، ساده‌تر است که به جای آن که از پیغام‌های خاص Windows برای دستگیره موردنظر پنجره استفاده کنید در صورتی که در

حال حاضر متدهای ارسال رویداد را در اجزاء خود باطل‌سازی کرده‌اید عمل تبدیل به CLX را انجام دهید.

در آغاز این فصل، بیان شد که تمامی تکنیک‌هایی که در فصول قبل در مورد ساخت اجزاء VCL فرا گرفته‌اید در ایجاد اجزاء CLX نیز به کار می‌آیند. توجه خواهید داشت که اعلان صفات، متدهای دسترسی، و حتی رویدادهای پیکربندی با روشی مشابه در CLX و VCL به کار گرفته می‌شوند.

متد ()Paint هنگام تبدیل یک جزء VCL به CLX نیاز به بیشترین تغییرات خواهد داشت.

هنگام تغییر فرم یک کنترل VCL به یک جزء CLX ، متدهای نمایش، نظیر ()Paint نیاز به بیشترین تغییرات را دارند حتی با این تصور که کلاس‌های TCanvas در هر دو معماری رابطهای تقریباً مشابهی دارند.

دو کاربر برای تغییر فرم جزء TdgSpinner بایستی انجام شوند. ابتدا، نگارش VCL مربوط به TdgSpinner از تابع GDI ، SetTextAlign استفاده می‌کند تا متن spinner را به صورت اتوماتیک در وسط ناحیه نمایش قرار دهد. البته، تحت Linux این تابع API وجود ندارد. حتی تحت Windows نیز این تابع کار نخواهد کرد زیرا منتظر دستگیره‌ای به یک متن ابزار GDI است و اجزاء CLX دسترسی به متن ابزار ندارند.

خوشبختانه اکثر متدهای TCanvas تحت Linux و Windows وجود دارند. بنابراین می‌توانیم با محاسبه محل مرکز به صورت دستی، بر این مشکل غلبه کنیم.

دومین مسأله نمایش مربوط به متد ()DrawButton است. در حالت خاص، سمبل‌های جمع و تفریق روی دکمه‌ها با استفاده از رنگ clActiveCaption در VCL رسم شده‌اند.

متأسفانه شناسه clActiveCaption به مقدار clActiveHighihtedText در یونیت QGraphics.pas نسبت داده شده است که چیزی که می‌خواهیم نیست.

تذکر

برای اجرای هر عمل ترسیم خارج از متد ()Paint جزء CLX خود بایستی ابتدا متد ()Canvas.Start را فراخوانی کرده و سپس هنگام خاتمه کار متد ()Canvas.Stop را فراخوانی کنید.

همه چیز به سادگی که انتظار دارید مبادله نمی‌شوند. ثابت‌های کد کلیدی مجازی که در VCL تعریف شده‌اند نظیر vk–Left در CLX قابل دسترسی نیستند. در عوض یک مجموعه کاملاً جدید از ثابت‌ها به منظور تشخیص این که کدام کلید فشرده شده است به کار رفته است. نتیجه می‌گیریم که کدهای مجازی کلید بخشی از Windows API بوده و بنابراین تحت Linux قابل دسترسی نیستند. اکنون یک جزء پیکربندی CLX کاملاً مناسب داریم که می‌توانیم از آن در برنامه‌های Windows که در Delphi 7 توسعه یافته‌اند و برنامه‌های Linux که توسط Kylix توسعه یافته‌اند استفاده نماییم.

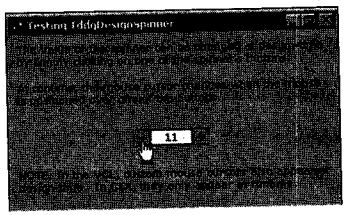
البته، مهمترین جنبه این مسأله آن است که برای هر دو محیط کد منبع مشابهی به کار می‌رود.

بهبودهای زمان طراحی

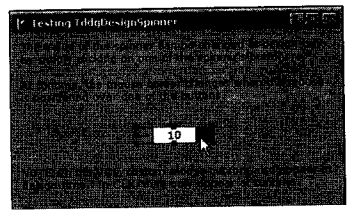
همه چیز در نظر گرفته شد، تبدیل جزء TddgSpinner VCL به CLX کاری آسان بود و نیاز به تکنیک خاص نداشت، علیرغم این که کشف صفت InputKeys نیاز به تلاش بیشتری داشت.

البته همانگونه که خواهید دید هنگامی که افزودن بخش‌های بیشتری را به اجزاء CLX خود آغاز کنیم تفاوت میان VCL و CLX آشکار خواهد شد.

کد منبع لیست ۲-۱۰ را در نظر بگیرید. این یونیت TddgDesignSpinner را پیاده‌سازی می‌کند که نسلی از TddgSpinner است. شکل ۴-۱۰ نحوه تغییر نشانگر ماوس در هنگامی که روی یکی از دکمه‌ها قرار می‌گیرد را نشان می‌دهد.



شکل ۴-۱۰ TddgDesingSpinner
هنگامی که ماوس روی دکمه‌ای قرار گیرد یک نشانگر خاص را نمایش می‌دهد.



شکل ۵-۱۰ TddgDesignSpinner صفت Value را قادر می‌سازد که با کلیک کردن بر روی دکمه‌های اجزاء در زمان طراحی تغییر نماید.

همانگونه که قبلاً بیان کردیم Linux از حلقه در پیغام‌ها استفاده نمی‌کند و بنابراین CLX بایستی ازمکانیزمی متفاوت برای پیاده‌سازی این جنبه‌ها استفاده کند.

اول از همه، مشخص کردن نشانگر ماوس به منظور این که در کنترل به کار گرفته شود کاری پیچیده‌تر است. در گونه VCL ، یک فایل منبع Windows که حاوی نشانگر پیکربندی است را الصاق نموده و سپس تابع API ، LoadCursor را فراخوانی می‌کنیم تا مرجعی به نشانگر حاصل کنیم (یک دستگیره).

لیست ۲-۱۰ QddgDsnSpin.pas-کد منبع مربوط به جزء TddgDesignSpinner

```
unit QddgDsnSpn;

interface

uses
  SysUtils, Classes, Qt, QddgSpin;
  (*
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  ddgSpin;
  *)
```

لیست ۲-۱۰ ادامه

```
type
  TddgDesignSpinner = class( TddgSpinner )
  private
    // VCL->CLX:  Custom cursor stored in QCursorH field
    FThumbCursor: QCursorH;

    (*
    // VCL->CLX:  Custom cursors and design-time interactions are
    //            handled differently under CLX. The following
    //            block is VCL-specific.

    FThumbCursor: HCursor;

    // Window Message Handling Method
    procedure WMSetCursor( var Msg : TWMSetCursor );
      message wm_SetCursor;

    // Component Message Handling Method
    procedure CMDesignHitTest( var Msg: TCMDesignHitTest );
      message cm_DesignHitTest;
    *)
  protected
    procedure Change; override;

    // VCL->CLX:  The following two methods are overridden for CLX
    procedure MouseMove( Shift: TShiftState;
                         X, Y: Integer ); override;
    function DesignEventQuery( Sender: QObjectH;
                               Event: QEventH ): Boolean; override;
  public
    constructor Create( AOwner: TComponent ); override;
    destructor Destroy; override;
  end;

implementation

(*
// VCL->CLX:  CLX does not support cursor resources
{$R DdgDsnSpn.res}                  // Link in custom cursor resource
*)

uses
  Types, QControls, QForms;              // VCL->CLX:  Add CLX units
```

لیست ۲-۱۰ ادامه

```
// VCL->CLX:  Two arrays of bytes (one for the image and one for
//            the mask) are used to represent custom cursors in CLX

const
  Bits: array[0..32*4-1] of Byte = (
    $00, $30, $00, $00, $00, $48, $00, $00,
    $00, $48, $00, $00, $00, $48, $00, $00,
    $00, $48, $00, $00, $00, $4E, $00, $00,
    $00, $49, $C0, $00, $00, $49, $30, $00,
    $00, $49, $28, $00, $03, $49, $24, $00,
    $04, $C0, $24, $00, $04, $40, $04, $00,
    $02, $40, $04, $00, $02, $00, $04, $00,
    $01, $00, $04, $00, $01, $00, $04, $00,
    $00, $80, $08, $00, $00, $40, $08, $00,
    $00, $40, $08, $00, $00, $20, $10, $00,
    $00, $20, $10, $00, $00, $7F, $F8, $00,
    $00, $7F, $F8, $00, $00, $7F, $E8, $00,
    $00, $7F, $F8, $00, $00, $00, $00, $00,
    $00, $00, $00, $00, $00, $00, $00, $00,
    $00, $00, $00, $00, $00, $00, $00, $00,
    $00, $00, $00, $00, $00, $00, $00, $00 );

  Mask: array[0..32*4-1] of Byte = (
    $00, $30, $00, $00, $00, $78, $00, $00,
    $00, $78, $00, $00, $00, $78, $00, $00,
    $00, $78, $00, $00, $00, $7E, $00, $00,
    $00, $7F, $C0, $00, $00, $7F, $F0, $00,
    $00, $7F, $F8, $00, $03, $7F, $FC, $00,
    $07, $FF, $FC, $00, $07, $FF, $FC, $00,
    $03, $FF, $FC, $00, $03, $FF, $FC, $00,
    $01, $FF, $FC, $00, $01, $FF, $FC, $00,
    $00, $FF, $F8, $00, $00, $7F, $F8, $00,
    $00, $7F, $F8, $00, $00, $3F, $F0, $00,
    $00, $3F, $F0, $00, $00, $7F, $F8, $00,
    $00, $7F, $F8, $00, $00, $7F, $E8, $00,
    $00, $7F, $F8, $00, $00, $00, $00, $00,
    $00, $00, $00, $00, $00, $00, $00, $00,
    $00, $00, $00, $00, $00, $00, $00, $00,
    $00, $00, $00, $00, $00, $00, $00, $00 );


{================================}
{== TddgDesignSpinner Methods ==}
{================================}
```

لیست ۲-۱۰ ادامه

```
constructor TddgDesignSpinner.Create( AOwner: TComponent );
var
  BitsBitmap: QBitmapH;
  MaskBitmap: QBitmapH;
begin
  inherited Create( AOwner );

  (*
  // VCL->CLX:  No LoadCursor in CLX
  FThumbCursor := LoadCursor( HInstance, 'DdgDSNSPN_BTNCURSOR' );
  *)

  // VCL->CLX:  Byte arrays are used to create a custom cursor
  BitsBitmap := QBitmap_create( 32, 32, @Bits, False );
  MaskBitmap := QBitmap_create( 32, 32, @Mask, False );
  try
    FThumbCursor := QCursor_create( BitsBitmap, MaskBitmap, 8, 0 );
  finally
    QBitmap_destroy( BitsBitmap );
    QBitmap_destroy( MaskBitmap );
  end;
end;


destructor TddgDesignSpinner.Destroy;
begin
  (*
  VCL->CLX:  In CLX, use QCursor_Destroy instead of DestroyCursor
  DestroyCursor( FThumbCursor );       // Release GDI cursor object
  *)
  QCursor_Destroy( FThumbCursor );
  inherited Destroy;
end;


// If the mouse is over one of the buttons, then change cursor to
// the custom cursor that resides in the DdgDsnSpn.res
// resource file

(*
// VCL->CLX:  There is no wm_SetCursor in CLX
procedure TddgDesignSpinner.WMSetCursor( var Msg: TWMSetCursor );
begin
  if MouseOverButton( btMinus ) or MouseOverButton( btPlus ) then
    SetCursor( FThumbCursor )
```

لیست ۲-۱۰ ادامه

```
  else
    inherited;
end;
*)

// VCL->CLX:  Override MouseMove to handle displaying custom cursor

procedure TddgDesignSpinner.MouseMove( Shift: TShiftState;
                                       X, Y: Integer );
begin
  if MouseOverButton( btMinus ) or MouseOverButton( btPlus ) then
    QWidget_setCursor( Handle, FThumbCursor )
  else
    QWidget_UnsetCursor( Handle );
  inherited;
end;


(*
// VCL->CLX:  cm_DesignHitTest does not exist in CLX.  Instead,
//            override the DesignEventQuery method (see below).

procedure TddgDesignSpinner.CMDesignHitTest( var Msg:
                                             TCMDesignHitTest );
begin
  // Handling this component message allows the Value of the
  // spinner to be changed at design-time using the left mouse
  // button.  If the mouse is positioned over one of the buttons,
  // then set the Msg.Result value to 1. This instructs Delphi to
  // allow mouse events to "get through to" the component.

  if MouseOverButton( btMinus ) or MouseOverButton( btPlus ) then
    Msg.Result := 1
  else
    Msg.Result := 0;
end;
*)


function TddgDesignSpinner.DesignEventQuery( Sender: QObjectH;
                                             Event: QEventH ): Boolean;

var
  MousePos: TPoint;
begin
  Result := False;
```

لیست ۲-۱۰ ادامه

```
  if ( Sender = Handle ) and
     ( QEvent_type(Event) in [QEventType_MouseButtonPress,
                              QEventType_MouseButtonRelease,
                              QEventType_MouseButtonDblClick]) then
  begin
    // Note: extracting MousePos is not actually needed in this
    //       example, but if you need to get the position of the
    //       mouse, this is how you do it.

    MousePos := Point( QMouseEvent_x( QMouseEventH( Event ) ),
                       QMouseEvent_y( QMouseEventH( Event ) ) );

    if MouseOverButton( btMinus ) or MouseOverButton( btPlus ) then
      Result := True
    else
      Result := False;
  end;
end;


procedure TddgDesignSpinner.Change;
var
  Form: TCustomForm;
begin
  inherited Change;

  // Force the Object Inspector to update what it shows for the
  // Value property of the spinner when changed via the mouse.

  if csDesigning in ComponentState then
  begin
    Form := GetParentForm( Self );

    (*
    // VCL->CLX:  Form.Designer replaced with DesignerHook in CLX
    if ( Form <> nil ) and ( Form.Designer <> nil ) then
      Form.Designer.Modified;
    *)

    if ( Form <> nil ) and ( Form.DesignerHook <> nil ) then
      Form.DesignerHook.Modified;
  end;
end;

end.
```

این دستگیره نشانگر سپس در به کارگیری پیغام WM_SetCursor مورد استفاده قرار می‌گیرد، Windows این پیغام را به همه کنترل‌هایی که نیاز به اشاره‌گر بهنگام‌سازی شده ماوس دارند ارسال می‌کند.

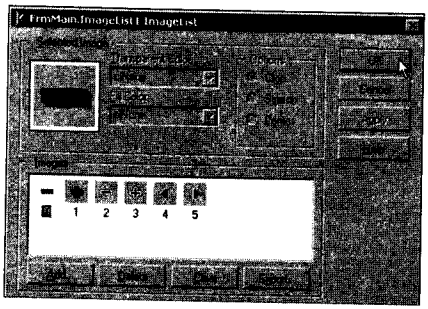
تحت CLX ، این روش را نمی‌توان به کار برد. اولاً Qt منابع نشانگر را پشتیبانی نمی‌کند. یونیت Qt.Pas چندین متد QCursor_Create() را پشتیبانی می‌کند که هر کدام روشی متفاوت برای ساخت یک نشانگر ماوس فراهم می‌کنند. می‌توانی با فرستادن یک مقدار عددی به متد QCursor_Create() یکی از نشانگرهای ذخیره ماوس را مورد استفاده قرار هید. ولی برای ایجاد یک نشانگر انتخابی، بایستی دو آرایه از نوع بایت را ایجاد کنید که حاوی طرح بیتی نشانگر باشد. اولین آرایه پیکسل‌های سیاه و سفید را مشخص می‌کند و در حالی که دومین آرایه پوشش را مشخص می‌کند که محل‌های شفاف درون نشانگر به کمک آن مشخص می‌شوند.

سپس، به منظور نمایش نشانگر ماوس در زمان مناسب، به جای به کارگیری پیغام WM_SetCursor متد ارسال رویداد MousMove() را باطل‌سازی می‌کنیم. برای تغییر نشانگر، هنگامی که ماوس روی یک دکمه قرار گیرد تابع QWidget_SetCursor() فراخوانی می‌شود. در غیر این صورت متد QWidget_UnsetCurtsor() فراخوانی می‌شود.

در VCL ، به کارگیری پیغام cm_DesignHitTest رویدادهای ماوس را قادر می‌سازد که در زمان طراحی توسط یک جزء به کار گرفته شوند. متأسفانه این پیغام در CLX وجود ندارد. در عوض، برای انجام کارهای مشابه بایستی متد جدید DesignEventQuery() را باطل‌سازی کنیم. این متد راهی برای نویسندگان اجزاء فراهم می‌کند تا هنگامی که Qt widget موردنظر در زمان طراحی ورودی دریافت کند از این موضوع مطلع شوند. اگر متد مقدار True را برگرداند کنترل بایستی به رویداد پاسخ دهد. در مثال مورد بحث فقط بر رویدادهای ورودی ماوس نظر داریم. بنابراین بایستی ابتدا مشخص کنیم که آیا رویداد ورودی محدودیت‌های ما را ارضاء می‌کند یا خیر. اگر چنین بود، بایستی مشخص کنیم که آیا ماوس روی یکی از دکمه‌ها قرار گرفته است یا خیر. متد Change() بایستی در TddgDesignSpinner باطل‌سازی شود به گونه‌ای که نمایش صفت Value در **Object Inspector** با جزء انتخاب شده، همگام باقی بماند. اگر این متد باطل‌سازی نشده است **Object Inspector** هنگامی که کاربر مستقیماً روی دکمه spinner در Form Designer کلیک نکند بهنگام‌سازی نخواهد شد. همانگونه که می‌توانید ببینید، تنها تغییر مراجعه Form.Designer به Form.DesignerHook است.

مرجع‌های اجزاء و لیست‌های تصویر

جزء بعدی یک بار دیگر کارائی spinner را توسعه می‌دهد. در حالت خاص، جزء TddgImgListSpinner از TddgDesingSpinner ارث برده و یک صفت مراجعه به جزء پیاده‌سازی می‌کند که کاربر را قادر می‌سازد تا spinner را به یک ImageList متصل سازد. در این صورت تصاویر درون ImageList را می‌توان در محل سمبل‌های پیش‌فرض جمع و تفریق همانند شکل ۶-۱۰ قرار داد.



شکل ۶-۱۰ TddgImglistSpinner نمایش تصاویر از یک ImageList را برای هر دکمه پشتیبانی می‌کند

لیست ۳-۱۰ کد یونیت QddgSpin.pas را نشان می‌دهد که جزء TddgImagListSpinner را پیاده‌سازی می‌نماید. برخلاف TddgDesignSpinner این جزء هنگام انتقال به CLX نیاز به تغییرات بسیار اندکی دارد.

طبق معمول بایستی بند uses یونیت تغییر کند تا یونیت‌های خاص CLX را دربرداشته باشد و یونیت‌های خاص VCL را حذف کند. در حالت خاص، توجه داشته باشید که یونیت QImgList یونیت ImagList را جایگزین می‌کند. این مسأله مهم است زیرا تحت VCL جزء TCustomImageList پوششی بر کنترل متداول ImageList است که در ComCtL32.dll پیاده‌سازی شده است. بورلند نگارش CLX ای از جزء TCustomImageList پدید آورده است که از عناوین گرافیکی Qt به جای ComCtl32.dll استفاده می‌کند.

لیست ۳-۱۰ QddgILspin.pas -کد منبع جزء سازندهٔ TddgImagListSpinner

```
unit QddgILSpin;

interface

uses
  Classes, Types, QddgSpin, QddgDsnSpn, QImgList;
  (*
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  ddgSpin, ddgDsnSpn, ImgList;
  *)

type
  TddgImgListSpinner = class( TddgDesignSpinner )
  private
    FImages: TCustomImageList;
    FImageIndexes: array[ 1..2 ] of Integer;
    FImageChangeLink: TChangeLink;
```

لیست ۳-۱۰ ادامه

```
  // Internal Event Handlers
  procedure ImageListChange( Sender: TObject );
protected
  procedure Notification( AComponent : TComponent;
                          Operation : TOperation ); override;

  procedure DrawButton( Button: TddgButtonType; Down: Boolean;
                        Bounds: TRect ); override;
  procedure CalcCenterOffsets( Bounds: TRect; var L, T: Integer);

  procedure CheckMinSize;

  // Property Access Methods
  procedure SetImages( Value: TCustomImageList ); virtual;
  function GetImageIndex( PropIndex: Integer ): Integer; virtual;
  procedure SetImageIndex( PropIndex: Integer;
                           Value: Integer ); virtual;
public
  constructor Create( AOwner: TComponent ); override;
  destructor Destroy; override;
published
  property Images: TCustomImageList
    read FImages
    write SetImages;

  property ImageIndexMinus: Integer
    index 1
    read GetImageIndex
    write SetImageIndex;

  property ImageIndexPlus: Integer
    index 2
    read GetImageIndex
    write SetImageIndex;
end;

implementation

uses
  QGraphics;                    // VCL->CLX:  Added for CLX support


{================================}
{== TddgImgListSpinner Methods ==}
{================================}

constructor TddgImgListSpinner.Create( AOwner: TComponent );
```

لیست ۳-۱۰ ادامه

```
begin
  inherited Create( AOwner );

  FImageChangeLink := TChangeLink.Create;
  FImageChangeLink.OnChange := ImageListChange;
  // NOTE: Since, the component user does not have direct access to
  // the change link, the user cannot assign custom event handlers.

  FImageIndexes[ 1 ] := -1;
  FImageIndexes[ 2 ] := -1;
end;


destructor TddgImgListSpinner.Destroy;
begin
  FImageChangeLink.Free;
  inherited Destroy;
end;


procedure TddgImgListSpinner.Notification( AComponent: TComponent;
                                           Operation: TOperation );
begin
  inherited Notification( AComponent, Operation );
  if ( Operation = opRemove ) and ( AComponent = FImages ) then
    SetImages( nil );              // Note the call to access method
end;


function TddgImgListSpinner.GetImageIndex( PropIndex:
                                           Integer ): Integer;
begin
  Result := FImageIndexes[ PropIndex ];
end;

procedure TddgImgListSpinner.SetImageIndex( PropIndex: Integer;
                                            Value: Integer );
begin
  if FImageIndexes[ PropIndex ] <> Value then
  begin
    FImageIndexes[ PropIndex ] := Value;
    Invalidate;
  end;
end;
```

لیست ۳-۱۰ ادامه

```
procedure TddgImgListSpinner.SetImages( Value: TCustomImageList );
begin
  if FImages <> nil then
    FImages.UnRegisterChanges( FImageChangeLink );

  FImages := Value;

  if FImages <> nil then
  begin
    FImages.RegisterChanges( FImageChangeLink );
    FImages.FreeNotification( Self );
    CheckMinSize;
  end;
  Invalidate;
end;

procedure TddgImgListSpinner.ImageListChange( Sender: TObject );
begin
  if Sender = Images then
  begin
    CheckMinSize;
    // Call Update instead of Invalidate to prevent flicker
    Update;
  end;
end;

procedure TddgImgListSpinner.CheckMinSize;
begin
  // Ensures button area will display entire image
  if FImages.Width > ButtonWidth then
    ButtonWidth := FImages.Width;
  if FImages.Height > Height then
    Height := FImages.Height;
end;


procedure TddgImgListSpinner.DrawButton( Button: TddgButtonType;
                                         Down: Boolean;
                                         Bounds: TRect );
var
  L, T: Integer;
begin
  with Canvas do
  begin
```

لیست ۳-۱۰ ادامه

```
Brush.Color := ButtonColor;
Pen.Color := clBtnShadow;
Rectangle( Bounds.Left, Bounds.Top,
           Bounds.Right, Bounds.Bottom );

if Button = btMinus then           // Draw the Minus (-) Button
begin
  if ( Images <> nil ) and ( ImageIndexMinus <> -1 ) then
  begin
    (*
    // VCL->CLX:  DrawingStyle does not exist in CLX TImageList
    //            BkColor is used instead.
    if Down then
      FImages.DrawingStyle := dsSelected
    else
      FImages.DrawingStyle := dsNormal;
    *)
    if Down then
      FImages.BkColor := clBtnShadow
    else
      FImages.BkColor := clBtnFace;

    CalcCenterOffsets( Bounds, L, T );

    (*
    // VCL->CLX:  TImageList.Draw is different in CLX
    FImages.Draw( Canvas, L, T, ImageIndexMinus, Enabled );
    *)
    FImages.Draw( Canvas, L, T, ImageIndexMinus, itImage,
                  Enabled );
  end
  else
    inherited DrawButton( Button, Down, Bounds );
end
else                                  // Draw the Plus (+) Button
begin
  if ( Images <> nil ) and ( ImageIndexPlus <> -1 ) then
  begin
    (*
    // VCL->CLX:  DrawingStyle does not exist in CLX TImageList
    //            BkColor is used instead.
    if Down then
      FImages.DrawingStyle := dsSelected
    else
```

لیست ۳-۱۰ ادامه

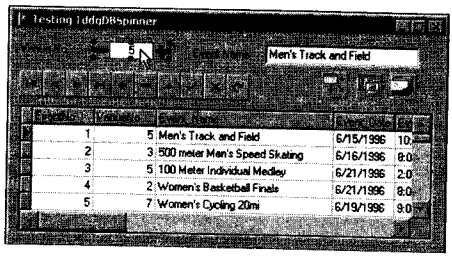
```
        FImages.DrawingStyle := dsNormal;
      *)
      if Down then
        FImages.BkColor := clBtnShadow
      else
        FImages.BkColor := clBtnFace;

      CalcCenterOffsets( Bounds, L, T );

      (*
      // VCL->CLX:  TImageList.Draw is different in CLX
      FImages.Draw( Canvas, L, T, ImageIndexPlus, Enabled );
      *)
      FImages.Draw( Canvas, L, T, ImageIndexPlus, itImage,
                    Enabled );
    end
    else
      inherited DrawButton( Button, Down, Bounds );
  end;
end;
end; {= TddgImgListSpinner.DrawButton =}


procedure TddgImgListSpinner.CalcCenterOffsets( Bounds: TRect;
                                                var L, T: Integer );
begin
  if FImages <> nil then
  begin
    L := Bounds.Left + ( Bounds.Right - Bounds.Left ) div 2 -
         ( FImages.Width div 2 );
    T := Bounds.Top + ( Bounds.Bottom - Bounds.Top ) div 2 -
         ( FImages.Height div 2 );
  end;
end;

end.
```

منفعت این کار در اعلان کلاس مشهود است. اعلان گونه CLX مربوط به TddgImagListSpinner مشابه با گونه VCL است. علاوه بر این، پیاده‌سازی تمامی بخش‌ها به جز متدهای اجزاء مشابه یکدیگر است. البته، این متد نمایش متدی ساده است، متد باطل‌سازی شده DrawButton()، که نیاز به کمی دستکاری دارد. در حالت خاص، دو بحث بایستی مطرح شود. اولین مورد، نکته‌ای کلیدی را در مقایسه کلاس‌هایی که در VCL و CLX وجود دارند مشخص می‌کند. یعنی، فقط به این دلیل که یک کلاس VCL دارای کلاسی متناظر در CLX است لزوماً این مفهوم حاصل نمی‌شود که همه کارائی کلاس VCL

درگونه CLX آن نیز قابل دسترسی است.
درکلاس TCustomImageList،گونه VCL آن صفت DrawingStyle را پیاده‌سازی می‌کندکه توسط گونه TddgImagListSpinnerVCL به منظور نمایش متفاوت تصویر دکمه هنگام کلیک کردن به کار می‌رود. صفت DrawingStyle درگونه CLX وجود ندارد و بنابراین روشی متفاوت بایستی در نظر گرفته شود.
دومین تغییر در متد DrawButton() از متد TCustomImageList.Draw() حاصل می‌شود که در دو معماری تفاوت دارد.

اجزاء CLX مراقب داده

در چهارمین جزء نمونه، مراقبت داده به جزء spinner اضافه شده است. یعنی جزء TddgDBSpinner را می‌توان به کمک صفات DataSource و DataFiled به یک فیلد عددی متصل نمود. شکل ۷-۱۰ یک جزء TddgDBSpinner که به فیلد VenueNo از مجموعه داده‌ای Events متصل شده است را نشان می‌دهد.
لیست ۴-۱۰ کد منبع یونیت QddgDBSpin.Pas را نشان می‌دهد که جزء TddgDBSpinner که از TddgImagListSpinner ارث می‌برد را پیاده‌سازی می‌نماید.
خوشبختانه قرار دادن امکان مراقبت داده در یک جزء CLX بسیار شبیه با پیاده‌سای VCL است. یعنی، هنگامی که یک جزء کاری CLX دارید که امکان مراقبت داده‌ای را ندارد نیاز به یک شیء TFieldDataLind در جزء CLX خود و پاسخ به رویدادهای DataChange و UpdateData دارید. البته، بایستی صفات DataSource ، DataField و ReadOnly را پیاده‌سازی کنید ولی این کار تفاوتی با انجام عمل مشابه در یک جزء VCL ندارد.



شکل ۷-۱۰ TddgDBSpinner را می‌توان به منظور نمایش و ویرایش فیلدهای عددی در یک مجموعه داده‌ای به کار برد

تذکر

فراموش نکنید که بایستی یونیت DBCtrls را به QDBCtrls تغییر دهید. علیرغم این که یونیت DB میان VCL و CLX اشتراکی است، یونیت DBCtrls چنین نیست. هر دو DBCtrls و QDBCtrls یک کلاس TFieldDataLink را تعریف می‌کنند. متأسفانه در دلفی ۷ در صورتی که از گونه TFieldDataLink به جای گونه CLX آن استفاده کنید هیچ پیغام خطائی مشاهده نخواهید کرد. در واقع این جزء حتی ممکن است تحت Windows به درستی کار کند. هنگامی که سعی در اجرای جزء سازنده در kylix کنید، خطاهای نحوی بسیاری از کامپایلر دریافت خواهید کرد.

لیست ۴-۱۰ QddgDBSpinner -کد منبع جزء TddgDBSpinner

```
unit QddgDBSpin;

interface

uses
  SysUtils, Classes, Qt, QddgILSpin, DB, QDBCtrls;
  (*
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  ddgILSpin, DB, DBCtrls;
  *)

type
  TddgDBSpinner = class( TddgImgListSpinner )
  private
    FDataLink: TFieldDataLink;          // Provides Access to Data

    // Internal Event Handlers for DataLink Events
    procedure DataChange( Sender: TObject );
    procedure UpdateData( Sender: TObject );
    procedure ActiveChange( Sender: TObject );

    (*
    // VCL->CLX:  Component Message handling methods not in CLX
    procedure CMExit( var Msg: TCMExit ); message cm_Exit;
    procedure CMDesignHitTest( var Msg: TCMDesignHitTest );
      message cm_DesignHitTest;
    *)
  protected
    procedure Notification( AComponent : TComponent;
                            Operation : TOperation ); override;
    procedure CheckFieldType( const Value: string ); virtual;

    // Overridden event dispatch methods
    procedure Change; override;
    procedure KeyPress( var Key : Char ); override;

    // VCL->CLX:  DoExit replaces CMExit
    procedure DoExit; override;
    // VCL->CLX:  DesignEventQuery replaces CMDesignHitTest
    function DesignEventQuery( Sender: QObjectH;
                               Event: QEventH ): Boolean; override;
```

لیست ۴-۱۰ ادامه

```
    // Overridden support methods
    procedure DecValue( Amount: Integer ); override;
    procedure IncValue( Amount: Integer ); override;

    // Property Access Methods
    function GetField: TField; virtual;
    function GetDataField: string; virtual;
    procedure SetDataField( const Value: string ); virtual;
    function GetDataSource: TDataSource; virtual;
    procedure SetDataSource( Value: TDataSource ); virtual;
    function GetReadOnly: Boolean; virtual;
    procedure SetReadOnly( Value: Boolean ); virtual;

    // Give Descendants Access to Field object and DataLink
    property Field: TField
      read GetField;

    property DataLink: TFieldDataLink
      read FDataLink;
  public
    constructor Create( AOwner: TComponent ); override;
    destructor Destroy; override;
  published
    property DataField: string
      read GetDataField
      write SetDataField;

    property DataSource: TDataSource
      read GetDataSource
      write SetDataSource;

    // This property controls the ReadOnly state of the DataLink
    property ReadOnly: Boolean
      read GetReadOnly
      write SetReadOnly
      default False;
  end;

type
  EInvalidFieldType = class( Exception );

resourcestring
  SInvalidFieldType = 'DataField can only be connected to ' +
                      'columns of type Integer, Smallint, Word, ' +
                      'and Float';
```

لیست ۴-۱۰ ادامه

```
implementation

uses
  Types;                          // VCL->CLX:  Added for CLS support


{============================}
{== TddgDBSpinner Methods ==}
{============================}

constructor TddgDBSpinner.Create( AOwner: TComponent );
begin
  inherited Create( AOwner );

  FDataLink := TFieldDataLink.Create;

  // To support the TField.FocusControl method, set the
  // FDataLink.Control property to point to the spinner.
  // The Control property requires a TWinControl component.
  FDataLink.Control := Self;

  // Assign Event Handlers
  FDataLink.OnDataChange := DataChange;
  FDataLink.OnUpdateData := UpdateData;
  FDataLink.OnActiveChange := ActiveChange;

  // NOTE: Since, the component user does not have direct access to
  // the data link, the user cannot assign custom event handlers.
end;


destructor TddgDBSpinner.Destroy;
begin
  FDataLink.Free;
  FDataLink := nil;
  inherited Destroy;
end;


procedure TddgDBSpinner.Notification( AComponent: TComponent;
                                      Operation: TOperation );

begin
  inherited Notification( AComponent, Operation );
  if ( Operation = opRemove ) and
     ( FDataLink <> nil ) and
```

لیست ۴-۱۰ ادامه

```
    ( AComponent = FDataLink.DataSource ) then
  begin
    DataSource := nil;            // Indirectly calls SetDataSource
  end;
end;


function TddgDBSpinner.GetField: TField;
begin
  Result := FDataLink.Field;
end;


function TddgDBSpinner.GetDataField: string;
begin
  Result := FDataLink.FieldName;
end;

procedure TddgDBSpinner.SetDataField( const Value: string );
begin
  CheckFieldType( Value );
  FDataLink.FieldName := Value;
end;


function TddgDBSpinner.GetDataSource: TDataSource;
begin
  Result := FDataLink.DataSource;
end;

procedure TddgDBSpinner.SetDataSource( Value: TDataSource );
begin
  if FDatalink.DataSource <> Value then
  begin
    FDataLink.DataSource := Value;

    // FreeNotification must be called b/c DataSource may be
    // located on another form or data module.
    if Value <> nil then
      Value.FreeNotification( Self );
  end;
end;
```

لیست ۴-۱۰ ادامه

```
function TddgDBSpinner.GetReadOnly: Boolean;
begin
  Result := FDataLink.ReadOnly;
end;

procedure TddgDBSpinner.SetReadOnly( Value: Boolean );
begin
  FDataLink.ReadOnly := Value;
end;



procedure TddgDBSpinner.CheckFieldType( const Value: string );
var
  FieldType: TFieldType;
begin
  // Make sure the field type corresponding to the column
  // referenced by Value is either ftInteger, ftSmallInt, ftWord,
  // or ftFloat.  If it is not, an EInvalidFieldType exception is
  // raised.

  if ( Value <> '' ) and
     ( FDataLink <> nil ) and
     ( FDataLink.Dataset <> nil ) and
     ( FDataLink.Dataset.Active ) then
  begin
    FieldType := FDataLink.Dataset.FieldByName( Value ).DataType;
    if ( FieldType <> ftInteger ) and
       ( FieldType <> ftSmallInt ) and
       ( FieldType <> ftWord ) and
       ( FieldType <> ftFloat ) then
    begin
      raise EInvalidFieldType.Create( SInvalidFieldType );
    end;
  end;
end;


procedure TddgDBSpinner.Change;
begin
  // Tell the FDataLink that the data has changed
  if FDataLink <> nil then
    FDataLink.Modified;
  inherited Change;                    // Generates OnChange event
end;
```

لیست ۴-۱۰ ادامه

```
procedure TddgDBSpinner.KeyPress( var Key: Char );
begin
  inherited KeyPress( Key );

  if Key = #27 then
  begin
    FDataLink.Reset;                              // Esc key pressed
    Key := #0;                 // Set to #0 so Esc won't close dialog
  end;
end;


procedure TddgDBSpinner.DecValue( Amount: Integer );
begin
  if ReadOnly or not FDataLink.CanModify then
  begin
    // Prevent change if FDataLink is ReadOnly
    (*
    // VCL->CLX:  MessageBeep is a Windows API function
    MessageBeep( 0 )
    *)
    Beep;
  end
  else
  begin
    // Try to put Dataset in edit mode--only dec if in edit mode
    if FDataLink.Edit then
      inherited DecValue( Amount );
  end;
end;


procedure TddgDBSpinner.IncValue( Amount: Integer );
begin
  if ReadOnly or not FDataLink.CanModify then
  begin
    // Prevent change if FDataLink is ReadOnly
    (*
    // VCL->CLX:  MessageBeep is a Windows API function
    MessageBeep( 0 )
    *)
    Beep;
  end
  else
```

لیست ۴-۱۰ ادامه

```
  begin
    // Try to put Dataset in edit mode--only inc if in edit mode
    if FDataLink.Edit then
      inherited IncValue( Amount );
  end;
end;


{=============================================================================
  TddgDBSpinner.DataChange

  This method gets called as a result of a number of
  different events:

  1. The underlying field value changes.  Occurs when changing the
     value of the column tied to this control and then move to a
     new column or a new record.
  2. The corresponding Dataset goes into Edit mode.
  3. The corresponding Dataset referenced by DataSource changes.
  4. The current cursor is scrolled to a new record in the table.
  5. The record is reset through a Cancel call.
  6. The DataField property changes to reference another column.
=============================================================================}

procedure TddgDBSpinner.DataChange( Sender: TObject );
begin
  if FDataLink.Field <> nil then
    Value := FDataLink.Field.AsInteger;
end;


{=============================================================================
  TddgDBSpinner.UpdateData

  This method gets called when the corresponding field value and
  the contents of the Spinner need to be synchronized.  Note that
  this method only gets called if this control was responsible for
  altering the data.
=============================================================================}

procedure TddgDBSpinner.UpdateData( Sender: TObject );
begin
  FDataLink.Field.AsInteger := Value;
end;
```

لیست ۴-۱۰ ادامه

```
{=====================================================================
  TddgDBSpinner.ActiveChange

  This method gets called whenever the Active property of the
  attached Dataset changes.

  NOTE: You can use the FDataLink.Active property to determine
        the *new* state of the Dataset.
=====================================================================}

procedure TddgDBSpinner.ActiveChange( Sender: TObject );
begin
  // If the Dataset is becoming Active, then check to make sure the
  // field type of the DataField property is a valid type.

  if ( FDataLink <> nil ) and FDataLink.Active then
    CheckFieldType( DataField );
end;


(*
// VCL->CLX:  CMExit replaced with DoExit (see below)

procedure TddgDBSpinner.CMExit( var Msg: TCMExit );
begin
  try   // Attempt to update the record if focus leaves the spinner
    FDataLink.UpdateRecord;
  except
    SetFocus;       // Keep the focus on the control if Update fails
    raise;                                    // Reraise the exception
  end;
  inherited;
end;
*)

procedure TddgDBSpinner.DoExit;
begin
  try   // Attempt to update the record if focus leaves the spinner
    FDataLink.UpdateRecord;
  except
    SetFocus;       // Keep the focus on the control if Update fails
    raise;                                    // Reraise the exception
  end;
  inherited;
end;
```

لیست ۴-۱۰ ادامه

```
(*
// VCL->CLX:  CMDesignHitTest replaced by DesignEventQuery

procedure TddgDBSpinner.CMDesignHitTest(var Msg: TCMDesignHitTest);
begin
  // Ancestor component allows Value to be changed at design-time.
  // This is not valid in a data-aware component because it would
  // put the connected dataset into edit mode.
  Msg.Result := 0;
end;
*)

function TddgDBSpinner.DesignEventQuery( Sender: QObjectH;
                                         Event: QEventH ): Boolean;

begin
  // Ancestor component allows Value to be changed at design-time.
  // This is not valid in a data-aware component because it would
  // put the connected dataset into edit mode.
  Result := False;
end;

end.
```

البته یک حالت نیاز به توجه بیشتری از طرف شما دارد. بسیاری از اجزاء VCL که امکان مراقبت داده‌ای را پشتیبانی می‌کنند به منظور فراخوانی متد UpdateRecord از پیوند داده‌ای پیغام جزء سازنده cm–Exit را به کار می‌گیرند. CLX پیغام cm–Exit را پیاده‌سازی نمی‌کند و بنابراین در عوض متد ارسال رویداد ()DoExit بایستی باطل‌سازی شود.

TddgDBSpinner یک وارث مستقیم از TddgImagListSpinner است که از TddgDesignSpinner ارث می‌برد. به خاطر بیاورید که یکی از جنبه‌های TddgDesignSpinner این بود که در زمان طراحی، با کمک ماوس امکان تغییر مقدار spinner را به کاربر می‌داد. این جنبهٔ دیگر در جزء مراقب داده مفید نیست زیرا اگر کاربر مقدار spinner را تغییر دهد مجموعه داده‌ای متناظر آن در حالت ویرایش قرار خواهد گرفت. متأسفانه در زمان طراحی در صورتی که چنین اتفاقی بیافتد راهی برای خروج از حالت ویرایش وجود ندارد. بنابراین TddgDBSpinner متد DesignEventQuery را باطل‌سازی کرده و برای جلوگیری از این که عملیات ماوس در زمان طراحی توسط جزء به کار گرفته شوند مقدار False را برمی‌گرداند.

ویرایشگرهای طراحی CLX

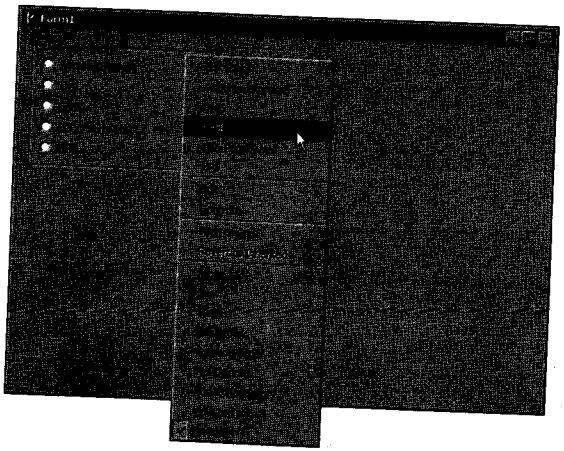
ویرایشگرهای طراحی اجزاء CLX دقیقاً به روشی مشابه با اجزاء VCL پیاده‌سازی شده‌اند. البته در این پیاده‌سازی تغییراتی پدید آمده است که شما بایستی از آن مطلع باشید. مهمترین نکته آن است که

یونیت‌هایی که کارائی زمان طراحی را پیاده‌سازی می‌کنند تفکیک شده و به یونیت‌های جدیدی تقسیم شده‌اند. بالاخص یونیت DsgnIntf به DesignIntf تغییر نام داده است. در اکثر حالات نیاز خواهید داشت که یونیت DesignEditors را به بند uses خود اضافه کنید. یونیت DesignIntf رابط به کار رفته توسط Form Designer و Object Inspector را تعریف می‌کند. یونیت DesignEditors کلاس‌های ویرایشگر صفت پایه و ویرایشگر جزء را پیاده‌سازی می‌کند.

متأسفانه تمام جنبه‌های زمان طراحی VCL به CLX منتقل نشده‌اند. به عنوان مثال، ویرایشگرهای صفت owner–draw فقط در VCL قابل دسترسی هستند. به عنوان نتیجه، ویرایشگرهای خاص CLX در یونیت CLXEditors پیاده‌سازی شده‌اند در حالی که ویرایشگرهای خاص VCL در یونیت VCLEditors تعریف شده‌اند.

در شکل ۸-۱۰، TddgRadioGroupEditor ، یک ویرایشگر صفت پیکربندی برای جزء TRadicGroup دیده می‌شود که یک کاربر را قادر می‌سازد که به سادگی صفت ItemIndex را تنظیم نماید. TddgRadioGroupEditor در یونیت QddgRgpEdt.Pas در لیست ۵-۱۰ تعریف شده است. تفکیک‌هایی که در TddgRadioGroupEditor شرح داده شدند در هر دو VCL و CLX به کار گرفته می‌شوند. در این مثال منوی متنی جزء TRadicGroup به منظور انعکاس آیتم‌هایی که در حال حاضر در گروه قرار دارند تغییر نموده است. انتخاب آیتم منوی متناظر یک آیتم گروه باعث می‌شود که صفت ItemIndex گروه به صورت مناسب تنظیم شود. اگر آیتمی در گروه نباشد فقط آیتم منوی Edit Items اضافه می‌شود.

اگر کاربر این آیتم منو را انتخاب کند ویرایشگر لیست رشته در صفت Items مربوط به TRadioGroup به کار گرفته می‌شود. متد EditPropertyByName() بخشی از CLX یا VCL نیست، این متد در کلاس TddgDefaultEditor تعریف شده است. این متد را می‌توان به منظور به



شکل ۸-۱۰ انتخاب یک CLX

لیست ۵-۱۰ QddgRgpEdt.pas- کد منبع برای ویرایشگر صفت TddgRadioGroupEditor

```
unit QddgRgpEdt;

interface

uses
  DesignIntf, DesignEditors, QExtCtrls, QDdgDsnEdt;

type
  TddgRadioGroupEditor = class( TddgDefaultEditor )
  protected
    function RadioGroup: TRadioGroup; virtual;
  public
    function GetVerbCount: Integer; override;
    function GetVerb( Index: Integer ) : string; override;
    procedure ExecuteVerb( Index: Integer ); override;
  end;


implementation

uses
  QControls;
{===================================}
{== TddgRadioGroupEditor Methods ==}
{===================================}

function TddgRadioGroupEditor.RadioGroup: TRadioGroup;
begin
  // Helper function to provide quick access to component being
  // edited.  Also makes sure Component is a TRadioGroup
  Result := Component as TRadioGroup;
end;

function TddgRadioGroupEditor.GetVerbCount: Integer;
begin
  // Return the number of new menu items to display
  Result := RadioGroup.Items.Count + 1;
end;


function TddgRadioGroupEditor.GetVerb( Index: Integer ): string;
begin
  // Menu item caption for context menu
  if Index = 0 then
    Result := 'Edit Items...
  else
    Result := RadioGroup.Items[ Index - 1 ];
end;
```

لیست ۵-۱۰ ادامه

```
procedure TddgRadioGroupEditor.ExecuteVerb( Index: Integer );
begin
  if Index = 0 then
    EditPropertyByName( 'Items' )       // Defined in QDdgDsnEdt.pas
  else
  begin
    if RadioGroup.ItemIndex <> Index - 1 then
      RadioGroup.ItemIndex := Index - 1
    else
      RadioGroup.ItemIndex := -1;              // Uncheck all items
    Designer.Modified;
  end;
end;

end.
```

کارگیری ویرایشگر صفتی که در حال حاضر برای صفات نامگذاری شد، مؤلفه‌ای درون متن ویرایشگر صفت ثبت شده است به کار گرفت. لیست ۶-۱۰ کد منبع یونیت QddgDsnEdt.pas را نشان می‌دهد که کلاس TddgDefaultEditor توسط آن پیاده‌سازی می‌شود.

لیست ۶-۱۰ QddgDsnEdt.pas -کد منبع ویرایشگر صفت TddgDefaultEditor

```
unit QddgDsnEdt;

interface

uses
  Classes, DesignIntf, DesignEditors;

type
  TddgDefaultEditor = class( TDefaultEditor )
  private
    FPropName: string;
    FContinue: Boolean;
    FPropEditor: IProperty;
    procedure EnumPropertyEditors(const PropertyEditor: IProperty);
    procedure TestPropertyEditor( const PropertyEditor: IProperty;
                                  var Continue: Boolean );
  protected
    procedure EditPropertyByName( const APropName: string );
  end;


implementation
```

لیست ۶-۱۰ ادامه

```
uses
  SysUtils, TypInfo;

{================================}
{== TddgDefaultEditor Methods ==}
{================================}
procedure TddgDefaultEditor.EnumPropertyEditors( const
                                      PropertyEditor: IProperty );
begin
  if FContinue then
    TestPropertyEditor( PropertyEditor, FContinue );
end;


procedure TddgDefaultEditor.TestPropertyEditor( const
                                        PropertyEditor: IProperty;
                                        var Continue: Boolean );
begin
  if not Assigned( FPropEditor ) and
     ( CompareText( PropertyEditor.GetName, FPropName ) = 0 ) then
  begin
    Continue := False;
    FPropEditor := PropertyEditor;
  end;
end;


procedure TddgDefaultEditor.EditPropertyByName( const
                                                APropName: string );
var
  Components: IDesignerSelections;
begin
  Components := TDesignerSelections.Create;
  FContinue := True;
  FPropName := APropName;
  Components.Add( Component );
  FPropEditor := nil;
  try
    GetComponentProperties( Components, tkAny, Designer,
                            EnumPropertyEditors );
    if Assigned( FPropEditor ) then
      FPropEditor.Edit;
  finally
    FPropEditor := nil;
  end;
end;

end.
```

بسته‌ها

اجزاء CLX نظیر اجزاء VCL بایستی به منظور نصب در IDE های kylix یا Delphi در یک بسته قرار گیرند. البته بایستی توجه داشت که یک بسته کامپایل شده Delphi 7 که حاوی یک جزء CLX است را نمی‌توان در kylix نصب کرد. دلیل این امر آن است که بسته‌های تحت ویندوز را می‌توان به عنوان DLL هایی نصب کرد که به صورت ویژه کامپایل شده‌اند در حالی که بسته‌های تحت Linux به عنوان فایل‌های اشیاء اشتراکی (so.) پیاده‌سازی شده‌اند. خوشبختانه قالب و ساختار فایل‌های منبع بسته‌ها تحت هر دو سیستم مشابه یکدیگرند.

اطلاعاتی که برای فراهم کردن بسته‌ها لازم دارید در Windows و Linux تفاوت دارند. به عنوان مثال بند requires بسته زمان طراحی Linux به طور خاص بسته‌های baseclx و visualclx را مشخص می‌کند. البته baseclx در دلفی ۷ موجود نیست. تحت Windows یک بسته زمان اجرای حاوی اجزاء CLX فقط نیاز به بسته Visualclx خواهد داشت. البته همانند بسته‌های VCL، بسته‌های طراحی CLX فقط نیاز به بسته‌های زمان اجرای حاوی اجزاء پیکربندی جدید شما خواهند داشت.

قواعد نامگذاری

اجزاء CLX که در این فصل ارائه شده‌اند در بسته‌هایی که در جداول ۱-۱۰ و ۲-۱۰ شرح داده‌ایم قرار دارد. جدول ۱-۱۰ فایل‌های BPL تولید شده تحت windows را نشان داده و بسته‌های مورد نیاز هر بسته پیکربندی را لیست می‌کند. جدول ۲-۱۰ فایل‌های شیء اشتراکی تولید شده در Linux را نشان داده و بسته‌های مورد نیاز را لیست می‌نماید. فایل‌های منبع بسته در دو جدول مشابه یکدیگرند. همانطور که می‌توانید ببینید برای نام بسته‌ها قواعد نامگذاری خاصی را اتخاذ کرده‌ایم.

جدول ۱-۱۰ بسته‌های CLX مربوط به (Delphi 7)

Package Source	*Compiled Version*	*Requires*
`QddgSamples.dpk`	`QddgSamples60.bpl`	`visualclx`

`QddgSamples_Dsgn.dpk`	`QddgSamples_Dsgn60.bpl`	`visualclx`
		`designide`
		`QddgSamples`

`QddgDBSamples.dpk`	`QddgDBSamples60.bpl`	`visualclx`
		`dbrtl`
		`visualdbclx`
		`QddgSamples`

`QddgDBSamples_Dsgn.dpk`	`QddgDBSamples_Dsgn60.bpl`	`visualclx`
		`QddgSamples_Dsgn`
		`QddgDBSamples`

جدول ۲-۱۰ بسته‌های CLX مربوط به Linux (Kylix)

Package Source	*Compiled Version*	*Requires*
QddgSamples.dpk	bplQddg6Samples.so.6	baseclx visualclx
QddgSamples_Dsgn.dpk	bplQddgSamples_Dsgn.so.6	baseclx visualclx designide QddgSamples
QddgDBSamples.dpk	bplQddgDBSamples.so.6	baseclx visualclx visualdbclx dataclx QddgSamples
QddgDBSamples_Dsgn.dpk	bplQddgDBSamples_Dsgn.so.6	baseclx visualclx QddgSamples_Dsgn QddgDBSamples

بسته‌های CLX که تحت Windows به کار می‌روند نوعاً گونه‌های محصول را در نام خود قید می‌کنند. به عنوان مثال، QddgSamples60.bpl مشخص می‌کند که این فایل مربوط به Delphi 7 بوده و همانگونه که از توسعه bpl. آن مشخص می‌شود یک کتابخانه بسته Borland است. تحت سیستم عامل Linux، بورلند روش سنتی نامگذاری اشیاء اشتراکی را برگزیده است. به عنوان مثال به جای استفاده از توسعه برای مشخص کردن نوع فایل، یک پیشوند bpl برای مشخص کردن بسته به کار می‌رود. بورلند برخی از بسته‌های طراحی با پیشوند Dsgn– بهتر شناخته می‌شوند. به علاوه تمامی بسته‌ها (زمان اجراء و طراحی) تحت Windows از یک توسعه bpl. استفاده می‌کند. با استفاده از یک پیشوند bpl برای تمامی بسته‌های تحت Linux سطح خوبی از سازگاری برقرار می‌شود. پسوند به کار رفته برای یک شیء اشتراکی Linux نوعاً so. است که در ادامه آن شماره نگارش می‌آید. پیشوندها و پسوندهای به کار رفته در تولید یک بسته کامپایل شده به کمک گزینه‌های بسته کنترل می‌شوند.

همچنین توجه خواهید داشت که فایل‌های منبع بسته شماره نگارشی را مشخص نمی‌کنند. در گونه‌های قبلی Delphi متداول بود که پسوندی به نام بسته اضافه کنیم تا مشخص شود کدام نگارش VCL مورد نیاز بسته است. البته در Kylix و Delphi 7، بورلند گزینه‌های متعددی اضافه کرده است که هنگام کامپایل کردن بسته‌ها، نام‌ها را کنترل می‌نمایند.

بسته‌های زمان اجرا

لیست‌های ۷-۱۰ و ۸-۱۰ کد منبع بسته‌های زمان اجرای مراقب داده و نامراقب داده‌ای را نشان می‌دهد که حاوی اجزاء مشخص شده در این فصل هستند. توجه کنید که از سمبل‌های وضعیتی MSWINDOWS و LINUX استفاده شده است تا دستورات مناسب مشخص شده و بسته‌های مناسب در قسمت requires ضمیمه شوند. MSWINDOWS هنگام کمپایل تحت Delphi 7 تعریف می‌شود در حالی که LINUX هنگام کمپایل تحت Kylix تعریف می‌شود.

لیست ۷-۱۰ QddgSamples.dpk

```
package QddgSamples;

{$R *.res}
{$ALIGN 8}
{$ASSERTIONS ON}
{$BOOLEVAL OFF}
{$DEBUGINFO ON}
{$EXTENDEDSYNTAX ON}
{$IMPORTEDDATA ON}
{$IOCHECKS ON}
{$LOCALSYMBOLS ON}
{$LONGSTRINGS ON}
{$OPENSTRINGS ON}
{$OPTIMIZATION ON}
{$OVERFLOWCHECKS OFF}
{$RANGECHECKS OFF}
{$REFERENCEINFO OFF}
{$SAFEDIVIDE OFF}
{$STACKFRAMES OFF}
{$TYPEDADDRESS OFF}
{$VARSTRINGCHECKS ON}
{$WRITEABLECONST ON}
{$MINENUMSIZE 1}
{$IMAGEBASE $400000}
{$DESCRIPTION 'DDG: CLX Components'}
 b
{$IFDEF MSWINDOWS}
{$LIBSUFFIX '60'}
{$ENDIF}

{$IFDEF LINUX}
{$SOPREFIX 'bpl'}
{$SOVERSION '6'}
{$ENDIF}
{$RUNONLY}
{$IMPLICITBUILD OFF}
```

لیست ۷-۱۰ ادامه

```
requires
  {$IFDEF LINUX}
  baseclx,
  {$ENDIF}
  visualclx;

contains
  QddgSpin in 'QddgSpin.pas',
  QddgDsnSpn in 'QddgDsnSpn.pas',
  QddgILSpin in 'QddgILSpin.pas';

end.
```

تذکر

هنگام مشخص کردن بلوک‌های کد خاص یک سیستم کاری از بلوک‌های مجزای {$IDFEF}..{$ENDIF} برای هر سیستم استفاده کنید. معمولاً از ساختارهایی نظیر ساختار زیر اجتناب می‌کنید.

```
{$IFDEF MSWINDOWS}
// Windows specific code here
{$ELSE}
// Linux specific code here
{$ENDIF}
```

اگر بورلند قصد پشتیبانی سیستم دیگری را نیز داشته باشد ساختارهای قبلی باعث خواهند شد که کد خاص Linux هنگامی که سیستم کاری Windows نباشد به اجرا در آید.

لیست ۸-۱۰ QddgSamples.dpk

```
package QddgDBSamples;

{$R *.res}
{$ALIGN 8}
{$ASSERTIONS ON}
{$BOOLEVAL OFF}
{$DEBUGINFO ON}
{$EXTENDEDSYNTAX ON}
{$IMPORTEDDATA ON}
{$IOCHECKS ON}
{$LOCALSYMBOLS ON}
{$LONGSTRINGS ON}
{$OPENSTRINGS ON}
{$OPTIMIZATION ON}
{$OVERFLOWCHECKS OFF}
```

لیست ۸-۱۰ ادامه

```
{$RANGECHECKS OFF}
{$REFERENCEINFO OFF}
{$SAFEDIVIDE OFF}
{$STACKFRAMES OFF}
{$TYPEDADDRESS OFF}
{$VARSTRINGCHECKS ON}
{$WRITEABLECONST ON}
{$MINENUMSIZE 1}
{$IMAGEBASE $400000}
{$DESCRIPTION 'DDG: CLX Components (Data-Aware)'}

{$IFDEF MSWINDOWS}
{$LIBSUFFIX '60'}
{$ENDIF}

{$IFDEF LINUX}
{$SOPREFIX 'bpl'}
{$SOVERSION '6'}
{$ENDIF}

{$RUNONLY}
{$IMPLICITBUILD OFF}

requires
  {$IFDEF MSWINDOWS}
  dbrtl,
  {$ENDIF}

  {$IFDEF LINUX}
  baseclx,
  dataclx,
  {$ENDIF}

  visualclx,
  visualdbclx,
  QddgSamples;

contains
  QddgDBSpin in 'QddgDBSpin.pas';

end.
```

بسته‌های زمان طراحی

علیرغم این که می‌توانید جزء پیکربندی خود را در یک بسته ترکیبی زمان اجراء/ طراحی قرار دهید این روش مطلوب نیست. در واقع این روش فقط هنگامی که ویرایشگر طراحی در بسته خود نداشته باشید.

اگر چنین کاری انجام دهید بسته شما نیاز به بسته designide خواهد داشت که نمی‌توان آن را مجدداً توزیع نمود.

راه‌حل کار ایجاد بسته‌های طراحی مجزا است که ثبت اجزاء درون بسته‌های زمان اجرای شما را به کار می‌گیرند. لیست‌های ۹-۱۰ و ۱۰-۱۰ کد منبع بسته‌های طراحی مراقب داده و نامراقب داده را نشان می‌دهند. مجدداً به استفاده از سمبل‌های وضعیتی MSWINDOWS و LINUX به منظور مشخص کردن دستورات مناسب و ضمیمه کردن بسته‌های مناسب در بند requires توجه کنید.

لیست ۹-۱۰ QddgSamples-Dsgn.dpk

```
package QddgSamples_Dsgn;

{$R *.res}
{$R 'QddgSamples_Reg.dcr'}
{$ALIGN 8}
{$ASSERTIONS OFF}
{$BOOLEVAL OFF}
{$DEBUGINFO OFF}
{$EXTENDEDSYNTAX ON}
{$IMPORTEDDATA ON}
{$IOCHECKS ON}
{$LOCALSYMBOLS OFF}
{$LONGSTRINGS ON}
{$OPENSTRINGS ON}
{$OPTIMIZATION ON}
{$OVERFLOWCHECKS OFF}
{$RANGECHECKS OFF}
{$REFERENCEINFO OFF}
{$SAFEDIVIDE OFF}
{$STACKFRAMES OFF}
{$TYPEDADDRESS OFF}
{$VARSTRINGCHECKS ON}
{$WRITEABLECONST ON}
{$MINENUMSIZE 1}
{$IMAGEBASE $400000}
{$DESCRIPTION 'DDG: CLX Components'}

{$IFDEF MSWINDOWS}
{$LIBSUFFIX '60'}
{$ENDIF}

{$IFDEF LINUX}
{$SOPREFIX 'bpl'}
{$SOVERSION '6'}
{$ENDIF}
```

لیست ۹-۱۰ ادامه

```
{$DESIGNONLY}
{$IMPLICITBUILD OFF}

requires
  {$IFDEF LINUX}
  baseclx,
  {$ENDIF}
  visualclx,
  designide,
  QddgSamples;

contains
  QddgSamples_Reg in 'QddgSamples_Reg.pas',
  QddgDsnEdt in 'QddgDsnEdt.pas',
  QddgRgpEdt in 'QddgRgpEdt.pas';

end.
```

تذکر

به منظور، پشتیبانی همزمان Kylix و Delphi 7 در یک بسته، فایل‌های منبع، نام‌های مشخص شده در قسمت‌های requires و contains بایستی با وضعیت نام فایل واقعی تطابق داشته باشند. به عنوان مثال، در VCL متداول است که بسته DesginIDE در وضعیتی ترکیبی مشخص شود. البته در Linux ، بسته designide از حروف کوچک استفاده می‌کند. اگر وضعیت ترکیبی در فال منبع به کار رفته باشد Kylix قادر به یافتن فایل designide.dcp نخواهد بود زیرا در DesignIDE.dcpLinux با designide.dcp تفاوت دارد.

لیست ۱۰-۱۰ QddgDBSamples-Dsgn.dpk

```
package QddgDBSamples_Dsgn;

{$R *.res}
{$ALIGN 8}
{$ASSERTIONS OFF}
{$BOOLEVAL OFF}
{$DEBUGINFO OFF}
{$EXTENDEDSYNTAX ON}
{$IMPORTEDDATA ON}
{$IOCHECKS ON}
{$LOCALSYMBOLS OFF}
{$LONGSTRINGS ON}
{$OPENSTRINGS ON}
{$OPTIMIZATION ON}
{$OVERFLOWCHECKS OFF}
```

لیست ۱۰-۱۰ ادامه

```
{$RANGECHECKS OFF}
{$REFERENCEINFO OFF}
{$SAFEDIVIDE OFF}
{$STACKFRAMES OFF}
{$TYPEDADDRESS OFF}
{$VARSTRINGCHECKS ON}
{$WRITEABLECONST ON}
{$MINENUMSIZE 1}
{$IMAGEBASE $400000}
{$DESCRIPTION 'DDG: CLX Components (Data-Aware)'}

{$IFDEF MSWINDOWS}
{$LIBSUFFIX '60'}
{$ENDIF}

{$IFDEF LINUX}
{$SOPREFIX 'bpl'}
{$SOVERSION '6'}
{$ENDIF}

{$DESIGNONLY}
{$IMPLICITBUILD OFF}

requires
  {$IFDEF LINUX}
  baseclx,
  {$ENDIF}
  visualclx,
  QddgSamples_Dsgn,
  QddgDBSamples;

contains
  QddgDBSamples_Reg in 'QddgDBSamples_Reg.pas';

end.
```

یونیت‌های ثبت

همانگونه که در لیست منبع بسته‌های طراحی می‌توان دید، یونیت‌های ثبت به منظور ثبت تمامی اجزاء به کار می‌روند. همانگونه که در هنگام ایجاد اجزاء VCL نیز مرسوم است این یونیت‌های ثبتی (QddgSamples-Reg و QddgDBSamples-Reg) فقط در یک بسته طراحی قرار دارند. لیست ۱۰-۱۱ کد منبع مربوط به یونیت QddgSamples-Reg را نشان می‌دهد که در ثبت اجزاء نامراقب داده‌ای و ویرایشگر جزء TddgRadioGroupEditor کاربرد دارد.

لیست ۱۰-۱۱ QddgSamples-Reg.pas

```
{=============================================================================
  QddgSamples_Reg Unit

  Registration Unit for all non-data-aware DDG-CLX components.

  Copyright © 2001 by Ray Konopka
=============================================================================}

unit QddgSamples_Reg;

interface

procedure Register;

implementation

uses
  Classes, DesignIntf, DesignEditors, QExtCtrls,
  QddgSpin, QddgDsnSpn, QddgILSpin,
  QddgRgpEdt;


{======================}
{== Register Procedure ==}
{======================}

procedure Register;
begin
  {== Register Components ==}

  RegisterComponents( 'DDG-CLX',
                      [ TddgSpinner,
                        TddgDesignSpinner,
                       TddgImgListSpinner ] );
  {== Register Component Editors ==}

  RegisterComponentEditor( TRadioGroup, TddgRadioGroupEditor );
end;

end.
```

نگاشت‌های بیتی اجزاء

به منظور شناسایی اجزاء CLXای که جدیداً در Component Pallette ایجاد کرده‌اید بایستی یک نگاشت بیتی جزء ایجاد کنید که نگاشت بیتی ۱۶ رنگی است که اندازه آن ۲۴ × ۲۴ نقطه است. راهنمای

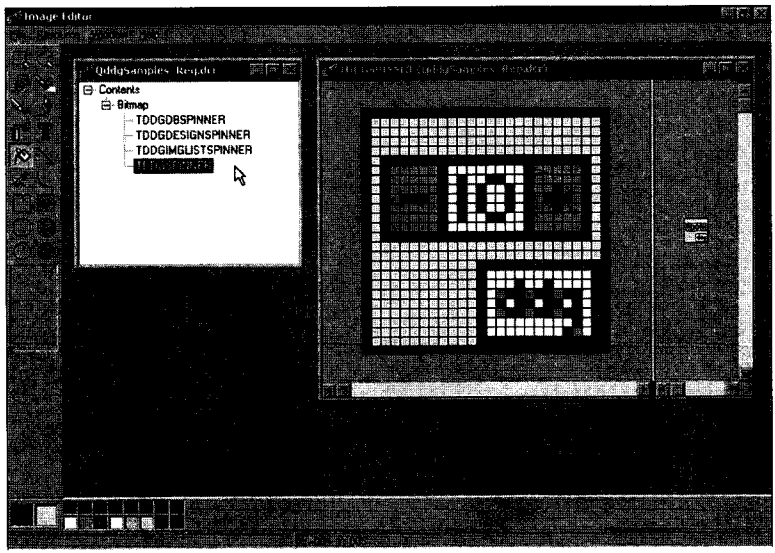
Kylix و Delphi پیشنهاد می‌کند که برای هر جزء یک فایل منبع مجزا ایجاد کنید.

البته Package Editor هنگامی که یک یونیت به بسته‌ای می‌افزایید به دنبال انطباق فایل‌های dcr. می‌گردد. متأسفانه، Package Editor این کار را برای هر دو بسته‌های زمان اجراء و زمان طراحی انجام می‌دهد و پیوند زدن نگاشت‌های بیتی پالت در یک بسته زمان اجراء کاری بیهوده است زیرا نگاشت‌های بیتی بلااستفاده خواهند بود و بخشی از حافظه به هدر می‌رود.

بنابراین به جای ایجاد فایل‌های dcr. مجزا برای هر یونیت جزء، یک فایل dcr. ایجاد کنید که حاوی تمام نگاشت‌های بیتی باشد. خوشبختانه منابع در Kylix همانند منابع به‌کار رفته در Delphi می‌باشند. یعنی، حتی با این تصور که Kylix برنامه‌های اجرایی Liunx را ایجاد می‌کند قالب به کار رفته برای افزودن منابع، قالب منبع Win32 است. به عنوان نتیجه می‌توانیم از هر ویرایشگر منبع که قادر به ایجاد فایل‌های res. در Windows است استفاده کرده و سپس توسعه نام فایل را به dcr. تغییر دهیم. به عنوان مثال، شکل ۹-۱۰ فایل QddgSamples-Reg.dcr را که در حال ویرایش شدن در Image Editor است را نشان می‌دهد.

توجه داشته باشید که نام فایل منبع همانند یونیت ثبتی است. به عنوان نتیجه هنگامی که یونیت ثبتی به بسته طراحی افزوده می‌شود فایل منبع جزء نیز اضافه می‌شود. علاوه بر این، از آنجا که یونیت ثبتی در بسته‌های زمان اجراء به کار برده نمی‌شود نگاشت‌های بیتی جزء بر آنها پیوند نخواهد خورد.

از اهمیت نگاشت‌های بیتی باطراحی مناسب در نمایش اجزاء خود غافل نشوید. نگاشت‌های بیتی اجزاء اولین چیزی هستند که کاربران مشاهده می‌کنند. نگاشت‌های بیتی غیرحرفه‌ای اجزاء غرحرفه‌ای را



شکل ۹-۱۰ نحوهٔ ایجاد فایل‌های dcr

در ذهن کاربران تداعی می‌کنند. اگر اجزائی برای فروشگاههای تجاری ایجاد می‌کنید احتمالاً نیاز خواهید داشت که از طراحان حرفه‌ای برای طراحی نگاشت‌های بیتی استفاده کنید.

خلاصه

می‌توانید به منظور کمک به تبدیل اجزاء VCL خود به اجزاء CLX کارهای زیادی انجام دهید. ابتدا هر زمان که ممکن بود از پوشش‌های VCL استفاده کنید. به عنوان مثال، از متدهای TCanvas به جای افزایش مستقیم توابع GDI استفاده کنید. متدهای موجود ارسال رویدادها نظیر ()MouseDown را باطل‌سازی کنید به جای آن که پیغام wm−LButtonDown را به کار بگیرید.

Linux از پیغام‌ها استفاده نمی‌کند؛ بنابراین پیغام wm−LButtonDown در Linux حتی وجود ندارند. یک تکنیک مفید دیگر ایجاد کلاس‌های انتزاع شخصی است که به جداسازی کدهای مستقل از سیستم کاری کمک می‌کند.

علیرغم این که CLX پس از VCL مدل‌سازی شده است تبدیل اجزاء VCL موجود به CLX نیاز به کمی تلاش دارد. فراخوانی‌های ویژه سیستم کاری نظیر فراخوانی‌های win32 API و یا libc بایستی حذف شوند و یا حداقل توسط دستورات وضعیتی سیستم محصور شوند.

البته همواره امکان ایجاد یک جزء سنتی CLX با استفاده از یک فایل منبع که تحت Delphi/Windows و یا Kylix/Linux کار می‌کند وجود دارد.

فصل یازدهم

بسته‌ها[1]

در این فصل می‌خوانید

- **چرا از بسته‌ها استفاده کنیم؟**
- **چرا از بسته‌ها استفاده نکنیم؟**
- **انواع بسته‌ها**
- **فایل‌های بسته**
- **استفاده از بسته‌های زمان اجراء**
- **نصب بسته‌ها در IDE دلفی**
- **ایجاد بسته‌ها**
- **نسخه‌بندی بسته‌ها**
- **دستورات کامپایلر بسته‌ها**
- **قواعد نامگذاری بسته‌ها**
- **برنامه‌های توسعه‌پذیر با استفاده از بسته‌های زمان اجراء (Add-In)**
- **صدور توابع از بسته‌ها**
- **حصول اطلاعات در مورد یک بسته**

بسته‌ها در Delphi 3 معرفی شدند و کاربران را قادر می‌سازند تا بخش‌هایی از برنامه را در ماژول‌های جداگانه قرار دهند که این ماژول‌ها را می‌توان بین برنامه‌های دیگر به اشتراک گذاشت. بسته‌ها به بیان ساده کتابخانه‌های پیوند دینامیک (DLL)[2] ویژه‌ای هستند که حاوی اطلاعات اضافی خاص Delphi می‌باشند. تفاوت بسته‌ها با DLL در نحوه استفاده از آنهاست. بسته‌ها معمولاً برای ذخیره‌سازی اجزاء در یک ماژول مجزا و اشتراکی (یک فایل BPL)[3] به کار می‌روند. هنگامی که برنامه‌ای از Delphi را ایجاد می‌کنید بسته‌هایی که ایجاد کرده‌اید را می‌توان به جای پیوند مستقیم در زمان کامپایل/ پیوند زدن، در زمان اجراء و توسط برنامه به کار گرفت. از آنجا که کد مربوط به این یونیت‌ها به جای آن که در قالب exe. و یا dll. قرار گیرند در قالب یک فایل bpl. جای می‌گیرد، اندازه برنامه‌های exe. و یا dll. برنامه شما بسیار کوچک می‌شود.

۱- Packages ۲- Dynamic Link Library ۳- Borland Package Library

بسته‌ها مخصوص گونه VCL هستند، یعنی برنامه‌هایی که در دیگر زبان‌ها نوشته شده‌اند قادر به استفاده از بسته‌های ایجاد شده توسط Delphi نمی‌باشند (به جز C++ Builder). یکی از اهداف استفاده از بسته‌ها رهایی از محدودیت‌های Delphi1 و 2 بوده است. در این نگارش‌ها VCL به هر فایل اجرائی حداقل ۱۵۰ تا ۲۰۰ کیلو بایت اضافه می‌کرد. بنابراین حتی اگر بخشی از برنامه خود را به صورت DLL تفکیک می‌کردید هر دو DLL و برنامه بایستی حاوی کد افزوده شده می‌بودند که این امر واقعاً مسأله‌ای مشکل‌ساز در طراحی برنامه‌های کاربردی بود. بسته‌ها شما را قادر می‌سازند که بر این مشکلات غلبه کنید و راهی مناسب برای توزیع اجزاء خود بیابید.

چرا از بسته‌ها استفاده کنیم؟

اهداف مختلفی وجود دارند که باعث می‌شوند از بسته‌ها استفاده کنیم. اهداف مهمی که برای این کار وجود دارند در قالب بخش‌های زیر لیست شده‌اند: کاهش کد، بخش‌بندی برنامه، و محدودسازی اجزاء.

کاهش کد

یک هدف اساسی در استفاده از بسته‌ها کاستن اندازه برنامه‌ها و DLL ها است. دلفی بسته‌های از پیش تعریف شده‌ای دارد که VCL را به گروههای منطقی تقسیم می‌کنند. در واقع می‌توانید برنامه خود را به گونه‌ای برای کامپایل کردن انتخاب کنید که به نظر برسد بسیاری از این بسته‌ها موجودند.

توزیع کمتر برنامه‌ها - بخش‌بندی برنامه‌ها

متوجه خواهید شد که بسیاری از برنامه‌ها به صورت دموهای قابل استفاده و یا بهنگام‌سازی برنامه‌های موجود در اینترنت وجود دارند. حالتی را در نظر بگیرید که کاربران امکان کپی کردن نگارش‌های کم‌حجم‌تر یک برنامه را داشته باشند.

محدودسازی اجزاء

یکی از متداول‌ترین اهداف استفاده از بسته‌ها، توزیع اجزاء است. اگر شما یک توزیع کننده اجزاء باشید بایستی نحوه ایجاد بسته‌ها را بدانید زیرا اجزاء اصلی زمان طراحی نظیر ویرایشگرهای صفت و ویزاردها همگی با کمک بسته‌ها فراهم شده‌اند.

بسته‌ها در مقابل DLL ها

هنگامی که DLL تخلیه شود Windows bandle توسط سیستم عامل مورد مراجعه مجدد قرار نمی‌گیرد. پیغام بعدی که برای پنجره‌های سطح بالایی از صفت گذر کند باعث بروز خطایی در برنامه می‌شود که باعث اتمام کار windows می‌شود زیرا برنامه در حالتی غیرمعتبر قرار دارد. استفاده از بسته‌ها به جای DLL ها باعث برطرف شدن این خطا می‌شود زیرا بسته‌ها به کپی اصلی Form.pas برنامه

مراجعه کرده و صف پیغام خواهد توانست با موفقیت با برنامه ارتباط برقرار کند.

چرا از بسته‌ها استفاده نکنیم؟

نبایستی از بسته‌های زمان اجراء استفاده کنیم مگر آن که مطمئن باشیم بقیه برنامه‌ها از این بسته‌ها استفاده خواهند کرد. در غیر این صورت این بسته‌ها نسبت به حالتی که قصد کامپایل کد منبع را در برنامه اجرایی نهایی خود داشته باشید فضای بیشتری را اشغال می‌کنند. چرا چنین اتفاقی رخ می‌دهد؟ اگر بسته نرم‌افزاری ایجاد کنید که فضای مورد نیاز خود را از ۲۰۰ کیلو بایت به ۳۰ کیلو بایت کاهش داده باشد چنین به نظر می‌رسد که فضای کمی از حافظه را صرفه‌جویی نموده‌اید. می‌توانید مشاهده کنید که این صرفه‌جوئی چیزی که مورد انتظار شما بوده است نمی‌باشد. هدف ما آن است که از بسته‌ها به منظور اشتراک کد در هنگام استفاده از چندین برنامه اجرایی بهره بگیرید. توجه داشته باشید که این کار فقط در بسته‌های زمان اجراء مناسب است. اگر شما یک تولید کننده اجزاء هستید بایستی یک بسته زمان طراحی آماده کنید که حاوی اجزائی است که می‌خواهید در IDE قابل دسترسی باشد.

انواع بسته‌ها

چهارگونه از بسته‌ها برای ایجاد و استفاده در دسترس شما هستند:

- بسته‌های زمان اجراء- بسته‌های زمان اجراء حاوی کد، اجزاء و مواردی نظیر این هستند که در زمان اجراء، مورد نیاز برنامه می‌باشند. اگر برنامه‌ای می‌نویسید که نیاز به یک بسته زمان اجرای خاص، دارد برنامه در غیاب آن بسته اجراء نخواهد شد.
- بسته‌های طراحی- بسته‌های طراحی حاوی اجزاء، ویرایشگرهای صفت/ جزء، expertها و مواردی نظیر این هستند که برای طراحی برنامه در IDE مورد نیاز می‌باشند. این نوع از بسته‌ها فقط توسط دلفی مورد استفاده قرار گرفته و هرگز به همراه برنامه‌های شما توزیع نمی‌شوند.
- بسته‌های زمان اجراء و طراحی- بسته‌ای که قابلیت‌های زمان اجراء و طراحی را داشته باشد نوعاً هنگامی به کار برده می‌شود که عناصر خاص طراحی نظیر ویرایشگرهای جزء/ صفت و expertها وجود نداشته باشند. می‌توانید این نوع بسته‌ها را به منظور ساده‌سازی توسعه و گسترش نرم‌افزار ایجاد کنید. البته اگر این بسته حاوی عناصر طراحی بود استفاده زمان اجرای آن بخشی از پشتیبانی طراحی را نیز به برنامه منتقل می‌کند. در بسیاری از عناصر زمان طراحی، ایجاد بسته‌های زمان اجراء، و زمان طراحی توصیه می‌شود تا عناصر ویژه‌ای جدا طراحی شوند.
- بسته‌های غیرزمان اجراء و طراحی- این محصول از بسته‌ها برای استفاده توسط دیگر بسته‌ها در نظر گرفته شده است و نمی‌توان آن را مستقیماً توسط برنامه‌ها به کار گرفت و یا در محیط طراحی مورد استفاده قرار داد. مفهوم این جمله آن است که بسته‌ها می‌توانند دیگر بسته‌ها را ضمیمه کرده و یا از آنهااستفاده کنند.

فایل‌های بسته

در جدول ۱-۱۱ انواع فایل‌های خاص بسته‌ها برحسب توسعه آنها لیست شده است.

جدول ۱-۱۱ فایل‌های بسته

توسعه فایل	نوع فایل	توضیحات
.dpk	فایل منبع بسته	این فایل هنگامی تولید می‌شود که شما Package Editor را به کار بگیرید. می‌توانید این فایل را همانند فایل dpr. در Delphi Project در نظر بگیرید.
.dcp	فایل سمبل بسته زمان اجراء / طراحی	این گونه کامپایل شده بسته است که حاوی اطلاعات سمبلیک بسته و یونیت‌های آن است. به علاوه، اطلاعات عنوانی مورد نیاز IDE دلفی نیز در آن وجود دارند.
.dcu	یونیت کامپایل شده	یک گونه کامپایل شده از یونیت در بسته‌ای قرار دارد. یک فایل dcu. برای هر یونیت داخل بسته ایجاد خواهد شد.
.bpl	زمان اجراء / طراحی	بسته کتابخانه‌ای زمان اجراء یا طراحی است که متناظر با یک DLL در Windows می‌باشد. اگر بسته موردنظر یک بسته زمان اجراء باشد فایل را به همراه برنامه‌های خود توزیع خواهید نمود. اگر این فایل یک بسته طراحی را مشخص کند آن را برای برنامه‌نویسانی که برای نوشتن برنامه‌ها از آن استفاده می‌کنند توزیع خواهید نمود. توجه داشته باشید که اگر کد منبع را توزیع نمی‌کنید بایستی فایل‌های dcp. متناظر آن را توزیع نمایید.

استفاده از بسته‌های زمان اجراء

برای استفاده از بسته‌های زمان اجراء در برنامه‌های دلفی خود بایستی کادر Build With Runtime Package را در کادر Project ، Options در صفحه Package خود علامت‌دار کنید. دفعه بعد که برنامه خود را می‌سازید، برنامه شما به صورت دینامیکی به بسته‌های زمان اجراء پیوند خواهد خورد به جای آن که یونیت‌ها به صورت استاتیکی به فایل‌های exe. یا dll. پیوند بخورند. نتیجه کار برنامه‌ای کم‌حجم‌تر خواهد بود.

نصب بسته‌ها در IDE دلفی

برخی اوقات لازم است که یک بسته در IDE دلفی نصب شود. در صورتی که مجموعه اجزاء ثالث و یا افزودنی‌های دلفی را بیابید که این کار را در زمان نصب انجام ندهند چنین کاری ضروری است.

ابتدا بایستی فایل‌های بسته را در محل‌های مناسب بیابید. جدول ۲-۱۱ محلی که فایل‌های بسته در آن قرار دارند را نشان می‌دهد.

جدول ۲-۱۱ محل‌های فایل‌های بسته

فایل بسته	محل
بسته‌های زمان اجراء(*.bpl)	بسته‌های زمان اجراء در شاخه‌های \Windows\System\ (برای گونه‌های Windows 95/98) و یا \WinNT\System32\ (برای گونه‌های Windows NT/2000) قرار دارند.
بسته‌های طراحی (*.bpl)	از آنجا که ممکن است بسته‌های متعددی را از فروشندگان مختلف تهیه کنید، بسته‌های طراحی شما بایستی در شاخه‌ای متعارف باشند. به عنوان مثال، شاخه‌ای با نام \PKG در شاخه \Delphi 7 ایجاد کرده و بسته‌های طراحی را در آن قرار دهید.
فایل‌های نمادین بسته (*.dcp)	می‌توانید فایل‌های نمادین بسته را در محلی مشابه با فایل‌های طراحی قرار دهید.
یونیت‌های کامپایل شده (*.dcu)	در صورتی که بسته‌های طراحی را بدون منبع توزیع می‌کنید بایستی یونیت‌های کامپایل شده را نیز توزیع نمایید. توصیه می‌شود که DCU هایی که از اشخاص دیگر دریافت می‌کنید را در شاخه‌ای مشابه با شاخه \Delphi 7\3PartyLib قرار دهید. به عنوان مثال، می‌توانید شاخه \Deplphi 7\3PartyLib را ایجاد کرده و جزءهای dcu.* را در آن قرار دهید. مسیر جستجوی شما این مسیر را نیز در برخواهد گرفت.

برای نصب یک بسته بایستی صفحه Packages جعبه مکالمه Project Options را با انتخاب جزء و نصب بسته‌ها از منوی Delphi 7 به کار گیرید.

با کلیک کردن روی دکمه Add می‌توانید فایل bpl. موردنظر را انتخاب نمایید. به محض انجام این کار فایل موردنظر در صفحه Project انتخاب می‌شود. هنگامی که روی OK کلیک کنید بسته جدید در IDE دلفی نصب خواهد شد. اگر این بسته حاوی جزءهایی باشد صفحه جزء جدید را به همراه اجزاء جدیداً نصب شده خواهید دید.

ایجاد بسته‌ها

قبل از ایجاد یک بسته بایستی در مورد برخی موارد تصمیم‌گیری کنید. اولاً بایستی بدانید که چه نوع بسته‌ای را می‌خواهید ایجاد نمایید (زمان اجراء، طراحی و ...). این مسأله بر پیش‌زمینه‌هایی استوار است که ارائه خواهیم کرد. دوماً بایستی نام بسته ایجاد شده و محل ذخیره آن را بدانید. به خاطر داشته باشید که شاخه‌ای که بسته‌های خود را در آن توسعه می‌دهید احتمالاً محل ایجاد بسته نخواهد بود. در پایان بایستی بدانید که بسته شما چه یونیت‌هایی را داشته و چه بسته‌هایی مورد نیاز آن می‌باشند.

Package Editor

بسته‌ها متداول‌ترین تولیدات Package Editor هستند که با انتخاب Object Repository Package آنها را به کار می‌گیرید. Package Editor دارای دو پوشه است: Contains ، Requires .

پوشه Contains

در پوشه Contains یونیت‌هایی را مشخص می‌کنید که بایستی در بسته جدید شما کامپایل شوند. قوانینی برای قرار دادن یونیت‌ها در صفحه Contains بسته وجود دارند.

- یونیت بایستی در بند contains بسته‌ای دیگر و یا بند uses یونیتی که در بسته‌ای دیگر قرار دارد لیست نشده باشد که با این کار به صورت همزمان به همراه بسته‌ای که یونیت در داخل آن قرار دارد باردهی می‌شود.
- یونیت‌هایی که در بند contains یک بسته لیست می‌شوند، به صورت مستقیم و یا غیرمستقیم (در بند uses یونیت‌هایی که در بند contains بسته لیست شده‌اند موجودند) نمی‌توانند در بند requires بسته لیست شوند. دلیل این امر آن است که این یونیت‌ها هنگامی که بسته کمپایل شده است به آن محدود شده‌اند.
- در صورتی که یونیت در بند contains بسته دیگری لیست شده باشد که توسط برنامه‌ای مشابه مورد استفاده قرار می‌گیرد نمی‌توانید یونیت را در بند contains یک بسته لیست کنید.

پوشه Requiers

در پوشه Requiers بسته‌های دیگری که توسط بسته جدید موردنیاز هستند را مشخص می‌کنید. این مشابه با بند uses یک یونیت Delphi است.

ترتیب نوعی در اینجا آن است که به عنوان مثال تمام جزءهای خود را در یک بسته زمان اجراء قرار دهید. سپس یک بسته طراحی ایجاد می‌کنید که در بند requires خود بسته زمان اجراء را در بردارد. برای قرار دادن بسته‌ها در پوشه Requires بسته‌های دیگر قوانینی وجود دارند:

- از مراجعات بازگشتی اجتناب کنید. (Package1 در قسمت requires خود Package1 را ندارد و همچنین بسته دیگری که در قسمت requires آن Package1 موجود باشد نیز را در برندارد.)
- زنجیر مراجعات نباید به بسته‌ای مراجعه کند که قبلاً در زنجیر به آن مراجعه شده است.

Package Editor نوار ابزار و منوهای حساس به متن را در خود جای داده است. به راهنمای Delphi 7 در Package Editor مراجعه کنید تا توضیح عملکرد این دکمه‌ها را مشاهده کنید.

دستورالعمل‌های طراحی بسته

قبلاً گفتیم که بایستی نوع بسته‌هایی که قصد ایجاد آنها را داریم مشخص باشد. در این بخش قصد داریم که چهار دستورالعمل مختلف که در آنها از بسته‌های طراحی و/ یا زمان اجراء استفاده می‌کنیم را ارائه نمائیم.

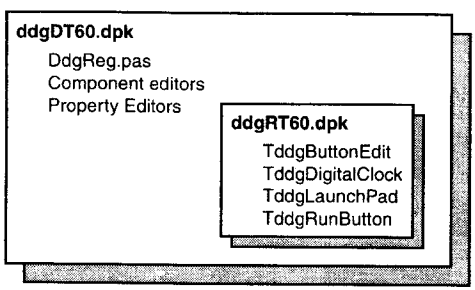
دستورالعمل ۱ : Design and Runtime Packages for Components

برای بسته‌های طراحی و زمان اجراء مربوط به جزءها شما یا نویسنده جزء هستید و یا یکی از دو حالت زیر (یا هر دوی آن) را به کار می‌گیرید:

- ممکن است بخواهید برنامه‌نویسان دلفی قادر به کامپایل/ Link اجزاء شما در برنامه‌های خود باشند و یا به صورت جدا اجزاء را به همراه برنامه‌ها توزیع کنید.
- یک بسته از اجزاء دارید و نمی‌خواهید کاربران قادر به کامپایل جنبه‌های مختلف طراحی در کد برنامه خود باشند.

با وجود این شرایط هر دو بسته زمان اجراء و طراحی را ایجاد خواهید نمود. در شکل ۱-۱۱ این آرایش نشان داده شده است. همانگونه که در شکل می‌بینید بسته طراحی (ddgDT60.dpk) هر دو جنبه طراحی (ویرایشگرهای جزء و صفت) و بسته زمان اجراء (ddgRT60.dpk) را برآورده می‌سازد. بسته زمان اجراء (dgRT60.dpk) فقط شامل اجزاء شما می‌شود. این آرایش با لیست کردن بسته زمان اجراء در بخش requires بسته طراحی، همانند شکل ۱-۱۱ حاصل می‌شود.

بایستی قبل از کامپایل کردن هر بسته برای آن گزینه‌های به کارگیری مناسب را اعمال کنید. این کار از طریق جعبه مکالمه Package Options انجام می‌شود. (با راست کلیک کردن در Package Editor به منظور به کارگیری منوی محلی می‌توانید این کار را انجام دهید.) برای بسته زمان اجراء، DDgRT60.dpk گزینه به کارگیری بایستی روی Runtime Only تنظیم شده باشد. با این کار اطمینان حاصل می‌کنید که بسته را نمی‌توان به صورت یک بسته طراحی در IDE نصب نمود. برای بسته طراحی، DdgDT60.dpk ، گزینه به کارگیری Design Time Only بایستی انتخاب شده باشد. با این کار کاربران قادر می‌شوند تا بسته را در IDE دلفی نصب کنند و هنوز هم از به کارگیری بسته به عنوان یک بسته زمان اجراء جلوگیری نمایند. افزودن بسته زمان اجراء به بسته طراحی باعث نمی‌شود که اجزاء داخل بسته زمان اجراء در دسترس IDE دلفی باقی بمانند. شما بایستی هنوز هم اجزاء خود را به همراه IDE ثبت نمایید. همانگونه که از قبل می‌دانید، هرگاه که یک جزء را ایجاد کنید، دلفی به صورت خودکار یک روال ()Register در یونیت جزء درج می‌کند که در عمل ()RegisterComponents را فراخوانی می‌نماید.



شکل ۱-۱۱ بسته‌های طراحی بخش‌های طراحی و بسته‌های زمان اجراء را میزبانی می‌کنند

RegisterProcedure() روالی است که جزء شما را هنگام نصب به همراه IDE دلفی ثبت می‌کند. هنگام کار با بسته‌ها، کاری که توصیه می‌شود آن است که روال Register() را از یونیت جزء به یونیت ثبتی مجزای دیگری منتقل کنید. این یونیت ثبتی تمامی اجزاء شما را با کمک فراخوانی RegisterComponents() ثبت می‌نماید. با این کار نه تنها مدیریت ثبت اجزاء برای شما آسان‌تر می‌شود بلکه از نصب و استفاده غیرقانونی بسته‌های زمان اجراء توسط دیگران جلوگیری می‌شود زیرا اجزاء در IDE دلفی قابل دسترسی نخواهند بود.

به عنوان یک مثال، اجزاء به کار رفته در این کتاب توسط بسته زمان اجراء DdgRT60.dpk میزبانی می‌شوند. ویرایشگرهای صفت، ویرایشگرهای جزء، و یونیت ثبتی (DdgReg.pas) . برای اجزاء موردنظر ما در بسته طراحی DdgDT60.dpk موجود می‌باشند. DdgDT60.dpk در بند requires خود DdgRT60.dpk را نیز دربردارد. در لیست ۱-۱۱ خواهید دید که یونیت ثبتی به چه صورتی می‌باشد.

لیست ۱-۱۱ یونیت ثبتی برای اجزاء Delphi 7

```
unit DDGReg;

interface

procedure Register;

implementation

uses Classes, ExptIntf, DsgnIntf, TrayIcon, AppBars, ABExpt, Worthless,
  RunBtn, PwDlg, Planets, LbTab, HalfMin, DDGClock, ExMemo, MemView,
  Marquee, PlanetPE, RunBtnPE, CompEdit, DefProp, Wavez,
  WavezEd, LnchPad, LPadPE, Cards, ButtonEdit, Planet, DrwPnel;

procedure Register;
begin

  // Register the components.
  RegisterComponents('DDG',
  [ TddgTrayNotifyIcon, TddgDigitalClock, TddgHalfMinute, tddgButtonEdit,
    TddgExtendedMemo, TddgTabListbox, TddgRunButton, TddgLaunchPad,
    TddgMemView, TddgMarquee, TddgWaveFile, TddgCard, TddgPasswordDialog,
    TddgPlanet, TddgPlanets, TddgWorthLess, TddgDrawPanel,
    TComponentEditorSample, TDefinePropTest]);


  // Register any property editors.
  RegisterPropertyEditor(TypeInfo(TRunButtons), TddgLaunchPad, '',
    TRunButtonsProperty);
  RegisterPropertyEditor(TypeInfo(TWaveFileString), TddgWaveFile, 'WaveName',
    TWaveFileStringProperty);
```

لیست ۱-۱۱ ادامه

```
  RegisterComponentEditor(TddgWaveFile, TWaveEditor);
  RegisterComponentEditor(TComponentEditorSample, TSampleEditor);
  RegisterPropertyEditor(TypeInfo(TPlanetName), TddgPlanet,
   'PlanetName', TPlanetNameProperty);
  RegisterPropertyEditor(TypeInfo(TCommandLine), TddgRunButton, '',
    TCommandLineProperty);
  // Register any custom modules, library experts.
  RegisterCustomModule(TAppBar, TCustomModule);
  RegisterLibraryExpert(TAppBarExpert.Create);

end;

end.
```

امنیت اجزاء

هر کسی قادر به ثبت اجزاء شما می‌باشد حتی با این تصور که فقط بسته‌های زمان اجرای شما را دارد. این کار را می‌توان با ایجاد یونیت ثبتی شخصی انجام داد که در آن اجزاء موردنظر را به ثبت برساند. سپس این یونیت را به یک بسته مجزا می‌افزاید که بسته زمان اجرای شما را در بند requiers خود دارد. پس از این که بسته جدید در IDE دلفی نصب شد، اجزاء شما در Component Palette ظاهر خواهند شد. البته هنوز کامپایل همه برنامه‌ها با استفاده از اجزاء شما ممکن نیست زیرا فایل‌های duc.* موردنیاز یونیت‌های جزء شما یافت نخواهند شد.

توزیع بسته

هنگام توزیع بسته‌ها بدون کد منبع به نویسندگان اجزاء بایستی بسته‌های کامپایل شده DdgDT6.bpl و DdgRT6.bpl، هر دو فایل‌های dcp.* و همه یونیت‌های کامپایل شده (dcu.*) موردنیاز برای کامپایل اجزاء را توزیع نمایید. برنامه‌نویسانی که از اجزاء شما استفاده می‌کنند و تمایل دارند که بسته‌های زمان اجرای برنامه‌های آنها در دسترس باشد بایستی بسته DdgRT6.bpl آنها به همراه برنامه‌ها و دیگر بسته‌های زمان اجرایی که ممکن است استفاده کنند توزیع شود.

دستورالعمل ۲ : Design Package Only for Components

این حالت زمانی رخ می‌دهد که می‌خواهید اجزایی که تمایل ندارید در بسته‌های زمان اجراء توزیع شوند را توزیع نمایید. در این حالت، اجزاء، ویرایشگرهای اجزاء، ویرایشگرهای فعالیت، یونیت ثبتی جزء، و غیره را در یک فایل بسته ضمیمه خواهید نمود.

توزیع بسته

هنگام توزیع بسته‌ها بدون کد منبع به نویسندگان اجزاء بایستی بسته کامپایل شده، DdgDT6.pbl، فایل DdgDT6.dcp و همه یونیت‌های کامپایل شده (*.dcu) موردنیاز برای کامپایل اجزاء خود را توزیع نمایید. برنامه‌نویسانی که از اجزاء شما استفاده می‌کنند بایستی اجزاء شما را در برنامه‌های خود کامپایل نمایند. این اشخاص همه اجزاء شما را به عنوان بسته‌های زمان اجراء توزیع نخواهند کرد.

دستورالعمل ۳:

Design Features Only (No Components) IDE Enhancements

این حالت هنگامی رخ می‌دهد که شما قصد بخش‌بندی برنامه خود به بخش‌های منطقی دارید که هر یک از این بخش‌ها را بتوانید به صورت جدا توزیع کنید. این کار با اهداف مختلفی انجام می‌شود:

- پشتیبانی این حالت آسان‌تر است.
- کاربران قادر خواهند بود که فقط توابع مورد نیاز خود را خریداری نمایند. اگر در آینده نیاز به توابع دیگری داشتند می‌توانند بسته مورد نیاز خود را download کنند.
- ترمیم بخش‌های برنامه آسان‌تر خواهد بود.

در این حالت شما فقط فایل‌های *.bpl موردنیاز برنامه خود را فراهم می‌کنید. این حالت مشابه حالت قبل است با این تفاوت که به جای فراهم کردن یک بسته برای IDE دلفی بسته‌ای برای برنامه خود فراهم خواهید کرد. هنگام بخش‌بندی برنامه به این شکل بایستی به نکات مربوط به نگارش‌بندی بسته‌ها که در بخش بعد ذکر می‌کنیم توجه داشته باشید.

نگارش‌بندی بسته‌ها

می‌توانید نگارش‌بندی بسته‌ها را به صورتی مشابه با نگارش‌بندی یونیت‌ها تصور کنید. یعنی هر بسته‌ای که برای برنامه خود فراهم می‌کنید بایستی با استفاده از نگارش مشابه Delphi که در کامپایل برنامه به کار گرفته شده است کامپایل شود. بنابراین، نمی‌توانید بسته‌ای که در Delphi 7 نوشته شده است، را برای برنامه‌ای در Delphi 7 به کار گیرید. برنامه‌نویسان Broland نگارش بسته‌ها را به عنوان code base در نظر می‌گیرند. بنابراین بسته‌ای که در Delphi 7 نوشته شده است code base 7.0 دارد.

دستورات کامپایلری بسته

برخی دستورات خاص کامپایلر هستند که می‌توانید در کد منبع بسته‌های خود قرار دهید. برخی از این دستورات خاص یونیت‌هایی هستند که در بسته قرار گرفته‌اند؛ بقیه دستورات خاص فایل بسته هستند. این دستورات در جداول ۳-۱۱ و ۴-۱۱ لیست شده و شرح داده شده‌اند.

جدول ۳-۱۱ دستورات کامپایلر یونیت‌هایی که در بسته‌ها قرار می‌گیرند.

دستور	مفهوم
{$G} یا {IMPORTEDDATA}	از این دستور هنگامی استفاده می‌شود که قصد دارید از بسته‌بندی شدن یونیت جلوگیری کنید. هنگامی که می‌خواهید یونیت مستقیماً به برنامه پیوند بخورد، این دستور را با دستور {$WEAKPACKAGEUNIT} مقایسه کنید که به یونیت این امکان را می‌دهد که در یک بسته ضمیمه شود ولی کد آن به صورت استاتیک به برنامه پیوند بخورد.
{$DENYPACKAGE UNIT}	همانند {$G}
{$WEAKPACKAGE UNIT}	بخش بعد را مطالعه کنید.

جدول ۴-۱۱ دستورات کامپایلر مربوط به فایل بسته dpk.

دستور	مفهوم
{#DESIGNONLY ON}	بسته‌ها را به صورت بسته‌های زمان طراحی کامپایل می‌کند.
{$RUNONLY ON}	بسته‌ها را به صورت بسته‌های زمان اجراء کامپایل می‌کند.
{$IMPLICITBUILD OFF}	از ساخت مجدد بسته جلوگیری می‌کند. از این گزینه هنگامی استفاده کنید که بسته متناوباً تغییر پیدا نمی‌کند.

مطالب بیشتری در مورد {$WEAKPACKAGEUNIT}

مفهوم یک بسته ضعیف[1] آسان است اساساً این مفهوم هنگامی به کار می‌رود که بسته شما به فراخوانی به کتابخانه‌هایی قرار دارد. (DLL) مراجعه کند که موجود نباشند. به عنوان مثال، بسته VCL60 هسته هایی WIN32 API انجام می‌دهد که در سیستم عامل Windows بسیاری از این فراخوانی‌ها در هستند. با DLLموجودند که در همه ماشین‌ها وجود ندارند. این فراخوانی‌ها توسط یونیت‌هایی افشا می‌شوند که حاوی دستور قرار دهید آن {$WEAKPACKAGEUNIT} ضمیمه کردن این دستور کد منبع آن یونیت را در بسته نگه خواهید داشت ولی به جای آن که آن را در فایل تصور کنید). DCP را در فایل مراجعه به توابعی BPL قرار می‌دهید (DCP را همانند DCU و BPL را همانند DLL بنابراین هر از این یونیت‌های بسته‌های ضعیف باعث می‌شود که به جای مراجعه دینامیک در بسته پیوندی استاتیک به برنامه حاصل شود.

دستور {$WEAKPAGKAGEUNIT} دستوری است که به ندرت مورد استفاده قرار می‌گیرد. این دستور توسط توسعه دهندگان دلفی ایجاد شده است تا حالات خاص را تحت پوشش قرار دهد. اگر

۱- Weak

دو جزء داشته باشیم که هر یک در بسته‌ای مجزا قرار داشته و به یونیت رابط مشابهی از یک DLL مراجعه کنند مشکل رخ می‌دهد. هنگامی که یک برنامه از هر دو جزء استفاده کند. این کار باعث می‌شود که دو نمونه از DLL باردهی شوند که با این کار در مقداردهی اولیه مراجعه به متغیرهای سراسری اشکال ایجاد می‌شود. راه‌حل کار قرار دادن یونیت رابط در یکی از بسته‌های استاندارد دلفی نظیر VCL60.bpl است. البته این کار باعث بروز مسأله دیگری برای DLL های خاص می‌شود که ممکن است موجود نباشند، نظیر PENWIN.DLL . اگر VCL60.bpl یونیت رابطی برای یک DLL داشته باشد که موجود نیست VCL60.bpl را منتقل کرده و دلفی برای آن مورد غیرقابل استفاده می‌شود.

توسعه دهندگان دلفی برای این مسأله راه‌حلی ارائه کردند. VCL60.bpl را به نحوی تغییر دادند که یونیت رابط را در یک بسته یکتا جای دهد ولی هر زمان که VCL60 به همراه IDE دلفی به کار رود به صورت استاتیک پیوند خورده و به صورت دینامیک باردهی نمی‌شود.

احتمالاً تاکنون هرگز نیاز به استفاده ازاین دستور نداشته‌اید مگر آن که حالتی مشابه را پیش‌فرض کرده باشید. که توسعه دهندگان با دلفی با آن مواجه شده‌اند و یا بخواهید حالتی را ایجاد کنید که یونیت خاص در بسته‌ای ضمیمه باشد ولی به صورت استاتیک به آن پیوند خورده باشد. هدف این کار می‌تواند بهینه‌سازی باشد. توجه داشته باید که یونیت‌هایی که به صورت ضعیف بسته‌بندی شده‌اند نمی‌توانند متغیرهای سراسری و یا کد در بخش‌های آغازین/ پایانی خود داشته باشند. برای بسته‌های ضعیف بایستی فایل‌های dcu.* را به همراه بسته‌های خود توزیع کنید.

قواعد نامگذاری بسته‌ها

قبلاً گفتیم که در مطالب مربوط به نگارش‌بندی بسته‌ها بایستی نحوه نامگذاری بسته‌ها مشخص شوند. برای نحوه نامگذاری بسته‌ها مجموعه قوانین خاصی وجود ندارد ولی در اینجا پیشنهاد می‌کنیم که از قواعد نامگذاری استفاده شود که رعایت آن باعث هماهنگی در نام بسته‌ها می‌شود. توجه داشته باشید که نام بسته‌های ما با شناسه سه حرفی طراح/ شرکت آغاز می‌شود، پس از آن RT می‌آید که بیانگر بسته زمان اجراء (Runtime) و یا DT که بیانگر بسته زمان طراحی (Design Time) است می‌آید. می‌تواند قاعده نامگذاری دلخواه خود را پیگیری کنید. سازگاری را حفظ کنید و در نام بسته‌های خود سعی کنید نگارش دلفی را مشخص نمایید.

برنامه‌های قابل بسط با استفاده از بسته‌های زمان اجراء (Add-In)

بسته‌های زمان اجراء (Add-In) شما را قادر می‌سازند برنامه‌های خود را به ماژول‌های مختلف تقسیم کرده و این ماژول‌ها را به صورت مجزا از برنامه اصلی توزیع کنید. این کار مفید است زیرا قادر می‌شوید که کارایی برنامه خود را بدون نیاز به کامپایل یا طراحی مجدد برنامه افزایش دهید. البته این کار نیاز به معماری دقیق دارد. علیرغم این که این مسأله ماوراء قلمرو این کتاب است باارائه این بحث نحوه بهره‌گیری از قابلیت‌های این تکنیک مشخص خواهد شد.

ایجاد فرم‌های (Add-In)

برنامه به سه بخش منطقی تقسیم شده است: برنامه اصلی (ChildTest.exe) . بسته TChildForm(AIChildFrm6.bpl) و کلاس‌های توارثی TChildForm که هر یک در بسته خود قرار گرفته‌اند.

بسته AIChildFrm6.bpl حاوی کلاس مبنای TCildForm است. بقیه بسته‌ها حاوی کلاس‌های توارثی TChildForm و یا TChildForm های ذاتی [1] می‌باشند.

برنامه اصلی از بسته مجرد [2] استفاده می‌کند (AIChildFrm6.bpl) . هر بسته ذاتی پر از بسته مجرد استفاده می‌کند. به منظور انجام صحیح این کار برنامه اصلی بایستی به همراه بسته زمان اجراء و همچنین بسته AIChildFrm6.dcp کامپایل‌شود. علاوه بر این هر بسته ذاتی بایستی به بسته AIChildFrm6.dcp نیاز داشته باشد. منبع TChildForm و وارث‌های ذاتی آن را برای هر یونیت توارثی TChildForm که بایستی حاوی بلوک‌های initialization و finalization به صورت زیر باشند لیست نخواهیم کرد.

```
initialization
  RegisterClass(TCF2Form);
finalization
  UnRegisterClass(TCF2Form);
```

فراخوانی ()RegisterClass برای این که کلاس وراثتی TChildForm در دسترس سیستم جریان داده برنامه در هنگام بار کردن بسته قرار گیرد ضروری می‌باشد این کار مشابه زمانی است که ()RegisterComponents اجزاء را در دسترس IDE دلفی قرار می‌دهد.

هنگامی که بسته تخلیه شد (unload) فراخوانی ()UnRegisterClass نیاز به حذف کلاس ثبت شده دارد. توجه داشته باشید که RegisterClass کلاس را فقط در اختیار برنامه اصلی قرار می‌دهد. برنامه اصلی هنوز نام کلاس را نمی‌داند. پس برنامه اصلی چگونه وهله‌ای از یک کلاس ایجاد می‌کند که نام کلاس آن شناخته شده نیست؟ آیا مفهوم این تمرین در دسترس قرار فرم‌ها برای برنامه اصلی، بدون نیاز به ذکر نام کلاس‌ها در منبع برنامه اصلی است؟ در لیست ۲-۱۱ کد منبع فرم اصلی برنامه Main نشان داده شده است.

لیست ۲-۱۱ فرم اصلی برنامه

```
unit MainFrm;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
```

۱- Concrete ۲- Abstract

لیست ۲-۱۱ ادامه

```
  StdCtrls, ExtCtrls, ChildFrm, Menus;

const
  { Child form registration location in the Windows Registry. }
  cAddInIniFile  = 'AddIn.ini';
  cCFRegSection  = 'ChildForms';  // Module initialization data section

  FMainCaption   = 'Delphi 6 Developer''s Guide Child Form Demo';
type

  TChildFormClass = class of TChildForm;

  TMainForm = class(TForm)
    pnlMain: TPanel;
    Splitter1: TSplitter;
    pnlParent: TPanel;
    mmMain: TMainMenu;
    mmiFile: TMenuItem;
    mmiExit: TMenuItem;
    mmiHelp: TMenuItem;
    mmiForms: TMenuItem;
    procedure mmiExitClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    // reference to the child form.
    FChildForm: TChildForm;
    // a list of available child forms used to build a menu.
    FChildFormList: TStringList;
    // Index to the Close Form menu which shifts position.
    FCloseFormIndex: Integer;
    // Handle to the currently loaded package.
    FCurrentModuleHandle: HModule;
    // method to create menus for available child forms.
    procedure CreateChildFormMenus;
    // Handler to load a child form and its package.
    procedure LoadChildFormOnClick(Sender: TObject);
    // Handler to unload a child form and its package.
    procedure CloseFormOnClick(Sender: TObject);
    // Method to retrieve the classname for a TChildForm descendant
    function GetChildFormClassName(const AModuleName: String): String;
  public
    { Public declarations }
  end;

var
  MainForm: TMainForm;
```

لیست ۲-۱۱ ادامه

```
  implementation
  uses IniFiles;

  {$R *.DFM}
function RemoveExt(const AFileName: String): String;
{ Helper function to remove the extension from a file name. }
begin
  if Pos('.', AFileName) <> 0 then
    Result := Copy(AFileName, 1,  Pos('.', AFileName)-1)
  else
    Result := AFileName;
end;

procedure TMainForm.mmiExitClick(Sender: TObject);
begin
  Close;
end;

procedure TMainForm.FormCreate(Sender: TObject);
begin
  FChildFormList := TStringList.Create;
  CreateChildFormMenus;
end;

procedure TMainForm.FormDestroy(Sender: TObject);
begin
  FChildFormList.Free;
  // Unload any loaded child forms.
  if FCurrentModuleHandle <> 0 then
    CloseFormOnClick(nil);
end;

procedure TMainForm.CreateChildFormMenus;
{ All available child forms are registered in the Windows Registry.
  Here, we use this information to create menu items for loading each of the
  child forms. }
var
  IniFile: TIniFile;
  MenuItem: TMenuItem;
  i:  integer;
begin
  inherited;

  { Retrieve a list of all child forms and build a menu based on the
    entries in the registry. }
  IniFile :=
TIniFile.Create(ExtractFilePath(Application.ExeName)+cAddInIniFile);
  try
```

لیست ۲-۱۱ ادامه

```
    IniFile.ReadSectionValues(cCFRegSection, FChildFormList);
  finally
    IniFile.Free;
  end;

  { Add Menu items for each module. Note the mmMain.AutoHotKeys property must
    bet set to maAutomatic }

  for i := 0 to FChildFormList.Count - 1 do
  begin
    MenuItem := TMenuItem.Create(mmMain);
    MenuItem.Caption := FChildFormList.Names[i];
    MenuItem.OnClick := LoadChildFormOnClick;
    mmiForms.Add(MenuItem);
  end;

  // Create Separator
  MenuItem := TMenuItem.Create(mmMain);
  MenuItem.Caption := '-';
  mmiForms.Add(MenuItem);

  // Create Close Module menu item
  MenuItem := TMenuItem.Create(mmMain);
  MenuItem.Caption := '&Close Form';
  MenuItem.OnClick := CloseFormOnClick;
  MenuItem.Enabled := False;
  mmiForms.Add(MenuItem);

  { Save a reference to the index of the menu item required to
    close a child form. This will be referred to in another method. }
  FCloseFormIndex := MenuItem.MenuIndex;
end;

procedure TMainForm.LoadChildFormOnClick(Sender: TObject);
var
  ChildFormClassName: String;
  ChildFormClass: TChildFormClass;
  ChildFormName: String;
  ChildFormPackage: String;
begin

  // The menu caption represents the module name.
  ChildFormName := (Sender as TMenuItem).Caption;
  // Get the actual Package file name.
  ChildFormPackage := FChildFormList.Values[ChildFormName];
```

لیست ۲-۱۱ ادامه

```
  // Unload any previously loaded packages.
  if FCurrentModuleHandle <> 0 then
    CloseFormOnClick(nil);

  try
    // Load the specified package
    FCurrentModuleHandle := LoadPackage(ChildFormPackage);

    // Return the classname that needs to be created
    ChildFormClassName :=  GetChildFormClassName(ChildFormPackage);

    { Create an instance of the class using the FindClass() procedure. Note,
      this requires that the class already be registered with the streaming
      system using RegisterClass(). This is done in the child form
      initialization section for each child form package. }
    ChildFormClass := TChildFormClass(FindClass(ChildFormClassName));
    FChildForm := ChildFormClass.Create(self, pnlParent);
    Caption := FChildForm.GetCaption;

    { Merge child form menus with the main menu }
    if FChildForm.GetMainMenu <> nil then
      mmMain.Merge(FChildForm.GetMainMenu);

    FChildForm.Show;

    mmiForms[FCloseFormIndex].Enabled := True;
  except
    on E: Exception do
    begin
      CloseFormOnClick(nil);
      raise;
    end;
  end;
end;

function TMainForm.GetChildFormClassName(const AModuleName: String): String;
{ The Actual class name of the TChildForm implementation resides in the
  registry. This method retrieves that class name. }
var
  IniFile:  TIniFile;
begin
  IniFile :=
TIniFile.Create(ExtractFilePath(Application.ExeName)+cAddInIniFile);
  try
```

لیست ۲-۱۱ ادامه

```
    Result := IniFile.ReadString(RemoveExt(AModuleName), 'ClassName',
      EmptyStr);
  finally
    IniFile.Free;
  end;
end;

procedure TMainForm.CloseFormOnClick(Sender: TObject);
begin
  if FCurrentModuleHandle <> 0 then
  begin
    if FChildForm <> nil then
    begin
      FChildForm.Free;
      FChildForm := nil;
    end;

    // Unregister any classes provided by the module
    UnRegisterModuleClasses(FCurrentModuleHandle);
    // Unload the child form package
    UnloadPackage(FCurrentModuleHandle);

    FCurrentModuleHandle := 0;
    mmiForms[FCloseFormIndex].Enabled := False;
    Caption := FMainCaption;
  end;
end;

end.
```

منطق برنامه بسیار آسان است. این برنامه از system registery برای تشخیص این که کدام بسته‌ها در دسترس هستند، از عنوان‌های منو هنگام ساخت منو برای بارکردن هر بسته و از نام کلاس درون هر بسته استفاده می‌کند.

LoadChildFormOnClick() محلی است که بیشتر کار در آن انجام می‌شود. پس از تشخیص نام فایل بسته، متد با استفاده از تابع LoadPackage() بسته را بار می‌کند. تابع LoadPackage() اساساً چیزی شبیه LoadLibrary() در DLL ها است. متد سپس نام کلاس فرم درون بسته باردهی شده را تشخیص می‌دهد.

به منظور ایجاد یک کلاس، به مرجع کلاسی نظیر TButton یا TForm1 نیاز دارید البته این برنامه اصلی نام کلاس TChildForm های ذاتی را در برندارد به همین دلیل نام کلاس را از system registory بازیابی می‌کنیم. برنامه اصلی قادر است این نام کلاس را به تابع FindClass() ارسال کند تا یک مراجعه به کلاس برای کلاس خاصی که قبلاً به همراه سیستم جریان داده ثبت شده است ارجاع شود. به خاطر

بیاورید که این کار را در مقداردهی اولیه یونیت فرم‌های ذاتی که هنگام باردهی بسته فراخوانی می‌شد انجام دادیم. سپس کلاس را با کمک دستورات زیر ایجاد کردیم.

```
ChildFormClass := TChildFormClass(FindClass(ChildFormClassName));
FChildForm := ChildFormClass.Create(self, pnlParent);
```

تذکر

یک مراجعه به کلاس ناحیه‌ای در حافظه است که حاوی اطلاعاتی در مورد کلاس می‌باشد. این همانند تعریف نوع در کلاس است و هنگامی کلاس توسط سیستم جریان داده‌ای VCL ثبت شود در حافظه قرار می‌گیرد (هنگامی که تابع RegisterClass() فراخوانی شود) تابع FindClass() ناحیه‌ای از حافظه که مربوط به یک کلاس خاص را می‌یابد و اشاره‌گری به آن ناحیه را برمی‌گرداند. این مشابه با یک نمونه کلاس نیست. نمونه‌های کلاس اغلب هنگامی ایجاد می‌شوند که سازنده، یک تابع کلاس، فراخوانی شود.

متغیر ChildFormClass یک مرجع کلاس از پیش اعلان شده TChildForm است و قادر است به صورت چند شکل به مرجع کلاس وارث TChildForm مراجعه کند.

CloseFormOnClick() فرم فرزند را بسته و بسته آن را خالی می‌کند. بقیه کد به طور ساده کد تنظیمی برای ایجاد منوهای بسته و خواندن اطلاعات از فایل INI است. با استفاده از این تکنیک برنامه‌هایی قابل توسعه پدید می‌آیند.

صدور توابع بسته‌ها

با در دسترس بودن بسته‌هایی که در ارتباط با DLL ها هستند مشاهده می‌شود که بایستی قادر به صدور توبع و روال‌ها از بسته‌ها همانند DLL ها باشید. این کار ممکن است در این بخش نحوه استفاده از بسته‌ها به صورت مشابه شرح داده خواهد شد.

به راه انداختن یک فرم از یک تابع بسته

لیست ۳-۱۱ یونیتی است که درون یک بسته قرار دارد.

لیست ۳-۱۱ یونیت بسته به همراه دو تابع صادر شده

```
unit FunkFrm;

interface

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

لیست ۳-۱۱ ادامه

```
type

  TFunkForm = class(TForm)
    Label1: TLabel;
    Button1: TButton;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

// Declare the package functions using the StdCall calling convention
procedure FunkForm; stdcall;
function AddEm(Op1, Op2: Integer): Integer; stdcall;

// Export the functions.
exports
  FunkForm,
  AddEm;

implementation

{$R *.dfm}

procedure FunkForm;
var
  FunkForm: TFunkForm;
begin
  FunkForm := TFunkForm.Create(Application);
  try
    FunkForm.ShowModal;
  finally
    FunkForm.Free;
  end;
end;

function AddEm(Op1, Op2: Integer): Integer;
begin
  Result := Op1+Op2;
end;

end.
```

روال FuncForm() فرمی را نشان می‌دهد که به عنوان یک فرم کیفیتی در یونیت اعلان شده است. AdEm() تابعی است که دو عملوند را گرفته و مجموع آنها را برمی‌گرداند. توجه داشته باشید که توابع با استفاده از قانون فراخوانی stdCall در بخش رابط اعلان شده‌اند.

لیست ۴-۱۱ برنامه‌ای است که نحوه به کارگیری یک تابع از یک بسته را نشان می‌دهد.

لیست ۴-۱۱ برنامه Demo

```
unit MainFrm;

interface

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

const
  cFunkForm = 'FunkForm';
  cAddEm    = 'AddEm';

type
  TForm1 = class(TForm)
    btnPkgForm: TButton;
    meOp1: TMaskEdit;
    meOp2: TMaskEdit;
    btnAdd: TButton;
    lblPlus: TLabel;
    lblEquals: TLabel;
    lblResult: TLabel;
    procedure btnAddClick(Sender: TObject);
    procedure btnPkgFormClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  // Defined the method signatures
  TAddEmProc = function(Op1, Op2: Integer): integer; stdcall;
  TFunkFormProc = procedure; stdcall;

var
  Form1: TForm1;

implementation
{$R *.dfm}

procedure TForm1.btnAddClick(Sender: TObject);
var
  PackageModule: THandle;
  AddEmProc: TAddEmProc;
  Rslt: Integer;
  Op1, Op2: integer;
```

لیست ۴-۱۱ ادامه

```
begin
  PackageModule := LoadPackage('ddgPackFunk.bpl');
  try

    @AddEmProc := GetProcAddress(PackageModule, PChar(cAddEm));
    if not (@AddEmProc = nil) then
    begin
      Op1 := StrToInt(meOp1.Text);
      Op2 := StrToInt(meOp2.Text);

      Rslt := AddEmProc(Op1, Op2);
      lblResult.Caption := IntToStr(Rslt);
    end;

  finally
    UnloadPackage(PackageModule);
  end;
end;

procedure TForm1.btnPkgFormClick(Sender: TObject);
var
  PackageModule: THandle;
  FunkFormProc: TFunkFormProc;
begin
  PackageModule := LoadPackage('ddgPackFunk.bpl');
  try
    @FunkFormProc := GetProcAddress(PackageModule, PChar(cFunkForm));
    if not (@FunkFormProc = nil) then
      FunkFormProc;
  finally
    UnloadPackage(PackageModule);
  end;
end;

end.
```

ابتدا توجه داشته باشید که بایستی دو نوع روالی را اعلان کرده باشیم، TAddEmProc و TFunkFormProc . این نوع‌ها به همان صورتی اعلان می‌شوند که در بسته وجود دارند.

ابتدا btnPkgFormClick() را شرح می‌دهیم. به جای ایجاد یک فراخوانی به LoadLibrary() از LoadPackage() استفاده می‌کنیم. در واقع، LoadPackage() کار خود را با فراخوانی LoadLibrary() به اتمام می‌رساند. سپس مرجع روال را با استفاده از تابع GerProcAddress() بازیابی می‌نماییم. ثابت cFunkForm نامی مشابه با نام تابع درون بسته دارد.

حصول اطلاعاتی راجع به بسته

می‌توانیم اطلاعاتی در مورد این که کدام یونیت در بسته موجود است و کدام بسته‌ها مورد نیاز می‌باشند را پرس‌وجو نماییم. دو تابع برای انجام این کار مورد استفاده قرار می‌گیرند EndModules() و GetPackageInfo() . هر دو این توابع نیاز به توابع بازفراخوانی دارند. در لیست ۵-۱۱ استفاده از این توابع شرح داده شده است.

EnumModules() ابتدا فراخوانی شده است. این تابع بخش اجرایی و تمامی بسته‌های درگیر آن را برمی‌شمرد. تابع بازفراخوانی که به EnumModules() ارسال می‌شود EnumModuleProc() است. این تابع یک جزء TTreeView را به همراه اطلاعاتی در مورد بسته‌های برنامه مقیم می‌کند. بخش زیادی از کد مربوط به تنظیمات جزء TTreeView است. تابع GetPackageDescription() رشته توصیفی درون منبع بسته را برمی‌گرداند. فراخوانی GetPackageInfo() تابع بازفراخوانی PackageInfoProc() را گذر می‌دهد.

در PackageInfoProc() قادر به پردازش اطلاعات جدول اطلاعاتی بسته هستیم. این تابع برای یونیت درون بسته و هر بسته مورد نیاز بسته فراخوانی می‌شود. در اینجا مجدداً جزء TTreeView را به همراه این اطلاعات با بررسی مقادیر پارامتر Falgs و پارامتر NameType مقیم می‌کنیم.

لیست ۵-۱۱ دموی اطلاعات بسته

```
unit MainFrm;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ComCtrls, DBXpress, DB, SqlExpr, DBTables;

type
  TForm1 = class(TForm)
    Button1: TButton;
    TreeView1: TTreeView;
    Table1: TTable;
    SQLConnection1: TSQLConnection;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
```

لیست ۵-۱۱ ادامه

```
implementation

{$R *.dfm}

type
  TNodeHolder = class
    ContainsNode: TTreeNode;
    RequiresNode: TTreeNode;
  end;

procedure RealizeLength(var S: string);
begin
  SetLength(S, StrLen(PChar(S)));
end;

procedure PackageInfoProc(const Name: string; NameType:
  TNameType; Flags: Byte; Param: Pointer);
var
  NodeHolder: TNodeHolder;
  TempStr: String;
begin
  with Form1.TreeView1.Items do
  begin

    TempStr := EmptyStr;

    if (Flags and ufMainUnit) <> 0 then
      TempStr := 'Main unit'
    else if (Flags and ufPackageUnit) <> 0 then
      TempStr := 'Package unit' else
    if (Flags and ufWeakUnit) <> 0 then
      TempStr := 'Weak unit';

    if TempStr <> EmptyStr then
      TempStr := Format(' (%s)', [TempStr]);

    NodeHolder := TNodeHolder(Param);
    case NameType of
      ntContainsUnit: AddChild(NodeHolder.ContainsNode,
        Format('%s %s', [Name,TempStr]));
      ntRequiresPackage: AddChild(NodeHolder.RequiresNode, Name);
    end; // case
  end;
end;
```

لیست ۵-۱۱ ادامه

```
function EnumModuleProc(HInstance: integer; Data: Pointer): Boolean;
var
  ModFileName: String;
  ModNode: TTreeNode;
  ContainsNode: TTreeNode;
  RequiresNode: TTreeNode;
  ModDesc: String;
  Flags: Integer;
  NodeHolder: TNodeHolder;

begin
  with Form1.TreeView1 do
  begin
    SetLength(ModFileName, 255);
    GetModuleFileName(HInstance, PChar(ModFileName), 255);
    RealizeLength(ModFileName);
    ModNode := Items.Add(nil, ModFileName);

    ModDesc := GetPackageDescription(PChar(ModFileName));
    ContainsNode := Items.AddChild(ModNode, 'Contains');
    RequiresNode := Items.Addchild(ModNode, 'Requires');


    if ModDesc <> EmptyStr then
    begin

      NodeHolder := TNodeHolder.Create;
      try
        NodeHolder.ContainsNode := ContainsNode;
        NodeHolder.RequiresNode := RequiresNode;

        GetPackageInfo(HInstance, NodeHolder, Flags, PackageInfoProc);
      finally
        NodeHolder.Free;
      end;

      Items.AddChild(ModNode, ModDesc);

      if Flags and pfDesignOnly = pfDesignOnly then
        Items.AddChild(ModNode, 'Design-time package');
      if Flags and pfRunOnly = pfRunOnly then
        Items.AddChild (ModNode, 'Run-time package');

    end;
```

لیست ۵-۱۱ ادامه

```
  end;
  Result := True;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  EnumModules(EnumModuleProc, nil);
end;

end.
```

خلاصه

بسته‌ها بخش کلیدی معماری VCL در دلفی هستند. با فراگیری نحوه استفاده از بسته‌ها می‌توانید بهینه‌سازی‌های بسیاری در معماری و طراحی برنامه‌های خود ایجاد کنید.

فصل دوازدهم

به‌کارگیری Open Tools API

در این فصل می‌خوانید

- **رابط‌های Open Tools**
- **به‌کارگیری و روش استفاده از Open Tools API**
- **نحوهٔ طراحی Form Wizard ها**

تاکنون فکر کرده‌اید که IDE دلفی چگونه شما را یاری می‌کند. IDE محیطی است که تمامی ابزارهای ضروری جهت طراحی، اجرا و آزمایش برنامه‌های کاربردی را فراهم می‌سازد. یونیت‌ها و ابزارهای API نیز در دلفی به همین شکل می‌باشند. این ابزارها به گونه‌ای با یکدیگر ارتباط یافته‌اند که تولید برنامه آسان باشد. در این فصل یک دید کلی نسبت به API و IDE دلفی بدست آورده و خواهید دید که چه عواملی دلفی را به عنوان یک ابزار منحصر به فرد و قدرتمند در تولید نرم‌افزار مطرح نموده است.

رابط‌های Open Tools

یونیت‌های Open Tools در ۱۴ واحد مجزا ارائه شده است که هر یک با اجزاء مربوطه، امکان برقراری ارتباط با IDE دلفی را برقرار می‌سازد. قطعاً می‌دانید که اصول برنامه‌نویسی تحت ویندوز بر مبنای یونیت‌های API پایه‌ریزی شده است. این یونیت‌ها به ساده‌ترین شکل، امکان دسترسی به امکانات سایر نرم‌افزارها نظیر IDE دلفی را فراهم می‌آورد. این ابزارها به قدری انعطاف‌پذیر و قدرتمند هستند که قادرید با به کارگیری آنها اجزاء سازنده مورد نیاز برنامه‌های خود را طراحی و پیاده‌سازی نمایید. البته باید به این نکته نیز توجه داشته باشید که شما نمی‌توانید خصوصیات و یا ویژگیهای شیء‌های تعریف شده را تغییر دهید.

یادداشت

یونیتهای API تنها در نسخه‌های Professional و Enterprise دلفی وجود دارند. در صورت نیاز به دانستن کد این یونیت‌ها می‌توانید به فهرست ذیل از محل نصب نرم‌افزار دلفی رجوع کنید.

▲ **\Delphi 7\Source\ToolsAPI**

شرح این یونیتها در جدول ۱-۱۲ آورده شده است. جدول ۲-۱۲ نیز دربرگیرنده یونیت‌های قدیمی API در نسخه‌های قبلی دلفی (نظیر دلفی ۴) است.

جدول ۱-۱۲ یونیت‌های API

یونیت	شرح
ToolsAPI	این یونیت حاوی رابطهایی جهت کار با فایل سیستم، منوها و ویراستار دلفی است. همچنین رابطهای جدیدی جهت خطایابی، مدیریت فایل‌های پروژه و کنترل IDE در آن گنجانده شده است.
VCSIntf	از این یونیت جهت تعریف کلاس TIVCSClient استفاده می‌شود. این کلاس، IDE دلفی را به نرم‌افزارهای version_control مرتبط می‌کند.
DesignConst	این یونیت در برگیرنده رشته‌های کاراکتری است که به منظورهای مختلف در API استفاده می‌شود.
DesignEditors	این یونیت امکان ویرایش خصوصیات اشیاء را فراهم می‌آورد.
DesignIntf	این یونیت که در نسخه‌های قبلی دلفی تحت عنوان DsgnIntf وجود داشته است، وظیفه کنترل IDE در زمان طراحی نرم‌افزار و ارتباط آن با سایر اشیاء را به عهده دارد. در اینجا لازم است چند رابط ابزاری دیگر را معرفی نمائیم. IDE از رابط IProperty برای ویرایش خصوصیات اشیاء استفاده می‌کند. IDesigNerSelection رابطی است که برای طراحی فرمها کاربرد دارد و IDesigner یکی از متداول‌ترین رابطهایی است که توسط ویزاردهای IDE مورد استفاده قرار می‌گیرد. IDesignerNotification وظیفه اعلان پنجره‌های اخطار (مانند حذف و اضافه کردن در جداول بانک اطلاعاتی) را به عهده دارد ICustomModule و TBaseCustomModule رابطهایی هستند که از آنها برای نصب ماژول‌های قابل ویرایش در IDE استفاده می‌شود.
DesignMenus	این یونیت شامل دو یونیت دیگر به نامهای IMenuItem و IMenuItems و برخی رابطهای مورد نیاز در زمان طراحی منوهای IDE است.
DesignWindows	این یونیت برای تعریف کلاس‌هایی استفاده می‌شود که در تعیین خصوصیات ویژه اشیاء کاربرد دارند. این یونیت به صورت خودکار و توسط Object Inspector مورد استفاده قرار می‌گیرد.
TreeIntf	از این یونیت جهت تولید یک جزء سازنده از نوع Tsprig و تخصیص کلاسها و

	Interfaceها جهت پشتیبانی از این جزء سازنده استفاده می‌شود.
VCLSprigs	از این یونیت جهت پیاده‌سازی Sprig برای اجزاء سازندهٔ VCL استفاده می‌شود.
VCLEditors	از این یونیت جهت تعریف و ویرایش خصوصیتهای ویژه برای VCLها استفاده می‌شود. این کار با کمک ICustomer ProperlyDrawing و ICustomerPropertyListDrawing انجام می‌گیرد.
ClxDesignWindows	تعریف کلاس TClxDesignWindows توسط این یونیت انجام می‌شود. این کلاس از لحاظ کاربرد همانند کلاس TDesignWindows می‌باشد.
ClxEditors	از این یونیت جهت تعریف و ویرایش در خصوصیات اجزاء CLX استفاده می‌شود. به بیان دیگر کاربردی نظیر یونیت VCLEditors دارد.
ClxSprigs	از این یونیت جهت پیاده‌سازی Sprig برای اجزاء سازندهٔ CLX استفاده می‌شود.

جدول ۲-۱۲ یونیت‌های API در نسخه‌های قبلی دلفی

یونیت	شرح
FileIntf	IDE دلفی عملیات مربوط به مدیریت فایل‌ها، ویرایش اجزاء و کنترل Wizard را توسط کلاس TIVirtualFileSystem انجام می‌دهد. این کلاس به وسیله یونیت FileIntf تعریف و تعیین می‌گردد.
EditIntf	از این یونیت جهت تعویض کلاس‌های مورد لزوم برای IDE استفاده می‌شود. این کلاس‌ها منحصراً در قسمتهای Form Designer و Code Editor محیط دلفی کاربرد دارند. برخی از این کلاسها را در اینجا شرح می‌دهیم. کلاس TIEditReader امکان خواندن از حافظه موقت را فراهم می‌آورد. کلاس TIEditView امکان نمایش را میسر ساخته و TIEditInterface رابط ویرایشی را مشخص می‌سازد.
ExptIntf	از این یونیت جهت تعریف یک کلاس انتزاعی TIExpert استفاده می‌شود.
VirtIntf	از این یونیت جهت تعریف یک کلاس پایه به نام TInterface استفاده می‌شود. کلاس TIStream نیز که در اجزاء VCL کاربرد دارد، در این یونیت تعریف می‌شود.
IStreams	این یونیت در برگیرنده سه کلاس به نامهای TIMemoryStream و TIFileStream و TIVirtualStream است. در حقیقت این یونیت رابطی برای Streamهای حافظه و فایل می‌باشد.
ToolIntf	این یونیت وظیفه تعریف کلاس‌های TIMenuItemIntf و TIMainMenuIntf را به عهده دارد. عملیاتهایی نظیر ایجاد و یا اعمال تغییرات در منوهای IDE به وسیله کلاس‌های ذکر شده انجام می‌پذیرد. از این یونیت همچنین در تعریف کلاس TIAddInNotifier برای اعلان اخطار در طراحی رویدادها استفاده می‌شود.

یادداشت

ممکن است شرح کامل این یونیت‌ها غیرضروری به نظر برسد. در صورت رجوع به هر کدام از این یونیت‌ها شرح کامل و مستند شده مربوط به آنها را در محتویات آنها مشاهده خواهید کرد. ▲

استفاده از API

وقت آن رسیده است که کارآیی API را به صورت عملی نشان دهیم. بهتر است قبل از این که به ادامه این فصل بپردازید، مروری کوتاه بر مطالب ارائه شده در فصل ۸ و ۹ داشته باشید.

ایجاد یک ویزارد

آسان‌ترین روش برای پیاده‌سازی یک ویزارد در دلفی، پیاده‌سازی رابط IOTAWizard است. IOTAWizard در ذیل تعریف شده است.

```
type
  IOTAWizard = interface(IOTANotifier)
  ['{B75C0CE0-EEA6-11D1-9504-00608CCBF153}']
  { Expert UI strings }
  function GetIDString: string;
  function GetName: string;
  function GetState: TWizardState;
  { Launch the AddIn }
  procedure Execute;
end;
```

پیاده‌سازی یک ویزارد مستلزم پیاده‌سازی چندین متد با عناوین Get*XXX*() می‌باشد. این متدها اطلاعات مربوط به ویزاردها را دربرداشته و بعضی اوقات پاسخگوی مناسبی برای رویدادها می‌باشند. متد Execute() که به صورت IOTAWizard.Execute() فراخوانی می‌شود، سطوح مختلفی از فرامین پیچیده را پاسخ می‌دهد. از آنجائی که IOTAWizard رابط دیگری به نام IOTANotifier را به ارث می‌برد، می‌بایست رابط مذکور را نیز پیاده‌سازی نمود. INTANotifier در ذیل تعریف شده است.

```
type
  IOTANotifier = interface(IUnknown)
    ['{F17A7BCF-E07D-11D1-AB0B-00C04FB16FB3}']
    { This procedure is called immediately after the item is successfully
      saved. This is not called for IOTAWizards }
    procedure AfterSave;
    { This function is called immediately before the item is saved. This is not
      called for IOTAWizard }
    procedure BeforeSave;
    { The associated item is being destroyed so all references should be
      dropped. Exceptions are ignored. }
    procedure Destroyed;
    { This associated item was modified in some way. This is not called for
```

```
    IOTAWizards }
  procedure Modified;
end;
```

می‌توانید از کلاس TNotifierObject به عنوان نسخه پیاده‌سازی شده IOTANotifier استفاده کنید. IOTANotifier دارای متدهایی برای پاسخگویی به رویدادها است.

در صورتی که می‌خواهید ویزارد خود را به یکی از منوهای اصلی دلفی اضافه کنید می‌بایست رابط IOTAMenuWizard را پیاده‌سازی کنید. IOTAMenuWizard گزینه‌ای را به عنوان یک ویزارد به منوی اصلی شما می‌افزاید. این رابط به صورت زیر تعریف می‌شود.

```
type
  IOTAMenuWizard = interface(IOTAWizard)
    ['{B75C0CE2-EEA6-11D1-9504-00608CCBF153}']
    function GetMenuText: string;
  end;
```

همانطوری که ملاحظه می‌کنید، تنها منویی که باید برای IOTAMenuWizard پیاده‌سازی کنید، GetMenuText است که نام متنی را باز می‌گرداند که باید در منوی اصلی نمایش داده شود.

لیست ۱-۱۲ یونیتی به نام DumbWiz.pas را نشان می‌دهد. این یونیت کد مربوط به TDumbWizard را نشان می‌دهد.

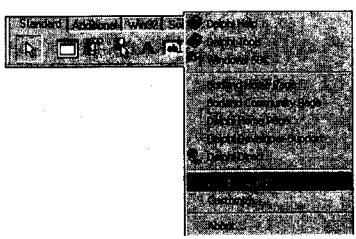
لیست ۱-۱۲ پیاده‌سازی یک ویزارد ساده

```
unit DumbWiz;

interface

uses
  ShareMem, SysUtils, Windows, ToolsAPI;

type
  TDumbWizard = class(TNotifierObject, IOTAWizard, IOTAMenuWizard)
    // IOTAWizard methods
    function GetIDString: string;
    function GetName: string;
    function GetState: TWizardState;
    procedure Execute;
    // IOTAMenuWizard method
    function GetMenuText: string;
  end;

implementation

uses Dialogs;
```

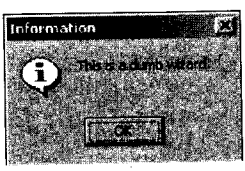
لیست ۱-۱۲ ادامه

```
function TDumbWizard.GetName: string;
begin
  Result := 'Dumb Wizard';
end;
function TDumbWizard.GetState: TWizardState;
begin
  Result := [wsEnabled];
end;

function TDumbWizard.GetIDString: String;
begin
  Result := 'DDG.DumbWizard';
end;

procedure TDumbWizard.Execute;
begin
  MessageDlg('This is a dumb wizard.', mtInformation, [mbOk], 0);
end;

function TDumbWizard.GetMenuText: string;
begin
  Result := 'Dumb Wizard';
end;

end.
```

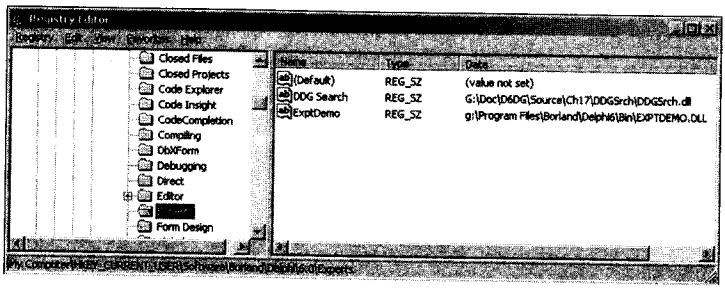
متد ()Getname وظیفه برگرداندن یک نام منحصر به فرد به عنوان نام wizard را به عهده دارد. رویه Register وظیفه فراخوانی RegisterPackageWizard را به عهده دارد. آسانترین روش برای نصب آن استفاده از گزینه Install Component از Component محیط دلفی است. این ویزارد بسیار ساده، زمانی فعال می‌شود که کاربر گزینه مربوطه را با ماوس برگزیند. به شکل ۱-۱۲ و ۲-۱۲ رجوع کنید. دانستن این نکته نیز جالب خواهد بود که در دلفی ۷ امکان پیاده‌سازی ویزاردهای نوع DLL و EXE



شکل ۱-۱۲ ویزارد Dumb به منوی اصلی افزوده شده است



شکل ۲-۱۲ ویزارد Dumb فراخوانی و فعال شده است



شکل ۳-۱۲ برنامهٔ RegEdit در ویندوز

نیز وجود دارد. نحوه تشخیص یک ویزارد DLL توسط RegistryEditor ویندوز در شکل ۳-۱۲ نمایش داده شده است.

رابط‌های یک ویزارد

منظور از طراحی و افزودن ویزارد، گسترش امکانات دلفی است. یک مثال کامل یونیت InitWiz.pas است که در لیست ۲-۱۲ آورده شده است. پس از آن به شرح قسمتهای مختلف آن می‌پردازیم تا شما را در شناسائی آن کمک کرده باشیم.

همانطور که از کد این یونیت پیداست، یونیتی متفاوت با یونیت DumbWizard می‌باشد. در این یونیت رویه‌ای به نام InitWizard() استفاده شده است. این رویه وظیفه انجام اعمال زیر را به عهده دارد:

- حاوی یک رابط به نام IOTAServices می‌باشد.
- ذخیره‌سازی اشاره‌گری که به رابط BorlandIDEServices اشاره دارد.
- تنظیمات مربوط به فایل‌های DLL و متغیرهایی که توسط IOTAServices.GetParentHandle() برگردانده می‌شود.
- ارسال یک نمونه از ویزارد ایجاد شده به رویه RegisterProc() و ثبت ویزاردی که پیاده‌سازی شده است.
- استفاده انتخابی از متد InitWizard() جهت مقداردهی اولیه و خاتمه سرویس‌دهی به یک ویزارد که پیاده‌سازی و نصب گردیده است.

نکته

فراخوانی متد initalization باید با استفاده از stdcall صورت پذیرد.

لیست ۲-۱۲ یک ویزارد از نوع DLL

```
unit InitWiz;

interface

uses Windows, ToolsAPI;

type
  TWizardWizard = class(TNotifierObject, IOTAWizard, IOTAMenuWizard)
    // IOTAWizard methods
    function GetIDString: string;
    function GetName: string;
    function GetState: TWizardState;
    procedure Execute;
    // IOTAMenuWizard method
    function GetMenuText: string;
  end;

function InitWizard(const BorlandIDEServices: IBorlandIDEServices;
  RegisterProc: TWizardRegisterProc;
  var Terminate: TWizardTerminateProc): Boolean stdcall;

var
  { Registry key where Delphi 6 wizards are kept.  EXE version uses default,
  { whereas DLL version gets key from ToolServices.GetBaseRegistryKey }
  SDelphiKey: string = '\Software\Borland\Delphi\6.0\Experts';

implementation

uses SysUtils, Forms, Controls, Main;
function TWizardWizard.GetName: string;
{ Return name of expert }
begin
  Result := 'WizardWizard';
end;

function TWizardWizard.GetState: TWizardState;
{ This expert is always enabled }
begin
  Result := [wsEnabled];
end;

function TWizardWizard.GetIDString: String;
{ "Vendor.AppName" ID string for expert }
```

لیست ۲-۱۲ ادامه

```
begin
  Result := 'DDG.WizardWizard';
end;

function TWizardWizard.GetMenuText: string;
{ Menu text for expert }
begin
  Result := 'Wizard Wizard';
end;

procedure TWizardWizard.Execute;
{ Called when expert is chosen from the main menu. }
{ This procedure creates, shows, and frees the main form. }
begin
  MainForm := TMainForm.Create(Application);
  try
    MainForm.ShowModal;
  finally
    MainForm.Free;
  end;
end;

function InitWizard(const BorlandIDEServices: IBorlandIDEServices;
  RegisterProc: TWizardRegisterProc;
  var Terminate: TWizardTerminateProc): Boolean stdcall;
var
  Svcs: IOTAServices;
begin
  Result := BorlandIDEServices <> nil;
  if Result then
  begin
    Svcs := BorlandIDEServices as IOTAServices;
    ToolsAPI.BorlandIDEServices := BorlandIDEServices;
    Application.Handle := Svcs.GetParentHandle;
    SDelphiKey := Svcs.GetBaseRegistryKey + '\Experts';
    RegisterProc(TWizardWizard.Create);
  end;
end;

end.
```

نکته

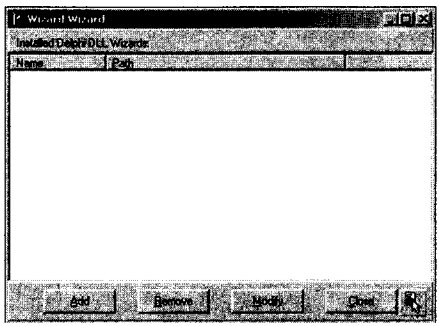
هر ویزاردی که از نوع DLL بوده و حاوی پارامترهای رشته‌ای (کاراکتری) نیز باشد، می‌باید یونیت shareMem را در کد خود داشته باشد. اگر یونیت مذکور وجود ندارد، باید آن را به کد ویزارد اضافه نمود.

رابط کاربر یک ویزارد

در مثالی که خواهیم آورد، مجدداً از متد ()Execute برای اجرا و پیاده‌سازی یک ویزارد استفاده شده است. در این مثال متد ()Execute عملیات پیچیده‌تری را انجام می‌دهد.

شکل ۴-۱۲، شکل این ویزارد و لیست ۳-۱۲ کد مربوط به این ویزارد را نمایش می‌دهد.

با استفاده از یونیت فوق که آن را ملاحظه کردید، یک رابط کاربر را برای اعمال حذف، اضافه و ویرایش ویزاردهای DLL طراحی و پیاده‌سازی نمودیم. در قسمت initialization این یونیت یک شیء از نوع Registery ساخته شده است. همانطور که می‌بینید مقدار HKEY_CURRENT_USER به خصوصیت Rootkey از DelReg اختصاص داده شده است. این مقدار معرف یک کلید در فهرست **\Software\Borland\Delphi\6.0\Experts** می‌باشد که از آن جهت پشتیبانی ویزاردهای نوع DLL استفاده می‌شود.



شکل ۴-۱۲ نمای فرم اصلی در ویزاردی که طراحی شده است

لیست ۳-۱۲ یونیت اصلی ویزارد موردنظر

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ExtCtrls, Registry, AddModU, ComCtrls, Menus;
```

لیست ۳-۱۲ ادامه

```
type
  TMainForm = class(TForm)
    TopPanel: TPanel;
    Label1: TLabel;
    BottomPanel: TPanel;
    WizList: TListView;
    PopupMenu1: TPopupMenu;
    Add1: TMenuItem;
    Remove1: TMenuItem;
    Modify1: TMenuItem;
    AddBtn: TButton;
    RemoveBtn: TButton;
    ModifyBtn: TButton;
    CloseBtn: TButton;
    procedure RemoveBtnClick(Sender: TObject);
    procedure CloseBtnClick(Sender: TObject);
    procedure AddBtnClick(Sender: TObject);
    procedure ModifyBtnClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    procedure DoAddMod(Action: TAddModAction);
    procedure RefreshReg;
  end;

var
  MainForm: TMainForm;

implementation

uses InitWiz;

{$R *.DFM}

var
  DelReg: TRegistry;

procedure TMainForm.RemoveBtnClick(Sender: TObject);
{ Handler for Remove button click. Removes selected item from registry. }
var
  Item: TListItem;
begin
  Item := WizList.Selected;
  if Item <> nil then
  begin
    if MessageDlg(Format('Remove item "%s"', [Item.Caption]), mtConfirmation,
      [mbYes, mbNo], 0) = mrYes then
      DelReg.DeleteValue(Item.Caption);
    RefreshReg;
  end;
end;
```

لیست ۳-۱۲ ادامه

```
procedure TMainForm.CloseBtnClick(Sender: TObject);
{ Handler for Close button click. Closes app. }
begin
  Close;
end;

procedure TMainForm.DoAddMod(Action: TAddModAction);
{ Adds a new expert item to registry or modifies existing one. }
var
  OrigName, ExpName, ExpPath: String;
  Item: TListItem;
begin
  if Action = amaModify then           // if modify...
  begin
    Item := WizList.Selected;
    if Item = nil then Exit;           // make sure item is selected
    ExpName := Item.Caption;           // init variables
    if Item.SubItems.Count > 0 then
      ExpPath := Item.SubItems[0];
    OrigName := ExpName;                 // save original name
  end;
  { Invoke dialog which allows user to add or modify entry }
  if AddModWiz(Action, ExpName, ExpPath) then
  begin
    { if action is Modify, and the name was changed, handle it }
    if (Action = amaModify) and (OrigName <> ExpName) then
      DelReg.RenameValue(OrigName, ExpName);
    DelReg.WriteString(ExpName, ExpPath);  // write new value
  end;
  RefreshReg;                              // update listbox
end;

procedure TMainForm.AddBtnClick(Sender: TObject);
{ Handler for Add button click }
begin
  DoAddMod(amaAdd);
end;

procedure TMainForm.ModifyBtnClick(Sender: TObject);
{ Handler for Modify button click }
begin
  DoAddMod(amaModify);
end;

procedure TMainForm.RefreshReg;
{ Refreshes listbox with contents of registry }
var
```

لیست ۳-۱۲ ادامه

```
  i: integer;
  TempList: TStringList;
  Item: TListItem;
begin
  WizList.Items.Clear;
  TempList := TStringList.Create;
  try
    { Get expert names from registry }
    DelReg.GetValueNames(TempList);
    { Get path strings for each expert name }
    for i := 0 to TempList.Count - 1 do
    begin
      Item := WizList.Items.Add;
      Item.Caption := TempList[i];
      Item.SubItems.Add(DelReg.ReadString(TempList[i]));
    end;
  finally
    TempList.Free;
  end;
end;

procedure TMainForm.FormCreate(Sender: TObject);
begin
  RefreshReg;
end;

initialization
  DelReg := TRegistry.Create;          // create registry object
  DelReg.RootKey := HKEY_CURRENT_USER; // set root key
  DelReg.OpenKey(SDelphiKey, True);    // open/create Delphi expert key
finalization
  Delreg.Free;                         // free registry object
end.
```

هنگامی که این ویزارد را اجرا می‌کنید متدهای DelReg.GetValueNames() و DelReg.ReadString() فراخوانی شده و اجزاء سازنده‌ای را به ExptList اضافه می‌نمایند. عملیات ثبت این ویزارد توسط متدهای RemoveBtnClick() و DoAddMod() انجام می‌گیرد. RemoveBtnClick() عملیات حذف عناوین مربوط به ویزارد جاری را به عهده دارد. DoAddMod() عملیات اضافه کردن یک عنوان جدید و یا ویرایش عناوین قبلی، در ویزارد را به انجام می‌رساند. اینکه عملیات مربوط به اضافه کردن یک عنوان جدید است و یا ویرایش عناوین قبلی، توسط پارامتری از نوع

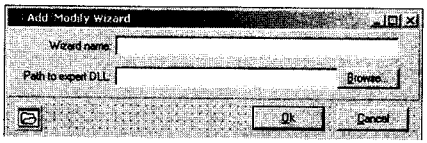
TAddModAction معین می‌گردد. این پارامتر به شکل زیر تعریف می‌شود.

```
Type
 TAddModAction = (amaAdd, amaModify);
```

اگر عنوان موردنظر از قبل وجود داشته باشد مقدار amaModify به این پارامتر ارسال شده و نمایانگر ایجاد تغییرات در ویزارد می‌باشد. اما چنانچه عنوان موردنظر موجود نباشد مقدار amaAdd به این پارامتر اختصاص داده شده و نشان دهنده اضافه نمودن یک عنوان جدید به ویزارد است.

در اینجا مثالی ارائه داده‌ایم که در آن با نمایش یک جعبه مکالمه، کاربر را در ایجاد و یا تغییر ویزارد یاری دهیم. این ویزارد در شکل ۵-۱۲ نمایش داده شده است.

لیست ۴-۱۲ کد مربوط به این ویزارد ارائه شده است.



شکل ۵-۱۲ ویزارد AddModForm

لیست ۴-۱۲ اضافه نمودن و یا اعمال تغییرات در یک ویزارد

```
unit AddModU;

interface

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

type
  TAddModAction = (amaAdd, amaModify);

  TAddModForm = class(TForm)
    OkBtn: TButton;
    CancelBtn: TButton;
    OpenDialog: TOpenDialog;
    Panel1: TPanel;
    Label1: TLabel;
    Label2: TLabel;
    PathEd: TEdit;
    NameEd: TEdit;
    BrowseBtn: TButton;
    procedure BrowseBtnClick(Sender: TObject);
  private
```

لیست ۴-۱۲ ادامه

```
    { Private declarations }
  public
    { Public declarations }
  end;

function AddModWiz(AAction: TAddModAction; var WizName,
  WizPath: String): Boolean;

implementation

{$R *.DFM}

function AddModWiz(AAction: TAddModAction; var WizName,
  WizPath: String): Boolean;
{ called to invoke dialog to add and modify registry entries }
const
  CaptionArray: array[TAddModAction] of string[31] =
    ('Add new expert', 'Modify expert');
begin
  with TAddModForm.Create(Application) do       // create dialog
  begin
    Caption := CaptionArray[AAction];            // set caption
    if AAction = amaModify then                  // if modify...
    begin
      NameEd.Text := WizName;                   // init name and
      PathEd.Text := WizPath;                   // path
    end;
    Result := ShowModal = mrOk;                 // show dialog
    if Result then                              // if Ok...
    begin
      WizName := NameEd.Text;                   // set name and
      WizPath := PathEd.Text;                   // path
    end;
    Free;
  end;
end;

procedure TAddModForm.BrowseBtnClick(Sender: TObject);
begin
  if OpenDialog.Execute then
    PathEd.Text := OpenDialog.FileName;
end;


end.
```

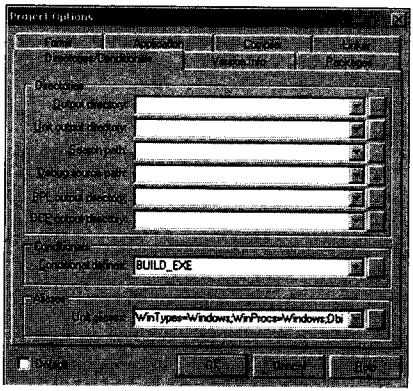
ویزارد EXE و DLL

همانطور که خواهید دید در دلفی ۷ امکان ساخت دو نوع ویزارد از یک قطعه کد وجود دارد. این بدان معنی است که با طراحی و پیاده‌سازی روتین‌های مربوط به یک ویزارد قادر خواهید بود آن را در دو حالت DLL و EXE کامپایل نمایید. فایل WizWiz.dpr نمونه‌ای از این کار را نشان می‌دهد. این فایل در لیست ۵-۱۲ ارائه گردیده است.

همانطور که از کد این فایل نمایان است، در صورتی که مقدار BUILD_EXE به عنوان شرط پیش‌فرض تعیین شده باشد، برنامه مذکور یک ویزارد از نوع EXE را تولید خواهد کرد و در غیر این صورت خروجی این برنامه ویزاردی از نوع DLL می‌باشد. شرطهای پیش‌فرض برای این کار را می‌توانید به وسیلهٔ تب Directories/Conditionals از منوی Project Options پیاده‌سازی نمایید. شکل ۶-۱۲ نحوه انجام این کار را نمایش دهید.

لیست ۵-۱۲ فایل اصلی مربوط به پروژه wizwiz

```
{$ifdef BUILD_EXE}
program WizWiz;      // Build as EXE
{$else}
library WizWiz;      // Build as DLL
{$endif}

uses
{$ifndef BUILD_EXE}
  ShareMem,                  // ShareMem required for DLL
  InitWiz in 'InitWiz.pas',  // Wizard stuff
{$endif}
  ToolsAPI,
  Forms,
  Main in 'Main.pas' {MainForm},
  AddModU in 'AddModU.pas' {AddModForm};

{$ifdef BUILD_EXE}
{$R *.RES}                                  // required for EXE
{$else}
exports                                     // required for DLL
  InitWizard name WizardEntryPoint;         // required entry point
{$endif}

begin
{$ifdef BUILD_EXE}                          // required for EXE...
  Application.Initialize;
  Application.CreateForm(TMainForm, MainForm);
  Application.Run;
{$endif}
end.
```



شکل ۶-۱۲ کادر مکالمه‌ای Project Options

یادداشت

توجه داشته باشید که در دلفی ۷ فایلی با عنوان ToolsAPI.dcu وجود ندارد. این نکته را به این جهت یادآوری می‌کنیم که بدانید در خود ویزاردهای DLL و EXE ارجاعی به ToolsAPI صورت می‌گیرد و این امر تنها با Packageها قابل اجرا می‌باشد. یعنی بدون Package قادر به ساختن ویزارد نمی‌باشد. ▲

در این بخش قصد داریم ویزارد دیگری را طراحی کنیم که در عملیات جستجو مورد استفاده قرار می‌گیرد. نام این ویزارد را DDGSearch خواهیم گذاشت.

در لیست ۶-۱۲ یونیتی به نام InitWiz.pas ارائه شده است که وظیفه مرتبط کردن ویزارد DDGSearch با IDE را به عهده دارد. این یونیت، شباهت زیادی با یونیت‌های قبل دارد، اما روش فراخوانی متد ()Execute در آن با سایر یونیت‌ها متفاوت است.

لیست ۶-۱۲ یونیت Initwiz.pas مربوط به ویزارد DDGSearch

```
unit InitWiz;

interface
uses
  Windows, ToolsAPI;

type
  TSearchWizard = class(TNotifierObject, IOTAWizard, IOTAMenuWizard)
    // IOTAWizard methods
    function GetIDString: string;
    function GetName: string;
    function GetState: TWizardState;
    procedure Execute;
    // IOTAMenuWizard method
```

لیست ۶-۱۲ ادامه

```
    function GetMenuText: string;
  end;

function InitWizard(const BorlandIDEServices: IBorlandIDEServices;
  RegisterProc: TWizardRegisterProc;
  var Terminate: TWizardTerminateProc): Boolean stdcall;

var
  ActionSvc: IOTAActionServices;

implementation

uses SysUtils, Dialogs, Forms, Controls, Main, PriU;

function TSearchWizard.GetName: string;
{ Return name of expert }
begin
  Result := 'DDG Search';
end;

function TSearchWizard.GetState: TWizardState;
{ This expert is always enabled on the menu }
begin
  Result := [wsEnabled];
end;

function TSearchWizard.GetIDString: String;
{ Return the unique Vendor.Product name of expert }
begin
  Result := 'DDG.DDGSearch';
end;

function TSearchWizard.GetMenuText: string;
{ Return text for Help menu }
begin
  Result := 'DDG Search Expert';
end;

procedure TSearchWizard.Execute;
{ Called when expert name is selected from Help menu of IDE. }
{ This function invokes the expert }
begin
  // if not created, created it and show it
  if MainForm = nil then
  begin
    MainForm := TMainForm.Create(Application);
    ThreadPriWin := TThreadPriWin.Create(Application);
    MainForm.Show;
  end
```

لیست ۶-۱۲ ادامه

```
  else
  // if created then restore window and show it
    with MainForm do
    begin
      if not Visible then Show;
      if WindowState = wsMinimized then WindowState := wsNormal;
      SetFocus;
    end;
end;

function InitWizard(const BorlandIDEServices: IBorlandIDEServices;
  RegisterProc: TWizardRegisterProc;
  var Terminate: TWizardTerminateProc): Boolean stdcall;
var
  Svcs: IOTAServices;
begin
  Result := BorlandIDEServices <> nil;
  if Result then
  begin
    Svcs := BorlandIDEServices as IOTAServices;
    ActionSvc := BorlandIDEServices as IOTAActionServices;
    ToolsAPI.BorlandIDEServices := BorlandIDEServices;
    Application.Handle := Svcs.GetParentHandle;
    RegisterProc(TSearchWizard.Create);
  end;
end;

end.
```

به این نکته باید توجه کنید که گاهی اوقات ممکن است در طراحی و یا ایجاد یک فرم متغیرهای زیادی استفاده شوند که باعث سردرگمی شما در طول ایجاد برنامه‌هایتان گردند. راه‌حل منطقی برای این روش تخصیص مقدار nil به آنها، آن هم در ابتدای برنامه است.

اما فایل پروژه مربوط به یونیت را چگونه طراحی کنیم؟ لیست ۷-۱۲ نحوه این کار را نمایش می‌دهد.

لیست ۷-۱۲ فایل پروژه DDGSearch.dpr

```
{$IFDEF BUILD_EXE}
program DDGSrch;
{$ELSE}
library DDGSrch;
{$ENDIF}

uses
{$IFDEF BUILD_EXE}
  Forms,
{$ELSE}
  ShareMem,
```

لیست ۷-۱۲ ادامه

```
  ToolsAPI,
  InitWiz in 'InitWiz.pas',
{$ENDIF}
  Main in 'MAIN.PAS' {MainForm},
  SrchIni in 'SrchIni.pas',
  SrchU in 'SrchU.pas',
  PriU in 'PriU.pas' {ThreadPriWin},
  MemMap in '..\..\Utils\MemMap.pas',
  DDGStrUtils in '..\..\Utils\DDGStrUtils.pas';

{$R *.RES}

{$IFNDEF BUILD_EXE}
exports
  { Entry point which is called by Delphi IDE }
  InitWizard name WizardEntryPoint;
{$ENDIF}

begin
{$IFDEF BUILD_EXE}
  Application.Initialize;
  Application.CreateForm(TMainForm, MainForm);
  Application.Run;
{$ENDIF}
end.
```

دوباره ملاحظه می‌کنید در این پروژه هم امکان ایجاد ویزارد DLL یا EXE فراهم شده است. همانطور که چندی پیش یادآور شدیم، می‌توانید تمامی متغیرهای فرم اصلی این ویزارد را در هنگام اجرا برابر با nil در نظر گرفته و در هنگام اتمام کار نیز مقادیر آنها را با nil جایگزین نمایید. این کار ضریب اطمینان برنامه‌تان را افزایش می‌دهد. متد زیر شما را در انجام این عملیات کمک خواهد کرد.

لازم است به روتین زیر توجه کنید. زمانی که این ویزارد را کامپایل می‌کنید متد FileLBDblClick() دو متد دیگر به نامهای OpenFile() و OpenProject() را فراخوانی خواهد نمود تا توسط آن امکان باز نمودن یک یونیت یا فایل پروژه میسر گردد.

```
procedure TMainForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  Action := caFree;
  Application.OnShowHint := FOldShowHint;
  MainForm := nil;
end;
procedure TMainForm.FileLBDblClick(Sender: TObject);
{ Called when user double-clicks in listbox. Loads file into IDE }
var
  FileName: string;
```

```
  Len: Integer;
begin
  { make sure user clicked on a file... }
  if Integer(FileLB.Items.Objects[FileLB.ItemIndex]) > 0 then
  begin
    FileName := FileLB.Items[FileLB.ItemIndex];
    { Trim "File " and ":" from string }
    FileName := Copy(FileName, 6, Length(FileName));
    Len := Length(FileName);
    if FileName[Len] = ':' then SetLength(FileName, Len - 1);
    { Open the project or file }
{$IFNDEF BUILD_EXE}
    if CompareText(ExtractFileExt(FileName), '.DPR') = 0 then
      ActionSvc.OpenProject(FileName, True)
    else
      ActionSvc.OpenFile(FileName);
{$ELSE}
    ShellExecute(0, 'open', PChar(FileName), nil, nil, SW_SHOWNORMAL);
{$ENDIF}
  end;
end;
```

در قطعه کد بالا روتینی به نام ShellExecute() را مشاهده می‌کنید. این روتین یکی از توابع API است که وظیفه باز کردن فایل‌ها را به عهده دارد.

در لیست ۸-۱۲ کد کامل مربوط به ویزارد DDGSearch ارائه شده است. نمای کلی این ویزارد را در شکل ۷-۱۲ مشاهده خواهید نمود.

لیست ۸-۱۲ ویزارد DDGSearch

```
unit Main;

interface

{$WARN UNIT_PLATFORM OFF}

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, Buttons, ExtCtrls, Menus, SrchIni,
  SrchU, ComCtrls;

type
  TMainForm = class(TForm)
    FileLB: TListBox;
    PopupMenu1: TPopupMenu;
    Font1: TMenuItem;
    N1: TMenuItem;
    Exit1: TMenuItem;
    FontDialog1: TFontDialog;
```

لیست ۸-۱۲ ادامه

```
    StatusBar: TStatusBar;
    AlignPanel: TPanel;
    ControlPanel: TPanel;
    ParamsGB: TGroupBox;
    LFileSpec: TLabel;
    LToken: TLabel;
    lPathName: TLabel;
    EFileSpec: TEdit;
    EToken: TEdit;
    PathButton: TButton;
    OptionsGB: TGroupBox;
    cbCaseSensitive: TCheckBox;
    cbFileNamesOnly: TCheckBox;
    cbRecurse: TCheckBox;
    SearchButton: TBitBtn;
    CloseButton: TBitBtn;
    PrintButton: TBitBtn;
    PriorityButton: TBitBtn;
    View1: TMenuItem;
    EPathName: TEdit;
    procedure SearchButtonClick(Sender: TObject);
    procedure PathButtonClick(Sender: TObject);
    procedure FileLBDrawItem(Control: TWinControl; Index: Integer;
      Rect: TRect; State: TOwnerDrawState);
    procedure Font1Click(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure PrintButtonClick(Sender: TObject);
    procedure CloseButtonClick(Sender: TObject);
    procedure FileLBDblClick(Sender: TObject);
    procedure FormResize(Sender: TObject);
    procedure PriorityButtonClick(Sender: TObject);
    procedure ETokenChange(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private
    FOldShowHint: TShowHintEvent;
    procedure ReadIni;
    procedure WriteIni;
    procedure DoShowHint(var HintStr: string; var CanShow: Boolean;
      var HintInfo: THintInfo);
  protected
    procedure WndProc(var Message: TMessage); override;
  public
    Running: Boolean;
    SearchPri: integer;
    SearchThread: TSearchThread;
    procedure EnableSearchControls(Enable: Boolean);
  end;
```

لیست ۸-۱۲ ادامه

```
var
  MainForm: TMainForm;

implementation
{$R *.DFM}

uses Printers, ShellAPI, MemMap, FileCtrl, PriU;

procedure PrintStrings(Strings: TStrings);
{ This procedure prints all of the string in the Strings parameter }
var
  Prn: TextFile;
  i: word;
begin
  if Strings.Count = 0 then // Are there strings?
  begin
    MessageDlg('No text to print!', mtInformation, [mbOk], 0);
    Exit;
  end;
  AssignPrn(Prn);                          // assign Prn to printer
  try
    Rewrite(Prn);                          // open printer
    try
      for i := 0 to Strings.Count - 1 do   // iterate over all strings
        WriteLn(Prn, Strings.Strings[i]);  // write to printer
    finally
      CloseFile(Prn);                      // close printer
    end;
  except
    on EInOutError do
      MessageDlg('Error Printing text.', mtError, [mbOk], 0);
  end;
end;

procedure TMainForm.EnableSearchControls(Enable: Boolean);
{ Enables or disables certain controls so options can't be modified }
{ while search is executing. }
begin
  SearchButton.Enabled := Enable;        // enabled/disable proper controls
  cbRecurse.Enabled := Enable;
  cbFileNamesOnly.Enabled := Enable;
  cbCaseSensitive.Enabled := Enable;
  PathButton.Enabled := Enable;
  EPathName.Enabled := Enable;
  EFileSpec.Enabled := Enable;
  EToken.Enabled := Enable;
  Running := not Enable;                 // set Running flag
  ETokenChange(nil);
```

لیست ۸-۱۲ ادامه

```
  with CloseButton do
  begin
    if Enable then
    begin                  // set props of Close/Stop button
      Caption := '&Close';
      Hint := 'Close Application';
    end
    else begin
      Caption := '&Stop';
      Hint := 'Stop Searching';
    end;
  end;
end;

procedure TMainForm.SearchButtonClick(Sender: TObject);
{ Called when Search button is clicked.  Invokes search thread. }
begin
  EnableSearchControls(False);          // disable controls
  FileLB.Clear;                         // clear listbox
  { start thread }
  SearchThread := TSearchThread.Create(cbCaseSensitive.Checked,
    cbFileNamesOnly.Checked, cbRecurse.Checked, EToken.Text,
    EPathName.Text, EFileSpec.Text, Handle);
end;

procedure TMainForm.ETokenChange(Sender: TObject);
begin
  SearchButton.Enabled := not Running and (EToken.Text <> '');
end;

procedure TMainForm.PathButtonClick(Sender: TObject);
{ Called when Path button is clicked.  Allows user to choose new path. }
var
  ShowDir: string;
begin
  ShowDir := EPathName.Text;
  if SelectDirectory(ShowDir, [], 0) then
    EPathName.Text := ShowDir;
end;

procedure TMainForm.FileLBDblClick(Sender: TObject);
{ Called when user double-clicks in listbox. Loads file into IDE }
var
  FileName: string;
  Len: Integer;
```

لیست ۸-۱۲ ادامه

```
begin
  { make sure user clicked on a file... }
  if Integer(FileLB.Items.Objects[FileLB.ItemIndex]) > 0 then
  begin
    FileName := FileLB.Items[FileLB.ItemIndex];
    { Trim "File " and ":" from string }
    FileName := Copy(FileName, 6, Length(FileName));
    Len := Length(FileName);
    if FileName[Len] = ':' then SetLength(FileName, Len - 1);
    { Open the project or file }
{$IFNDEF BUILD_EXE}
    if CompareText(ExtractFileExt(FileName), '.DPR') = 0 then
      ActionSvc.OpenProject(FileName, True)
    else
      ActionSvc.OpenFile(FileName);
{$ELSE}
    ShellExecute(0, 'open', PChar(FileName), nil, nil, SW_SHOWNORMAL);
{$ENDIF}
  end;
end;

procedure TMainForm.FileLBDrawItem(Control: TWinControl;
  Index: Integer; Rect: TRect; State: TOwnerDrawState);
{ Called in order to owner draw listbox. }
var
  CurStr: string;
begin
  with FileLB do
  begin
    CurStr := Items.Strings[Index];
    Canvas.FillRect(Rect);                    // clear out rect
    if not cbFileNamesOnly.Checked then       // if not filename only...
    begin
      { if current line is file name... }
      if Integer(Items.Objects[Index]) > 0 then
        Canvas.Font.Style := [fsBold]; // bold font
      end
    else
      Rect.Left := Rect.Left + 15;            // otherwise, indent
    DrawText(Canvas.Handle, PChar(CurStr), Length(CurStr), Rect,
dt_SingleLine);
  end;
end;
```

لیست ۸-۱۲ ادامه

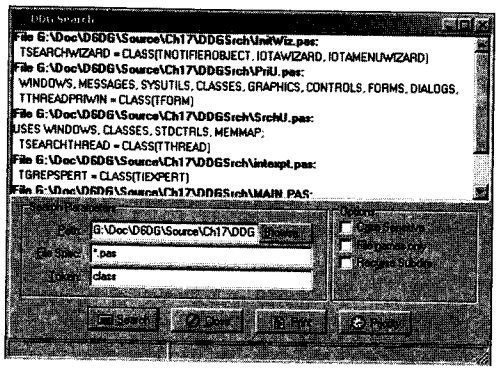
```
procedure TMainForm.Font1Click(Sender: TObject);
{ Allows user to pick new font for listbox }
begin
  { Pick new listbox font }
  if FontDialog1.Execute then
    FileLB.Font := FontDialog1.Font;
end;


procedure TMainForm.FormDestroy(Sender: TObject);
{ OnDestroy event handler for form }
begin
  WriteIni;
end;

procedure TMainForm.FormCreate(Sender: TObject);
{ OnCreate event handler for form }
begin
  Application.HintPause := 0;             // don't wait to show hints
  FOldShowHint := Application.OnShowHint; // set up hints
  Application.OnShowHint := DoShowHint;
  ReadIni;                                // read reg INI file
end;

procedure TMainForm.DoShowHint(var HintStr: string; var CanShow: Boolean;
  var HintInfo: THintInfo);
{ OnHint event handler for Application }
begin
  { Display application hints on status bar }
  StatusBar.Panels[0].Text := HintStr;
  { Don't show tool tip if we're over our own controls }
  if (HintInfo.HintControl <> nil) and
    (HintInfo.HintControl.Parent <> nil) and
    ((HintInfo.HintControl.Parent = ParamsGB) or
    (HintInfo.HintControl.Parent = OptionsGB) or
    (HintInfo.HintControl.Parent = ControlPanel)) then
    CanShow := False;
  if Assigned(FOldShowHint) then
    FOldShowHint(HintStr, CanShow, HintInfo);
end;

procedure TMainForm.PrintButtonClick(Sender: TObject);
{ Called when Print button is clicked. }
begin
  if MessageDlg('Send search results to printer?', mtConfirmation,
```

لیست ۸-۱۲ ادامه

```
    [mbYes, mbNo], 0) = mrYes then
    PrintStrings(FileLB.Items);
end;

procedure TMainForm.CloseButtonClick(Sender: TObject);
{ Called to stop thread or close application }
begin
  // if thread is running then terminate thread
  if Running then SearchThread.Terminate
  // otherwise close app
  else Close;
end;


procedure TMainForm.FormResize(Sender: TObject);
{ OnResize event handler. Centers controls in form. }
begin
 { divide status bar into two panels with a 1/3 - 2/3 split }
  with StatusBar do
  begin
    Panels[0].Width := Width div 3;
    Panels[1].Width := Width * 2 div 3;
  end;
  { center controls in the middle of the form }
  ControlPanel.Left := (AlignPanel.Width div 2) - (ControlPanel.Width div 2);
end;

procedure TMainForm.PriorityButtonClick(Sender: TObject);
{ Show thread priority form }
begin
  ThreadPriWin.Show;
end;

procedure TMainForm.ReadIni;
{ Reads default values from Registry }
begin
  with SrchIniFile do
  begin
    EPathName.Text := ReadString('Defaults', 'LastPath', 'C:\');
    EFileSpec.Text := ReadString('Defaults', 'LastFileSpec', '*.*');
    EToken.Text := ReadString('Defaults', 'LastToken', '');
    cbFileNamesOnly.Checked := ReadBool('Defaults', 'FNamesOnly', False);
    cbCaseSensitive.Checked := ReadBool('Defaults', 'CaseSens', False);
    cbRecurse.Checked := ReadBool('Defaults', 'Recurse', False);
```

لیست ۸-۱۲ ادامه

```
    Left := ReadInteger('Position', 'Left', 100);
    Top := ReadInteger('Position', 'Top', 50);
    Width := ReadInteger('Position', 'Width', 510);
    Height := ReadInteger('Position', 'Height', 370);
  end;
end;

procedure TMainForm.WriteIni;
{ writes current settings back to Registry }
begin
  with SrchIniFile do
  begin
    WriteString('Defaults', 'LastPath', EPathName.Text);
    WriteString('Defaults', 'LastFileSpec', EFileSpec.Text);
    WriteString('Defaults', 'LastToken', EToken.Text);
    WriteBool('Defaults', 'CaseSens', cbCaseSensitive.Checked);
    WriteBool('Defaults', 'FNamesOnly', cbFileNamesOnly.Checked);
    WriteBool('Defaults', 'Recurse', cbRecurse.Checked);
    WriteInteger('Position', 'Left', Left);
    WriteInteger('Position', 'Top', Top);
    WriteInteger('Position', 'Width', Width);
    WriteInteger('Position', 'Height', Height);
  end;
end;

procedure TMainForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  Action := caFree;
  Application.OnShowHint := FOldShowHint;
  MainForm := nil;
end;

procedure TMainForm.WndProc(var Message: TMessage);
begin
  if Message.Msg = DDGM_ADDSTR then
  begin
    FileLB.Items.AddObject(PChar(Message.WParam), TObject(Message.LParam));
    StrDispose(PChar(Message.WParam));
  end
  else
    inherited WndProc(Message);
end;

end.
```



شکل ۷-۱۲ اجرای ویزارد DDGSearch

Form Wizards

ویزاردها می‌توانند بر خستگی حاصل از نوشتن روتین‌ها غلبه کنند و برنامه‌سازانی که به دنبال امکانات جدید دلفی هستند را در پشت سر گذاشتن این مانع یاری نمایند.

ابزار دیگری در دلفی ۷ وجود دارد که امکان طراحی و پیاده‌سازی Form Wizard را نیز فراهم می‌آورد. نکتهٔ حائز اهمیت در طراحی Form Wizard ها پیاده‌سازی متدهای ارتباطی قوی است. چنانچه این متدها به درستی طرح‌ریزی شوند، ویزاردهای ساخته شده نیز قابلیت اجرایی بالائی خواهند داشت. برای ایجاد Form Wizard پنج گام اساسی لازم است:

۱- ایجاد یک کلاس از اجزاء سازندهٔ TCustomForm ، TDataModule و TWinControl که از آن به عنوان یک کلاس پایه برای سایر کلاسها استفاده می‌شود.

۲- ایجاد یک TNotifierObject که از آن جهت پیاده‌سازی IoTAWizard ، IOTARepositoryWizard ، IOTACreator و IOTAModuleCreator استفاده می‌شود.

۳- IOTAModuleServices.GetNewModuleAndClassName() متدی است که وظیفهٔ تعریف یک یونیت و کلاس را برای یک ویزارد به عهده دارد. IOTAModuleServies.CreateModule() متدی است که IDE دلفی را در ایجاد ماژول جدیدی یاری می‌کند، در این گام لازم است تا دو متد فوق را در متد IOTAWizard.Execute() اجرا نمائیم.

۴- فراخوانی متدهای NewFormFile() و NewImpFile() که وظیفهٔ ارجاع کدهای مربوط به یونیت و فرم برنامه را به عهده دارند.

۵- تکمیل روتین Register() به وسیلهٔ متد RegisterCustomModule() و ایجاد ویزارد موردنظر توسط فراخوانی روتین RegisterPackageWizard() .

ABWizard.pas که کد مربوط به یک ویزارد AppBar است در لیست ۹-۱۲ ارائه شده است.

در اینجا می‌خواهیم مثال فوق را با استفاده از فایل‌های Resource بسازیم. بیشتر اوقات ویزاردها

لیست ۹-۱۲ یونیت مربوط به پیاده‌سازی ویزارد AppBar

```
unit ABWizard;

interface

uses Windows, Classes, ToolsAPI;

type
  TAppBarWizard = class(TNotifierObject, IOTAWizard, IOTARepositoryWizard,
    IOTAFormWizard, IOTACreator, IOTAModuleCreator)
  private
    FUnitIdent: string;
    FClassName: string;
    FFileName: string;
  protected
    // IOTAWizard methods
    function GetIDString: string;
    function GetName: string;
    function GetState: TWizardState;
    procedure Execute;
    // IOTARepositoryWizard / IOTAFormWizard methods
    function GetAuthor: string;
    function GetComment: string;
    function GetPage: string;
    function GetGlyph: HICON;
    // IOTACreator methods
    function GetCreatorType: string;
    function GetExisting: Boolean;
    function GetFileSystem: string;
    function GetOwner: IOTAModule;
    function GetUnnamed: Boolean;
    // IOTAModuleCreator methods
    function GetAncestorName: string;
    function GetImplFileName: string;
    function GetIntfFileName: string;
    function GetFormName: string;
    function GetMainForm: Boolean;
    function GetShowForm: Boolean;
    function GetShowSource: Boolean;
    function NewFormFile(const FormIdent, AncestorIdent: string): IOTAFile;
    function NewImplSource(const ModuleIdent, FormIdent,
      AncestorIdent: string): IOTAFile;
    function NewIntfSource(const ModuleIdent, FormIdent,
      AncestorIdent: string): IOTAFile;
    procedure FormCreated(const FormEditor: IOTAFormEditor);
  end;

implementation

uses Forms, AppBars, SysUtils, DsgnIntf;
```

لیست ۹-۱۲ ادامه

```
{$R CodeGen.res}

type
  TBaseFile = class(TInterfacedObject)
  private
    FModuleName: string;
    FFormName: string;
    FAncestorName: string;
  public
    constructor Create(const ModuleName, FormName, AncestorName: string);
  end;

  TUnitFile = class(TBaseFile, IOTAFile)
  protected
    function GetSource: string;
    function GetAge: TDateTime;
  end;
  TFormFile = class(TBaseFile, IOTAFile)
  protected
    function GetSource: string;
    function GetAge: TDateTime;
  end;

{ TBaseFile }

constructor TBaseFile.Create(const ModuleName, FormName,
  AncestorName: string);
begin
  inherited Create;
  FModuleName := ModuleName;
  FFormName := FormName;
  FAncestorName := AncestorName;
end;

{ TUnitFile }

function TUnitFile.GetSource: string;
var
  Text: string;
  ResInstance: THandle;
  HRes: HRSRC;
begin
  ResInstance := FindResourceHInstance(HInstance);
  HRes := FindResource(ResInstance, 'CODEGEN', RT_RCDATA);
  Text := PChar(LockResource(LoadResource(ResInstance, HRes)));
  SetLength(Text, SizeOfResource(ResInstance, HRes));
  Result := Format(Text, [FModuleName, FFormName, FAncestorName]);
end;
```

لیست ۹-۱۲ ادامه

```
function TUnitFile.GetAge: TDateTime;
begin
  Result := -1;
end;

{ TFormFile }

function TFormFile.GetSource: string;
const
  FormText =
    'object %0:s: T%0:s'#13#10'end';
begin
  Result := Format(FormText, [FFormName]);
end;

function TFormFile.GetAge: TDateTime;
begin
  Result := -1;
end;

{ TAppBarWizard }

{ TAppBarWizard.IOTAWizard }

function TAppBarWizard.GetIDString: string;
begin
  Result := 'DDG.AppBarWizard';
end;

function TAppBarWizard.GetName: string;
begin
  Result := 'DDG AppBar Wizard';
end;

function TAppBarWizard.GetState: TWizardState;
begin
  Result := [wsEnabled];
end;

procedure TAppBarWizard.Execute;
begin
  (BorlandIDEServices as IOTAModuleServices).GetNewModuleAndClassName(
    'AppBar', FUnitIdent, FClassName, FFileName);
  (BorlandIDEServices as IOTAModuleServices).CreateModule(Self);
end;

{ TAppBarWizard.IOTARepositoryWizard / TAppBarWizard.IOTAFormWizard }
```

لیست ۹-۱۲ ادامه

```
function TAppBarWizard.GetGlyph: HICON;
begin
  Result := 0;  // use standard icon
end;

function TAppBarWizard.GetPage: string;
begin
  Result := 'DDG';
end;

function TAppBarWizard.GetAuthor: string;
begin
  Result := 'Delphi 5 Developer''s Guide';
end;

function TAppBarWizard.GetComment: string;
begin
  Result := 'Creates a new AppBar form.'
end;

{ TAppBarWizard.IOTACreator }

function TAppBarWizard.GetCreatorType: string;
begin
  Result := '';
end;

function TAppBarWizard.GetExisting: Boolean;
begin
  Result := False;
end;

function TAppBarWizard.GetFileSystem: string;
begin
  Result := '';
end;

function TAppBarWizard.GetOwner: IOTAModule;
var
  I: Integer;
  ModServ: IOTAModuleServices;
  Module: IOTAModule;
  ProjGrp: IOTAProjectGroup;
begin
  Result := nil;
  ModServ := BorlandIDEServices as IOTAModuleServices;
```

لیست ۹-۱۲ ادامه

```
      for I := 0 to ModServ.ModuleCount - 1 do
      begin
        Module := ModSErv.Modules[I];
        // find current project group
    if CompareText(ExtractFileExt(Module.FileName), '.bpg') = 0 then
      if Module.QueryInterface(IOTAProjectGroup, ProjGrp) = S_OK then
      begin
        // return active project of group
        Result := ProjGrp.GetActiveProject;
        Exit;
      end;
  end;
end;

function TAppBarWizard.GetUnnamed: Boolean;
begin
  Result := True;
end;

{ TAppBarWizard.IOTAModuleCreator }

function TAppBarWizard.GetAncestorName: string;
begin
  Result := 'TAppBar';
end;

function TAppBarWizard.GetImplFileName: string;
var
  CurrDir: array[0..MAX_PATH] of char;
begin
  // Note: full path name required!
  GetCurrentDirectory(SizeOf(CurrDir), CurrDir);
  Result := Format('%s\%s.pas', [CurrDir, FUnitIdent, '.pas']);
end;

function TAppBarWizard.GetIntfFileName: string;
begin
  Result := '';
end;

function TAppBarWizard.GetFormName: string;
begin
  Result := FClassName;
end;

function TAppBarWizard.GetMainForm: Boolean;
begin
```

لیست ۹-۱۲ ادامه

```
  Result := False;
end;
function TAppBarWizard.GetShowForm: Boolean;
begin
  Result := True;
end;

function TAppBarWizard.GetShowSource: Boolean;
begin
  Result := True;
end;

function TAppBarWizard.NewFormFile(const FormIdent,
  AncestorIdent: string): IOTAFile;
begin
  Result := TFormFile.Create('', FormIdent, AncestorIdent);
end;

function TAppBarWizard.NewImplSource(const ModuleIdent, FormIdent,
  AncestorIdent: string): IOTAFile;
begin
  Result := TUnitFile.Create(ModuleIdent, FormIdent, AncestorIdent);
end;

function TAppBarWizard.NewIntfSource(const ModuleIdent, FormIdent,
  AncestorIdent: string): IOTAFile;
begin
  Result := nil;
end;

procedure TAppBarWizard.FormCreated(const FormEditor: IOTAFormEditor);
begin
  // do nothing
end;

end.
```

دارای فرامین مشابه و یکسان می‌باشند. مناسب‌ترین روش برای تعریف فرامین جهت تولید روتین‌ها، استفاده از یک رشتهٔ پارامتری در یک فایل Resource است.

ابتدا یک فایل متنی متشکل از فرامین پارامتری ایجاد کرده و بعد از آن فایل Resource تعریف می‌کنیم تا به کامپایلر فرمان دهد که فایل RES را تولید کند. پس از آن از BRCC32.EXE برای کامپایلر کردن Resoure استفاده می‌کنیم. در لیست ۱۲-۱۰ CodeGen.txt و در لیست ۱۲-۱۱ CodeGen.rc ارائه شده است.

لیست ۱۰-۱۲ یک Resource برای ویزارد AppBar

```
unit %0:s;

interface

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

type
  T%1:s = class(%2:s)
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  %1:s: T%1:s;

implementation

{$R *.DFM}

end.
```

در مثال فوق نام یونیت با %0:S جایگزین شده است.

لیست ۱۱-۱۲ CODEGEN.RC

```
CODEGEN RCDATA CODEGEN.TXT
```

خلاصه

در این فصل شیوه استفاده از رابطهای موجود در Open Tools API را فراگرفتید و با شیوه نوشتن ویراستار اجزاء ساخت آشنا شدید. در فصل آینده با امکانات قدرتمندتر و هیجان‌انگیزتری آشنا خواهید شد که سبب جدا شدن شما از دیگر برنامه‌نویسان عادی خواهد شد. با درک محتوای ابزارهایی همچون COM+ و MTS در زمره برنامه‌نویسان حرفه‌ای قرار خواهید گرفت.

فصل سیزدهم

استفاده از سرویس‌های COM+/MTS در طراحی برنامه‌های تراکنشی

در این فصل می‌خوانید

- مفاهیم COM+
- ضرورت استفاده از COM
- سرویس‌های COM+
- Rumtime
- ایجاد برنامه‌های کاربردی COM+
- COM+ و دلفی

در این فصل شما را با اجزاء سازندهٔ COM+ در دلفی آشنا خواهیم ساخت. امروزه COM+ به عنوان جزئی از سیستم عامل‌های ویندوز ۲۰۰۰ و XP جایگاه مناسب خود را پیدا کرده و کاربران بسیاری در سراسر دنیا از آن استفاده می‌کنند.

کارشناسان، COM+ را مکمل بسته‌های نرم‌افزاری امروزی می‌دانند و شما هم به عنوان یک برنامه‌نویس باید آن را بیاموزید و با نحوهٔ بکارگیری آن در دلفی آشنا شوید.

COM+ چیست؟

COM ابزاری است که قابلیتهای برنامه‌های موجود تحت محیطهای ویندوز را افزایش می‌دهد. اجزاء COM را با زبانهای مختلفی می‌توان نوشت که دلفی یکی از قویترین آنهاست. COM سرنام Component Object Model است، یعنی مجموعه مشخصاتی که میکروسافت برای ساختن اجزاء نرم‌افزارهایی مطرح کرده که می‌توان در برنامه‌ها گنجاند و قابلیت برنامه‌های موجود تحت محیط‌های پنجره را افزایش داد. اجزای COM را با زبانهای مختلفی می‌توان نوشت. COM پایه و شالودهٔ مجموعه مشخصات OLE (گنجاندن و مرتبط کردن شیءها)، ActiveX و DriectX است. COM+ نیز چیزی منفک از COM نیست، تنها امکانات و قدرت آن ارتقاء پیدا کرده است. اگر با *Microsoft Transaction Server (MTS)* و *Microsoft Message Queue (MSMQ)* و قابلیتهای آنها آشنایی دارید، باید گفت

که +COM قابلیتهای متناظر با دو ابزار فوق و COM را داراست و آنچه که در اینجا اهمیت دارد پشتیبانی دلفی از این ابزار است.

COM دارای چه قابلیتهایی است؟

COM اصلی‌ترین پایهٔ تکنولوژی‌هایی است که به اجزاء نرم‌افزاری امکان می‌دهند تا بدون توجه به زبانی که با آن ایجاد شده‌اند، با یکدیگر ارتباط برقرار سازند. دو قابلیت مهم COM که امروزه بیش از ۱۵۰ میلیون کاربر دارد عبارت است از:

- COM مدلی مستقل از زبانهای برنامه‌سازی است و نحوهٔ بکارگیری آن وابستگی به زبانی خاص ندارد.
- تمامی ابزارها و نرم‌افزارهای کاربردی تحت ویندوز از COM پشتیبانی می‌کنند.

در اینجا با دسته‌بندی ویژگیهای COM تحت سه عنوان مدیریت اجزاء سازنده، سرویس‌ها و runtime به توضیح آنها خواهیم پرداخت.

سرویس‌ها

سرویس‌های +COM ترکیبی از سرویس‌های MTS و MSMQ می‌باشند، چنانکه برخی از کارشناسان بر این باورند که MTS و MSMQ در ارتقاء سرویس‌های +COM مؤثر بوده‌اند. مدیریت تراکنشها[1] مهمترین بخش این سرویس‌ها بوده که در ادامه به تشریح آنها خواهیم پرداخت.

تراکنش‌ها

از آنجا که +COM بر مبنای طراحی شیءگرا استوار است و محیط‌های شیءگرا نیازمند سیستمی هستند که پردازش تراکنشها را به محض دریافت آنها انجام دهد، لذا مدیریت تراکنش‌ها یکی از ضروری‌ترین سرویس‌های +COM به شمار می‌آید.

به طور مثال هنگامی که یک برنامهٔ کاربردی تحت ویندوز مانند سفارش یک مشتری، در حال پردازش داده‌ها می‌باشد، به لحاظ ارتباط تنگاتنگ آن با محیط‌های شیءگرا و همچنین بانک‌های اطلاعاتی، لازم است تا عملیات پردازش تراکنشها نیز به درستی انجام پذیرد، به طوری که تراکنشها بلافاصله پس از دریافت توسط سیستم اجرا شوند. کنترل تراکنشها در +COM به وسیلهٔ *Distributed Transaction Coordinator (DTC)* انجام می‌گیرد. در روش مدیریت تراکنشها به وسیلهٔ DTC، پردازش تراکنشها بین یک یا چند سیستم به اشتراک گذاشته خواهد شد.

۱- تراکنش (Transaction) به مفهوم بروزرسانی هم زمان همه فایل‌هاست. هنگامی که یک سیستم حین انجام یک تراکنش دچار اشکال شود، قسمتی از اطلاعات به روز در آمده و قسمتهای دیگر هنوز به حالت اولیه باقی مانده‌اند. به این ترتیب اطلاعات خروجی سیستم قابل اطمینان نبوده و فاقد ارزش می‌باشند. در سیستمی که از مدیریت تراکنش‌ها استفاده می‌کند، یا تمامی اطلاعات در فایل‌های مختلف به روز در می‌آیند و یا در صورت به وجود آمدن اشکال، کلیهٔ فایل‌ها به حالت اولیه برمی‌گردند.

امنیت

برنامه‌های کاربردی، بخصوص برنامه‌های سرویس‌دهنده/ سرویس‌گیرنده، دربرگیرندهٔ حجم وسیعی از اطلاعات می‌باشند، بدین ترتیب کنترل شدید بر صلاحیت و چگونگی کار کاربران آنها امری حیاتی است.

مفهوم امنیت در برنامه‌های کاربردی امروزی که توسط ابزارها و زبان‌های برنامه‌نویسی شیءگرا طراحی می‌گردند، مفهومی گسترده‌تر از کنترل عملیات کاربران داراست. مدیریت عناصر، اجزاء سازنده و کنترل انتقال داده‌ها در این سیستم‌ها و همچنین کنترل انتقال داده‌ها بین عناصر یک برنامه مقوله‌هایی است که به امنیت سیستم مربوط می‌شود.

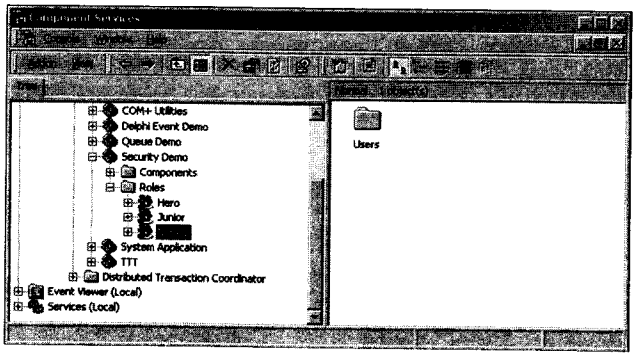
در ادامهٔ این بخش به بررسی ساختار امنیتی COM+ پرداخته و نشان خواهیم داد که باز هم به سادگی و با استفاده از طبیعت شیءگرای دلفی قادرید امنیت سیستم‌های مبتنی بر تکنولوژی COM خود را افزایش دهید.

ساختار امنیتی Role_Based

در ساختار امنیتی COM+ از روش role_based استفاده گردیده است. در این روش تمامی عناصر و گروههای کاری تحت عنوان Roles معرفی شده و در کنار روش امنیتی system_based (این روش در همهٔ سیستم‌های عامل پیاده‌سازی شده است)، امنیت برنامه‌های کاربردی را تضمین می‌نمایند.

اما Roles به خودی خود تولید نمی‌شود. برای ایجاد Roles می‌بایست ابزار Comopnent Services administration دلفی را به کار گرفت. در سمت چپ منوی Component Services ، یک لیست درختی از عناصر موجود در آن نمایش داده خواهد شد. اکنون با انتخاب گرهٔ Roles از این منو قادرید users را به آن اضافه نمائید (برای انجام این کار دکمهٔ سمت راست ماوس را بر روی Roles بفشارید). شکل ۱-۱۳ نحوهٔ انجام این کار را نمایش می‌دهد.

همانطور که در شکل می‌بینید، سه نوع Roles به نامهای Hero ، Junior و Normal برای COM+



شکل ۱-۱۳ منوی مربوط به Component Services administration

تعریف شده است. هر کدام از این عناوین گروههای کاری متفاوتی را برای کاربران مشخص می‌سازند. توجه داشته باشید که کنترل اصلی امنیت توسط سیستم عامل انجام می‌شود و +COM به عنوان جزئی از سرویس‌های سیستم عامل مطرح است.

پیکربندی Role_Based

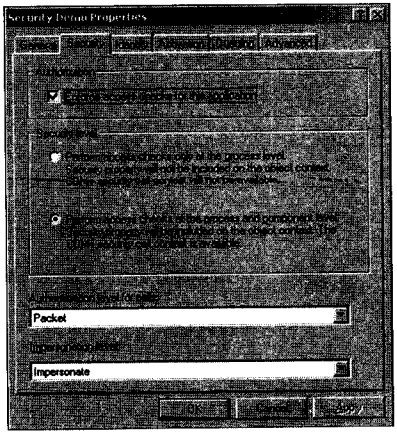
Role_Based علاوه بر امنیت برنامه‌ها، قابلیت پیاده‌سازی و برقراری امنیت در سطوح اجزاء سازنده، Interface ها و حتی متدها را نیز داراست.

اولین گام برای پیاده‌سازی امنیت در +COM ارزیابی خود برنامه است. این کار با فراخوانی ابزار Component Services administration و انتخاب تب Security همانند شکل ۲-۱۳ امکان‌پذیر است.

ابتدا گزینهٔ Enforce Access Checks For This Application را فعال نمائید. همانطور که در شکل می‌بینید، قسمتی از کادر نمایش داده شده مربوط به تعیین سطح امنیتی است. یکی از این سطوح امنیتی فقط مربوط به فرآیندها (process) و دیگری مربوط به عناصر سازنده و فرآیندها است.

چنانچه تمایل دارید که برنامهٔ COM شما تنها برای گروهی از کاربران قابل استفاده باشد و امنیت را تنها در همین سطح برقرار سازید، لازم است تا گزینهٔ اول را انتخاب کنید. با انتخاب این گزینه هیچگونه اقدام امنیتی بر روی متدها، Interface ها و عناصر سازندهٔ برنامه‌تان انجام نمی‌گیرد. اما کارآیی و سرعت برنامهٔ شما تا حد قابل توجهی افزایش می‌یابد. این افزایش کارآیی برنامه به دلیل کاهش سطوح امنیتی در سیستم است. با انتخاب گزینهٔ دوم از این قسمت، امنیت برنامه‌تان را تا جزئی‌ترین سطوح آن تسری داده‌اید. البته توجه داشته باشید که با انجام این کار ممکن است کارآیی سیستم تا حدی کاهش یابد.

گزینه‌های دیگری نیز بر روی جعبهٔ مکالمهٔ نشان داده شده در شکل ۲-۱۳ وجود دارند. یکی از این گزینه‌ها Authentication level for calls می‌باشد که سطح تصدیق یک برنامهٔ +COM را تعیین



شکل ۲-۱۳ پیکربندی امنیت یک برنامهٔ کاربردی مبتنی بر +COM

می‌نماید. سطح تصدیق، مربوط به کنترل ریز عملیات بین یک برنامهٔ +COM و یک سرویس‌گیرنده می‌باشد، بدین معنی که با تنظیم گزینه‌های آن می‌توان وضعیت ارتباطی کاربران، وضعیت انتقال داده‌ها و یا عملیات رمزگذاری و رمزگشایی بسته‌ها را کنترل نمود. پارامترهای مجاز برای این گزینه در جدول ۱-۱۳ آورده شده است.

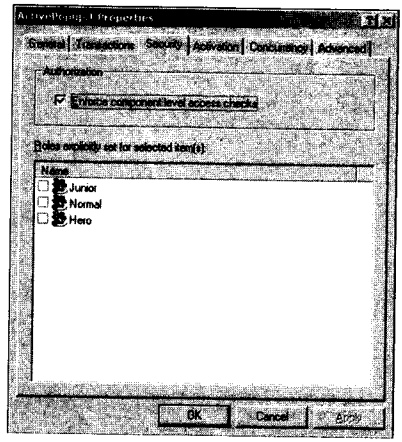
جدول ۱-۱۳ سطوح تصدیق +COM

سطح	شرح
None	هیچ سطح تصدیقی استفاده نخواهد شد.
Connect	این سطح تصدیق تنها اتصال سرویس‌گیرنده با برنامهٔ +COM را تأیید می‌نماید.
Call	این سطح تصدیق تأیید کنندهٔ مراحل فراخوانی روتین‌ها و یا داده‌ها است.
Packet	دریافت داده‌ها توسط این سطح تصدیق تأیید خواهد شد (مقدار پیش‌فرض گزینهٔ Authentication level for calls).
Packet Integrity	این سطح تصدیق، عدم تغییر بسته‌ها را در زمان ارسال دریافت آنها تأیید می‌کند.
Packet Privacy	از این سطح تصدیق جهت رمزگذاری بسته‌ها (شامل داده‌ها و مشخصات ارسال کنندهٔ آن) استفاده می‌شود.

آنچه که در بالا گفته شد همگی مربوط به تنظیم سطوح امنیتی برنامه‌ها و اجزاء سازندهٔ +COM بود. اما هنوز یک گام دیگر باقی است و آن هم تعیین اعتبار سرویس‌گیرنده در استفاده از یک برنامهٔ +COM است. از این تعیین اعتبار برای جلوگیری از دسترسی سرویس‌گیرنده‌های غیرمجاز به اجزاء سازندهٔ +COM استفاده می‌شود. این عملیات توسط پارامترهای Impersonation level (شکل ۲-۱۳) پیاده‌سازی می‌گردد. مقادیر این پارامترها به شرح جدول ۲-۱۳ ارائه شده است.

جدول ۲-۱۳ پارامترهای مجاز برای گزینهٔ Impersonation level

پارامتر	شرح
Anonymous	سرویس‌گیرنده از طرف سرویس‌دهنده قابل شناسایی نمی‌باشد (سرویس‌گیرندهٔ غیرمجاز)
Identify	سرویس‌دهنده، پس از دریافت شمارهٔ شناسایی (ID) سرویس‌گیرنده، امکان برقراری ارتباط را میسر می‌سازد.
Impersonate	ارتباط بین سرویس‌دهنده و سرویس‌گیرنده تنها تحت شرایطی خاص امکان‌پذیر می‌باشد. مقدار پیش‌فرض گزینهٔ Impersonation level برابر با این پارامتر می‌باشد.
Delegate	این پارامتر گسترده‌ترین مجوزی است که بین سرویس‌گیرنده و سرویس‌دهنده تعریف می‌شود. برای دریافت اطلاعات بیشتر در خصوص این پارامتر به راهنمای دلفی مراجعه کنید.



شکل ۳-۱۳ پیکربندی امنیت اجزاء سازندهٔ COM+

تا اینجا پیاده‌سازی امنیت برای برنامه‌های کاربردی COM+ خود را آموخته‌اید. اما برای امنیت اجزاء سازنده چه کاری باید انجام داد؟ برای این کار لازم است تا پس از قرار دادن یک جزء سازنده در فرم مربوط به یک برنامه، خصوصیات آن را فراخوانی نمائید. بر روی منوی ظاهر شده یک تب تحت عنوان security وجود دارد که عملیات فوق را به انجام می‌رساند (همانند شکل ۳-۱۳).

برنامه‌های Multitier

چنانچه یک برنامهٔ کاربردی Multitier طراحی کرده‌اید و قصد دارید تکنیک‌های امنیتی COM+ را بر روی آن پیاده‌سازی کنید، برای بهبود کارآیی برنامه‌تان می‌بایست نکاتی را مدنظر داشته باشید. یک سیستم Multitier ترکیبی از سیستم‌های سرویس‌گیرنده، برنامه‌های کاربردی سرویس‌دهنده و بانک‌های اطلاعاتی است. از آنجا که این سیستم‌ها با حجم وسیعی از داده‌ها روبه‌رو هستند، لذا پیاده‌سازی روش‌های امنیتی در آنها به سادگی امکان‌پذیر نبوده و همواره می‌بایست بهبود کارآیی سیستم را مدنظر قرار دهید.

امنیت برنامه‌ای

گاهی اوقات امنیت مفهومی فراتر از آنچه تاکنون آموخته‌اید پیدا می‌کند. این که شما می‌خواهید بر کارکرد برنامه‌تان اعمال مدیریت کنید و یا زیرمجموعه‌های آن را کنترل نمائید، همگی مقوله‌هایی است که مربوط به امنیت سیستم‌ها می‌گردد. اما اگر خواسته باشید علاوه بر کنترل متدهای برنامه‌تان، رفتار آنها را هم کنترل نمائید وضع به چه منوال است؟ بلی. این هم مربوط به امنیت سیستم می‌شود. ساختارهای امنیتی COM+، شما را در انجام این کار یاری خواهند داد.

کلاس IObjectContext حاوی متدی به نام IsCallerInRole() است که با ارجاع یک مقدار منطقی، انتساب یک متد به یک Role را مشخص می‌سازد. Role نامی برای متدهایی است که رفتار و

مشخصات مشابه با هم دارند.

این متد به صورت زیر تعریف می‌شود:

```
function IsCallerInRole(const bstrRole: WideString): Bool; safecall;
```

مثال ارائه شده در زیر با استفاده از یک تابع API به نام GetObjectContext() و متد ذکر شده، انتساب یک role "Hero" و معتبر بودن آن را بررسی می‌نماید.

```
var
  Ctx: IObjectContext;
begin
  Ctx := GetObjectContext;
  if (Ctx <> nil) and (Ctx.IsCallerInRole('Hero')) then
  begin
    // do something interesting
  end;
end;
```

مفهوم Just_In_Time (JIT)

Just_In_Time (JIT) واژه‌ای برای توصیف کامپایلری است که برنامه‌های شبکه‌ای نظیر Java را بسیار سریع ترجمه می‌کند، اما در COM+ به مفهوم ایجاد و تخریب یک برنامه بدون توجه به ساختار آن برنامه است. آنهایی که با تکنیک‌های برنامه‌نویسی شیءگرا بیشتر آشنایی دارند، از این قابلیت COM+ به عنوان یک فرصت طلایی برای طراحی بهتر برنامه‌های کاربردی بهره خواهند گرفت. متدهای SetComplete() و SetAbort() از کلاس IObjectContext و متد SetDeactivateOnReturn() از کلاس IContextState جهت این امر کاربرد دارند.

اجزاء سازندهٔ صف‌بندی شده

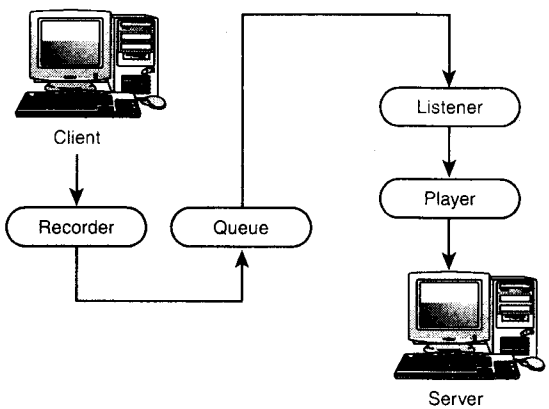
دلفی ۳ با معرفی ابزار جدیدی تحت عنوان MIDAS هیجان خاصی را در دل برنامه‌نویسان برانگیخت. MIDAS مجموعه‌ای از اجزاء سازنده بود که آماده‌سازی برنامه‌های کاربردی سرویس‌گیرنده/ سرویس‌دهنده Multitier را آسان می‌نمود. به خاطر داشته باشید که منظور از برنامهٔ کاربردی سرویس‌گیرنده، برنامه‌ای است که کاربران با استفاده از یک رابط گرافیکی با آن ارتباط برقرار می‌کنند. برنامهٔ کاربردی سرویس‌دهنده نیز برنامه‌ای است که خدماتی را برای برنامهٔ کاربردی سرویس‌گیرنده فراهم می‌کند. عنوان سرویس‌دهنده/ سرویس‌گیرنده به معنی وجود یک رابط گرافیکی و یک سرویس‌دهندهٔ بانک اطلاعاتی است.

بعد از آن، نسخه‌های جدید دلفی با ابزارهای متنوع‌تری به بازار عرضه گردیدند. امروزه دلفی ۷ با پشتیبانی از تکنولوژی‌هایی نظیر COM+، MTS و CORBA به عنوان یکی از قدرتمندترین ابزارهای برنامه‌نویسی مطرح و مورد استفادهٔ جمع کثیری از خبرگان این امر می‌باشد.

اکنون اجزاء دیگری را تحت عنوان اجزاء سازندهٔ صف‌بندی شده به شما معرفی می‌کنیم. از اجزاء

سازندهٔ صف‌بندی شده برای همزمانی درخواستهای ارسالی مابین برنامه‌های سرویس‌دهنده/سرویس‌گیرندهٔ مبتنی بر COM+ استفاده می‌شود. در این حالت یک سرویس‌گیرنده قادر است بدون دسترسی به سیستم سرویس‌دهنده، یک متد و یا یک مدل از یک شیء را درخواست و فراخوانی نماید. COM+ این کار را با قرار دادن درخواستهای ارسالی در یک صف و پاسخ‌دهی عندالزوم آنها به هنگام در دسترس بودن سرور انجام می‌دهد.

شکل ۴-۱۳ نحوهٔ پیاده‌سازی اجزاء سازندهٔ صف‌بندی شده را نمایش می‌دهد. همانطور که در شکل می‌بینید، با ارسال یک درخواست از سوی یک سرویس‌گیرنده، متد درخواست شده به همراه پارامترهای آن به وسیلهٔ Recorder ثبت و در Queue قرار می‌گیرد. Recorder قسمتی است که می‌تواند در برگیرندهٔ اطلاعاتی از طرف سرور نیز باشد. این اطلاعات محدود به پیکربندی، توابع کتابخانه‌ای و اطلاعات اولیهٔ سرور است. Listener وظیفهٔ ارجاع یک پیام (اطلاعات درخواستی) به بخش Player را به عهده دارد. عملیات نهایی باز کردن پیام (ممکن است پیام دارای کدهای حفاظتی از طرف سرویس‌گیرنده باشد) و اجرای متد درخواستی در طرف سرویس‌دهنده به وسیلهٔ Player انجام می‌پذیرد.



شکل ۴-۱۳ ساختار داخلی اجزاء سازندهٔ صف‌بندی شده در COM+

چرا از اجزاء سازندهٔ صف‌بندی شده استفاده می‌کنیم؟

قبل از این که نحوهٔ پیاده‌سازی این اجزاء را شرح دهیم، لازم است تا برخی از مزایای به کارگیری آنها را برشماریم.

- بهبود بهره‌وری سیستم‌ها: در سیستم‌هایی که از اجزاء سازندهٔ صف‌بندی شده استفاده نمی‌کنند، پاسخدهی به درخواستها تنها همزمان با ارسال آنها انجام می‌شود. در این صورت چنانچه همزمان با یک درخواست، درخواست‌های دیگری ارسال گردد، هیچ پاسخی به آنها داده نشده، یک چرخهٔ انتظار نامحدود بوجود می‌آید. اما در اجزاء سازندهٔ صف‌بندی شده با قرار دادن درخواست‌ها در یک

صف، به محض اتمام یک درخواست، درخواستهای بعدی وارد شده و به آنها پاسخ داده می‌شود.

- مدل Briefcase : در اجزاء سازندهٔ صف‌بندی شده، امکان فعالیت مجزای سرویس‌گیرنده بدون استفاده از سرویس‌دهنده وجود دارد. یعنی اگر سرویس‌دهنده دچار اشکال شود، سرویس‌گیرنده می‌تواند به طور مستقل عملیات خود را انجام داده و پس از رفع عیب با سرویس‌دهنده ارتباط برقرار نماید.
- مدیریت بحران: چنانچه قصد دارید یک برنامهٔ سرویس‌دهنده با تضمین امنیت بالا (همانند یک برنامهٔ تجارتی) پیاده‌سازی کنید، ویژگیهای منحصر به فرد اجزای سازندهٔ صف‌بندی شده کمک شایانی به ارتقاء سطح امنیت برنامه‌تان می‌کند.
- زمان‌بندی در بارگذاری برنامه‌ها: مزیت دیگر اجزاء سازندهٔ صف‌بندی شده استفاده از تکنیک زمان‌بندی در بارگذاری برنامه‌هاست. این بدین معنی است که یک مدیریت زمان‌بند برای پاسخ‌دهی به فرآیندها در اجزاء سازندهٔ صف‌بندی شده وجود دارد.

ایجاد یک سرویس‌دهنده

روشهای ایجاد اجزاء سازندهٔ صف‌بندی شده و اجزاء سازندهٔ +COM با هم تفاوت دارند. با ارائه یک مثال نحوهٔ ایجاد آنها را مقایسه خواهیم کرد.

می‌خواهیم یک سرور دلفی ایجاد کنیم که حاوی یک کلاس +COM ، یک Interface و تنها یک متد خاص باشد.

با انتخاب گزینهٔ Automation Object Wizard و New Main menu از منوی File کار را ادامه دهید. شیء ایجاد شده را QTest و Interface مربوطه را IQTest نام‌گذاری کرده و کد زیر را برای Interface ذکر شده وارد نمائید.

```
procedure SendText(Value: WideString; Time: TDateTime) [dispid $00000001];
  safecall;
```

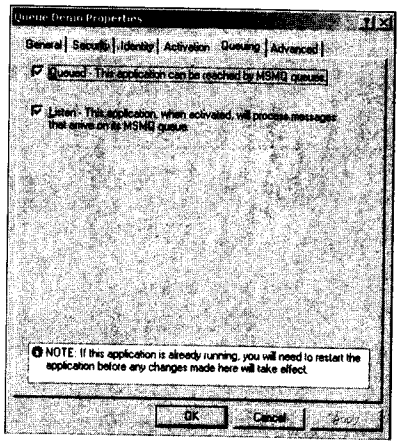
همانطور که ملاحظه می‌کنید متد ارائه شده دارای دو پارامتر است. یکی از آنها در برگیرندهٔ پیام درخواستی بوده و دیگری زمان فراخوانی متد در سیستم سرویس‌گیرنده را مشخص می‌سازد. برای آگاهی از نحوهٔ پردازش این پیام در سیستم سرویس‌دهنده، لازم است تا اطلاعات آن را در یک فایل متنی نگهداری نمائیم. لیست ۱-۱۳ نحوهٔ پیاده‌سازی این عملیات را نشان می‌دهد. چگونگی پردازش پیام‌ها در فایل queue.txt از درایو C نگهداری خواهد شد.

بعد از اجرای کد فوق می‌بایست آن را پیکربندی و در یک برنامهٔ +COM نصب نمائید.

از ابزار Component Services Management و یا توابع کتابخانه‌ای API در +COM برای انجام این کار استفاده کنید. بعد از این که برنامه را ایجاد نمودید، همانند شکل ۵-۱۳ گزینهٔ Queued و گزینهٔ Listen را انتخاب کنید. این گزینه بر نحوهٔ پردازش پیام‌ها در هنگام اجرای برنامهٔ کاربردی دلالت دارد.

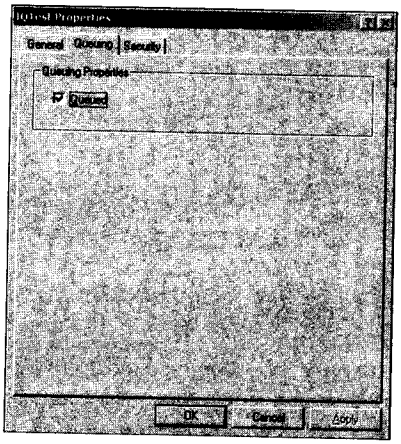
لیست ۱-۱۳ پیاده‌سازی یک شیء از نوع صف‌بندی شده، TestImp.pas

```
unit TestImpl;

interface

uses
  Windows, ComObj, ActiveX, Srv_TLB, StdVcl;

type
  TQTest = class(TAutoObject, IQTest)
  protected
    procedure SendText(const Value: WideString; Time: TDateTime); safecall;
  end;

implementation

uses ComServ, SysUtils;

procedure TQTest.SendText(const Value: WideString; Time: TDateTime);
const
  SFileName = 'c:\queue.txt';
  SEntryFormat = 'Send time:  %s'#13#10'Write time: %s'#13#10 +
    'Message: %s'#13#10#13#10;
var
  F: THandle;
  WriteStr: string;
begin
  F := CreateFile(SFileName, GENERIC_WRITE, FILE_SHARE_READ, nil, OPEN_ALWAYS,
    FILE_ATTRIBUTE_NORMAL, 0);
  if F = INVALID_HANDLE_VALUE then RaiseLastWin32Error;
  try
    FileSeek(F, 0, 2);  // go to EOF
    WriteStr := Format(SEntryFormat, [DateTimeToStr(Time),
      DateTimeToStr(Now), Value]);
    FileWrite(F, WriteStr[1], Length(WriteStr));
  finally
    CloseHandle(F);
  end;
end;

initialization
  TAutoObjectFactory.Create(ComServer, TQTest, Class_QTest,
    ciMultiInstance, tmApartment);
end.
```



شکل ۵-۱۳ پیکربندی یک
برنامهٔ صف‌بندی شدهٔ COM+

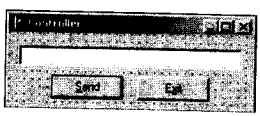
حال می‌خواهیم سروری را در برنامهٔ COM+ خود نصب نمائیم. با انتخاب گزینهٔ New مربوط به گرهٔ Component (به شکل ۱-۱۳ مراجعه کنید) این کار را انجام دهید. اکنون با فراخوانی پنجرهٔ مربوط به IQTest آن را طبق شکل ۶-۱۳ تنظیم نمائید.



شکل ٦-۱۳ تعریف یک Interface
از نوع صف‌بندی شده

ایجاد یک سرویس‌گیرنده

اکنون به ایجاد یک برنامهٔ سرویس‌گیرنده با استفاده از اجزاء سازندهٔ صف‌بندی شده خواهیم پرداخت.
فرم اصلی این برنامه را همانند شکل ۷-۱۳ طراحی کنید.
قرار است پس از اجرای برنامه، آنچه که در قسمت ویرایشی آن تایپ شده است با فشردن دکمهٔ send به سرویس‌دهنده ارسال شود.



شکل ۷-۱۳ یک برنامهٔ سرویس‌گیرنده
برای اجزاء سازندهٔ صف‌بندی شده

متد SendText() که در لیست ۲-۱۳ ارائه شده است، وظیفهٔ ارسال پیام را به عهده دارد.

لیست ۲-۱۳ متد SendText()

```
unit Ctrl;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ColorGrd, ExtCtrls, Srv_TLB, Buttons;

type
  TControlForm = class(TForm)
    BtnExit: TButton;
    Edit: TEdit;
    BtnSend: TButton;
    procedure BtnExitClick(Sender: TObject);
    procedure BtnSendClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    FIntf: IQTest;
  end;

var
  ControlForm: TControlForm;

implementation

{$R *.DFM}

uses ComObj, ActiveX;

// Need to import CoGetObject because import in the ActiveX unit is incorrect:
function MyCoGetObject(pszName: PWideChar; pBindOptions: PBindOpts;
  const iid: TIID; out ppv): HResult; stdcall;
  external 'ole32.dll' name 'CoGetObject';

procedure TControlForm.BtnExitClick(Sender: TObject);
begin
  Close;
end;

procedure TControlForm.BtnSendClick(Sender: TObject);
begin
  FIntf.SendText(Edit.Text, Now);
  Edit.Clear;
end;
```

لیست ۲-۱۳ ادامه

```
procedure TControlForm.FormCreate(Sender: TObject);
const
  SMoniker: PWideChar = 'queue:/new:{64C576F0-C9A7-420A-9EAB-0BE98264BC9D}';
begin
  // Create object using a moniker that specifies queued creation
  OleCheck(MyCoGetObject(SMoniker, nil, IQTest, FIntf));
end;

end.
```

متد CoGetObject() وظیفهٔ ایجاد یک شیء به وسیلهٔ Moniker را به عهده دارد. رشتهٔ کاراکتری درج شده در Moniker تنها برای COM+ معتبر بوده و از آن، جهت فراخوانی اجزاء سازندهٔ صف‌بندی شده استفاده می‌شود. نحوهٔ نگارش این رشته عموماً به صورت ... : queue:/new بوده که در زیر نمونه‌هایی از آن ارائه شده است.

```
queue:/new:Srv.IQTest
queue:/new:{64C576F0-C9A7-420A-9EAB-0BE98264BC9D}
queue:/new:64C576F0-C9A7-420A-9EAB-0BE98264BC9D
```

ضمناً شما قادر خواهید بود تا مقصد صف‌ها و یا رفتار اجزاء سازندهٔ آنها را تغییر داده و کنترل کنید. برای انجام این کار چندین پارامتر queue moniker تعریف شده است که به شرح آنها خواهیم پرداخت.

- ComputerName : از رشتهٔ کاراکتری موجود در این پارامتر برای شناسایی سیستم دربرگیرندهٔ صف استفاده خواهد شد. این نام به عنوان بخشی از مسیر (محل قرارگیری) صف مطرح می‌باشد. چنانچه مقداری برای این پارامتر مشخص نگردد، عملیات مقداردهی آن به صورت خودکار و در هنگام پیکربندی انجام خواهد شد.
- QueueName : این پارامتر مشخص کنندهٔ نام صف، بر روی سیستم سرویس‌دهنده می‌باشد. در صورت عدم تعریف آن، مقداردهی آن به صورت خودکار و در هنگام پیکربندی برنامه انجام خواهد شد.
- PathName : این پارامتر که محل قرار گرفتن صف را مشخص می‌سازد بایستی به صورت Computername\QueueName تعریف می‌شود.
- FormatName : این پارامتر نام قالب و یا فرمت یک صف را مشخص می‌سازد. مثلاً عبارت DIRECT=9CA3600F-11D2-88C5-00A0C90AB40E معرف یک قالب برای صف می‌باشد.
- AppSpecific : این پارامتر که یک عدد صحیح بدون علامت می‌باشد، مشخصات برنامهٔ کاربردی را تعیین می‌نماید. به طور مثال AppSpecific=8675309 .
- AuthLevel : این پارامتر حاوی یکی از مقادیر MQMSG_AUTH_LEVEL_NONE(0) و یا

MQMSG_AUTH_LEVEL_ALWAYS(1) خواهد بود. این دو مقدار، سطح تضمین یک پیام را مشخص می‌سازد. همانطور که قبلاً اشاره شد سطح تضمین یک پیام بر درستی ارسال پیام از طرف کاربر دلالت دارد.

- Delivery : این پارامتر بر درستی تحویل یک پیام دلالت داشته و حاوی یکی از مقادیر MQMSG_DELIVERY_EXPRESS(0) یا MQMSG_DELIVERY_RECOVERABLE(1) خواهد بود.
- EncrypAlgorithm : الگوریتم رمزگذاری برای پیام‌های +COM به وسیلهٔ این پارامتر مشخص می‌شود. عموماً مقدار این پارامتر برابر با CALG_RC4 و CALG_RC2 خواهد بود.
- HashAlgorithm : چنانچه قصد دارید الگوریتم Hash را جهت رمزگذاری پیام‌های +COM بکار ببرید بایستی از این پارامترها استفاده کنید. CALG_MD2 ، CALG_MD4 ، CALG_MD5 ، CALG_SHA ، CALG_SHA1 ، CALG_MAC ، CALG_SSL3_SHAMD5 ، CALG_HMAC ، CALG_TLS1PRF مقادیر معتبر برای این پارامترها می‌باشند.
- Journal : چنانچه صف انتخاب شده در +COM از نوع Journal می‌باشد، از این پارامتر برای تعریف آن استفاده خواهد شد. MQMSG_JOURNAL_NONE(0) ، MQMSG_DEALETTER(1) و یا MQMSG_JOURNAL(2) مقادیر معتبر برای این پارامتر می‌باشند.
- Lable : برچسب مربوط به پیام ارسالی توسط این پارامتر مشخص می‌شود. حداکثر طول این برچسب برابر با پارامتر MQ_MAX_MSG_LABEL_LEN خواهد بود.
- MaxTimeResearchQueue : مدت زمانی که پیام ارسالی عملیات جستجو برای یافتن یک صف را انجام می‌دهد توسط این پارامتر مشخص می‌شود. هنگام ارسال پیام، مدت زمانی طول خواهد کشید تا پیام ارسالی یک صف را پیدا کرده و به آن دسترسی یابد. زمان معتبر جهت انجام این عملیات (به ثانیه) توسط این پارامتر مشخص می‌شود. INFINITE ، LONG_LIVED و یا هر مقدار صحیح دیگر، مقادیر مجاز برای این پارامتر خواهند بود.
- MaxTimeToReceive : مدت زمان بین ارسال یک پیام تا دریافت آن در مقصد توسط این پارامتر مشخص می‌گردد. INFINITE ، LONG_LIVED و یا هر مقدار صحیح دیگر، مقادیر مجاز برای این پارامتر خواهند بود.
- Priority : این پارامتر مشخص کنندهٔ سطح اولویت پیام‌ها خواهد بود. MQ_MIN_PRIORITY(0) ، Q_MAX_PRIORITY(7) و MQ_DEDAULT_PRIORITY(3) (همچنین مقادیر صحیح بین ۰ تا ۷) مقادیر مجاز برای این پارامتر می‌باشند.
- PrivLevel : وقتی پیام‌ها به صورت رمز در می‌آیند، سطحی به نام سطح پنهان‌سازی نیز برای عملیات رمزگذاری تعریف می‌گردد. این پارامتر معرف این سطح خواهد بود.

MQMSG_PRIV_LEVEL_BODY، NONE، BODY، MQMSG_PRIV_LEVEL_BODY،
MQMSG_PRIV_LEVEL_BODY_ENHANCED، MQMSG_PRIV_LEVEL_BODY_BASE
و یا BODY_ENHANCED مقادیر مجاز این پارامتر خواهند بود.

- Trace : از این پارامتر برای ردیابی کردن روتین‌های COM+ استفاده می‌شود. MQMSG_TRACE_NONE(0) یا QMSG_SEND_ROUTE_TO_REPORT_QUEUE(1) مقادیر مجاز برای این پارامتر می‌باشند.

به مثالهای زیر توجه کنید. متوجه خواهید شد که این پارامترها در همهٔ مواقع کاربرد نداشته و فقط در مواردی خاص از برخی از آنها استفاده می‌شود.

```
queue:Priority=6,ComputerName=foo/new:{64C576F0-C9A7-420A-9EAB-0BE98264BC9D}
queue:PathName=drevil\myqueue/new:{64C576F0-C9A7-420A-9EAB-0BE98264BC9D}
```

اجرای یک سرویس‌دهنده

اکنون مثال قبل را پس از وارد کردن یک رشتهٔ کاراکتری دلخواه در فرم اصلی آن اجرا کنید، متوجه خواهید شد که فایل queue.txt (که آن را ایجاد کرده‌اید) بر روی هارد دیسک شما وجود ندارد. چه اتفاقی افتاده است؟ آیا خطایی رخ داده است؟

هیچ خطایی رخ نداده است. توجه داشته باشید که قبل از ارسال یک پیام، ابتدا باید برنامهٔ سرویس‌دهنده اجرا گردد. این کار به سه روش امکان‌پذیر است:

۱- روش دستی: با فراخوانی ابزار Component Services و انتخاب گزینهٔ start از منوی ظاهر شده قادر خواهید بود تا برنامهٔ سرویس‌دهنده را اجرا کنید.

۲- روش برنامه‌نویسی: در این روش از توابع کتابخانه‌ای API در COM+ برای اجرای برنامهٔ سرویس‌دهنده استفاده می‌شود.

۳- روش زمان‌بندی شده: در این روش از یک اسکریپت برای صدور فرامین لازم جهت اجرای برنامهٔ سرویس‌دهنده، استفاده می‌شود. این اسکریپت می‌تواند کدی مانند کد زیر داشته باشد.

```
dim cat
set cat = CreateObject("COMAdmin.COMAdminCatalog");
cat.StartApplication("YourApplication");
```

حال فایل queue.txt را باز کرده و محتویات آن را بررسی کنید. نمونه‌ای از محتویات احتمالی این فایل در زیر ارائه شده است.

```
Send time:  7/6/2001 7:15:08 AM
Write time: 7/6/2001 7:15:18 AM
Message: this is a test

Send time:  7/6/2001 7:15:10 AM
```

```
Write time: 7/6/2001 7:15:18 AM
Message: this is another
```

روش Pooling

هدف کلی از طراحی Pooling مانند JIT[1] ، افزایش بهره‌وری سیستم و ارتقاء کارآیی برنامه‌هاست. در این روش، اشیائی که دارای ویژگیهای تقریباً مشابه می‌باشند، در یک منبع نگهداری می‌شوند.

+COM از Pooling پشتیبانی می‌کند، اما این پشتیبانی با محدودیتهایی روبروست که در زیر به آنها اشاره می‌کنیم:

- تمامی اشیاء از نوع Stateless[2] باشند.
- اشیاء هیچگونه وابستگی با Thread[3] ها نداشته باشند.
- تمامی اشیاء حالت پیوسته داشته باشند.
- عملیات مدیریت منابع و یا ثبت و مقداردهی اشیاء به صورت خودکار انجام نگیرد.
- قابلیت پیاده‌سازی کلاس IObjectControl وجود داشته باشد.

رویدادها

همگی ما با رویدادها و نحوهٔ پاسخ‌دهی آنها آشنایی داریم. یک رویداد، عملی است که مسبب آن یک کاربر است و یک برنامه ممکن است به آن پاسخ دهد. به عنوان مثال فشرده شدن کلیدها، برگزیدن دکمه‌ها با ماوس و یا حرکت دادن ماوس. آنچه که در محیط دلفی برای مدیریت رویدادهای یک برنامهٔ کاربردی اتفاق می‌افتد، تفاوت آشکاری با مدیریت رویدادها در برنامه‌ها +COM دارد. +COM از شیوهٔ جدیدی برای مدیریت رویدادها استفاده می‌کند.

بحث خود را با بررسی دقیق‌تر ارتباط بین سیستم‌های سرویس‌دهنده و سرویس‌گیرنده ادامه می‌دهیم. یک سیستم سرویس‌گیرنده، درخواستی را برای استفاده از یک متد به سیستم سرویس‌دهنده ارسال می‌دارد. اگر همه چیز به درستی انجام پذیرد، دستگاه سرویس‌دهنده به تقاضای ارسالی پاسخ گفته و قاعدتاً مقداری داده برای آن ارسال می‌کند. به جرأت می‌توان گفت که ۹۰٪ سیستم‌های سرویس‌دهنده/ سرویس‌گیرندهٔ مبتنی بر COM به همین منوال کار می‌کنند.

پیاده‌سازی این کار بسیار ساده بوده ولی بدون شک مشکلاتی را به دنبال دارد. با استفاده از این روش همواره چرخه‌ای از رویدادها در سیستم وجود خواهد داشت که ممکن است تا مدت زمانی هیچ پاسخی به آنها داده نشود. این مسئله باعث بروز ترافیک در شبکه شده و کارآیی سیستم را تا حد قابل ملاحظه‌ای

۱- Just_In_Time

۲- Stateless : اصطلاحی در رابطه با فرآیندهایی که بدون نظارت بر جزئیات وضعیت یک فعالیت، در آن شرکت می‌کنند. به عنوان مثال سیستمی که پیامها را مدیریت می‌کند ممکن است تنها مبدأ و مقصد پیامها را در نظر داشته باشد و به محتوای آنها توجهی نکند.

۳- در برنامه‌سازی به فرآیندی گفته می‌شود که بخشی از یک فرآیند بزرگتر یا یک برنامه باشد.

کاهش می‌دهد.

اما COM راه‌حل ساخت یافته و مؤثری برای این مشکل ارائه کرده است. در این روش با کنترل انتقال اطلاعات بین سرویس‌دهنده و سرویس‌گیرنده از بروز ترافیک کاری جلوگیری به عمل می‌آید. این تکنیک TCE[1] نام دارد. در این حالت یک ارتباط دو طرفه بین سیستم سرویس‌گیرنده و سرویس‌دهنده برقرار می‌شود.

توجه داشته باشید که سیستم‌های سرویس‌دهنده و سرویس‌گیرنده می‌بایست هم زمان با هم و تحت شرایط مساوی شروع به کار نمایند و در غیر این صورت امکان پیاده‌سازی TCE وجود نخواهد داشت.

در COM+ این روش کمی تغییر کرده است. COM+ از تکنیک دیگری به نام LCE[2] برای مدیریت رویدادها استفاده می‌کند. در این روش، سیستم سرویس‌دهنده به عنوان توزیع کننده رویدادها و سیستم سرویس‌گیرنده به عنوان متقاضی رویدادها معرفی می‌شوند. شکل ۸-۱۳ ساختار داخلی مربوط به مدیریت رویدادها را در COM+ نمایش می‌دهد.

همانطور که در شکل ملاحظه می‌کنید، به محض ثبت شدن یک Event Class جدید توسط Publisher، فرآیند سیستم آغاز می‌شود (این کار به وسیلهٔ ابزار Component Services administration و یا از طریق متد ICOMAdminCatalog.InstallEventClass() انجام می‌گیرد). بعد از آن، خود Publisher و یا هر یک از اشیاء با فراخوانی متد CoCreateInstance نمونه‌ای از اشیاء را ایجاد می‌نمایند.

عموماً برای طراحی و ساخت رویدادهای مبتنی بر COM+، اجرای فرآیندهای زیرلازم وضروری‌است.

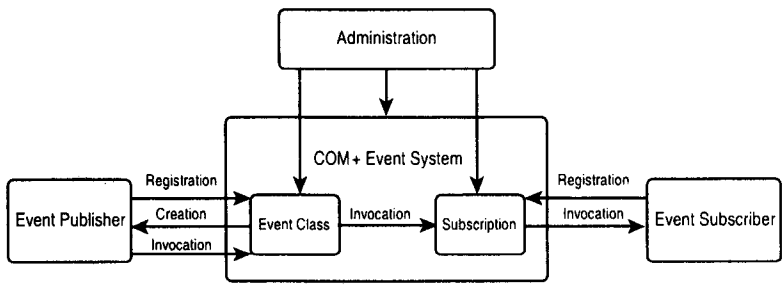
۱- ایجاد یک Event Class Server.

۲- ثبت و پیکربندی Event Class Server.

۳- ایجاد یک Subscriber Server.

۴- ثبت و پیکربندی Subscriber Servers.

۵- توزیع رویدادها.



شکل ۸-۱۳ ساختار مدیریت رویدادها در COM+

۱- Tightly Coupled Event (TCE) : در این روش موفقیت و میزان کارآیی هر یک از سیستم‌ها به دیگری وابسته است.

۲- Loosely Coupled Event (LCE) : در این روش وابستگی سیستم‌ها در مقایسه با روش TCE کمتر خواهد بود.

ایجاد یک Event Class Server

ActiveX Library Wizard ابزاری است که از آن برای ایجاد یک Event Class Server استفاده می‌شود. در اینجا با ارائه یک مثال، نحوهٔ انجام این کار را توضیح خواهیم داد.

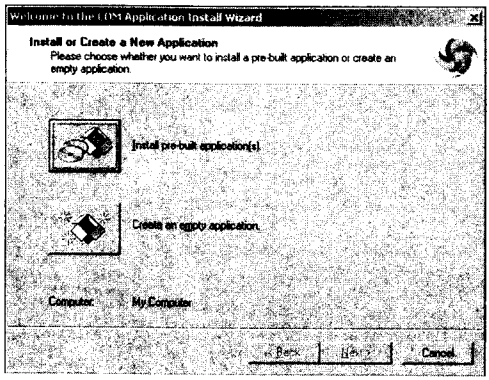
ابتدا با فراخوانی ActiveX Library Wizard ، یک COM Server DLL جدید ایجاد کنید. برای تولید Interface و کلاس‌های این برنامه از Automation Object Wizard استفاده نمائید. شیء ساخته شده را EventObj بنامید. سپس متدی تحت عنوان MyEvent را در قسمت Interface مربوط به IEventObj وارد نمائید. لیست ۳-۱۳ نحوهٔ پیاده‌سازی یک تابع کتابخانه‌ای برای ایجاد Event Class Server را نشان می‌دهد. این کد همهٔ آن چیزی است که برای ایجاد یک Event Class Server لازم است. در گام بعدی نحوهٔ ثبت و پیکربندی آن را توضیح می‌دهیم.

ثبت و پیکربندی Event Class Server

دوباره به ابزار Component Services administration نیاز پیدا کردیم. اکنون با این ابزار به خوبی آشنایی دارید، چراکه تقریباً در طراحی تمام برنامه‌های COM+ از آن استفاده کرده‌اید.

لیست ۳-۱۳ ایجاد Event Class Server

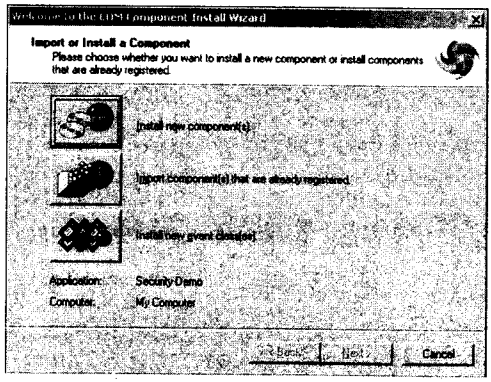
```
unit PubMain;

interface

uses
  ComObj, ActiveX, Publisher_TLB, StdVcl;

type
  TEventObj = class(TAutoObject, IEventObj)
  protected
    function MyEvent(const EventParam: WideString): HResult; safecall;
  end;

implementation
uses ComServ;

function TEventObj.MyEvent(const EventParam: WideString): HResult;
begin

end;

initialization
  TAutoObjectFactory.Create(ComServer, TEventObj, Class_EventObj,
    ciMultiInstance, tmApartment);
end.
```



شکل ۹-۱۳ اضافه کردن یک برنامهٔ +COM با استفاده از ابزار Component Services

ابتدا با استفاده از این ابزار یک برنامهٔ جدید +COM (همانند شکل ۹-۱۳) به نمایش در خواهد آمد. اکنون گزینهٔ Create an empty application را انتخاب کنید. با این کار شما یک برنامهٔ +COM را نصب کرده‌اید.

مجدداً Component Services را فراخوانی کرده و بر روی عنوان Component در سمت چپ منوی ظاهر شده (شکل ۱-۱۳)، گزینهٔ New را انتخاب کنید. هم اینک کادری مانند شکل ۱۰-۱۳ نمایش داده می‌شود. از این کادر گزینهٔ Install new event class را انتخاب کرده و آن را در قالب یک فایل (با تخصیص نام به آن) ذخیره نمائید.



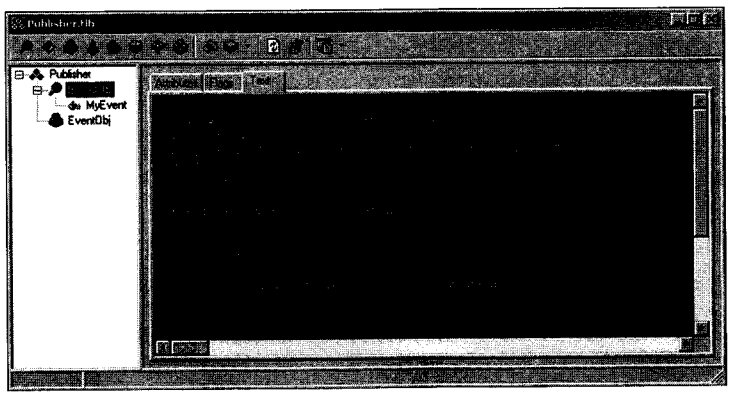
شکل ۱۰-۱۳ اضافه کردن یک جزء سازندهٔ +COM با استفاده از ابزار Component Services

ایجاد یک Subscriber Server

Subscriber Server جزء سرورهای استاندارد دلفی بوده و پیاده‌سازی آن وقت زیادی از شما نخواهد گرفت. لیست ۴-۱۳ پیاده‌سازی یک Subscriber Server را نشان می‌دهد. نحوهٔ تعریف Interface نیز در شکل ۱۱-۱۳ نمایش داده شده است.

لیست ۴-۱۳ پیاده‌سازی Subscriber Server

```
unit SubMain;

interface

uses
  ComObj, ActiveX, Subscriber_TLB, StdVcl, Publisher_TLB;

type
  TSubObj = class(TAutoObject, ISubObj, IEventObj)
  protected
    function MyEvent(const EventParam: WideString): HResult; safecall;
    { Protected declarations }
  end;

implementation

uses ComServ, Windows;

function TSubObj.MyEvent(const EventParam: WideString): HResult;
begin
  MessageBox(0, PChar(string(EventParam)), 'COM+ Event!', MB_OK);
  Result := S_OK;
end;

initialization
  TAutoObjectFactory.Create(ComServer, TSubObj, Class_SubObj,
    ciMultiInstance, tmApartment);
end.
```



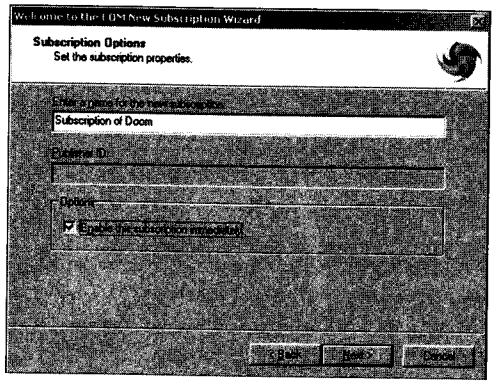
شکل ۱۱-۱۳ اضافه نمودن یک Interface با نام IEventObj.
این کار با استفاده از ویراستار Type Library دلفی صورت گرفته است

ثبت و پیکربندی Subscriber Server

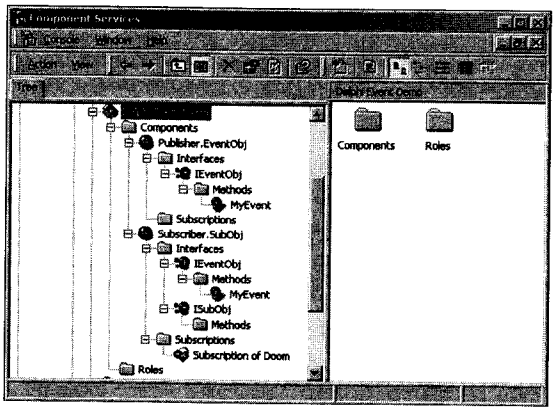
برای انجام این کار پس از فراخوانی Com Componenet Install Wizard (شکل ۱۰-۱۳) گزینهٔ Install new component را انتخاب کرده و پس از آن عنوان Subscriber DLL را برگزینید. اکنون بر روی گرهٔ Subscription از Subscriber Server و New را انتخاب کنید.

با طی مراحل فوق کادر New Subscription Wizard به نمایش درمی‌آید. برای متد Subscriber، متد IEventObj و برای Event class گزینهٔ Publisher.EventObj را انتخاب نمائید. نام Subscription را Subscription Of Doom گذاشته و گزینهٔ Enable This Subscription Immediately را فعال نمائید. (شکل ۱۲-۱۳)

نمای کامل این برنامه COM+ در شکل ۱۳-۱۳ به نمایش در آمده است.



شکل ۱۲-۱۳ منوی Subscription Wizard مربوط به Component Services



شکل ۱۳-۱۳ اجزاء برنامهٔ طراحی شده بر روی Component Services

توزیع رویدادها

تا کامل شدن برنامه تنها یک گام دیگر باقی است. برای توزیع رویدادها ابتدا می‌بایست یک نمونه از کلاس EventObj ایجاد کرده و سپس متد IEventObj.MyEvent را فراخوانی کنید. نحوهٔ انجام این کار در لیست ۵-۱۳ ارائه شده است.

لیست ۵-۱۳ یونیتی برای توزیع رویدادها

```
unit TestU;

interface

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

type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    FEvent: IEventObj;
  end;

var
  Form1: TForm1;

implementation

uses ComObj, ActiveX;

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
begin
  OleCheck(CoCreateInstance(CLASS_EventObj, nil, CLSCTX_ALL, IEventObj,
    FEvent));
  FEvent.MyEvent('This is a clever string');
end;

end.
```

پس از اجرای کامل برنامه و با فشردن دکمهٔ موجود در روی فرم اصلی، شکلی همانند شکل ۱۴-۱۳ نمایان خواهد شد.

شکل ۱۴-۱۳ اجرای برنامه

Runtime

مفهوم Runtime در COM+ به معنی مدیریت و دستکاری اشیاء، مدیریت proxy، کنترل چرخهٔ حیات اشیاء، مدیریت حافظه و در حالت کلی انجام تمامی اعمال سطح پایین نرم‌افزاری است.

در اینجا لازم می‌دانیم شما را با مفاهیم ذیل در خصوص Runtime آشنا سازیم.

Registration Database (RegDB)

دو مکان مختلف برای نگهداری صفات و ویژگیهای اشیاء مبتنی بر COM، وجود دارد. این دو مکان عبارت از type library و system registary می‌باشند.

COM+ این اطلاعات را در محلی به نام registration database ثبت و نگهداری می‌نماید. برخی اوقات یک registration database اطلاعات تراکنشی اشیاء را نیز در خود نگهداری می‌نماید.

اجزای سازندهٔ پیکربندی شده

تمامی اجزایی که صفات و ویژگیهای آنها در یک registration database نگهداری می‌شود به اجزای سازندهٔ پیکربندی شده معروف‌اند. عموماً در محیط‌های COM+ از این اجزاء برای طراحی برنامه‌های کاربردی استفاده می‌شود.

Context

این عبارت وضعیت یک فرآیند اجرایی در سیستم را توصیف می‌نماید. منظور از فرآیند اجرایی، فرآیندهای نظیر تراکنش‌ها در سیستم می‌باشد.

Neutral Threading

مفهوم Thread را قبلاً توضیح داده‌ایم. طراحان COM+ با ارائه مدل جدیدی از Thread ها، سعی در ارتقاء کارآیی سیستم‌ها و توسعه‌پذیری آن داشته‌اند.

ساخت یک برنامهٔ کاربردی COM+

تا به اینجا کم و بیش با دنیای COM+ آشنا شده‌اید. اما اکنون وقت آن رسیده است تا از ویژگیهای COM+ (نظیر مدیریت تراکنشها، منابع و چرخهٔ حیات اشیاء) در برنامه‌های کاربردی خود استفاده کنید. اکنون پس از بیان چند مفهوم دیگر به سراغ برنامه‌نویسی COM+ خواهیم رفت.

هدف افزایش کارآیی سیستم‌های کاربردی است

آنچه که هر روز دنیای نرم‌افزار را دچار تحول می‌سازد، ارائهٔ تکنولوژی‌هایی است که هدف آنها افزایش سرعت و بهره‌وری سیستم‌هاست. اگر تا چند سال پیش، برقراری ارتباط بین کامپیوترها محدود به

داشتن نرم‌افزارهای پیچیده و خاص می‌شد، امروزه اینترنت آنچنان شگفتی در عرصهٔ ارتباطات به وجود آورده که در عرض چند ثانیه حجم وسیعی از اطلاعات را در اختیارتان قرار می‌دهد.

سیستم‌های سرویس‌دهنده/ سرویس‌گیرنده نیز تکنولوژی دیگری است که در راستای هدف فوق، روش طراحی و پیاده‌سازی برنامه‌های کاربردی امروزی را متحول ساخته است. این سیستم‌ها عموماً بر پایهٔ خصوصیتهای شیءگرا استوار بوده و بدین لحاظ از قابلیت انعطاف بالایی برخوردارند.

اشیاء Stateful و Stateless

در این قسمت یک دید کلی از اشیاء Stateful و Stateless ارائه می‌کنیم. مفهوم Stateful در خصوص اشیاء و فرآیندهایی مصداق دارد که بر تمامی جزئیات فعالیتی که در آن شرکت می‌کنند، نظارت دارند. به عنوان مثال سیستمی که پیام‌ها را مدیریت می‌کند، محتوای پیام‌ها را نیز در نظر بگیرد. در مقابل، ویژگی Stateless به اشیاء و فرآیندهایی گفته می‌شود که بدون نظارت بر جزئیات یک فعالیت، در آن شرکت می‌کنند. به عنوان مثال یک سیستم مدیریت پیام‌ها ممکن است تنها مبدأ و مقصد پیام‌ها را در نظر داشته باشد و به محتوای آنها توجهی نکند.

اما COM و +COM از چه الگویی استفاده می‌کنند؟ COM از اشیاء و فرآیندهای Stateful استفاده می‌کند. لازمهٔ این کار نگهداری اطلاعات وضعیتی اشیاء و فرآیندها، در زمان ایجاد آنهاست. لازم به ذکر است که COM از این اطلاعات تا زمان تخریب یک شیء محافظت می‌کند.

مشکل استفاده از اشیاء Stateful ، پایین آمدن کارآیی سیستم است. +COM با هدف رفع این مشکل از اشیاء Stateless استفاده می‌کند. با این کار ضمن بهبود کارآیی سیستم، در استفاده از منابع موجود نیز صرفه‌جویی خواهد شد. تنها نکته‌ای که باقی می‌ماند، طراحی و پیاده‌سازی Interface های لازم برای کنترل و مدیریت این اشیاء است. به عنوان مثال Interface زیر را در نظر بگیرید.

```
ICheckbook = interface
['{2CCF0409-EE29-11D2-AF31-0000861EF0BB}']
  procedure SetAccount(AccountNum: WideString); safecall;
  procedure AddActivity(Amount: Integer); safecall;
end;
```

فرض کنید این Interface را به صورت زیر به کار برده‌اید:

```
var
  CB: ICheckbook;
begin
  CB := SomehowGetInstance;
  CB.SetAccount('12345ABCDE');  // open my checking account
  CB.AddActivity(-100);         // add a debit for $100
  ...
end;
```

مشکل موجود در این نوع از Interface ها، Stateless نبودن اشیاء در هنگام فراخوانی متدهاست. با

استفاده از متد ()AddActivity در COM+ که وظیفهٔ گذر از اطلاعات ضروری را دارد، این مشکل مرتفع گردیده است. نحوهٔ فراخوانی این متد به صورت زیر است:

```
procedure AddActivity(AccountNum: WideString; Amount: Integer); safecall;
```

توجه داشته باشید که تنها برخی از فعالیتهای سیستم در فرآیندهای Stateless نادیده گرفته می‌شوند. اطلاعات تراکنشی اشیاء، نحوهٔ امنیت و پنهان‌سازی آنها از جمله فعالیتهایی هستند که در سیستم نگهداری خواهند شد. هر فرآیند در محیط‌های COM+ با فراخوانی متد ()GetObjectContext به این اطلاعات دسترسی پیدا می‌کند. Interface ای که برای انجام این کار از آن استفاده می‌شود IObjectContext نام داشته و در یونیت Mtx به شکل زیر تعریف شده است.

```
IObjectContext = interface(IUnknown)
    ['{51372AE0-CAE7-11CF-BE81-00AA00A2FA25}']
    function CreateInstance(const cid, rid: TGUID; out pv): HResult; stdcall;
    procedure SetComplete; safecall;
    procedure SetAbort; safecall;
    procedure EnableCommit; safecall;
    procedure DisableCommit; safecall;
    function IsInTransaction: Bool; stdcall;
    function IsSecurityEnabled: Bool; stdcall;
    function IsCallerInRole(const bstrRole: WideString): Bool; safecall;
  end;
```

متدهای ()SetComplete و ()SetAbort وظیفهٔ ذخیره‌سازی اطلاعات ذکر شده را به عهده دارند. تأثیر و یا نادیده انگاشتن تراکنشها (Rollback) نیز جزو وظایف متدهای ()SetAbort و ()SetComplete است.

چرخهٔ حیات اشیاء

در برنامه‌نویسی COM، عملیات ذخیره‌سازی اطلاعات اشیاء، همزمان با ایجاد آنها انجام می‌پذیرد. این ذخیره‌سازی اطلاعات تا زمان تخریب شیء به طول می‌انجامد. همانطور که گفته شد، این کار افت کارآیی سیستم را به همراه دارد. COM+ با فراخوانی خودکار متدهای ()SetComplete و ()SetAbort از این اطلاعات تنها در مواقعی نگهداری می‌کند که ارجاعی به اشیاء صورت گرفته است.

سازماندهی برنامه‌های کاربردی COM+

COM+ تمامی اجزاء موجود در یک Package را تحت یک فرآیند به اجرا در می‌آورد. این کار شما را قادر خواهد ساخت تا در شرایط بهینه‌تری به خطایابی Package هایتان بپردازید.

دو روش برای سازماندهی و ساخت برنامه‌های COM+ در دلفی وجود دارد. یک روش فراخوانی ابزار Component Services بوده که تا اینجا با آن آشنا شده‌اید. روش دیگر استفاده از منوی

Install COM+Object می‌باشد. معطل چه هستید؟ به منوی مربوطه بروید و کارهایی را که تاکنون آموخته‌اید به اجرا در آورید.

مروری بر تراکنشها

همانطور که قبلاً اشاره شد، مدیریت تراکنشها یک امر ضروری در برنامه‌های کاربردی امروزی و بخصوص برنامه‌های سرویس‌دهنده/ سرویس‌گیرنده است.

اما آنچه که برنامه‌نویسان را نگران می‌سازد، چگونگی مدیریت تراکنشهاست. باید توجه داشته باشید که این کار به سادگی قابل پیاده‌سازی است. مهمترین مسئله در تراکنش‌ها، بروزرسانی همزمان همهٔ فایل‌هاست. یک تراکنش، یا تمامی اطلاعات در فایل‌های مختلف را بروز در می‌آورد و یا در صورت بوجود آمدن اشکال کلیهٔ فایل‌ها به حالت اولیه برمی‌گردند (Rollback).

با طرح یک مثال به بررسی بیشتر این مسئله می‌پردازیم. فرض کنید برنامهٔ کاربردی برای یک سیستم بانکی پیاده‌سازی کرده‌اید. بدون شک یکی از قسمتهای این برنامه پرداخت و وصول چکهای مشتریان است. حال فرض کنید که یک مشتری چندین چک را برای یک روز خاص و از یک حساب خاص ارائه داده است. به محض ورود چک به باجهٔ بانک ابتدا لازم است تا حساب موردنظر مسدود گردد و پس از آن عملیات پرداخت مبلغ چک صورت گیرد. پس از این کار لازم است تا تمامی اطلاعات مالی مربوط به همان مشتری در این حساب خاص سریعاً و همزمان بروز آورده شود. هرگونه اشکال در نحوهٔ پیاده‌سازی این عملیات، باید سیستم را به حالت قبل از تغییرات (Rollback) هدایت نماید. این تمام آن چیزی است که در سیستم به نام تراکنش معروف است.

COM+ در دلفی

آنچه که تاکنون آموخته‌اید محدود به ویژگیها، مشخصات و مزایای COM+ بوده است. اما دلفی چگونه از COM+ پشتیبانی می‌کند؟ توجه داشته باشید که تنها نسخهٔ Client/Server دلفی قابلیت پشتیبانی برنامه‌های COM+ را داراست. در نسخه‌های Professional دلفی نیز امکاناتی جهت ساخت اجزاء سازندهٔ COM+ ارائه شده است. اکنون قصد داریم نحوهٔ پیاده‌سازی برنامه‌های COM+ در دلفی را همراه با ارائه یک مثال بررسی نمائیم.

ویزاردهای COM+

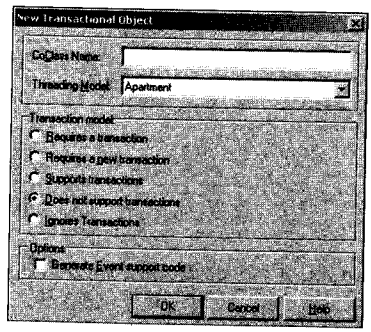
دلفی از دو نوع ویزارد برای ساخت اجزاء سازندهٔ COM+ استفاده می‌کند.

۱- Transactional Data Wizard که بر روی تب Multitier دلفی وجود دارد. (برای ساخت سرورهای MIDAS که بر روی محیطهای COM+ نصب و اجرا می‌گردند از این ویزارد استفاده می‌شود).

۲- Transactional Object Wizard که بر روی تب ActiveX دلفی قرار دارد.

بحث خود را بر روی Transactional Object Wizard متمرکز می‌کنیم. با فراخوانی این ویزارد کادری همانند شکل ۱۵-۱۳ به نمایش در خواهد آمد. اولین گام در این قسمت، تنظیم مدل تراکنشی است که بر روی جعبهٔ مکالمهٔ Transaction Model وجود دارد. شرح این مدلها در زیر ارائه شده است.

- Requiers a Transaction : اجزاء سازنده همواره در متن یا محتوای تراکنش ایجاد خواهند شد و خصوصیت تراکنش سازندهٔ خود را در صورت وجود به ارث می‌برند.
- Requiers a New Transaction : تراکنشی برای یک جزء سازنده ایجاد و در درون آن اجرا می‌شود.
- Supports Transaction : اجزاء سازنده خصوصیت مربوط به تراکنش سازندهٔ خود را به ارث خواهند برد.
- Dose Not Support Transation : اجزاء سازنده در متن یا محتوای تراکنش ایجاد نمی‌شوند.
- Ignore Transaction : هیچ تراکنشی برای سیستم در نظر گرفته نمی‌شود.



شکل ۱۵-۱۳ COM+ Transaction Object Wizard

چارچوب‌کاری COM+

کلاس TMtsAutoObject مهمترین کلاس برای طراحی شیءهای مبتی بر COM+ است. این کلاس که در یونیت MtsObject وجود دارد به صورت زیر پیاده‌سازی شده است.

```
type
  TMtsAutoObject = class(TAutoObject, IObjectControl)
  private
    FObjectContext: IObjectContext;
  protected
    { IObjectControl }
    procedure Activate; safecall;
    procedure Deactivate; stdcall;
    function CanBePooled: Bool; stdcall;

    procedure OnActivate; virtual;
    procedure OnDeactivate; virtual;
    property ObjectContext: IObjectContext read FObjectContext;
  public
```

```
  procedure SetComplete;
  procedure SetAbort;
  procedure EnableCommit;
  procedure DisableCommit;
  function IsInTransaction: Bool;
  function IsSecurityEnabled: Bool;
  function IsCallerInRole(const Role: WideString): Bool;
end;
```

مقداردهی اولیه اجزاء سازندهٔ +COM به وسیلهٔ IObjectControl (یک Interface می‌باشد) انجام می‌گیرد. این Interface از سه متد به شرح زیر استفاده می‌کند:

- ()Activate : این متد وظیفهٔ مقداردهی اولیهٔ مشخصات یک شیء در هنگام ایجاد آن را به عهده دارد.
- ()Deactivate : این متد وظیفهٔ حذف مشخصات یک شیء در سیستم را در هنگام تخریب آن به عهده دارد.
- ()CanBePooled : این متد تنها توسط +COM پشتیبانی می‌شود. برای دریافت اطلاعات بیشتر در مورد این متد به راهنمای دلفی مراجعه کنید.

شرح متدهای IObjectContext در زیر آورده شده است.

- ()CreateInstance : این متد وظیفهٔ ایجاد یک نمونه از اشیاء +COM را به عهده دارد. این متد در محیطهای COM به شکل ()IClassFactory.CreateInstance فراخوانی شده است.
- ()SetComplete : این متد با ارسال یک سیگنال به +COM، اتمام کار یک شیء یا یک فرآیند را اطلاع می‌دهد. ضمناً تأیید عملیات بروزرسانی فایل‌ها (در سیستم‌های تراکنشی) نیز به عهدهٔ این متد است.
- ()SetAbort : به نوعی کارآیی این متد مشابه پارامتر ()SetComplete است. پاسخ‌دهی به خطاها و به تعویق انداختن تراکنشها نیز به عهدهٔ این متد می‌باشد.
- ()EnableCommit : این متد بر تأیید ایجاد یک شیء و یا تأیید یک تراکنش دلالت دارد.
- ()DisableCommit : این متد بر بی‌ثباتی (کامل نشدن) یک شیء و یا یک فرآیند دلالت دارد.
- ()IsInTransaction : از این متد جهت اجرای یک جزء سازنده در یک تراکنش استفاده می‌شود.
- ()IsSecutityEnabled : از این متد جهت تشخیص پیاده‌سازی امنیت یک شیء در +COM استفاده می‌شود.
- ()IsCallerInRole : این متد به عنوان قلب سیستم امنیتی +COM عمل می‌کند. همانطور که گفته شد امنیت در +COM به وسیلهٔ روش Role_Based پیاده‌سازی شده است. متد مذکور صلاحیت یک کاربر برای فراخوانی اجزاء سازنده را تعیین می‌کند.

بازی Tic-Tac-Toe، یک برنامهٔ کاربردی ساده

تا اینجا تمام نکات مربوط به برنامه‌نویسی مبتنی بر +COM شامل تعریف کلاس‌ها، متدها،

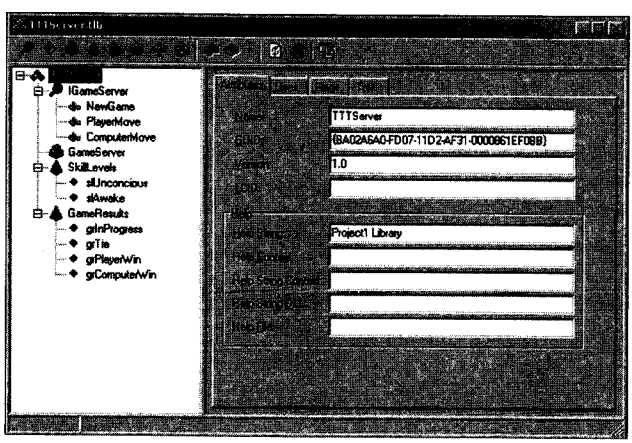
Inteface ها و تعریف اجزاء سازندهٔ مناسب را فراگرفته‌اید. اکنون مثال کاربردی خود تحت عنوان "طراحی بازی Tic_Tac_Toe" را مطرح می‌نمائیم.

با استفاده از Transactional Object Wizard یک شیء جدید ساخته و آن را GameServer نام‌گذاری کنید. اکنون ویراستار Type Library را فراخوانی کنید و IGameServer را به Interface پیش‌فرض این شیء اضافه کنید. طبق همین روال سه متد جدید به نام‌های NewGame()، ComputerMove() و PlayerMove() و دو enum جدید به نام‌های SkillLevels و GameResults به آن بیافزائید. در صورت نیاز می‌توانید به شکل ۶-۱۳ مراجعه کنید. این شکل ترتیب پیاده‌سازی اجزاء و متدها را در Type Library نمایش می‌دهد.

مد NewGame() را برای مقداردهی اولیه بازی جهت یک سرویس گیرنده تنظیم خواهیم نمود. ComputerMove() را طوری طراحی می‌کنیم که وظیفهٔ تعیین حرکات مجاز و ساخت این حرکات از طرف کامپیوتر را برعهده گیرد. پیاده‌سازی متد PlayerMove() نیز براساس حرکات مجاز کاربر در طرف سرویس‌گیرنده خواهد بود.

توجه داشته باشید که بر طبق گفته‌های گذشته، تمامی اشیاء و فرآیندهای COM+ از نوع Sateless هستند، با توجه به این اصل پیاده‌سازی متد NewGame() به صورت زیر خواهد بود.

```
procedure TGameServer.NewGame(out GameID: Integer);
var
  SPG: ISharedPropertyGroup;
  SProp: ISharedProperty;
  Exists: WordBool;
  GameData: OleVariant;
begin
  // Use caller's role to validate security
  CheckCallerSecurity;
  // Get shared property group for this object
```



شکل ۱۶-۱۳ اجزاء برنامهٔ Tic-Tac-Toe در ویراستار Type Library

```
  SPG := GetSharedPropertyGroup;
  // Create or retrieve NextGameID shared property
  SProp := SPG.CreateProperty('NextGameID', Exists);
  if Exists then GameID := SProp.Value
  else GameID := 0;
  // Increment and store NextGameID shared property
  SProp.Value := GameID + 1;
  // Create game data array
  GameData := VarArrayCreate([1, 3, 1, 3], varByte);
  SProp := SPG.CreateProperty(Format(GameDataStr, [GameID]), Exists);
  SProp.Value := GameData;
  SetComplete;
end;
```

متد CheckCallerSecurity() در بالاکه عهده‌دار امنیت برنامه است به صورت زیر پیاده‌سازی می‌شود. این متد از روش Role_Based برای تضمین امنیت برنامه استفاده می‌کند.

```
procedure TGameServer.CheckCallerSecurity;
begin
  // Just for fun, only allow those in the "TTT" role to play the game.
  if IsSecurityEnabled and not IsCallerInRole('TTT') then
    raise Exception.Create('Only those in the TTT role can play tic-tac-toe');
end;
```

اما چگونه یک "TTT" به یک Role اختصاص داده می‌شود و تشخیص ارتباط کاربران با این Role به چه ترتیبی است؟ پاسخ این سؤال بسیار ساده است. اگر چه که برنامه‌نویسی یک راه‌حل این مسأله است اما ما راه‌حل آسان‌تری را پیشنهاد می‌کنیم و آن هم استفاده از Transaction Server Explorer است. با فراخوانی آن، نحوهٔ کاربری آن را یاد خواهید گرفت. متدهای ComputerMove() و PlayerMove() نیز به صورت زیر پیاده‌سازی می‌شوند.

```
procedure TGameServer.ComputerMove(GameID: Integer;
  SkillLevel: SkillLevels; out X, Y: Integer; out GameRez: GameResults);
var
  Exists: WordBool;
  PropVal: OleVariant;
  GameData: PGameData;
  SProp: ISharedProperty;
begin
  // Get game data shared property
  SProp := GetSharedPropertyGroup.CreateProperty(Format(GameDataStr, [GameID]),
    Exists);
  // Get game data array and lock it for more efficient access
  PropVal := SProp.Value;
  GameData := PGameData(VarArrayLock(PropVal));
  try
```

```
    // If game isn't over, then let computer make a move
    GameRez := CalcGameStatus(GameData);
    if GameRez = grInProgress then
    begin
      CalcComputerMove(GameData, SkillLevel, X, Y);
      // Save away new game data array
      SProp.Value := PropVal;
      // Check for end of game
      GameRez := CalcGameStatus(GameData);
    end;
  finally
    VarArrayUnlock(PropVal);
  end;
  SetComplete;
end;

procedure TGameServer.PlayerMove(GameID, X, Y: Integer;
  out GameRez: GameResults);
var
  Exists: WordBool;
  PropVal: OleVariant;
  GameData: PGameData;
  SProp: ISharedProperty;
begin
  // Get game data shared property
  SProp := GetSharedPropertyGroup.CreateProperty(Format(GameDataStr, [GameID]
    Exists);
  // Get game data array and lock it for more efficient access
  PropVal := SProp.Value;
  GameData := PGameData(VarArrayLock(PropVal));
  try
    // Make sure game isn't over
    GameRez := CalcGameStatus(GameData);
    if GameRez = grInProgress then
    begin
      // If spot isn't empty, raise exception
      if GameData[X, Y] <> EmptySpot then
        raise Exception.Create('Spot is occupied!');
      // Allow move
      GameData[X, Y] := PlayerSpot;
      // Save away new game data array
      SProp.Value := PropVal;
      // Check for end of game
      GameRez := CalcGameStatus(GameData);
    end;
  finally
    VarArrayUnlock(PropVal);
  end;
  SetComplete;
end;
```

لیست ۶-۱۳ ، کد کامل این برنامه را نشان می‌دهد. فراموش نکنید که این برنامه بر پایهٔ COM+ پیاده‌سازی شده است.

لیست ٦-۱۳ یونیت اصلی برنامهٔ Tic-Tac-Toe

```
unit ServMain;

interface

uses
  ActiveX, MtsObj, Mtx, ComObj, TTTServer_TLB;

type
  PGameData = ^TGameData;
  TGameData = array[1..3, 1..3] of Byte;

  TGameServer = class(TMtsAutoObject, IGameServer)
  private
    procedure CalcComputerMove(GameData: PGameData; Skill: SkillLevels;
      var X, Y: Integer);
    function CalcGameStatus(GameData: PGameData): GameResults;
    function GetSharedPropertyGroup: ISharedPropertyGroup;
    procedure CheckCallerSecurity;
  protected
    procedure NewGame(out GameID: Integer); safecall;
    procedure ComputerMove(GameID: Integer; SkillLevel: SkillLevels; out X,
      Y: Integer; out GameRez: GameResults); safecall;
    procedure PlayerMove(GameID, X, Y: Integer; out GameRez: GameResults);
      safecall;
  end;

implementation

uses ComServ, Windows, SysUtils;

const
  GameDataStr = 'TTTGameData%d';
  EmptySpot = 0;
  PlayerSpot = $1;
  ComputerSpot = $2;

function TGameServer.GetSharedPropertyGroup: ISharedPropertyGroup;
var
  SPGMgr: ISharedPropertyGroupManager;
  LockMode, RelMode: Integer;
  Exists: WordBool;
begin
  if ObjectContext = nil then
    raise Exception.Create('Failed to obtain object context');
```

لیست ۶-۱۳ ادامه

```
  // Create shared property group for this object
  OleCheck(ObjectContext.CreateInstance(CLASS_SharedPropertyGroupManager,
    ISharedPropertyGroupManager, SPGMgr));
  LockMode := LockSetGet;
  RelMode := Process;
  Result := SPGMgr.CreatePropertyGroup('DelphiTTT', LockMode, RelMode, Exists);
  if Result = nil then
    raise Exception.Create('Failed to obtain property group');
end;

procedure TGameServer.NewGame(out GameID: Integer);
var
  SPG: ISharedPropertyGroup;
  SProp: ISharedProperty;
  Exists: WordBool;
  GameData: OleVariant;
begin
  // Use caller's role to validate security
  CheckCallerSecurity;
  // Get shared property group for this object
  SPG := GetSharedPropertyGroup;
  // Create or retrieve NextGameID shared property
  SProp := SPG.CreateProperty('NextGameID', Exists);
  if Exists then GameID := SProp.Value
  else GameID := 0;
  // Increment and store NextGameID shared property
  SProp.Value := GameID + 1;
  // Create game data array
  GameData := VarArrayCreate([1, 3, 1, 3], varByte);
  SProp := SPG.CreateProperty(Format(GameDataStr, [GameID]), Exists);
  SProp.Value := GameData;
  SetComplete;
end;

procedure TGameServer.ComputerMove(GameID: Integer;
  SkillLevel: SkillLevels; out X, Y: Integer; out GameRez: GameResults);
var
  Exists: WordBool;
  PropVal: OleVariant;
  GameData: PGameData;
  SProp: ISharedProperty;
begin
  // Get game data shared property
  SProp := GetSharedPropertyGroup.CreateProperty(Format(GameDataStr, [GameID]),
    Exists);
  // Get game data array and lock it for more efficient access
  PropVal := SProp.Value;
  GameData := PGameData(VarArrayLock(PropVal));
```

لیست ۶-۱۳ ادامه

```
  try
    // If game isn't over, then let computer make a move
    GameRez := CalcGameStatus(GameData);
    if GameRez = grInProgress then
    begin
      CalcComputerMove(GameData, SkillLevel, X, Y);
      // Save away new game data array
      SProp.Value := PropVal;
      // Check for end of game
      GameRez := CalcGameStatus(GameData);
    end;
  finally
    VarArrayUnlock(PropVal);
  end;
  SetComplete;
end;
procedure TGameServer.PlayerMove(GameID, X, Y: Integer;
  out GameRez: GameResults);
var
  Exists: WordBool;
  PropVal: OleVariant;
  GameData: PGameData;
  SProp: ISharedProperty;
begin
  // Get game data shared property
  SProp := GetSharedPropertyGroup.CreateProperty(Format(GameDataStr, [GameID]),
    Exists);
  // Get game data array and lock it for more efficient access
  PropVal := SProp.Value;
  GameData := PGameData(VarArrayLock(PropVal));
  try
    // Make sure game isn't over
    GameRez := CalcGameStatus(GameData);
    if GameRez = grInProgress then
    begin
      // If spot isn't empty, raise exception
      if GameData[X, Y] <> EmptySpot then
        raise Exception.Create('Spot is occupied!');
      // Allow move
      GameData[X, Y] := PlayerSpot;
      // Save away new game data array
      SProp.Value := PropVal;
      // Check for end of game
      GameRez := CalcGameStatus(GameData);
    end;
  finally
    VarArrayUnlock(PropVal);
  end;
```

لیست ۶-۱۳ ادامه

```
  SetComplete;
end;

function TGameServer.CalcGameStatus(GameData: PGameData): GameResults;
var
  I, J: Integer;
begin
  // First check for a winner
  if GameData[1, 1] <> EmptySpot then
  begin
    // Check top row, left column, and top left to bottom right diagonal for win
  if ((GameData[1, 1] = GameData[1, 2]) and
    (GameData[1, 1] = GameData[1, 3])) or
    ((GameData[1, 1] = GameData[2, 1]) and
    (GameData[1, 1] = GameData[3, 1])) or
    ((GameData[1, 1] = GameData[2, 2]) and
    (GameData[1, 1] = GameData[3, 3])) then
  begin
    Result := GameData[1, 1] + 1; // Game result is spot ID + 1
    Exit;
  end;
end;
if GameData[3, 3] <> EmptySpot then
begin
  // Check bottom row and right column for win
  if ((GameData[3, 3] = GameData[3, 2]) and
    (GameData[3, 3] = GameData[3, 1])) or
    ((GameData[3, 3] = GameData[2, 3]) and
    (GameData[3, 3] = GameData[1, 3])) then
  begin
    Result := GameData[3, 3] + 1; // Game result is spot ID + 1
    Exit;
  end;
end;
if GameData[2, 2] <> EmptySpot then
begin
  // Check middle row, middle column, and bottom left to top right
  // diagonal for win
  if ((GameData[2, 2] = GameData[2, 1]) and
    (GameData[2, 2] = GameData[2, 3])) or
    ((GameData[2, 2] = GameData[1, 2]) and
    (GameData[2, 2] = GameData[3, 2])) or
    ((GameData[2, 2] = GameData[3, 1]) and
    (GameData[2, 2] = GameData[1, 3])) then
  begin
    Result := GameData[2, 2] + 1; // Game result is spot ID + 1
    Exit;
  end;
```

لیست ۶-۱۳ ادامه

```
    end;
    // Finally, check for game still in progress
    for I := 1 to 3 do
      for J := 1 to 3 do
        if GameData[I, J] = 0 then
        begin
          Result := grInProgress;
          Exit;
        end;
    // If we get here, then we've tied
    Result := grTie;
end;

procedure TGameServer.CalcComputerMove(GameData: PGameData;
    Skill: SkillLevels; var X, Y: Integer);
type
    // Used to scan for possible moves by either row, column, or diagonal line
    TCalcType = (ctRow, ctColumn, ctDiagonal);
    // mtWin = one move away from win, mtBlock = opponent is one move away from
    // win, mtOne = I occupy one other spot in this line, mtNew = I occupy no
    // spots on this line
    TMoveType = (mtWin, mtBlock, mtOne, mtNew);
var
    CurrentMoveType: TMoveType;

    function DoCalcMove(CalcType: TCalcType; Position: Integer): Boolean;
    var
      RowData, I, J, CheckTotal: Integer;
      PosVal, Mask: Byte;
    begin
      Result := False;
      RowData := 0;
      X := 0;
      Y := 0;
      if CalcType = ctRow then
      begin
        I := Position;
        J := 1;
      end
      else if CalcType = ctColumn then
      begin
        I := 1;
        J := Position;
      end
      else begin
        I := 1;
```

لیست ۶-۱۳ ادامه

```
    case Position of
      1: J := 1; // scanning from top left to bottom right
      2: J := 3; // scanning from top right to bottom left
    else
      Exit;   // bail; only 2 diagonal scans
    end;
  end;
  // Mask masks off Player or Computer bit, depending on whether we're
  //thinking
  // offensively or defensively. Checktotal determines whether that is a row
  // we need to move into.
  case CurrentMoveType of
    mtWin:
      begin
        Mask := PlayerSpot;
        CheckTotal := 4;
      end;
    mtNew:
      begin
        Mask := PlayerSpot;
        CheckTotal := 0;
      end;
    mtBlock:
      begin
        Mask := ComputerSpot;
        CheckTotal := 2;
      end;
  else
    begin
      Mask := 0;
      CheckTotal := 2;
    end;
  end;
  // loop through all lines in current CalcType
  repeat
    // Get status of current spot (X, O, or empty)
    PosVal := GameData[I, J];
    // Save away last empty spot in case we decide to move here
    if PosVal = 0 then
    begin
      X := I;
      Y := J;
    end
    else
      // If spot isn't empty, then add masked value to RowData
      Inc(RowData, (PosVal and not Mask));
```

لیست ۶-۱۳ ادامه

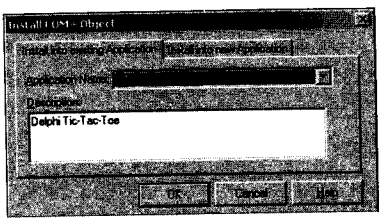
```
        if (CalcType = ctDiagonal) and (Position = 2) then
        begin
          Inc(I);
          Dec(J);
        end
       else begin
         if CalcType in [ctRow, ctDiagonal] then Inc(J);
         if CalcType in [ctColumn, ctDiagonal] then Inc(I);
       end;
    until (I > 3) or (J > 3);
    // If RowData adds up, then we must block or win, depending on whether
    // we're thinking offensively or defensively.
    Result := (X <> 0) and (RowData = CheckTotal);
    if Result then
    begin
      GameData[X, Y] := ComputerSpot;
      Exit;
    end;
  end;

var
  A, B, C: Integer;
begin
  if Skill = slAwake then
  begin
    // First look to win the game, next look to block a win
    for A := Ord(mtWin) to Ord(mtBlock) do
    begin
      CurrentMoveType := TMoveType(A);
      for B := Ord(ctRow) to Ord(ctDiagonal) do
        for C := 1 to 3 do
          if DoCalcMove(TCalcType(B), C) then Exit;
    end;
    // Next look to take the center of the board
    if GameData[2, 2] = 0 then
    begin
      GameData[2, 2] := ComputerSpot;
      X := 2;
      Y := 2;
      Exit;
    end;
    // Next look for the most advantageous position on a line
    for A := Ord(mtOne) to Ord(mtNew) do
```

لیست ۶-۱۳ ادامه

```
    begin
      CurrentMoveType := TMoveType(A);
      for B := Ord(ctRow) to Ord(ctDiagonal) do
        for C := 1 to 3 do
          if DoCalcMove(TCalcType(B), C) then Exit;
    end;
  end;
  // Finally (or if skill level is unconscious), just find the first open place
  for A := 1 to 3 do
    for B := 1 to 3 do
      if GameData[A, B] = 0 then
      begin
        GameData[A, B] := ComputerSpot;
        X := A;
        Y := B;
        Exit;
      end;
end;

procedure TGameServer.CheckCallerSecurity;
begin
  // Just for fun, only allow those in the "TTT" role to play the game.
  if IsSecurityEnabled and not IsCallerInRole('TTT') then
    raise Exception.Create('Only those in the TTT role can play tic-tac-toe');
end;

initialization
  TAutoObjectFactory.Create(ComServer, TGameServer, Class_GameServer,
    ciMultiInstance, tmApartment);
end.
```

نصب سرور

پس‌ازپیاده‌سازی برنامهٔ سرویس‌دهنده‌بایدآن را در COM+ دلفی نصب‌کنید. ابزار Install COM+ Objects را از منوی اصلی اجرا کرده، سپس Install COM+ Object را فراخوانی کنید. این کادر محاوره‌ای به شما اجازه می‌دهد تا اشیاء مورد نظرتان را در یک Package قرار دهید. (همانند شکل ۱۷-۱۳)



شکل ۱۷-۱۳ نصب یک شیء COM+ در دلفی

برنامهٔ سرویس‌گیرنده

علاوه بر پیاده‌سازی برنامهٔ سرویس‌دهنده و نصب آن می‌بایست یک برنامهٔ سرویس‌گیرنده +COM نیز برای این مثال طراحی شود. برنامهٔ سرویس‌گیرنده برای اجزاء سازندهٔ +COM مثال Tic-Tac-Toe در لیست ۷-۱۳ ارائه شده است.

لیست ۷-۱۳ برنامهٔ سرویس‌گیرندهٔ +COM

```
unit UiMain;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Buttons, ExtCtrls, Menus, TTTServer_TLB, ComCtrls;

type
  TRecord = record
    Wins, Loses, Ties: Integer;
  end;

  TFrmMain = class(TForm)
    SbTL: TSpeedButton;
    SbTM: TSpeedButton;
    SbTR: TSpeedButton;
    SbMM: TSpeedButton;
    SbBL: TSpeedButton;
    SbBR: TSpeedButton;
    SbMR: TSpeedButton;
    SbBM: TSpeedButton;
    SbML: TSpeedButton;
    Bevel1: TBevel;
    Bevel2: TBevel;
    Bevel3: TBevel;
    Bevel4: TBevel;
    MainMenu1: TMainMenu;
    FileItem: TMenuItem;
    HelpItem: TMenuItem;
    ExitItem: TMenuItem;
    AboutItem: TMenuItem;
    SkillItem: TMenuItem;
    UnconItem: TMenuItem;
    AwakeItem: TMenuItem;
    NewGameItem: TMenuItem;
    N1: TMenuItem;
    StatusBar: TStatusBar;
    procedure FormCreate(Sender: TObject);
    procedure ExitItemClick(Sender: TObject);
    procedure SkillItemClick(Sender: TObject);
```

لیست ۷-۱۰۳ ادامه

```
    procedure AboutItemClick(Sender: TObject);
    procedure SBClick(Sender: TObject);
    procedure NewGameItemClick(Sender: TObject);
  private
    FXImage: TBitmap;
    FOImage: TBitmap;
    FCurrentSkill: Integer;
    FGameID: Integer;
    FGameServer: IGameServer;
    FRec: TRecord;
    procedure TagToCoord(ATag: Integer; var Coords: TPoint);
    function CoordToCtl(const Coords: TPoint): TSpeedButton;
    procedure DoGameResult(GameRez: GameResults);
  end;

var
  FrmMain: TFrmMain;

implementation

uses UiAbout;

{$R *.DFM}

{$R xo.res}

const
  RecStr = 'Wins: %d, Loses: %d, Ties: %d';

procedure TFrmMain.FormCreate(Sender: TObject);
begin
  // load "X" and "O" images from resource into TBitmaps
  FXImage := TBitmap.Create;
  FXImage.LoadFromResourceName(MainInstance, 'x_img');
  FOImage := TBitmap.Create;
  FOImage.LoadFromResourceName(MainInstance, 'o_img');
  // set default skill
  FCurrentSkill := slAwake;
  // init record UI
  with FRec do
   StatusBar.SimpleText := Format(RecStr, [Wins, Loses, Ties]);
  // Get server instance
  FGameServer := CoGameServer.Create;
  // Start a new game
  FGameServer.NewGame(FGameID);
end;
```

لیست ۷-۱۳ ادامه

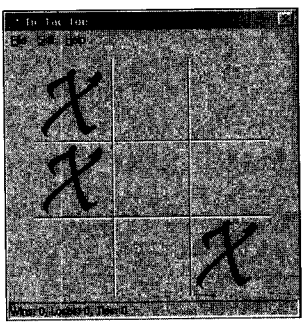
```
procedure TFrmMain.ExitItemClick(Sender: TObject);
begin
  Close;
end;

procedure TFrmMain.SkillItemClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    Checked := True;
    FCurrentSkill := Tag;
  end;
end;

procedure TFrmMain.AboutItemClick(Sender: TObject);
begin
  // Show About box
  with TFrmAbout.Create(Application) do
    try
      ShowModal;
    finally
      Free;
    end;
end;

procedure TFrmMain.TagToCoord(ATag: Integer; var Coords: TPoint);
begin
  case ATag of
    0: Coords := Point(1, 1);
    1: Coords := Point(1, 2);
    2: Coords := Point(1, 3);
    3: Coords := Point(2, 1);
    4: Coords := Point(2, 2);
    5: Coords := Point(2, 3);
    6: Coords := Point(3, 1);
    7: Coords := Point(3, 2);
  else
    Coords := Point(3, 3);
  end;
end;

function TFrmMain.CoordToCtl(const Coords: TPoint): TSpeedButton;
begin
  Result := nil;
  with Coords do
    case X of
      1:
        case Y of
          1: Result := SbTL;
```

لیست ۷-۱۳ ادامه

```
          2: Result := SbTM;
          3: Result := SbTR;
        end;
      2:
        case Y of
          1: Result := SbML;
          2: Result := SbMM;
          3: Result := SbMR;
        end;
      3:
        case Y of
          1: Result := SbBL;
          2: Result := SbBM;
          3: Result := SbBR;
        end;
    end;
end;

procedure TFrmMain.SBClick(Sender: TObject);
var
  Coords: TPoint;
  GameRez: GameResults;
  SB: TSpeedButton;
begin
  if Sender is TSpeedButton then
  begin
    SB := TSpeedButton(Sender);
    if SB.Glyph.Empty then
    begin
      with SB do
      begin
        TagToCoord(Tag, Coords);
        FGameServer.PlayerMove(FGameID, Coords.X, Coords.Y, GameRez);
        Glyph.Assign(FXImage);
      end;
      if GameRez = grInProgress then
      begin
        FGameServer.ComputerMove(FGameID, FCurrentSkill, Coords.X, Coords.Y,
          GameRez);
        CoordToCtl(Coords).Glyph.Assign(FOImage);
      end;
      DoGameResult(GameRez);
    end;
  end;
end;

procedure TFrmMain.NewGameItemClick(Sender: TObject);
var
  I: Integer;
begin
```

لیست ۷-۱۳ ادامه

```
  FGameServer.NewGame(FGameID);
  for I := 0 to ControlCount - 1 do
    if Controls[I] is TSpeedButton then
      TSpeedButton(Controls[I]).Glyph := nil;
end;

procedure TFrmMain.DoGameResult(GameRez: GameResults);
const
  EndMsg: array[grTie..grComputerWin] of string = (
    'Tie game', 'You win', 'Computer wins');
begin
  if GameRez <> grInProgress then
  begin
    case GameRez of
      grComputerWin: Inc(FRec.Loses);
      grPlayerWin: Inc(FRec.Wins);
      grTie: Inc(FRec.Ties);
    end;
   with FRec do
    StatusBar.SimpleText := Format(RecStr, [Wins, Loses, Ties]);
   if MessageDlg(Format('%s! Play again?', [EndMsg[GameRez]]), mtConfirmation
     [mbYes, mbNo], 0) = mrYes then
     NewGameItemClick(nil);
  end;
end;

end.
```



شکل ۱۸-۱۳ اجرای برنامهٔ Tic-Tac-Toe

خلاصه

در این فصل با مفاهیم، اصول اولیه و نحوهٔ پیاده‌سازی برنامه‌های +COM آشنا شدید. +COM با در برداشتن سرویس‌هایی همچون مدیریت چرخهٔ حیات اشیاء، مدیریت تراکنشها و سرویس‌های مختلف امنیتی برای برنامه‌های کاربردی، به عنوان یکی از ابزارهای منحصر به فرد دنیای نرم‌افزار مطرح شده است. همچنین در این فصل آموختید که دلفی چگونه شما را در پیاده‌سازی این گونه برنامه‌ها یاری می‌رساند.

فصل چهاردهم

طراحی و پیاده سازی برنامه‌های CORBA

در این فصل می‌خوانید

- **ویژگیها و خصوصیات CORBA**
- **معماری CORBA**
- **زبان تعریف Interface**
- **انواع داده‌ای پیچیده**
- **دلفی، CORBA و EJB**
- **CORBA و سرویسهای وب**

CORBA سر نام عبارت Common Object Request Broker Architecture بوده و مجموعه مشخصاتی است که توسط Object Management Group (OMG) ابداع شده است و طی آن شیءهای یک برنامه با شیءهای برنامهٔ دیگر ارتباط برقرار می‌کنند، حتی اگر برنامه‌ها با زبانهای برنامه‌سازی مختلف نوشته شده باشند و در محیطهای مختلف اجرا شوند.

برنامه‌ها درخواست خود را برای اشیاء از طریق Object Management Group (OMG) ارائه می‌دهند و از این رو نیازی به دانستن ساختار برنامه‌ای که شیءها متعلق به آن هستند، ندارند. CORBA برای کار با محیطهای شیءگرا و افزایش قابلیت این گونه برنامه‌های کاربردی طراحی شده است. از طریق اینترنت می‌توانید به مثالهای متنوعی از نحوهٔ پیاده‌سازی برنامه‌های CORBA دسترسی داشته باشید. در این فصل قصد داریم طریقه طراحی و پیاده‌سازی چنین برنامه‌هایی را با کمک توابع کتابخانه‌ای و ویزاردهای موجود در Delphi 7 تشریح نمائیم.

ویژگیهای CORBA

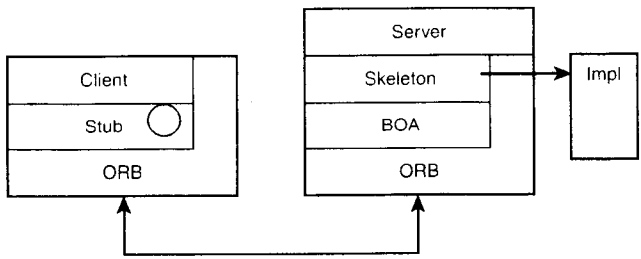
CORBA دارای چندین ویژگی است که کار با محیط‌های توزیع یافته و شیءگرا را آسان‌تر می‌نماید.

- **شیءگرا بودن:** CORBA مبتنی بر برنامه‌نویسی شیءگرا است. هر سرویس‌دهندهٔ CORBA شامل یک Interface می‌باشد که عملیات پشتیبانی از متدها و داده‌های برنامه را انجام می‌دهد.
- **موقعیت مکانی مستقل:** قدرت واقعی CORBA در نحوهٔ استقرار اشیاء است. هنگامی که یک سرویس‌گیرنده CORBA متقاضی یک شیء از طرف سرویس‌دهنده می‌باشد، تنها چیزی که اهمیت ندارد نوع و محل استقرار سرویس‌دهنده (و داده‌های موجود در آن) است. CORBA در حقیقت با ایجاد یک برنامهٔ سرویس‌دهنده، به درخواستهای ارسالی پاسخ می‌دهد.
- **مستقل از زبانهای برنامه‌سازی:** ساختار CORBA مستقل از زبانهای برنامه‌سازی است. بدین معنی که حتی اگر با برنامه‌های مختلف پیاده‌سازی شده باشند باز هم امکان برقراری ارتباط آنها از طریق آن میسر است.
- **سیستم عامل / بستر سخت‌افزاری چندگانه:** CORBA بر روی سیستم عامل‌های مختلف با بسترهای سخت‌افزاری مختلف پیاده‌سازی و نصب می‌گردد. این به معنی استقلال CORBA از بسترهای سخت‌افزاری، نرم‌افزاری است.

معماری CORBA

معماری و ساختار داخلی CORBA در شکل ۱-۱۴ نمایش داده شده است. ORB قسمتی است که در هر دو طرف سرویس‌دهنده و سرویس‌گیرنده وجود دارد. برنامه‌ها درخواستهای خود را از طریق ORB ارسال می‌کنند لذا نیازی به دانستن ساختار برنامه‌ای که اشیاء متعلق به آن هستند ندارند. این کار توسط *Internet Inter-ORB Protocol (IIOP)* که لایه‌ای از پروتکل TCP/IP است انجام می‌گیرد.

تشخیص مد بسترهای سخت‌افزاری[1] و هماهنگی بین آنها با ORB است. در طرف سرویس‌گیرنده قسمتی تحت عنوان Stub وجود دارد. Stub فایلی است که به وسیلهٔ ابزارهای موجود در دلفی (تنها



شکل ۱-۱۴ معماری CORBA

۱- Big-Endian یا Little-Endian

نسخه Enterpise دلفی) و به صورت خودکار تولید می‌شود. این فایل حاوی اطلاعاتی است که با معرفی Interface های لازم و تبدیل کدهای دلفی، امکان برقراری ارتباط با ORB را فراهم می‌آورد. کامپایلر IDL2Pas در دلفی وظیفهٔ انجام این کار را به عهده دارد. (مستندات مربوط به این کمپایلر در فهرست Delphi7\Doc\Corba از CD نصب دلفی ۷ وجود دارد). فایل همچنین دربرگیرنده چندین کلاس است که از آنها جهت شبیه‌سازی سرویس‌دهنده استفاده می‌شود.[۱]

Interface موجود در ORB در طرف سرویس‌دهنده تحت عنوان Basic Object Adapter (BOA) شناسایی می‌شود. وظیفه این Interface هدایت و مسیریابی پیام‌ها برای انتقال به بخش Skeleton می‌باشد. دلفی با استفاده از ابزار دیگری تحت عنوان Portable Object Adapter (POA) به Interface های موجود در طرف سرویس‌دهنده، انعطاف‌پذیری بیشتری ارائه کرده است. Skeleton نام کلاسی است که به وسیله کامپایلر IDL2pas ساخته می‌شود و کلاس Skeleton حاوی چندین کلاس دیگر است که وظیفه توزیع Interface های CORBA در طرف سرویس‌دهنده را به عهده دارند. هنگامی که به پیاده‌سازی یک برنامهٔ CORBA در دلفی می‌پردازید، متوجه می‌شوید که هیچ Interface ای در کلاس Skeleton وجود ندارد (برای طرف سرویس‌دهنده). این جزئیات در فایل‌هایی با عنوان IMPL پیاده‌سازی شده‌اند.

OSAgent

همانطور که گفتیم CORBA مجموعهٔ مشخصاتی است که اجازه می‌دهد شیءهای یک برنامه با شیءهای برنامهٔ دیگر و بدون توجه به ساختار برنامه‌ها با هم ارتباط برقرار سازند. برنامه‌ای که اجازهٔ ارتباط بین سرویس‌گیرنده و سرویس‌دهنده را ارائه می‌دهد به OSAgent معروف است. OSAgent ابزاری است که تنها به وسیله ORB بورلند ارائه شده است. OSAgent برنامه‌ای است که قبل از ایجاد یک برنامهٔ CORBA به اجرا در می‌آید و به مثابه یک پل جهت ارتباط بین سرویس‌گیرنده و سرویس‌دهنده عمل می‌نماید.

Interface ها

هر یک از اشیاء CORBA یک Interface مربوط به خود دارد که به وسیلهٔ آن شناسائی و تعریف می‌شود.

برنامهٔ سرویس‌دهنده، توزیع متدها، Interface ها و داده‌ها را با ارسال درخواست از طرف سرویس‌گیرنده انجام می‌دهد. Interface ها تنها با یک بار توزیع در سیستم مقیم می‌شوند. OMG برای

۱- اصطلاحی که در ارتباط با پردازنده‌ها و سایر تراشه‌هایی که می‌توانند برای کار در بین یکی از دو مد Big-Endian یا Little-Endian رفت و برگشت کنند. تراشه Power PC دارای این ویژگی است و می‌تواند مد Little-Endian وبندوز NT و مد Big-Endian سیستم عامل Macos/ppc را اجرا نماید.

توصیف و تعریف این Interface ها اقدام به ارائه زبانی تحت عنوان زبان تعریف Interface (IDL) نموده است. کد IDL چیزی مشابه با کدهای C و java است.

زبان تعریف Interface

همانطور که قبلاً اشاره شد مستندات مربوط به CORBA در فهرست **Delphi6\Doc\Corba** از CD نصب دلفی ۶ وجود دارد. در این مستندات شرح توصیفی تمامی متدها، انواع داده‌ای، وراثت، Interface ها و همچنین توضیحاتی در رابطه با IDL ارائه شده است. لذا پیشنهاد می‌کنیم برای دریافت اطلاعات بیشتر در زمینه‌های فوق به فهرست مذکور مراجعه داشته باشید.

در فایلهای IDL عمل حساسیت به حروف بزرگ یا کوچک اعمال شده است. به طور مثال تعریف یک Interface با نام FOO متفاوت با تعریف یک Interface تحت عنوان foo خواهد بود.

جملات توضیحی این فایل‌ها نیز به صورت زیر می‌باشد. (همانند C++ و java) .

```
// This is a single line comment.
/* This is an example of a block comment
   that can be spread
   over several lines */
```

تمامی کلمات کلیدی IDL با حروف کوچک نوشته می‌شوند. توجه داشته باشید که تعریف یک فایل IDL تنها در فایل‌های IDL مجاز می‌باشد. این کار به وسیلهٔ اعلان Include # به همراه نام فایل IDL انجام می‌پذیرد.

انواع داده‌ها در IDL

IDL حاوی چندین نوع داده برای تعریف متغیرهاست. انواع این داده‌ها به همراه مقادیر متناظرشان در پاسکال شیءگرا در جدول ۱-۱۴ ارائه شده است.

داده‌های نوع‌دار

نوع داده‌ای Int در IDL تعریف شده است اما در صورت لزوم می‌توان از انواع داده‌ای short ، long ، unsigneed ، short و unsigne برای انجام این کار استفاده نمود.

نام‌های مستعار

نامهای مستعار عناوینی است که از آنها جهت افزایش قابلیت فهم داده‌ها استفاده می‌شود. یک نام مستعار برچسبی برای شیءهایی چون فایل‌ها یا مجموعه‌ای از داده‌ها می‌باشند. به مثال زیر توجه کنید:

```
typedef short yearType;
```

جدول ۱-۱۴ انواع داده‌ها در IDL

IDL Type	*Pascal Type*
boolean	Boolean
Char	Char
wchar	Wchar
octet	Byte
string	AnsiString
wstring	WideString
short	SmallInt
unsigned short	Word
long	Integer
unsigned long	Cardinal
long long	Int64
unsigned long long	Int64
float	Single
double	Double
long double	Extended
fixed	not implemented—no corresponding type

ساختمان‌ها

همانطور که می‌دانید تعریف ساختمان چیزی مشابه با تعریف رکورد در پاسکال است. نمونه‌ای از این تعریف در اینجا ارائه شده است.

```
struct TimeOfDay {
   short hour;
    short minute;
   short seconds;
};
```

آرایه‌ها

در IDL امکان تعریف آرایه‌های یک بعدی و یا چند بعدی وجود دارد. نمونه‌ای از تعریف آرایه‌ها در زیر ارائه شده است.

typedef Color ColorArray [4]; // single dimensional array of the Color enum
typedef string StringArray [10][20]; //10 strings of max length 20

دنباله‌ها

دنباله‌ها می‌توانند به صورت متناهی یا غیرمتناهی در IDL استفاده شوند. البته کاربرد دنباله‌ها در IDL بسیار نادر می‌باشد. در زیر دو نوع تعریف از دنباله‌ها ارائه شده است.

```
typedef sequence<Color> Colors;
typedef sequence<long, 1000> Numseq;
```

مهمترین کاربرد دنباله‌ها در برنامه‌نویسی CORBA در پردازش رکوردهای یک بانک اطلاعاتی سرویس‌دهنده/ سرویس‌گیرنده است.

آرگومانهای مربوط به متدها

هر آرگومان معرفی شده در یک متد با سه مشخصه in ، out و یا inout شناسائی می‌شود. آرگومانهایی که مقدارشان توسط سرویس‌گیرنده تنظیم و جایگزین می‌گردد in معرفی می‌شوند و نوع out نمایانگر آرگومانهایی است که مقدارشان به وسیله سرویس‌دهنده تنظیم و جایگزین می‌شود. اگر مقداردهی اولیه یک آرگومانی به وسیله سرویس‌گیرنده انجام شود و امکان تغییر آن از طرف سرویس‌دهنده هم وجود داشته باشد، آنگاه آرگومان از نوع inout تعریف می‌شود.

ماژول‌ها

کلمهٔ کلیدی Module واژه‌ای است که از آن جهت تعریف گروهی از Interface ها و یا انواع داده‌ای استفاده می‌شود. IDL2pas از نام ماژول برای تبدیل آن به یک یونیت خاص در دلفی استفاده می‌کند. ارجاع به ماژولها نیز با یک روش فراخوانی خاص انجام می‌پذیرد. به عنوان مثال ماژول Foo که دربرگیرنده Interface ای با نام Bar می‌باشد به صورت Foo::Bar فراخوانی می‌شود.

توجه داشته باشید که در IDL امکان تعریف انواع داده‌ای از نوع Protected یا Private وجود ندارد و تمامی متدها و Interface ها در IDL از نوع Public می‌باشند. یک راه برای یادگیری بهتر کدنویسی IDL مراجعه به برنامه‌ها و مستندات دیگر برنامه‌نویسان است. در صورتی که علاقمند به رؤیت این برنامه هستید، می‌توانید به CD نصب دلفی ۷ مراجعه نمائید. CD مذکور حاوی مثال‌های متنوعی از این رابطه می‌باشد. جستجو در اینترنت نیز بی‌فایده نخواهد بود.

اکنون با درک مفاهیم اولیه CORBA و آنچه که در مورد IDL ها آموخته‌اید به ارائه مثال‌های کاربردی خواهیم پرداخت. تا از این حیث شما را با قدرت CORBA در محیط شیءگرا آشنا سازیم.

برنامهٔ کاربردی Bank

بدون شک همگی شما با برنامهٔ "Hello World" در C و Java و عمومیت آن به عنوان یک مثال کاربردی آشنائی دارید. برنامهٔ Bank در CORBA نیز به همین شکل است. چنانچه کتاب‌های دیگری را

در خصوص برنامه‌نویسی CORBA مطالعه کرده‌اید، حتماً این مثال را به یاد خواهید آورد. دو متد با نام deposit() و withdraw() عملیات اساسی این برنامه را انجام می‌دهند. این متدها کنترل عملیات سپرده‌گذاری و دریافت سپردهٔ یک مشتری را به عهده دارند. یکی دیگر از اعمال اصلی این برنامه کنترل موجودی افراد در هنگامی است که فرد تقاضای مبلغی بیش از مبلغ سپرده‌اش را دارد. این کار به عنوان یک استثناء (exception) در سیستم مطرح می‌شود.

فایل این مثال در زیر ارائه شده است.

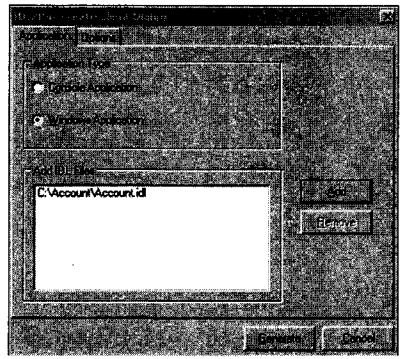
```
module Bank {
   exception WithdrawError {
      float current_balance;
   };

   interface Account {
      void deposit(in float amount);
      void withdraw(in float amount) raises (WithdrawError);
      float balance();
   };
};
```

با بروز یک استثناء در برنامهٔ سرویس‌گیرنده، یک پیغام خطا به همراه مبلغ تراز حساب برای یک مشتری خاص داده می‌شود. متدهای deposit و withdraw روالی مشابه با هم دارند. هر دوی آنها از نوع void تعریف شده‌اند. عملیات سپرده‌گذاری به وسیلهٔ متد deposit (با اضافه کردن مبلغ سپرده به حساب) و عملیات برداشت سپرده توسط متد withdraw (با کم کردن مبلغ سپرده به حساب) انجام می‌گیرد.

آرگومانهای متدهای deposit و withdraw به صورت پارامتر تعریف شده‌اند که همانطور که ملاحظه می‌کنید این امر تنها به خاطر ارسال آنها از برنامهٔ سرویس‌گیرنده به برنامهٔ سرویس‌دهنده می‌باشد.

در IDE دلفی ۷ چندین ویزارد برای طراحی برنامه‌های سرویس‌دهنده/ سرویس‌گیرنده CORBA وجود دارد. اکنون قصد داریم تا برنامهٔ سرویس‌دهندهٔ خود را برای مثال ذکر شده پیاده‌سازی نمائیم. برای فراخوانی ویزارد مرتبط با CORBA به منوی File بروید گزینهٔ New و سپس Other را انتخاب کنید و بعد از آن تب CORBA را از جعبهٔ مکالمه نمایش داده شده برگزینید. دکمهٔ ماوس را دو بار بر روی نشانهٔ CORBA server بفشارید تا جعبهٔ مکالمه‌ای همانند شکل ۲-۱۴ به نمایش در آید.

همانطور که در شکل می‌بینید، قسمتی از این پنجره در برگیرنده فایل‌های IDL برنامه است. برای درج یک یا چند فایل IDL جدید دکمهٔ Add را بفشارید و از روی کادر نمایش داده شده مسیر و فایل‌های موردنظر را انتخاب کنید. اکنون فایل Bank.idl را از فهرست موجود فراخوانی کرده و آن را به لیست فایل‌های IDL بیافزایید. در اینجا لازم است تا دکمهٔ Generate را بفشارید. همانطور که قبلاً گفتیم پردازش فایل‌های IDL توسط کامپایلر IDL2Pas انجام می‌پذیرد. ویزارد ارائه شده نیز پس از تولید فایل‌ها توسط IDL2Pas ، عملیات ایجاد برنامه‌های سرویس‌دهنده/ سرویس‌گیرنده را انجام



شکل ۲-۱۴ ویزارد مربوط
به طراحی برنامه‌های CORBA

می‌دهد. در این حالت برای برنامهٔ سرویس‌دهندهٔ Bank ، ۴ فایل تولید می‌شود.

- Bank_I.pas : تعریف داده‌ها و Interface ها در این فایل قرار می‌گیرد.
- Bank_C.pas : این فایل شامل انواع داده‌ای از پیش تعیین شده، استثنائات برنامه و کلاس‌های طرف سرویس‌دهنده می‌باشد.
- Bank_S.pas : تعریف کلاس‌های Skeleton سرویس‌دهنده در این فایل قرار دارد.
- Bank_Impl.pas : این فایل شامل کلاس‌های عمومی برنامه برای پیاده‌سازی برنامهٔ سرویس‌دهنده می‌باشد.

نحوهٔ تعریف Interface مرتبط به این برنامه در زیر آورده شده است. برای این برنامه تنها یک Interface با نام Account تعریف شده است که از دو متد به شرح قبل استفاده می‌کند.

```
unit Bank_i;
 interface

uses
  CORBA;

type
  Account = interface;

  Account = interface
    ['{99FCA96D-77B2-4A99-7677-E1E0C32F8C67}']
    procedure deposit (const amount : Single);
    procedure withdraw (const amount : Single);
    function  balance : Single;
  end;

implementation

initialization

end.
```

کد زیر محتویات مربوط به فایل Bank_C.pas می‌باشد. همانطور که خواهید دید در این فایل یک استثناء به نام Overdrawn تعریف شده است. این در پیاده‌سازی این استثناء از کلاس UserException استفاده گردیده است.

```
unit Bank_c;

interface

uses
  CORBA, Bank_i;

type
  EWithdrawError = class;
  TAccountHelper = class;
  TAccountStub = class;

  EWithdrawError = class(UserException)
  private
    Fcurrent_balance : Single;
protected
  function  _get_current_balance : Single; virtual;
public
  property  current_balance : Single read _get_current_balance;
  constructor Create; overload;
  constructor Create(const current_balance : Single); overload;
  procedure Copy(const _Input : InputStream); override;
  procedure WriteExceptionInfo(var _Output : OutputStream); override;
end;

TAccountHelper = class
  class procedure Insert (var _A: CORBA.Any; const _Value : Bank_i.Account);
  class function  Extract(var _A: CORBA.Any) : Bank_i.Account;
  class function  TypeCode     : CORBA.TypeCode;
  class function  RepositoryId : string;
  class function  Read (const _Input  : CORBA.InputStream) : Bank_i.Account;
  class procedure Write(const _Output : CORBA.OutputStream;
➥       const _Value : Bank_i.Account);
  class function  Narrow(const _Obj   : CORBA.CORBAObject; _
➥       IsA : Boolean = False) : Bank_i.Account;
  class function  Bind(const _InstanceName : string = ''; _
➥       HostName : string = '') : Bank_i.Account; overload;
  class function  Bind(_Options : BindOptions;
➥       const _InstanceName : string = ''; _HostName: string = '') :
➥       Bank_i.Account; overload;
end;

TAccountStub = class(CORBA.TCORBAObject, Bank_i.Account)
public
  procedure deposit ( const amount : Single); virtual;
  procedure withdraw ( const amount : Single); virtual;
```

```
    function  balance : Single; virtual;
  end;

implementation

var

  WithdrawErrorDesc : PExceptionDescription;

function  EWithdrawError._get_current_balance : Single;
begin
  Result := Fcurrent_balance;
end;
constructor EWithdrawError.Create;
begin
  inherited Create;
end;

constructor EWithdrawError.Create(const current_balance : Single);
begin
  inherited Create;
  Fcurrent_balance := current_balance;
end;

procedure EWithdrawError.Copy(const _Input: InputStream);
begin
  _Input.ReadFloat(Fcurrent_balance);
end;

procedure EWithdrawError.WriteExceptionInfo(var _Output : OutputStream);
begin
  _Output.WriteString('IDL:Bank/WithdrawError:1.0');
  _Output.WriteFloat(Fcurrent_balance);
end;

function  WithdrawError_Factory: PExceptionProxy; cdecl;
begin
  with Bank_c.EWithdrawError.Create() do Result := Proxy;
end;

class procedure TAccountHelper.Insert(var _A : CORBA.Any;
➥    const _Value : Bank_i.Account);
begin
  _A := Orb.MakeObjectRef( TAccountHelper.TypeCode, _
➥        Value as CORBA.CORBAObject);
end;

class function TAccountHelper.Extract(var _A : CORBA.Any): Bank_i.Account;
var
  _obj : Corba.CorbaObject;
begin
```

```
  _obj := Orb.GetObjectRef(_A);
  Result := TAccountHelper.Narrow(_obj, True);
end;

class function TAccountHelper.TypeCode : CORBA.TypeCode;
begin
  Result := ORB.CreateInterfaceTC(RepositoryId, 'Account');
end;

class function TAccountHelper.RepositoryId : string;
begin
  Result := 'IDL:Bank/Account:1.0';
end;

class function TAccountHelper.Read(const _Input : CORBA.InputStream)
➥     : Bank_i.Account;
var
  _Obj : CORBA.CORBAObject;
begin
  _Input.ReadObject(_Obj);
  Result := Narrow(_Obj, True)
end;

class procedure TAccountHelper.Write(const _Output : CORBA.OutputStream;
➥     const _Value : Bank_i.Account);
begin
  _Output.WriteObject(_Value as CORBA.CORBAObject);
end;

class function TAccountHelper.Narrow(const _Obj : CORBA.CORBAObject; _
➥     IsA : Boolean) : Bank_i.Account;
begin
  Result := nil;
  if (_Obj = nil) or (_Obj.QueryInterface(Bank_i.Account, Result) = 0) then
    exit;
  if _IsA and _Obj._IsA(RepositoryId) then
    Result := TAccountStub.Create(_Obj);
end;

class function TAccountHelper.Bind(const _InstanceName : string = ''; _
➥     HostName: string = '') : Bank_i.Account;
begin
  Result := Narrow(ORB.bind(RepositoryId, _InstanceName, _HostName), True);
end;

class function TAccountHelper.Bind(_Options : BindOptions;
➥    const_InstanceName : string = ''; HostName : string = '') :
➥     Bank_i.Account;
```

```
begin
  Result := Narrow(ORB.bind(RepositoryId, _Options, _InstanceName, _
➥      HostName), True);
end;

procedure TAccountStub.deposit ( const amount : Single);
var
  _Output: CORBA.OutputStream;
  _Input : CORBA.InputStream;
begin
  inherited _CreateRequest('deposit',True, _Output);
  _Output.WriteFloat(amount);
  inherited _Invoke(_Output, _Input);
end;

procedure TAccountStub.withdraw ( const amount : Single);
var
  _Output: CORBA.OutputStream;
  _Input : CORBA.InputStream;
begin
  inherited _CreateRequest('withdraw',True, _Output);
  _Output.WriteFloat(amount);
  inherited _Invoke(_Output, _Input);
end;

function  TAccountStub.balance : Single;
var
  _Output: CORBA.OutputStream;
  _Input : CORBA.InputStream;
begin
  inherited _CreateRequest('balance',True, _Output);
  inherited _Invoke(_Output, _Input);
  _Input.ReadFloat(Result);
end;

initialization

Bank_c.WithdrawErrorDesc := RegisterUserException('WithdrawError',
➥    'IDL:Bank/WithdrawError:1.0', @Bank_c.WithdrawError_Factory);

finalization

UnRegisterUserException(Bank_c.WithdrawErrorDesc);

end.
```

اکنون نوبت پیاده‌سازی کلاس Account است. این کلاس حاوی متدهایی است که در فایل Bank_idl تعریف شده‌اند. پیاده‌سازی این کلاس در زیر ارائه شده است.

```
unit Bank_impl;

interface

uses
  SysUtils, CORBA, Bank_i, Bank_c;

type
  TAccount = class;

 unit Bank_impl;

interface

uses
  SysUtils, CORBA, Bank_i, Bank_c;

type

TAccount = class(TInterfacedObject, Bank_i.Account)
  protected
    _balance : Single;
  public
    constructor Create;
    procedure deposit (const amount : Single);
    procedure withdraw (const amount : Single);
    function  balance : Single;
  end;


implementation

constructor TAccount.Create;
begin
  inherited;
  _balance := random(10000);
end;
procedure TAccount.deposit(const amount : Single);
begin
  if amount > 0 then
    _balance := _balance + amount;
end;

procedure TAccount.withdraw(const amount : Single);
begin
```

```
  if amount < _balance then
    _balance := _balance - amount
  else
    raise EWithdrawError.Create(_balance);
end;

function  TAccount.balance : Single;
begin
  result := _balance;
end;

initialization
   randomize;

end.
```

کار دیگری که متد deposit می‌بایست انجام دهد، جلوگیری از ارسال مقادیر منفی توسط کاربر است. همانطور که گفتیم چنانچه متد withdraw مقداری کمتر از مقدار تراز حساب را برگردانده بایستی استثناء مربوطه فعال شده و کنترل برنامه را در دست بگیرد.

کد زیر تعریف کلاس stub را برای این برنامه نشان می‌دهد. این کلاس به عنوان یک شیء میانجی مابین سرویس‌دهنده و سرویس‌گیرنده قرار گرفته و از داده‌ها و شیء‌های برنامه نیز محافظت می‌کند.

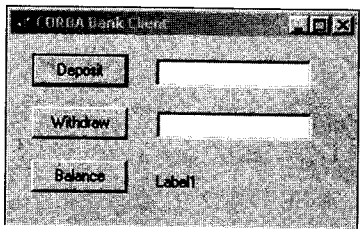
```
TAccountStub = class(CORBA.TCORBAObject, Bank_i.Account)  public

public

public
  procedure deposit ( const amount : Single); virtual;
  procedure withdraw ( const amount : Single); virtual;
  function  balance : Single; virtual;
end;
```

کد زیر نحوهٔ پیاده‌سازی متد deposit را نمایش می‌دهد، همانطور که می‌بینید در این متد دو بافر CORBA به عنوان متغیرهای اصلی تعریف شده‌اند.

```
procedure TAccountStub.deposit(const amount : Single);
var
  _Output: CORBA.OutputStream;
  _Input : CORBA.InputStream;
begin
  inherited _CreateRequest('deposit', True, Output);
  _Output.WriteFloat(amount);
  inherited _Invoke(Output, Input);
end;
```

پیاده‌سازی متد Invoke گام بعدی برای این برنامه است. از این متد جهت ارسال درخواستها به سوی سرویس‌دهنده استفاده می‌شود. از آنجا که تمام کدهای stub به وسیلهٔ IDL2Pas و به صورت خودکار تولید می‌شوند، لذا توصیه می‌شود به هیچ عنوان این گونه فایل‌ها را ویرایش نکرده و تغییری در آنها ایجاد ننمائید.

آخرین قدم طراحی GUI برای سرویس‌گیرنده است. نمونه فرم مربوط به این کار در شکل ۳-۱۴ ارائه شده است.



شکل ۳-۱۴ برنامهٔ سرویس‌گیرنده مبتنی بر CORBA

کد مربوط به پیاده‌سازی این GUI در زیر آورده شده است.

```
unit ClientMain;

interface

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

type
  TForm1 = class(TForm)
    btnDeposit: TButton;
    btnWithdraw: TButton;
    btnBalance: TButton;
    Edit1: TEdit;
    Edit2: TEdit;
    Label1: TLabel;
    procedure btnDepositClick(Sender: TObject);
    procedure btnWithdrawClick(Sender: TObject);
    procedure btnBalanceClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
  { private declarations }
  protected
    Acct : Account;
    procedure InitCorba;
  { protected declarations }
```

```
  public
  { public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.InitCorba;
begin
  CorbaInitialize;
  // Bind to the Corba server
  Acct := TAccountHelper.bind;
end;

procedure TForm1.btnDepositClick(Sender: TObject);
begin
  Acct.deposit(StrToFloat(Edit1.text));
end;

procedure TForm1.btnWithdrawClick(Sender: TObject);
begin
  try
    Acct.withdraw(StrToFloat(Edit2.Text));
  except
    on e: EWithdrawError do
      ShowMessage('Withdraw Error. The balance = ' +
      FormatFloat('$##,##0.00', E.current_balance));
  end;
end;

procedure TForm1.btnBalanceClick(Sender: TObject);
begin
  label1.caption := FormatFloat('Balance = $##,##0.00', acct.balance);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
 InitCorba;
end;

end.
```

بعد از کامپایل نمودن برنامه‌های سرویس‌دهنده و سرویس‌گیرندهٔ این مثال می‌بایست OSAgent را اجرا نمود. برای اجرای OSAgent در سیستم عاملهای ویندوز 9X کافی است در منوی RUN ،

OSAgent را تایپ نموده و دکمهٔ OK را بفشارید. پس از این کار نشانهٔ OSAgent در میلهٔ ابزار ویندوز ظاهر خواهد شد.

اما در ویندوز NT ، OSAgent به عنوان یک سرویس موجود است که به راحتی قابل نصب می‌باشد. بعد از اجرای OSAgent باید برنامهٔ سرویس‌دهنده را اجرا نمود.

انواع داده‌ای پیچیده

در این قسمت می‌خواهیم شما را با نحوهٔ به کارگیری داده‌های پیچیده در CORBA آشنا سازیم. IDL زیر نمونه‌ای از به کارگیری انواع داده‌ای پیشرفته (ADT) را نمایش می‌دهد.

```
// ADT IDL file
//
// Demonstrates various data structures in IDL
//

// use an alias for string types

typedef string Identifier;

enum EnumType
{
  first,
  second,
  third
};

struct StructType
{
  short s;
  long l;
  Identifier i;
};

const unsigned long ArraySize = 3;

typedef StructType StructArray[ArraySize];

typedef sequence<StructType> StructSequence;


interface ADT
{

  void Test1(in Identifier st, in EnumType myEnum, inout StructType myStruct);

  void Test2(out StructType myStruct, in StructArray myStructArray,
➥       out StructSequence myStructSeq);

};
```

همانطور که ملاحظه می‌کنید، تمام رشته‌ها در این IDL از نوع identifier تعریف گردیده‌اند. در EnumType که یک ساختار عددی می‌باشد، سه متغیر first، second و third تعریف شده‌اند. تعریف ساختمان StructTypes نیز مشابه با تعریف رکوردها در پاسکال بوده و شامل دو متغیر عددی و یک متغیر از نوع رشته‌ای است. در ادامهٔ این کد structArray (با حداکثر سه عضو) از نوع structType و یک دنباله از همین نوع تعریف شده است. همانطور که قبلاً نیز گفته شده، از دنباله‌ها به عنوان آرایه‌های پویا استفاده می‌گردد. بعد از آن یک Interface تحت عنوان ADT تعریف شده است که شامل دو متد به نامهای Test1، Test2 می‌باشد. همانطور که می‌بینید آرگومانهای این متدها از پارامترهای in، out و inout می‌باشند که تعریف آنها را قبلاً ارائه داده‌ایم. پس از تعریف فایل ADT.idl که در بالا مشاهده کردید. فایل ADT_I.pas را ارائه می‌کنیم. همانطور که قبلاً متذکر شدیم تعریف Interface در این فایل صورت می‌گیرد.

```
unit adt_i;

interface

uses
  CORBA;

type

  EnumType = (first, second, third);

const
  { (Do not edit the values assigned to these constants.) }

  ArraySize : Cardinal = 3;

type
  StructType = interface;
  ADT = interface;

  Identifier = AnsiString;

  StructArray = array[0..2] of adt_i.StructType;

  StructSequence = array of adt_i.StructType;

  StructType = interface
    ['{B4A1845D-4DB0-9B2E-A2E3-001F2D6B8C81}']
    function  _get_s : SmallInt;
    procedure _set_s (const s : SmallInt);
    function  _get_l : Integer;
    procedure _set_l (const l : Integer);
    function  _get_i : adt_i.Identifier;
    procedure _set_i (const i : adt_i.Identifier);
    property  s : SmallInt read _get_s write _set_s;
```

```
  property  l : Integer read _get_l write _set_l;
  property  i : adt_i.Identifier read _get_i write _set_i;
end;

ADT = interface
  ['{203B9E07-735F-2980-CB02-353A7C6A5B68}']
  procedure Test1 (const st : adt_i.Identifier;
                   const myEnum : adt_i.EnumType;
                   var   myStruct : adt_i.StructType);
  procedure Test2 (out   myStruct : adt_i.StructType;
                   const myStructArray : adt_i.StructArray;
                   out   myStructSeq : adt_i.StructSequence);
end;

implementation

initialization

end.
```

کد زیر پیاده‌سازی این ADT را در برنامهٔ سرویس‌دهنده نشان می‌دهد.

```
unit adt_impl;

interface

uses
  SysUtils, CORBA, adt_i, adt_c;

type
  TADT = class;

  TADT = class(TInterfacedObject, adt_i.ADT)
  public
    constructor Create;
    procedure Test1 ( const st : adt_i.Identifier;
                      const myEnum : adt_i.EnumType;
                      var   myStruct : adt_i.StructType);
    procedure Test2 ( out   myStruct : adt_i.StructType;
                      const myStructArray : adt_i.StructArray;
                      out   myStructSeq : adt_i.StructSequence);
  end;

implementation

uses ServerMain;

constructor TADT.Create;
```

```
begin
  inherited;
end;

procedure TADT.Test1 ( const st : adt_i.Identifier;
                       const myEnum : adt_i.EnumType;
                       var   myStruct : adt_i.StructType);
begin
  Form1.Memo1.Lines.Add('String from Client : ' + st);

  case myEnum of
    first : Form1.Memo1.Lines.Add('Enum value is "first"');
    second: Form1.Memo1.Lines.Add('Enum value is "second"');
    third:  Form1.Memo1.Lines.Add('Enum value is "third"');
  end;

  Form1.Memo1.Lines.Add(Format('myStruct.s = %d', [myStruct.s]));
  Form1.Memo1.Lines.Add(Format('myStruct.l = %d', [myStruct.l]));
  Form1.Memo1.Lines.Add(Format('myStruct.i = %s', [myStruct.i]));

  myStruct.s := 10;
  myStruct.l := 1000;
  myStruct.i := 'This is the return string from the Server';
end;

procedure TADT.Test2 ( out   myStruct : adt_i.StructType;
                       const myStructArray : adt_i.StructArray;
                       out   myStructSeq : adt_i.StructSequence);
var
  k : integer;
  tempSeq : StructSequence;
begin
  myStruct := TStructType.Create(20, 2000,
➥      'Hello from the server structType Test 2');

  for k := 0 to ArraySize - 1 do
    With Form1.Memo1.Lines do
    begin
      Add(Format('myStructArray[%d].s = %d', [k, myStructArray[k].s]));
      Add(Format('myStructArray[%d].l = %d', [k, myStructArray[k].l]));
      Add(Format('myStructArray[%d].i = %s', [k, myStructArray[k].i]));
    end;

  SetLength(tempSeq, 2);
```

```
  for k := 0 to 1 do
    tempSeq[k] := TStructType.Create(k + 100, k + 1000, Format('k = %d', [k]));
  myStructSeq := tempSeq;
end;


initialization

end.
```

اکنون می‌بایست GUI لازم برای این کار را پیاده‌سازی نمائیم. فرم اصلی این GUI دربرگیرندهٔ یک memo control و دو دکمه برای فراخوانی متدهای test می‌باشد. نحوهٔ پیاده‌سازی GUI برای سرویس‌گیرنده در زیر ارائه شده است.

```
unit ClientMain;

interface

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

type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Memo1: TMemo;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
  { private declarations }
  protected
    myADT : ADT;
    procedure InitCorba;
  { protected declarations }
  public
  { public declarations }
  end;

var
  Form1: TForm1;
implementation

{$R *.DFM}

procedure TForm1.InitCorba;
begin
  CorbaInitialize;
```

```
  myADT := TADTHelper.bind;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  initCorba;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  temp : StructType;
begin
  temp := TStructType.Create(50, 500, 'This is the client struct in Test 1');
  myADT.Test1('Hello from the Test1 Client', first, temp);

  with Memo1.Lines do
  begin
    Add('Response from server inout struc var:');
    Add(Format('myStruct.s = %d', [temp.s]));
    Add(Format('myStruct.l = %d', [temp.l]));
    Add(Format('myStruct.i = %s', [temp.i]));
  end;
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  I: Integer;
  temp : StructType;
  tempSeq : StructSequence;
  tempArray : StructArray;
begin
   temp := TStructType.Create(0,0,'test');
   SetLength(tempSeq, 2);

   for I := 0 to ArraySize -1 do
     tempArray[I] := TStructType.Create(200 + I, 2000 + I,
➥          Format('Stuct %d in Array', [I]) );
  myADT.Test2(temp, tempArray, tempSeq);

  with Memo1.Lines do
  begin
    Add(Format('struct.s = %d', [temp.s]) );
    Add(Format('struct.l = %d', [temp.l]) );
    Add(Format('struct.i = %s', [temp.i]) );
  end;
```

```
  for I := 0 to 1 do
  with Memo1.Lines do
  begin
    Add( Format('tempSeq[%d].s = %d', [I, tempSeq[I].s]) );
    Add( Format('tempSeq[%d].l = %d', [I, tempSeq[I].l]) );
    Add( Format('tempSeq[%d].i = %s', [I, tempSeq[I].i]) );
  end;
end;

end.
```

برای اجرای این مثال، ابتدا کدهای موجود را کامپایل کرده و سپس OSAgent را فعال نمائید. با فشردن دکمه موجود بر روی فرم، ساختمان داده‌ای روی سرویس‌دهنده و سرویس‌گیرنده با هم تعویض خواهند شد. داده‌های دریافت شده در هر طرف به وسیلهٔ memo control به نمایش در می‌آید.

دلفی، CORBA و Enterprise Java Beans (EJBs)

Enterprise Java Beans یک API است که برای گسترش مدل Javabean به چند محیط مختلف طراحی شده است. این بدان معنی است که برنامه‌های کاربردی سرویس‌دهنده بتوانند در محیطهای مختلفی اجرا شوند. هدف از این کار استفاده از تکنولوژی جاوا در راستای ایجاد روتین‌های سرویس‌دهندهٔ قابل استفاده مجدد در برنامه‌های اداری و تجاری است.

در اینجا قصد داریم نحوهٔ برقراری ارتباط بین برنامه‌های CORBA در دلفی و Enterprise Java Beans را بررسی کنیم. برای انجام این کار به نرم‌افزارهای Borland Jbuilder5 و Application Server 4.51 نیاز خواهید داشت. (این نرم‌افزارها در سایت **www.borland.com** قابل دسترسی می‌باشند).

EJB به عنوان یک جزء سازنده

دلفی با EJB به عنوان یک جزء سازنده رفتار می‌کند. بدین معنی که EJB برای دلفی مشابه جزء سازنده‌ای است که از آن جهت اعمالی نظیر برقراری ارتباط با یک بانک اطلاعاتی، پردازش اطلاعات و پاسخ‌دهی رویدادها استفاده می‌شود.

Interface های Home و Remote

EJBs نیز همانند سایر API ها شامل دو Interface به شرح زیر است:

Home : این Interface در برگیرندهٔ متدی است که برنامه‌ها از آن برای فراخوانی و به دست آوردن یک نمونه از EJB استفاده می‌کنند.

Remote : این Interface در برگیرندهٔ تمام متدهایی است که در برنامه‌هایی از این قبیل کاربرد دارند.

روش کار و پیاده‌سازی این Interface ها مشابه با فایل‌های IDL است.

انواع EJBs

EJBs ها به دو گروه به شرح زیر تقسیم‌بندی می‌شوند:

Session beans : Session beans به EJB هایی گفته می‌شود که از نوع Stateless می‌باشند. این بدان معنی است که در بین فراخوانی برنامه‌ها، هیچ اطلاعاتی که مربوط به این فراخوانی باشد، ذخیره نخواهد شد.

Entity beans : Entity beans به EJB هایی گفته می‌شود که از نوع Stateful می‌باشند. Stateful به معنی ذخیره‌سازی اطلاعات بین فراخوانی برنامه‌هاست.

مفهوم Stateless و Stateful را در فصل ۱۳ نیز ارائه داده‌ایم. تفاوت دیگری که بین Session beans و Entity beans وجود دارد مربوط به اشتراک‌پذیری آنهاست. توجه داشته باشید که برای هر درخواست ارسال شده از سوی سرویس‌گیرنده یک Session beans مجزا ایجاد می‌شود و این در حالی است که Entity beans ایجاد شده برای یک سرویس‌گیرنده واحد بوده و بین همهٔ سرویس‌گیرنده‌ها به اشتراک گذاشته می‌شود.

پیکربندی JBuilder 5 برای ساخت برنامه‌های EJB

معمولاً برای ساخت برنامه‌های EJB از نرم‌افزار JBuilder استفاده می‌شود. همانطور که گفتیم فایل‌های مربوط به نصب این نرم‌افزار در سایت **www.Borland.com** موجود می‌باشد. توصیه می‌کنیم قبل از کار با JBuilder 5، مسیری را برای ذخیره‌سازی پروژه‌های JBuilder خود مشخص سازید. (به عنوان مثال **c:\Myprojects**) طراحی EJB در JBuilder 5 توسط ویزاردهای موجود در آن انجام می‌شود. اما قبل از استفاده از آنها می‌بایست محیط JBuilder 5 به شرح ذیل پیکربندی نمود.

۱- JBuilder 5 را اجرا کنید. سپس با رفتن به منوی Tools، گزینهٔ Enterprise setup را انتخاب نمائید. با این کار یک جعبهٔ مکالمه برای پیکربندی CORBA نمایش داده می‌شود.

۲- بر روی تب CORBA، گزینهٔ Visibroker را انتخاب کنید.

۳- دکمهٔ Edit را بفشارید و مسیر مربوط به ORB را در جعبهٔ مکالمه نمایش داده شده وارد کنید. منظور از این مسیر، محل قرارگیری IDLJava است که معمولاً در فهرست **c:\Borland\AppServer\bin** وجود دارد.

۴- تب Application Server را انتخاب کنید. سپس گزینهٔ BAS 4.5 را انتخاب کرده و از اشاره آن به فهرست AppServer مطمئن شوید.

c:\Borland\AppServer\bin

۵- در قسمت Default project Properties تب Server را انتخاب کنید و دقت کنید که گزینهٔ

Borland Application Server فعال باشد. در صورت غیرفعال بودن این گزینه، دکمهٔ ellipse را فشرده و آن را به پیکربندی اضافه نمائید.

ساخت یک برنامه سادهٔ EJB

اکنون اولین برنامهٔ EJB خود را ایجاد می‌نمائیم. طبق معمول به سراغ برنامهٔ "Hello, World" می‌رویم.

ابتدا Borland Application و JBuilder را اجرا کرده و سپس گامهای زیر را برای ایجاد این برنامهٔ ("HelloWorld") طی کنید.

۱- تمامی پروژه‌های فعال در JBuilder را ببندید و سپس گزینهٔ New Project را از منوی فایل انتخاب کنید. پروژه را تحت عنوان "HelloWorld" ذخیره سازید.

۲- قدم دوم اضافه کردن یک گروه EJB است. برای انجام این کار به منوی File رفته و گزینهٔ New و سپس Enterprise را انتخاب کنید. اکنون نشان Empty EJB Group را برگزینید. و نام "HelloWorld" را به آن اختصاص دهید. چنانچه در این حالت یک فایل jar (فایل‌های jar همانند فایل‌های zip بوده و حاوی کلیه کدهای java برای یک پروژه می‌باشند) تولید شد، آن را به HelloWorld.jar تغییر نام دهید.

۳- اکنون نوبت EJB است. به منوی File رفته و به ترتیب گزینه‌های New ، Enterprise و Enterprise Java Beans را انتخاب کنید. در صورتی که اعلان تغییر نام ظاهر شد، نام HelloBean را وارد نمائید. توجه داشته باشید که JBuilder5 تمامی Interface های لازم را به صورت خودکار ایجاد خواهید نمود.

۴- در پنجرهٔ Project ، فایل HelloBean.java را انتخاب کرده و تب source را برگزینید. اکنون مطمئن شوید که کد نمایش داده شده همانند کد ارائه شده در زیر است.

```
package helloworld;
import java.rmi.*;
import javax.ejb.*;
import java.lang.String;
public class HelloBean implements SessionBean
{
   private SessionContext sessionContext;
   public void ejbCreate()
   {
   }
   public void ejbRemove() throws RemoteException
   {
   }
   public void ejbActivate() throws RemoteException
   {
   }
   public void ejbPassivate() throws RemoteException
```

```
    {
    }
    public void setSessionContext(SessionContext sessionContext) throws
    RemoteException
    {
       this.sessionContext = sessionContext;
    }
    public String sayHello() {
       return "Hello, world";
    }
}
```

۵- متد ()SayHello را برابر آنچه که در بالا ارائه شده است مقداردهی کنید.

۶- اکنون می‌بایست با استفاده از Remote (که یک Interface می‌باشد) متد ()SayHello را نمایش دهیم. برای انجام این کار تب Bean را از قسمت پایین پنجرهٔ Codes انتخاب کنید. سپس در پنجرهٔ نمایش داده شده تب ()Methods برگزیده و متد ()SayHello را فعال سازید. برای این که از انجام این کار مطمئن شوید، در پنجرهٔ Project فایل Hello.java را انتخاب کرده و از وجود متد ()SayHello در آن مطمئن شوید.

۷- تمام تغییرات را ذخیره نموده و گزینه build project را انتخاب کنید.

تست و خطایابی EJB

JBuilder این امکان را برای شما فراهم می‌آورد تا با نوشتن یک برنامهٔ Java در طرف سرویس‌گیرنده، عملیات تست و خطایابی EJB های خود را انجام دهید. روش انجام این کار را برای برنامهٔ "Hello, world" شرح می‌دهیم.

۱- در منوی File محیط JBuilder، گزینهٔ New و سپس Enterprise را برگزینید. اکنون از منوی ظاهر شده، گزینه EJB Test Client را انتخاب کرده و نام Hello Test Client.java را به آن اختصاص دهید.

۲- پس از آن JBuilder برنامهٔ مربوط به این کار را به صورت خودکار ایجاد می‌نماید. دقت کنید که برنامهٔ ایجاد شده کدی مشابه کد زیر داشته باشید.

```
public static void main(String[] args) {
   HelloTestClient1 client = new HelloTestClient1();
   client.create();   //add these two lines
   client.sayHello();
   // Use the client object to call one of the Home interface wrappers
   // above, to create a Remote interface reference to the bean.
   // If the return value is of the Remote interface type, you can use it
   // to access the remote interface methods. You can also just use the
   // client object to call the Remote interface wrappers.
}
```

توجه داشته باشید که متدهای ()SayHello و ()Create را شما به این برنامه اضافه می‌کنید.

اجرای EJB

برای اجرای EJB ارائه شده در مثال فوق، لازم است تا مراحل زیر را طی نمائید.

۱- در پنجرهٔ Project، موجودیت HelloGroup را انتخاب کرده و دکمهٔ سمت راست ماوس را بر روی آن بفشارید. سپس از منوی ظاهر شده، گزینهٔ Run را انتخاب کنید.

۲- با انتخاب برنامهٔ سرویس‌گیرنده، پنجره‌ای حاوی پیام "Hello, world" به نمایش در خواهد آمد.

۳- برای خاتمه کار این EJB، کافی است تا پنجرهٔ ظاهر شده را ببندید.

طراحی و ساخت EJB برای AppServer

مراحل طراحی و ساخت یک EJB برای AppServer به شرح زیر می‌باشد:

۱- گزینهٔ EJB Developent از منوی Tools را انتخاب کنید. (از EJB Development به عنوان یک ویزارد برای طراحی EJB ها استفاده می‌شود).

۲- گزینهٔ Next را برای چهار بار متوالی از کادرهای مکالمهٔ نمایشی داده شده در این ویزارد، انتخاب نمائید.

۳- در مرحلهٔ چهارم از این ویزارد، گزینهٔ EJB Container را برگزینید. با اطمینان یافتن از اجرای AppServer، دکمهٔ Add EJB Container را بفشارید. سپس از منوی ظاهر شده AppServer Container را انتخاب کرده و دکمهٔ OK را بفشارید و مراحل این ویزارد را تا کامل شدن آن ادامه دهید.

ساخت یک برنامهٔ سرویس‌گیرندهٔ EJB در دلفی

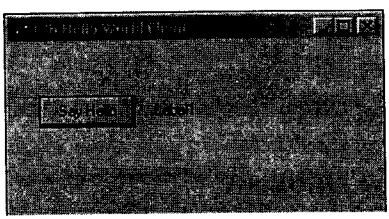
اکنون یک برنامهٔ سرویس‌گیرندهٔ CORBA طراحی می‌نمائیم تا از EJB ساخته شده استفاده نماید. مراحل انجام این کار به شرح زیر است:

۱- از منوی File گزینهٔ New و سپس Other را انتخاب کنید. در این حالت با انتخاب عنوان CORBA گزینهٔ CORBA Application را انتخاب نمائید.

۲- فایل HelloHome.idl را به فایل‌های برنامه‌تان بیفزائید.

۳- برنامه را با نام HelloClient ذخیره کنید.

۴- بر روی فرم اصلی برنامه، یک دکمه و یک برچسب همانند شکل ۴-۱۴ قرار دهید.



شکل ۴-۱۴ برنامهٔ سرویس‌گیرندهٔ EJB

۵- در متد ()Oncreate این فرم، عبارت initcorba را وارد نمائید.

۶- یونیت اصلی برنامه را همانند کد زیر پیاده‌سازی کنید.

```
unit ClientMain;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Corba, HelloHome_c, HelloHome_helloworld_c, HelloHome_helloworld_i,
  HelloHome_i, HelloHome_sidl_javax_ejb_c, HelloHome_sidl_javax_ejb_i,
  HelloHome_sidl_java_lang_c, HelloHome_sidl_java_lang_i,
  HelloHome_sidl_java_math_c, HelloHome_sidl_java_math_i,
  HelloHome_sidl_java_sql_c, HelloHome_sidl_java_sql_i,
  HelloHome_sidl_java_util_c, HelloHome_sidl_java_util_i,
  StdCtrls;

type
  TForm1 = class(TForm)
   Button1: TButton;
   Label1: TLabel;
   procedure FormCreate(Sender: TObject);
   procedure Button1Click(Sender: TObject);
 private
 { private declarations }
 protected
   myHome : HelloHome;
   myRemote : Hello;
   procedure InitCorba;
 { protected declarations }
 public
 { public declarations }
 end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.InitCorba;
begin
   CorbaInitialize;

   myHome := THelloHomeHelper.Bind;
   myRemote := myHome._create;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
   initCorba;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
   Label1.Caption := myRemote.sayHello;
end;

end.
```

طراحی و ساخت یک سرویس وب

برای ساخت یک سرویس وب مراحل زیر را طی نمائید.

۱- تمام فایل‌های پروژه جاری را ببندید.

۲- از منوی File گزینهٔ New و سپس Other را انتخاب کنید. در این حالت با انتخاب عنوان Web Server ، نشانهٔ Soap Server Application را برگزینید.

۳- مقدار Web App Debugger Executable را برای گزینهٔ target Application web Server انتخاب کرده و گزینهٔ Colcass Name را با عنوان coHelloWorld مقداردهی نمائید.

۴- برنامهٔ تولید شده را در همان فهرستی که حاوی فایل‌های IDL است ذخیره کنید. سپس فایل مربوط به ماژول وب را ServerMod.pas و فایل مربوط به فرم اصلی برنامه را ServerMain.pas بنامید و برنامه را با عنوان Server.dpr ذخیره سازید.

۵- از منوی File گزینهٔ New و سپس Other را انتخاب کنید. دراین حالت با انتخاب عنوان Web Services نشانهٔ Invokamatic Wizard را برگزینید.

۶- سپس یک جعبهٔ مکالمه برای شما به نمایش در می‌آید و شما نیز باید نامی را در این قسمت وارد نمائید. نام HelloWorldsoap را در این جعبهٔ مکالمه درج کنید. این نام مربوط به فایل‌ها و Interface های برنامه‌تان می‌باشد. بعد از آن عنوان TInvokable class را برای کلاس برنامه‌تان انتخاب کنید. اکنون با فشردن دکمهٔ OK ، دو یونیت جدید (یونیت مربوط به بخش Interface و Implementaion) به برنامهٔ شما اضافه می‌گردد.

۷- به یونیت مربوط به تعریف Interfac ها مراجعه کرده و متد زیر را به IHelloworldsoapIntf اضافه کنید.

```
function sayHello: string; stdcall;
```

۸- روال فوق را برای متد THelloWorldsoapIntf یونیت Implementaion انجام دهید.

۹- برنامه را ذخیره، کامپایل و اجرا نمائید. البته قبل از آن از اجرای برنامهٔ Web App Debugger

مطمئن شوید.

ساخت یک برنامهٔ سرویس‌گیرندهٔ SOAP

برای ساخت برنامهٔ سرویس‌گیرندهٔ SOAP برای مثال ذکر شده گام‌های زیر را طی نمائید.

۱- از منوی File، گزینهٔ New و سپس Application را انتخاب نمائید.

۲- یک label و یک edit control به فرم اصلی برنامه‌تان بیفزائید.

۳- فایل‌های soapHTTPClient و HelloWorldsoapIntf را نیز به برنامهٔ خود اضافه کنید. (این فایل‌ها مربوط به تعریف Interface ها می‌باشند).

۴- در متد OnClick این برنامه، کد زیر را وارد نمائید.

۵- برنامه را ذخیره و کامپایل کنید.

۶- برنامه را اجرا کرده و دکمهٔ Say Hello را بفشارید. بعد از باز کردن برنامهٔ سرویس‌دهنده پیغام "Hello World" برای شما به نمایش در خواهد آمد.

```
unit ClientMain;

interface

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

type
  TForm1 = class(TForm)
    Button1: TButton;
    Label1: TLabel;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
    mySoap : IHelloWorldSoap;
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
var x : THTTPRio;
begin
   x := THTTPRio.Create(nil);
```

```
    x.URL := 'http://localhost:1024/Server.exe/SOAP/';
    mySoap := x as IHelloWorldSoap;
    Label1.Caption := mySoap.sayHello;
end;

end.
```

افزودن کد CORBA به یک سرویس وب

۱- فایل‌هایی با عناوین i.pas_* و c.pas_* را به فهرست ذکر شده در کد بالا کپی نمائید.

۲- یونیت Interface مربوط به SOAP را به شکل زیر پیاده‌سازی کنید.

```
{ Invokable interface declaration unit for IHelloWorldSoap }

unit HelloWorldSoapIntf;

interface

uses
  Types, XSBuiltIns;

type
  IHelloWorldSoap = interface(IInvokable)
    ['{CA738F7B-B111-4F12-BEBD-C2ADDD80C3E2}']
    // Declare your invokable logic here using standard Object Pascal code
    // Remember to include a calling convention! (usually stdcall)
    // For example:
    // function Add(const First, Second: double): double; stdcall;
    // function Subtract(const First, Second: double): double; stdcall;
    // function Multiply(const First, Second: double): double; stdcall;
    // function Divide(const First, Second: double): double; stdcall;
    function sayHello : String; stdcall;
  end;

implementation
uses
  InvokeRegistry;

initialization
  InvRegistry.RegisterInterface(TypeInfo(IHelloWorldSoap), '', '');

end.
```

۳- فایل اصلی برنامه را به صورت زیر پیاده‌سازی کنید.

```
unit ServerMain;

interface

uses
  SysUtils, Classes, Graphics, Controls, Forms, Dialogs, Corba,
  HelloHome_helloworld_i, HelloHome_helloworld_c;

type
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    myHome :  HelloHome;
    myRemote : Hello;
  end;
var
  Form1: TForm1;

implementation

uses ComApp;

{$R *.DFM}

const
  CLASS_ComWebApp: TGUID = '{63859D3A-005F-43BB-8E64-85A466D9C364}';

procedure TForm1.FormCreate(Sender: TObject);
begin
  CorbaInitialize;
  myHome := THelloHomeHelper.bind;
  myRemote := myHome._create;
end;

initialization
  TWebAppAutoObjectFactory.Create(Class_ComWebApp,
    'coHelloWorld', 'coHelloWorld Object');

end.
```

۴- در متد sayHello() در فایل HelloWorldsoapImpl.pas را به شکل زیر تصحیح نمائید.

```
{ Invokable implementation declaration unit for THelloWorldSoap,
  which implements IHelloWorldSoap }

unit HelloWorldSoapImpl;
```

```
interface

uses
  HelloWorldSoapIntf, InvokeRegistry, ServerMain;

type
  THelloWorldSoap = class(TInvokableClass, IHelloWorldSoap)
    // Make sure you have your invokable logic implemented in IHelloWorldSoap
    // first, save the file, then use CodeInsight(tm) to fill in this
    // implementation section by pressing Ctrl+Space, marking all the interface
    // declarations for IHelloWorldSoap, and pressing Enter.
    // Once the declarations are inserted here, use ClassCompletion(tm)
    // to write the implementation stubs by pressing Ctrl+Shift+C
    function sayHello : String; stdcall;
  end;

implementation

{ THelloWorldSoap }

function THelloWorldSoap.sayHello: String;
begin
//  result := 'Hello, world';  //test for soap client
  result := ServerMain.Form1.myRemote.sayHello;
end;

initialization
  InvRegistry.RegisterInvokableClass(THelloWorldSoap);

end.
```

۵- برنامه را ذخیره و کامپایل نمائید.

۶- از اجرای برنامه‌های OSAgent و Borland AppServer مطمئن شوید.

۷- برنامهٔ سرویس‌گیرنده را اجرا نمائید.

خلاصه

در این فصل مطالب زیادی در خصوص برنامه‌های CORBA بررسی شد. در بخش نخست، ویژگیها و خصوصیات برنامه‌های CORBA و نحوهٔ کارآیی آنها در محیطهای سرویس‌گیرنده/ سرویس‌دهنده ارائه شد و در بخش‌های بعدی نحوهٔ پیاده‌سازی برنامه‌های CORBA تحت عنوان سرویس‌های وب ارائه گردید.

با آنچه که در این فصل آموخته‌اید، می‌توانید مطمئن باشید که CORBA مزایای قابل توجهی برای پشتیبانی از تولید برنامه‌های کاربردی توزیع یافته دارد.

فصل پانزدهم

طراحی سرویس‌های وب بر پایهٔ SOAP

در این فصل می‌خوانید

- **سرویس‌های وب**
- **SOPA چیست؟**
- **طراحی یک سرویس وب**
- **روشهای فراخوانی و اجرای سرویس‌های وب**

سرویس‌های یک شبکه، عبارت است از فعالیت‌ها و امکاناتی که آن شبکه در اختیار کامپیوترهای متصل به خود قرار می‌دهد. این سرویس‌ها ممکن است از طریق خدمات Online، اینترنت و یا از طریق تبادل الکترونیکی داده‌ها در اختیار مشترکان قرار گیرد. در شبکهٔ جهانی اینترنت و وب از این سرویسها تحت عنوان سرویس‌های وب یاد می‌شود.

سرویس‌های وب

بورلند، تعریف زیر را برای سرویس‌های وب ارائه می‌دهد:

"سرویس‌های وب با استفاده از بستر اینترنت، امکان برقراری ارتباط بین برنامه‌های کاربردی، عملیاتی نظیر تجارت الکترونیکی، انتقال داده‌ها و ... را فراهم می‌آورند. نکته حائز اهمیت این است که تمامی این سرویس‌ها با یک زبان استاندارد و پروتکل‌های مستقل از ساختارهای فیزیکی سیستم‌ها پیاده‌سازی می‌شوند".

همانطور که می‌دانید سیستم‌های توزیع یافته از یک سری سرویس‌دهنده و سرویس‌گیرنده تشکیل شده‌اند. این سیستمها، شبکهٔ غیرمتمرکزی از چندین کامپیوتر و برنامه هستند که می‌توانند با یکدیگر

ارتباط برقرار نمایند و از نظر کاربران به صورت یک سیستم بزرگ و واحد به نظر می‌رسند.

هر سیستم توزیع یافته ممکن است از چندین سرور تشکیل شده باشد که برخی از این سرورها به عنوان یک سرویس‌گیرنده هم، مطرح می‌باشند. سرویسهای وب، نوع جدیدی از سرویس‌دهنده‌ها در سیستم‌های توزیع یافته هستند و از پروتکل‌های اینترنت جهت انجام اعمال خود استفاده می‌کنند.

SOAP چیست؟

SOAP سرنام عبارت Simple Object Access Protocol بوده و معرف پروتکلی است که برای تبادل اطلاعات بین سیستم‌های توزیع یافته کاربرد دارد. پروتکل SOAP به عنوان جزئی از سیستم‌های CORBA نیز مطرح است که با استفاده از سندهای XML و پروتکل HTTP عمل تبادل اطلاعات را انجام می‌دهد. چنانچه مایلید اطلاعات بیشتری را در خصوص SOAP بدست آورید به آدرس **http://www.w3.org/TR/SOAP** بر روی اینترنت مراجعه نمائید.

WSDL نوع خاصی از XMLهاست که در سرویس‌دهنده‌های وب کاربرد دارد. WSDL سرنام Web Services Description Language بوده و به وسیلهٔ یک برنامهٔ سرویس‌گیرنده مورد استفاده قرار می‌گیرد. WSDL معرف محل قرارگیری سرویس‌های وب و چگونگی فراخوانی آنهاست. آنچه که در بالا گفته شد تنها خلاصه‌ای در تعریف SOAP و WSDL بود، با این حال توضیحات خود را در این دو مورد در همین جا خاتمه می‌دهیم.

در این فصل ابتدا شما را با نحوهٔ ایجاد یک سرویس وب آشنا ساخته و سپس روش دسترسی به این سرویس را از یک برنامهٔ سرویس‌گیرنده تشریح می‌نمائیم.

نوشتن یک سرویس وب

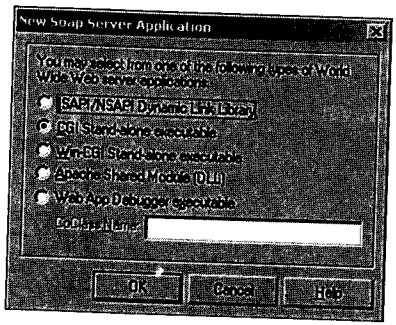
یک سرویس وب در دلفی از سه جزء به شرح ذیل تشکیل می‌شود.

۱- یک ماژول وب به همراه چند جزء سازندهٔ SOAP (که در ادامه به توضیح آن خواهیم پرداخت). این ماژول با اجرای SOAP Server Wizard به صورت خودکار ایجاد می‌شود.

۲- یک کلاس که به وسیلهٔ خود شما پیاده‌سازی شده و نحوهٔ کار سرویس وب را مشخص می‌سازد.

۳- یک Interface که برقراری ارتباط با کلاس موردنظر را به عهده دارد.

دلفی ۶ برای طراحی سرویس‌های وب از ویزاردی به نام Web Service Wizard استفاده می‌کند. با ارائهٔ یک مثال نحوهٔ انجام این کار را بیشتر توضیح خواهیم داد. این ویزارد بر روی تب WebServices از Object Repository قرار دارد. بر روی این تب سه عنوان وجود دارد که ما تنها مورد Soap Server Application Wizard را بررسی خواهیم کرد. با انتخاب این گزینه (Soap Server Application Wizard)، جعبهٔ مکالمه‌ای همچون شکل ۱-۱۵ به نمایش در می‌آید. در جعبهٔ مکالمهٔ ظاهر شده (شکل ۱-۱۵) گزینهٔ CGI Stand_alone را انتخاب کرده و دکمهٔ OK را



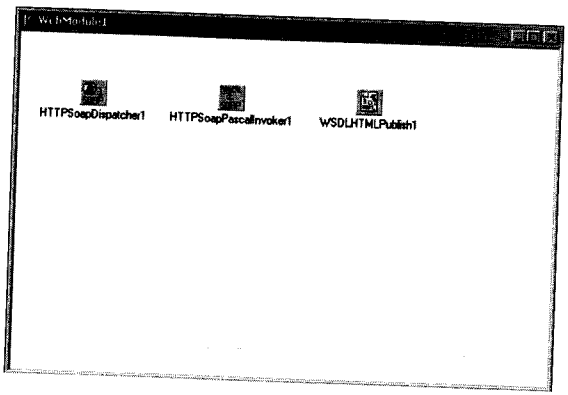
شکل ۱-۱۵ جعبهٔ مکالمهٔ New Soap Server Application

بفشارید. با این کار یک TWebModule توسط این ویزارد تولید می‌شود. (شکل ۲-۱۵)

TWebModule

همانطور که ملاحظه می‌کنید TWebModule از سه جزء سازنده به شرح زیر تشکیل شده است:

- THTTPSoapDispatcher : جزء سازنده‌ای است که از آن جهت دریافت پیام‌های SOAP و توزیع آنها در سیستم استفاده می‌شود. عملیات توزیع پیام‌ها با تنظیم خصوصیت Dispatcher این جزء سازنده انجام می‌گیرد.
- THTTPSoapPascalInvoker : این جزء سازنده که به وسیلهٔ THTTPSoapDispatcher.Dispatcher مورد ارجاع قرار می‌گیرد، عملیات دریافت و تفسیر پیام‌های SOAP و تعریف *Interface* فراخوانندهٔ این پیام‌ها را به عهده دارد.
- TWSDLHTMLPublish : جزء سازنده‌ای است که از آن برای توزیع سندهای WSDL استفاده می‌شود.



شکل ۲-۱۵ ماژول وب تولید شده به وسیلهٔ ویزارد

اکنون می‌بایست یک *Interface* از نوع Invokable را پیاده‌سازی نمائید. منظور از این *Interface*، *Interface* استفاده شده در جزء سازندهٔ THTTPSoapPascalInvoker است.

تعریف یک Interface فراخواننده

در این قسمت یونیت جدیدی را برای تعریف *Interface* موردنظر ایجاد می‌کنیم. کد مربوط به این یونیت در لیست ۱-۱۵ ارائه شده است. این یونیت را با نام TempConverterIntf.pas ذخیره کنید. این یونیت در برگیرندهٔ *Interface*ای است که متدهای مورد لزوم سرویس‌های وب را تعریف می‌کند. توجه داشته باشیدکه این *Interface* از IInvokable اشتقاق یافته است.

مثال مطرح شده در این قسمت در رابطه با تبدیل درجه حرارت از مقیاس فارنهایت به سانتیگراد و بالعکس می‌باشد، لذا از دو متد برای تبدیل درجهٔ حرارت استفاده خواهیم کرد. متدی به نام Purpose() هم تعریف می‌کنیم که یک رشتهٔ کاراکتری را به نمایش در آورد (نمایش این رشته به معنی اتمام عملیات خواهد بود).

توجه داشته باشید که می‌بایست برای این *Interface* یک GUID هم ایجاد نمائیم. (برای ایجاد GUID در محیط ویراستار، کلیدهای Ctrl + Shift + G را بفشارید).

لیست ۱-۱۵ تعریف یک Interface فراخواننده

```
unit TempConverterIntf;

interface

type
  ITempConverter = Interface(IInvokable)
    ['{6D239CB5-6E74-445B-B101-F76F5C0F6E42}']
    function FahrenheitToCelsius(AFValue: double): double; stdcall;
    function CelsiusToFahrenheit(ACValue: double): double; stdcall;
    function Purpose: String; stdcall;
  end;

implementation
uses InvokeRegistry;
initialization
  InvRegistry.RegisterInterface(TypeInfo(ITempConverter));

end.
```

یادداشت

در تعریف *Interface*های فراخواننده، ذکر عنوان stdcall الزامی است (همانطور که در ادامه ملاحظه خواهید کرد). در غیر این صورت *Interface* مورد نظر هیچ کاری انجام نخواهد داد. ▲

THTTPSoapPascalInvoker جزء سازنده‌ای است که تعریف *Interface* فراخواننده را در هنگام ارسال یک پیام SOAP به عهده می‌گیرد. در بخش initialization این یونیت متدی به نام RegisterInterface() قرار داده شده است. از این متد برای تأیید انجام کار در زمان اجرای سرویس استفاده می‌شود.

پیاده‌سازی *Interface*های فراخواننده با سایر *Interface*ها هیچ تفاوتی ندارد. Interface مربوط به مثال ذکر شده (برنامه تبدیل درجه حرارت) در لیست ۲-۱۵ پیاده‌سازی گردیده است. با دقت در این کد، نحوهٔ تعریف *Interface*های فراخواننده را خواهید دید و ملاحظه خواهید کرد که هیچ تفاوتی با تعریف سایر *Interface*ها ندارند.

لیست ۲-۱۵ پیاده‌سازی Interface

```
unit TempConverterImpl;

interface
uses InvokeRegistry, TempConverterIntf;
type

  TTempConverter = class(TInvokableClass, ITempConverter)
  public
    function FahrenheitToCelsius(AFValue: double): double; stdcall;
    function CelsiusToFahrenheit(ACValue: double): double; stdcall;
    function Purpose: String; stdcall;
  end;

implementation

{ TTempConverter }

function TTempConverter.CelsiusToFahrenheit(ACValue: double): double;
begin
// Tf = (9/5)*Tc+32
  Result := (9/5)*ACValue+32;
end;

function TTempConverter.FahrenheitToCelsius(AFValue: double): double;
begin
// Tc = (5/9)*(Tf-32)

  Result := (5/9)*(AFValue-32);
end;

function TTempConverter.Purpose: String;
begin
  Result := 'Temperature converstions';
end;

initialization
  InvRegistry.RegisterInvokableClass(TTempConverter);
end.
```

همانطور که ملاحظه می‌کنید، در پیاده‌سازی *Interface* این برنامه از شیء TInvokableClass استفاده شده است. دلیل این کار براساس آنچه که در راهنمای دلفی ارائه شده است به شرح زیر می‌باشد:

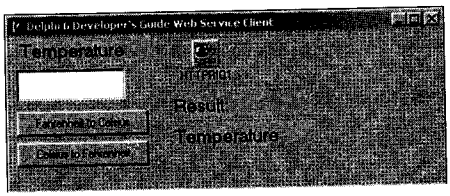
- ایجاد یک نمونه از TInvokableClass برای InvRegistry بسیار آسان است. یعنی InvRegistry به سادگی یک نمونه از TInvokableClass را ایجاد می‌نماید.
- TInvokableClass قابلیت آزادسازی خود به صورت خودکار را داراست.

اکنون این برنامه (سرویس وب) را کامپایل کرده، آن را در فهرست اجرایی یک سرور وب همانند Apache یا IIS قرار دهید (معمولاً این فهرست معادل Scripts\ یا cgi_bin\ می‌باشد).

اکنون لازم است تا شما را با نحوهٔ فراخوانی سرویس‌های وب آشنا سازیم.

نحوه فراخوانی یک سرویس وب از یک سرویس‌گیرنده

برای فراخوانی یک سرویس وب، ابتدا لازم است تا آدرس URL آن را بدانید. (از این URL جهت بازیابی سندهای WSDL استفاده می‌شود). چگونگی انجام این کار را نیز با ارائهٔ یک مثال پیش می‌گیریم. مثال قبل را در نظر بگیرید. فرم اصلی آن را همانند شکل ۳-۱۵ پیاده‌سازی کنید. کاربر درجهٔ حرارتی را در قسمت ویرایشی این فرم وارد می‌کند و با فشردن دکمه‌های موجود بر روی فرم، درجهٔ حرارت تبدیل شده به فارنهایت یا سانتیگراد در محل برچسب Temperature نمایش داده می‌شود.



شکل ۳-۱۵ فرم اصلی برنامهٔ سرویس‌گیرندهٔ سرویس وب

کد این برنامه در لیست ۳-۱۵ آورده شده است.

لیست ۳-۱۵ برنامهٔ سرویس‌گیرنده

```
unit MainFrm;

interface

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

type
  TMainForm = class(TForm)
```

لیست ۳-۱۵ ادامه

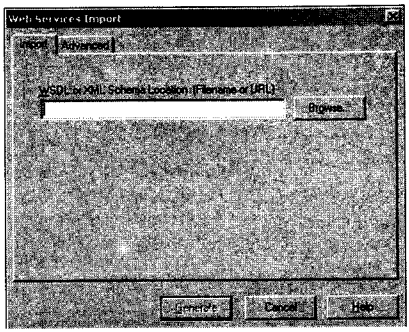
```
    btnFah2Cel: TButton;
    btnCel2Fah: TButton;
    edtArguement: TEdit;
    lblTemperature: TLabel;
    lblResultValue: TLabel;
    lblResult: TLabel;
    HTTPRIO1: THTTPRIO;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  MainForm: TMainForm;

implementation

uses TempConvImport;

{$R *.dfm}

end.
```

جزء سازندهٔ دیگری به نام THTTPRIO نیز در فرم اصلی برنامه قرار داده شده است (شکل ۳-۱۵). از این جزء سازنده برای دسترسی به یک شیء قابل فراخوانی استفاده می‌شود، ضمن این که می‌توان از آن به عنوان یک proxy برای سرویس‌های وب استفاده نمود. رویدادهای مربوط به این فرم باید به گونه‌ای تنظیم شوند که بتوانند یک شیء را از سرویس وب فراخوانی نمایند. توجه داشته باشید که قبل از نصب جزء سازندهٔ THTTPRIO در برنامه می‌بایست آن را تنظیم نمائیم. یعنی قبل از نصب، باید عملیات آماده‌سازی آن را انجام دهیم.

ایجاد یک یونیت Import برای دسترسی به یک شیء قابل فراخوانی

اگر قصد دارید از جزء سازندهٔ THTTPRIO در برنامه‌هایتان استفاده کنید، قبل از هر چیز لازم است تا یک یونیت Import برای شیء قابل فراخوانی پیاده‌سازی نمائید. این کار در دلفی به وسیلهٔ یک ویزارد انجام می‌پذیرد. این ویزارد در صفحهٔ WebServices از Object Repository قرار دارد و با اجرای آن جعبهٔ مکالمه‌ای همانند شکل ۴-۱۵ نمایش داده می‌شود.

برای وارد نمودن یک سرویس وب در برنامهٔ سرویس‌گیرنده، لازم است تا مسیر WSDL (مسیر URL) را در این ویزارد قرار داده و دکمهٔ Generate را بفشارید. این یونیت در لیست ۴-۱۵ ارائه شده است.



شکل ۴-۱۵ ویزارد
Web Services Import

لیست ۴-۱۵ پیاده‌سازی یونیت Import

```
Unit TempConvImport;

interface

uses Types, XSBuiltIns;

type

  ITempConverter = interface(IInvokable)
    ['{684379FC-7D4B-4037-8784-B58C63A0280D}']
    function FahrenheitToCelsius(const AFValue: Double): Double;  stdcall;
    function CelsiusToFahrenheit(const ACValue: Double): Double;  stdcall;
    function Purpose: WideString;  stdcall;
  end;


implementation

uses InvokeRegistry;

initialization
  InvRegistry.RegisterInterface(TypeInfo(ITempConverter),
➥ 'urn:TempConverterIntf-ITempConverter', '');

end.
```

بعد از ایجاد این یونیت، به فرم اصلی برنامهٔ سرویس‌گیرنده بازگردید و این یونیت را در آن به کار ببرید.

استفاده از جزء سازندهٔ THTTPRIO

برای استفاده از جزء سازندهٔ THTTPRIO لازم است تا سه خصوصیت آن را تنظیم نمائید. ابتدامسیر

WSDL را به خصوصیت WSDLLocation اختصاص دهید. سپس خصوصیت Service را با مقداری که نمایش داده می‌شود تنظیم کنید (با انتخاب این خصوصیت، خواهید دید که تنها یک مقدار جهت تخصیص به آن وجود دارد).

سپس به سراغ خصوصیت Port رفته و همین روال را برای آن انجام دهید. اکنون جزء سازندهٔ THTTPRIO آماده است تا از آن استفاده کنید.

اجرای یک سرویس وب

اکنون که تمامی اجزای برنامه آماده شده است، لازم است تا روالی را جهت اجرای این سرویس وب پیاده‌سازی کنید. برای این کار لیست ۵-۱۵ را در رویداد OnClick دکمه‌ها وارد نمائید.

لیست ۵-۱۵ رویداد OnClick

```
procedure TMainForm.btnFah2CelClick(Sender: TObject);
var
  TempConverter: ITempConverter;
  FloatVal: Double;
begin
  TempConverter := HTTPRIO1 as ITempConverter;
  FloatVal := TempConverter.FahrenheitToCelsius(StrToFloat(edtArguement.Text));
  lblResultValue.Caption := FloatToStr(FloatVal);
end;

procedure TMainForm.btnCel2FahClick(Sender: TObject);
var
  TempConverter: ITempConverter;
  FloatVal: Double;
begin
  TempConverter := HTTPRIO1 as ITempConverter;
  FloatVal := TempConverter.CelsiusToFahrenheit(StrToFloat(edtArguement.Text));
  lblResultValue.Caption := FloatToStr(FloatVal);
end;
```

تا اینجا مراحل آماده‌سازی یک سرویس وب در دلفی را فراگرفتید و ملاحظه کردید که دلفی تا چه اندازه شما را در انجام این کار یاری می‌رساند. تمام سرویس‌های وبی که در دلفی ایجاد می‌کنید به صورت مستقل از بسترهای نرم‌افزاری/ سخت‌افزاری به اجرا در خواهند آمد، بنابراین نگران اجرا نشدن سرویس‌های وب خود در دلفی نباشید. سیستم عاملهای ویندوز، Linux ، Solaris و حتی Mainframe ها از این سرویس‌ها پشتیبانی می‌کنند و این هم به خاطر ساختار مستقل آنهاست.

خلاصه

سرویسهای وب ابزارهای سازمان یافته‌ای هستند که در ارتقاء کارآیی سیستم‌های توزیع‌یافته تأثیر بسزا دارند. در این فصل نشان داده شد که چگونه می‌توان با دنبال کردن چند مرحله از یک ویزارد، سرویسهای وب را با کیفیت برنامه‌های خدماتی ایجاد نمود.

مثال‌های متنوعی در رابطه با ایجاد سرویس‌های وب و پیاده‌سازی آنها در سایت بورلند وجود دارند. توصیه می‌کنیم برای درک بهتر از این سرویس‌ها، مثالهای موجود را نیز بررسی کنید. برای دریافت این مثالها به همراه مستندات آنها به آدرس **http://community.borland.com/article** مراجعه نمائید.

فصل شانزدهم

طراحی برنامه‌های Multitier

در این فصل می‌خوانید

- **معماری و ساختار سیستم‌های Multitier**
- **مزایای استفاده از سیستم‌های Multitier**
- **طراحی برنامه‌های Multitier دقت وب**
- **روش برخورد با شرایط بحرانی**

اصول برنامه‌نویسی Multitier

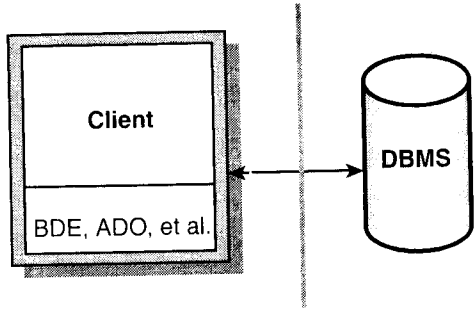
قبل از شروع بحث در رابطه با برنامه‌های Multitier و اصول و مفاهیم آنها، لازم است تا مطالبی را در رابطه با واژه "tier" ارائه نمائیم. tier به لایه‌ای از یک برنامه گفته می‌شود که خود این لایه از چندین تابع (مجموعهٔ تابع) تشکیل شده است. هر سیستم Multitier حاوی چندین tier یا به عبارتی چندین لایه است. به عنوان مثال برنامه‌های بانک اطلاعاتی Multitier شامل سه لایه به شرح زیر می‌باشند:

۱- *Data*: لایه‌ای است که عملیات ذخیره‌سازی داده‌ها را انجام می‌دهد. به عنوان مثال این لایه می‌تواند شامل نرم‌افزارهایی همچون Oracle ، InterBase ، RDBMS و Microsoft SQL Server باشد.

۲- *Business*: این لایه وظیفهٔ بازیابی داده‌ها از لایهٔ Data را به عهده دارد. گاهی اوقات از این لایه به عنوان لایه Application Server نیز یاد می‌شود.

۳- *Presentation*: عملیات نمایش داده‌ها در این لایه صورت می‌پذیرد. ضمناً توجه داشته باشید که ارتباط این لایه و لایه Data تنها از طریق لایهٔ Business انجام‌پذیر است. به عنوان مثال پیاده‌سازی یک GUI برای یک برنامه Multitier از این لایه انجام می‌شود.

در شکل ۱-۱۶ ساختار عمومی یک برنامهٔ بانک اطلاعاتی سرویس‌دهنده/ سرویس‌گیرنده ارائه شده است. همانطور که در شکل نشان داده شده است لازم است تا بر روی هر سرویس گیرنده

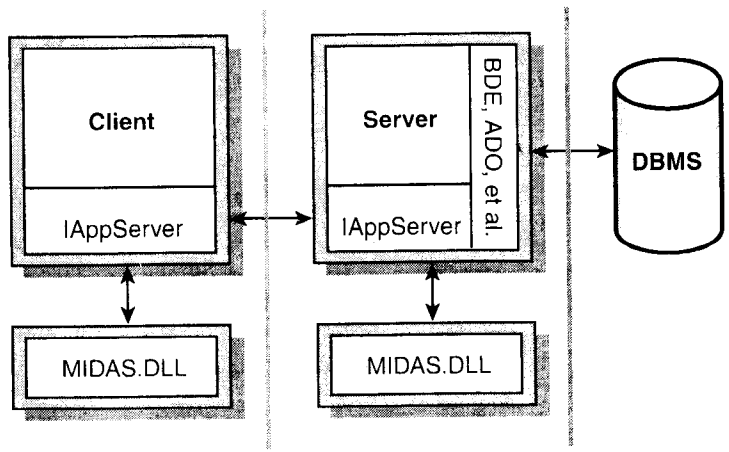


شکل ۱-۱۶ ساختار عمومی یک سیستم بانک اطلاعاتی سرویس‌دهنده/ سرویس‌گیرنده

مجموعه‌ای از توابع کتابخانه‌ای برای دسترسی به داده‌ها نصب گردد. این کار به معنی پیاده‌سازی لایه Business بر روی سیستم سرویس گیرنده می‌باشد.

ساختار عمومی سیستم‌های Multitier نیز در شکل ۲-۱۶ ارائه گردیده است. در ادامهٔ این فصل به مزایای استفاده از این ساختار و یا به عبارتی مزایای سیستم‌های Multitier خواهیم پرداخت.

در اکثر سیستم‌های سرویس‌گیرنده/ سرویس‌دهنده روال کار بدین ترتیب است که هر سیستم سرویس‌گیرنده عملیات ارتباطی را از طریق لایهٔ Business مربوط به خود انجام می‌دهد. این بدان معنی است که برای هر دستگاه سرویس‌گیرنده یک لایهٔ Business بر روی آن نصب و راه‌اندازی خواهد شد. از آنجا که لایهٔ Business نیز از چندین نرم‌افزار تشکیل می‌شود، لذا ضرورت دارد تا تمامی این نرم‌افزارها بر روی سیستم سرویس‌گیرنده نصب گردند. حال فرض کنید که نسخهٔ پیاده‌سازی شده این نرم‌افزار برای



شکل ۲-۱۶ ساختار عمومی سیستم‌های Multitier

چند یا همهٔ سرویس‌گیرنده‌ها یکسان نباشد. به نظر شما چه اتفاقی خواهد افتاد؟

بدون شک این کار باعث خواهد شد تا هر سرویس‌گیرنده پاسخی متناسب با توابع و نرم‌افزارهای نصب شده در لایهٔ Business ارائه نماید و این یعنی عدم پاسخ‌دهی متناظر سرویس‌گیرنده‌ها و بروز خطای منطقی در سیستم!!

اما اگر این لایه بر روی برنامهٔ سرویس‌دهنده نصب و تعریف شود آنگاه تمامی سرویس‌گیرنده‌ها نیز از یک نسخهٔ واحد در لایهٔ Business بهره‌مند خواهند شد. این درست همان روشی است که سیستم‌های Multitier از آن استفاده می‌کنند. (شکل ۲-۱۶)

استفاده از معماری Thin- Client

معماری سیستم‌های سرویس‌گیرنده/ سرویس‌دهنده شکلهای متفاوتی دارند. منظور از این معماری، نوع و نحوهٔ پردازش داده‌ها توسط سیستم‌های سرویس‌دهنده/ سرویس‌گیرنده است. سیستم‌های Multitier از نوع معماری Thin_clien استفاده می‌کنند. در این نوع معماری سرویس‌گیرنده/ سرویس‌دهنده، سیستم سرویس‌گیرنده توانایی پردازش مستقل اطلاعات را داراست ولیکن برای ذخیره‌سازی داده‌ها و مدیریت آنها به سرویس‌دهنده‌ها متکی است. در مقابل این معماری مدل Fat_ Client وجود دارد. Fat_ Client در معماری سرویس‌گیرنده/ سرویس‌دهنده به ماشین سرویس‌گیرنده‌ای گفته می‌شود که تمام یا بیشتر پردازش‌ها را انجام می‌دهد.

تلفیق خودکار خطاها

"تلفیق خودکار خطاها" مکانیزم جدیدی است که در نرم‌افزار دلفی گنجانیده شده است. در سیستم‌های Multitier عملیات تفسیر داده‌ها ابتدا بر روی دستگاههای سرویس‌گیرنده انجام می‌پذیرد. و اگر در یک زمان تنها یک سرویس‌گیرنده عملیات تغییر و بروزرسانی رکوردی از بانک اطلاعاتی را انجام می‌دهد، مشکلی پیش نخواهد آمد، اما اگر درخواست برای تغییر و بروزرسانی یک رکورد خاص توسط چندین سرویس‌گیرنده ارسال شود، روال کار به چه صورت است؟

مسئولیت انجام بدون نقص این کار به عهدهٔ مکانیزم تلفیق خودکار خطاهاست که به صورت توکار در دلفی گنجانیده شده است.

مدل چمدانی

در این مدل امکان تغییر و بروزرسانی فایل‌ها بدون برآوری ارتباط با سرویس‌دهنده، وجود دارد. ارائه این مدل برای سیستم‌های Multitier در دلفی، شما را قادر می‌سازد تا ابتدا بر روی سرویس‌گیرنده فایل را ویرایش نموده و بعد از برقراری ارتباط با سرویس‌دهنده، تمام فایل‌ها را بروز برسانید.

Fault Tolerance

این امکان به مفهوم توانائی یک کامپیوتر یا سیستم عامل در پاسخگویی به یک رویداد فاجعه‌انگیز مثل قطع برق یا خرابی سخت‌افزار است، به گونه‌ای که داده‌ها از دست نرفته و کار در دست اجرا خراب نشود.

در این حالت سیستم می‌تواند به هنگام بروز مشکلی، عمل جاری را بدون از دست رفتن داده‌ها ادامه دهد و یا عملکرد تمام قسمت‌ها را قطع و مجدداً راه‌اندازی نماید.

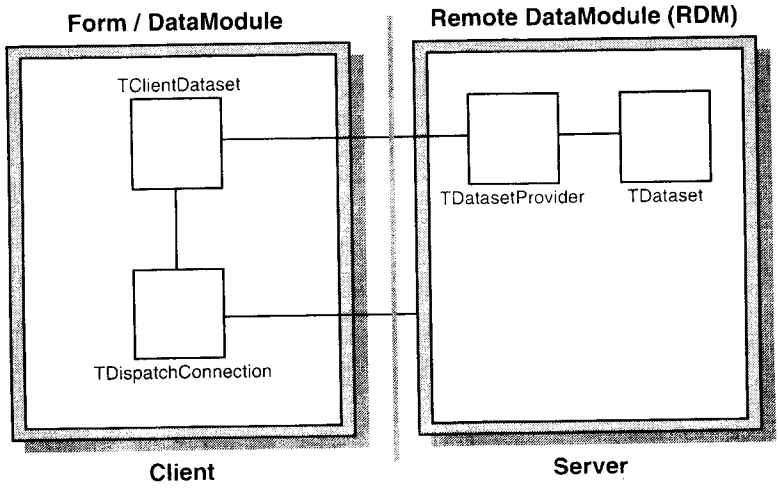
توزیع یکنواخت در بارگذاری داده‌ها

در مدیریت سیستم‌های سرویس‌گیرنده/ سرویس‌دهنده، این عبارت به مفهوم فرآیند کاهش ترافیک‌های موجود گفته می‌شود. این کار یا از طریق تقسیم یک قسمت شلوغ از شبکه به چندین قسمت کوچکتر و یا از طریق بکارگیری یک نرم‌افزار برای توزیع ترافیک انجام می‌شود. این امکان در نسخه‌های دلفی ۴ به بالا قرار داده شده است.

ساختار برنامه‌های کاربردی DataSnap

ساختار عمومی برنامه‌های کاربردی DataSnap در شکل ۳-۱۶ نشان داده شده است. همانطور که ملاحظه می‌کنید این ساختار از دو بخش Data Module و Remote Data Module (RDM) تشکیل گردیده است. Romote Data Module بر پایه خصوصیتهای COM پیاده‌سازی می‌شود و به عنوان ماژول سرویس‌دهنده مورد استفاده قرار می‌گیرد.

ابتدا نحوهٔ ایجاد Remote Data Module (RDM) در دلفی را بررسی می‌کنیم.

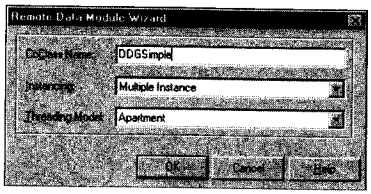


شکل ۳-۱۶ ساختار برنامه‌های کاربردی DataSnap

سرور

همانطور که در شکل ۳-۱۶ ملاحظه می‌کنید، برای ساخت یک سرور ابتدا لازم است تا یک Remote Data Module (RDM) ایجاد کنید. برای ایجاد RDM گزینه New از منوی File را انتخاب کرده و در پنجرهٔ نمایش داده شده، نشانهٔ Remote Data Module را برگزینید. با انجام این کار جعبهٔ مکالمه‌ای همانند شکل ۴-۱۶ به نمایش در می‌آید.

بر روی این جعبهٔ مکالمه دو گزینهٔ انتخابی به نامهای Instancing و Threading Model وجود دارد که در زیر به تشریح آنها خواهیم پرداخت.

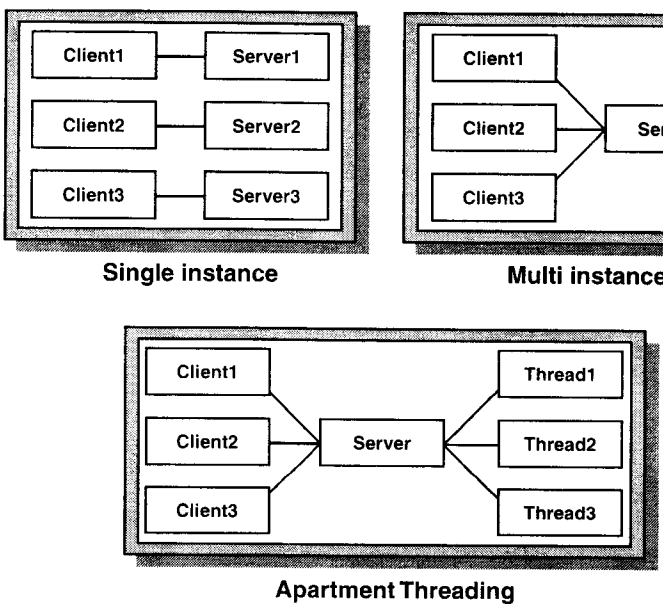


شکل ۴-۱۶ جعبهٔ مکالمهٔ New Remote Data Module

گزینهٔ Instancing

این گزینه نحوهٔ ایجاد فرآیندهای سرور و تخصیص آنها به سرویس‌گیرنده‌ها رامشخص می‌سازد. سه روش برای تخصیص این فرآیند وجود دارد که در شکل ۵-۱۶ به آنها اشاره شده است.

سه انتخاب برای گزینهٔ Instancing در جعبهٔ مکالمهٔ Remote Data Module (شکل ۴-۱۶) به شرح



شکل ۵-۱۶ نحوهٔ تخصیص فرآیندهای موجود بر روی سرور

زیر وجود دارد. (گزینهٔ CoClassName در قسمت بالای این جعبهٔ مکالمه معرف سرورهای COM می‌باشد که در فصل ۱۸ به آنها اشاره شده است).

- Multipie Instance (CiMultiInstance) : در این حالت یک نمونه از فرآیند موردنظر در طرف سرویس‌دهنده ساخته شده و به صورت مشترک در اختیار سرویس‌گیرنده‌ها قرار می‌گیرد. توجه داشته باشید که گزینهٔ Threading Model (شکل ۴-۱۶) نیز در نحوهٔ کارکرد این مدل تأثیر مستقیم دارد.
- Single Instance (CiSingleInstance) : در این حالت برای هر سرویس‌گیرنده یک نمونه مجزا از فرآیند موردنظر در طرف سرویس‌دهنده ایجاد و اختصاص داده می‌شود. این کار امکان دسترسی موازی سرویس‌گیرنده‌ها به سرویس‌دهنده‌ها را میسر می‌سازد.
- Internal (CiInternal) : در این حالت امکان ایجاد فرآیند سرویس‌دهندهٔ COM از طرف برنامهٔ کاربردی وجود ندارد. این گزینه در برنامه‌های سرویس‌دهنده دارای Proxy کاربرد خواهد داشت.

گزینهٔ Threading Model

چهار انتخاب به نام‌های Single ، Apartment ، Free و Both در این گزینه وجود دارد که برای درک بهتر آنها لازم است تا به شکل ۵-۱۶ مراجعه نمائید.

دسترسی به داده‌ها

دلفی ۷ اجزاء سازندهٔ متنوعی را برای دسترسی به داده‌ها فراهم آورده است. TDBDataset ، TTable ، TQuery و TStoredProc همگی اجزاء سازنده‌ای هستند که روش‌های خاصی را برای دسترسی به داده‌ها فراهم می‌آورند. جدا از این اجزاء سازنده، دلفی ۷ از ابزارهایی همچون dbExpress (در فصل ۶ به معرفی آن پرداخته‌ایم) نیز برای دسترسی به داده‌ها بهره می‌گیرد. منظور این است که با استفاده از قابلیت‌ها و امکانات منحصر به فرد دلفی ۷ قادر خواهید بود تا به راحتی به منابع مختلف داده‌ای دستیابی داشته باشید.

سرویس‌ها

یکی از وظایف RDM ارائه و تخصیص سرویس‌های درخواستی به سرویس‌گیرنده‌هاست. اجرای یک پرس‌وجو، ویرایش داده‌ها و ... تنها عناوینی از این سرویسها می‌باشند که به وسیلهٔ بخش RDM در اختیار سرویس‌گیرنده‌ها قرار می‌گیرند. به طور مثال اگر قرار باشد، RDM یک سرویس پرس‌وجو را در اختیار سرویس‌گیرنده‌ها قرار دهد لازم است تا جزء سازندهٔ TQuery را در آن قرار دهید و خصوصیت Dataset از جزء سازندهٔ TDatasetProvider آن را با TQuery مرتبط سازید.

سرویس‌گیرنده

پس از ساخت و طراحی سرویس‌دهنده، نوبت طراحی سرویس‌گیرنده فرا می‌رسد. ولی قبل از هر چیز

اجازه دهید تا برخی از گزینه‌های انتخابی را که در زمان ساخت سرویس‌گیرنده DataSnap با آن مواجه می‌شوید بررسی کنیم.

Connection Choice

برقراری ارتباط اولیه بین سرویس‌گیرنده‌ها و سرویس‌دهنده‌ها در ساختار داخلی دلفی توسط جزء سازندهٔ TDispatchConnection صورت می‌پذیرد. هر جزء سازندهٔ دیگری که برای برقراری ارتباط بین سرویس‌گیرنده و سرویس‌دهنده در این فصل معرفی خواهیم کرد از اجزاء سازندهٔ TDispatchConnection اشتقاق یافته است.

TDCOMConnection جزء سازندهٔ دیگری است که در پیاده‌سازی سطوح امنیتی ارتباط بین سرویس‌دهنده‌ها و سرویس‌گیرنده‌ها کاربرد دارد. این جزء سازنده عموماً در برنامه‌های Intranet/Extranet کاربرد داشته و کارآیی خوبی را از خود به نمایش می‌گذارد، اما پیاده‌سازی آن کمی مشکل خواهد بود.

سه دلیل عمده پیچیده بودن این پیاده‌سازی عبارتند از:

- پیکربندی این جزء سازنده مراحل پیچیده‌ای دارد.
- این جزء سازنده از نوع اجزاء سازندهٔ firewall نمی‌باشد.
- بکارگیری آن در سیستم عامل ویندوز ۹۵ مستلزم نصب نرم‌افزاز DCOM95 می‌باشد.

TSocketConnection همانند جزء سازندهٔ TDCOMConnection بوده و با این تفاوت که از مراحل پیکربندی آسانتری برخوردار است. توجه داشته باشید که هنگامی که از جزء سازنده TSocketConnection استفاده می‌کنید قبل از استفاده از جزء سازندهٔ TSocketConnection لازم است تا فایل Scktsrvr را از فهرست Bin دلفی اجراء نمائید.

اگر می‌خواهید در برنامه‌هایتان با یک برنامه کاربردی CORBA ارتباط برقرار سازید، می‌بایست از جزء سازندهٔ TCORBAConection استفاده نمائید. TWebConnection جزء سازندهٔ قدرتمند دیگری است که در برقراری ارتباط با پروتکل‌های HTTP و HTTPS کاربرد دارد، ولی بکارگیری آن در برنامه‌ها محدودیت‌های زیر را اعمال خواهد کرد.

- لازم است تا WININET.DLL بر روی سرویس‌گیرنده نصب شود.
- سیستم عامل دستگاه سرویس‌دهنده لزوماً IIS4 (یا بالاتر) Netscap3.6 (یا بالاتر) باشد.

با معرفی مختصر این اجزاء ملاحظه کردید که دلفی اجزاء سازندهٔ متنوعی را برای برنامه‌های کاربردی سرویس‌دهنده/ سرویس‌گیرنده ارائه می‌نماید، و این در حالی است که نسخه‌های جدید دلفی هم با اجزاء سازندهٔ قدرتمندتری عرضه می‌شوند. یکی از اجزاء TSOAPconnection در دلفی ۷ است که برقراری ارتباط با سرویس‌های وب DataSnap را میسر می‌سازد.

ارتباط بین اجزاء سازنده

چگونگی برقراری ارتباط بین لایه‌های یک برنامهٔ کاربردی DataSnap را ملاحظه نمودید (شکل

۳-۱۶). اکنون جزئیات بیشتری را در این خصوص بررسی خواهیم کرد.

همانطور که در قسمت قبل گفته شد، برای برقراری ارتباط سرویس‌گیرنده‌ها و سرویس‌دهنده‌ها در برنامه‌های DataSnap از جزء سازندهٔ TDispatchConnection استفاده می‌شود. کارآیی این جزء سازندهٔ مشابه با TDataBase در برنامه‌های سرویس‌دهنده/ سرویس‌گیرنده است. برای استفاده از این جزء سازنده می‌بایست جزء سازندهٔ دیگری به نام TClientDataset را در برنامهٔ سرویس‌گیرنده قرار داده و خصوصیت RemoteServer از TDispatchConnection را با آن مرتبط سازید. این امر باعث می‌شود که TClientDataset همانند یک TTable عمل کند.

استفاده از DataSnap برای ساخت برنامه‌های کاربردی

در این قسمت به ساخت اولین برنامهٔ کاربردی به وسیلهٔ DataSnap خواهیم پرداخت.

تنظیم سرور و Remote Data Module

RDM مرکزی‌ترین بخش از برنامهٔ سرویس‌دهنده است. در این قسمت با ارائه یک مثال به بررسی بیشتر RDM خواهیم پرداخت.

برای ایجاد یک RDM نشانهٔ Remote Data Module از Object Repository را برگزینید (به وسیلهٔ گزینهٔ New از منوی File). پس از انجام این کار یک جعبهٔ مکالمه برای مقداردهی و تنظیم RDM نمایش داده خواهد شد.

بعضی از خصوصیات و پارامترهای RDM از اهمیت بیشتری نسبت به سایر پارامترها برخوردار می‌باشند. یکی از این پارامترها ProgID است که مستقیماً و از روی نام فایل پروژه برنامه و نام RDM مقداردهی می‌شود. به طور مثال اگر فایل پروژه برنامه (dpr) معادل Appserver و نام RDM برابر با MyRDM باشد آنگاه مقدار Appserver.MyRDM به ProgID اختصاص داده می‌شود. از اجزاء سازندهٔ TSocketConnection و TWebconnection هم، برای اعمال تصدیق فرآیندها (سطح تصدیق پیش‌فرض ویندوز) استفاده می‌گردد. تمام این کارها در متدی به نام UpdateRegistry و به شرح لیست ۱-۱۶ پیاده‌سازی می‌شود.

کد فوق، پس از ایجاد یک Remote Data Module و به صورت خودکار ایجاد گردیده است.

یکی از وظایف برنامه‌های سرویس‌دهنده، انتقال بدون نقص داده‌ها به سیستم‌های سرویس‌گیرنده است. مهمترین بخش این عملیات، هماهنگی و حفظ قالب داده‌ها بر روی سیستم‌های سرویس‌دهنده و سرویس‌گیرنده می‌باشد. TDatasetProvider جزء سازنده‌ای است که پیاده‌سازی این عملیات را به عهده می‌گیرد. قبل از قرار دادن این جزء سازنده، یک جزء سازندهٔ TQuery به بخش RDM اضافه نمائید چنانچه از یک RDBMS در برنامه‌تان استفاده می‌کنید، جزء سازندهٔ TDatabase را هم به بخش RDM بیفزائید). حال به عنوان مثال، یک کد SQL ساده (نظیر Select * From Customer) را به خصوصیت SQL جزء سازندهٔ TQuery اختصاص دهید. اکنون زمان قرار دادن TDatasetProvider فرا

لیست ۱-۱۶ پیاده‌سازی متد UpdateRegistry

```
class procedure TDDGSimple.UpdateRegistry(Register:  Boolean;
 const ClassID, ProgID: string);
begin
  if Register then
  begin
    inherited UpdateRegistry(Register, ClassID, ProgID);
    EnableSocketTransport(ClassID);
    EnableWebTransport(ClassID);
  end else
  begin
    DisableSocketTransport(ClassID);
    DisableWebTransport(ClassID);
    inherited UpdateRegistry(Register, ClassID, ProgID);
  end;
end;
```

رسیده است. TDatasetProvider را در RDM قرار داده و آن را با جزء سازندهٔ TQuery (از طریق خصوصیت Dataset) مرتبط سازید. برای اعلان نتیجه این SQL بر روی سرویس‌گیرنده، می‌توانید از خصوصیت Exported جزء سازندهٔ TDatasetProvider استفاده کنید. (تخصیص مقدار Ture به این خاصیت به مفهوم نمایش نتایج فرآیندها بر روی سرویس‌گیرنده می‌باشد).

نکته

اجزاء سازنده‌ای که تا اینجا مطرح شد همانند TDBDataset و TDataset ، اجزائی هستند که در موتور بانک اطلاعاتی دلفی (BDE) بکار برده می‌شوند. توجه داشته باشید که دلفی ۷ در برگیرنده ابزار دیگری نظیر DBExpress ، ADO و InterBase Express است که از آنها برای طراحی بانک‌های اطلاعاتی سرویس‌گیرنده/ سرویس‌دهنده استفاده می‌شود.

ثبت کردن یک سرور

ثبت یک برنامهٔ سرویس‌دهنده پس از ایجاد آن انجام می‌گیرد. دانستن این نکته ضروری است که بدون ثبت یک برنامهٔ سرویس‌دهنده، امکان برقراری ارتباط آن با سرویس‌گیرنده وجود ندارد. ضمناً توجه داشته باشید که این سرویس‌دهنده‌ها با عنوان سرورهای COM ثبت خواهند شد. (نحوهٔ ثبت آنها را در فصل ۱۵ ارائه کرده‌ایم).

یک راه ساده برای انجام این کار اجرای برنامهٔ سرویس‌دهنده با استفاده از regserver/ و unregserver/ می‌باشد. regserver/ پارامتری است که عملیات ثبت برنامهٔ سرویس‌دهنده را انجام می‌دهد. از unregserver/ نیز برای خارج کردن برنامهٔ سرویس‌دهنده از حالت ثبت استفاده می‌شود.

ایجاد سرویس‌گیرنده

تا این قسمت نحوهٔ ایجاد برنامهٔ سرور (سرویس‌دهنده) را آموختید. اکنون بحث خود را در رابطه با طرف سرویس‌گیرنده تعمیم داده و مطالبی را در خصوص بازیابی، ویرایش و بروزرسانی داده‌ها در بانک‌های اطلاعاتی سرویس‌گیرنده/ سرویس‌دهنده مطرح می‌نمائیم.

بازیابی داده‌ها

یکی از فرآیندهای مهم بانک‌های اطلاعاتی سرویس‌گیرنده/ سرویس‌دهنده عملیات مربوط به بازیابی داده‌هاست. یکی از گامهای مهم در پیاده‌سازی این عملیات شناسایی و دانستن مکان برنامهٔ سرویس‌دهنده است. این مکان می‌بایست به جزء سازندهٔ TDispatchConnection شناسانده شود. برای این کار از خصوصیت Servername این جزء سازنده استفاده می‌شود. بعد از اینکه یک نام را به این خصوصیت اختصاص دادید نام اختصاص داده شده در این قسمت به خصوصیت ServerGUID نیز تخصیص داده می‌شود.

ServerGUID مهمترین خصوصیت در جزء سازندهٔ TDispatchConnection برای تعیین نام و محل سرویس‌دهنده است.

یادداشت

خصوصیت Servername در تمام اجزاء سازنده به یک شکل رفتار نمی‌کنند. این خصوصیت در جزء سازندهٔ TDCOMConnection لیستی از برنامه‌های سرویس‌دهندهٔ ثبت شده بر روی همان ماشین را نمایش می‌دهد و در جزء سازندهٔ TSocketConnection معرف برنامه‌های سرویس‌دهنده‌ای است که بر روی ماشین سرور ثبت گردیده‌اند. ▲

برای برقراری ارتباط بین جزء سازندهٔ TDispatchConnection و برنامهٔ سرویس‌دهنده، ابتدا مقدار True را به TDispatchConnection.Connected اختصاص دهید. بعد از برقراری ارتباط لازم است تا روشی را برای بازیابی داده‌ها از طرف سرویس‌دهنده پیاده‌سازی کنید. برای این کار از جزء سازنده TClientDataset استفاده کنید. اکنون باید بین این جزء سازنده و TDispatchConnection ارتباط برقرار نمائید. برای این کار از خصوصیت Remoteserver جزء سازندهٔ TClientDataset استفاده کنید. اکنون با فعال نمودن TClientDataset بصورت TClientDataset.Active قادر خواهید بود تا به داده‌های برنامهٔ سرویس‌دهنده دسترسی یابید.

ویرایش داده‌ها در طرف سرویس‌گیرنده

یکی دیگر از وظایف جزء سازندهٔ TClientDataset ذخیره‌سازی رکوردهایی است که از طرف سرویس‌دهنده به طرف سرویس‌گیرنده ارسال می‌شود. این رکوردها در خصوصیت Data این جزء

سازنده ذخیره می‌شوند.

از این پس هر تغییری که بر روی داده‌ها انجام گیرد در خصوصیت Delta از جزء سازنده TClientDataset ذخیره‌سازی می‌گردد. این تغییرات می‌توانند به عنوان حذف، اضافه و ویرایش یک یا چندین رکورد در سیستم مطرح شوند. اما توجه داشته باشید که برای هر عملیات درج و یا حذف تنها یک رکورد (رکورد موردنظر) در خصوصیت Delta ذخیره می‌شود. این در حالی است که برای عملیات بروزرسانی (update) ، دو رکورد (رکورد موردنظر قبل از بروزرسانی و بعد از بروزرسانی) در خصوصیت Delta ذخیره‌سازی خواهند شد.

لغو کردن تغییرات (Undo)

فرآیند احیای اطلاعات تغییر یافته (Undo) در بسیاری از برنامه‌های کاربردی وجود داشته و به کاربر امکان می‌دهد عمل انجام شده را لغو یا تکرار نماید. این عملیات در TClientDataset به وسیلهٔ فراخوانی متد ()cdsCustomer.undoLastChange انجام می‌پذیرد. مکانیزم Undo از یک پشته در حافظهٔ سیستم برای لغو و برگرداندن تغییرات استفاده می‌کند. اندازهٔ این پشته در جزء سازنده TClientDataset به صورت نامحدود در نظر گرفته شده است.

اکنون اگر قصد داشته باشید که کلیه تغییرات اعمال شده را لغو نمائید، چه روالی را پیش خواهید گرفت؟ یک روش استفاده از متد ()cdsCustomer.undoLastChange به تعداد مراحل موردنیاز است. روش دیگر که به نسبت روش اول بهینه‌تر می‌باشد استفاده از متد ()cdsCustomer.CancelUpdates می‌باشد. متد اخیر کلیه تغییرات اعمال شده را لغو می‌نماید.

تراکنش

عملیات تراکنش‌ها و نحوهٔ مدیریت آنها را در فصل ۱۳ مطالعه کرده‌اید. ClientDataset.SavePoint متدی است که انجام عملیات تراکنشی در طرف سرویس‌گیرنده را به عهده دارد.

Reconciling Data

به دلیل آن که اعمال تغییرات در رکوردهای یک بانک اطلاعاتی در طرف سرویس‌گیرنده انجام می‌گیرد لذا پس از تکمیل عملیات، می‌بایست عملیات بروزرسانی در طرف سرویس‌دهنده نیز انجام پذیرد. این کار به وسیلهٔ فراخوانی متد ()cdsCustomer.ApplyUpdates صورت می‌گیرد.

در هنگام فراخوانی این متد تغییرات درج شده در خصوصیت Delta (که قبلاً به آن اشاره نمودیم) به طرف سرویس‌دهنده ارسال می‌شوند. به عنوان مثال هنگامی که در طرف سرویس‌گیرنده، عملیات حذف یک رکورد صورت می‌گیرد لازم است تا پس از تأیید کار، این تغییرات بر روی برنامهٔ سرویس‌دهنده نیز اعمال گردد. این کار می‌تواند توسط مدیر سیستم پس از ملاحظه لیست رکوردهای حذف شده و تأیید عمل حذف آنها انجام گیرد. نحوهٔ پیاده‌سازی این عملیات در زیر ارائه شده است.

```
procedure TDataModule1.Provider1BeforeUpdateRecord(Sender: TObject;
  SourceDS: TDataset; DeltaDS: TClientDataset; UpdateKind: TUpdateKind;
  var Applied: Boolean);
begin
  if UpdateKind=ukDelete then
  begin
    Query1.SQL.Text:='update CUSTOMER set STATUS="DEL" where ID=:ID';
    Query1.Params[0].Value:=DeltaDS.FieldByName('ID').OldValue;
    Query1.ExecSQL;
    Applied:=true;
  end;
end;
```

برنامه‌های کاربردی با قابلیت Robust [۱]

در این بخش راه‌کارهای لازم برای طراحی برنامه‌های کاربردی با قابلیت Robust را بررسی خواهیم کرد. این تحلیل راه هم برای برنامه‌های سرویس‌گیرنده و هم برای برنامه‌های سرویس‌دهنده ارئه خواهیم نمود.

روشهای بهینه‌سازی برنامهٔ سرویس‌گیرنده

روش‌های بازیابی داده‌ها یک اصل اساسی در بهینه‌سازی برنامه‌های سرویس‌گیرنده (یا سیستم‌های سرویس‌گیرنده) است. نحوهٔ پیاده‌سازی و اجرای این روشها تأثیر بسزایی در کارآیی اینگونه برنامه‌ها خواهد داشت.

در قسمت قبل جزء سازندهٔ TClientDataset را معرفی نمودیم. همانطور که گفته شد، در هنگام استفاده از این جزء سازنده داده‌ها در حافظه اصلی ذخیره‌سازی می‌شوند. از این رو نحوهٔ مدیریت و تخصیص حافظه اصلی سیستم می‌تواند نقش مؤثری را در بهبود کارآیی این جزء سازنده و همچنین برنامهٔ سرویس‌گیرنده ایفا نماید. در ادامه تکنیک‌هایی برای پیاده‌سازی این عمل ارائه گردیده است.

محدودسازی اندازهٔ بسته‌ها

خصوصیت TClientDataset.PacketRecords مشخص کننده تعداد رکوردهایی است که در هر رتبه از فراخوانی TClientDataset بازیابی می‌شوند. بطور مثال با تخصیص مقدار عددی به این خصوصیت، عملیات واکشی برای تعداد ۱۰ رکورد از بانک اطلاعاتی صورت خواهد پذیرفت. تخصیص مقدار عددی ۱- به این خصوصیت معرف بازیابی کلیه رکوردهای موجود در بانک اطلاعاتی است.

با بکارگیری این خصوصیت قادر خواهید بود تا تعداد رکوردهای موجود در حافظه را در هنگام بازیابی آنها محدود نمائید. بدون شک این کار در عدم اشغال نادرست منابع حافظه‌ای مؤثر خواهد بود. مورد دیگری که اهمیت دارد موقعیت اشاره‌گر مربوط به فایل است. به این نکته توجه داشته باشید که هر

۱- Roubust : عنوان Robust به برنامه‌هایی گفته می‌شود که قابلیت کار با ارائه کار در شرایط بحرانی را دارند.

سرویس‌گیرنده از یک اشاره‌گر مخصوص به خود برای این کار استفاده می‌کند، اما لازم است تا مکان آخرین رکورد بازیابی شده توسط هر سرویس‌گیرنده را مشخص سازیم. در زیر قطعه کدی به عنوان مثال ارائه گردیده است که نحوهٔ اجرای این عملیات را نشان می‌دهد.

```
Server RDM:
procedure TStateless.DataSetProvider1BeforeGetRecords(Sender: TObject;
  var OwnerData: OleVariant);
begin
  with Sender as TDataSetProvider do
  begin
    DataSet.Open;
    if VarIsEmpty(OwnerData) then
      DataSet.First
    else
    begin
      while not DataSet.Eof do
      begin
        if DataSet.FieldByName('au_id').Value = OwnerData then
          break;
      end;
    end;
  end;
end;

procedure TStateless.DataSetProvider1AfterGetRecords(Sender: TObject;
  var OwnerData: OleVariant);
begin
  with Sender as TDataSetProvider do
  begin
    OwnerData := Dataset.FieldValues['au_id'];
    DataSet.Close;
  end;
end;

Client:
procedure TForm1.ClientDataSet1BeforeGetRecords(Sender: TObject;
  var OwnerData:  OleVariant);
begin
  // KeyValue is a private OleVariant variable
  if not (Sender as TClientDataSet).Active then
    KeyValue := Unassigned;
  OwnerData := KeyValue;
end;

procedure TForm1.ClientDataSet1AfterGetRecords(Sender: TObject;
  var OwnerData: OleVariant);
begin
  KeyValue := OwnerData;
end;
```

یادداشت

کد ارائه شده در بالا برای داده‌های تک سویه (در فصل ۶ معرفی شده‌اند) نیز قابل اعمال می‌باشد. به عنوان مثال چنانچه از DBExpress برای طراحی بانک اطلاعاتی برنامه‌تان استفاده کرده‌اید، می‌توانید کد فوق را در قسمتی از این برنامه پیاده‌سازی نمائید. ▲

استفاده از مدل چمدانی

یک راه دیگر برای کاهش ترافیک شبکه بین برنامه‌های سرویس‌گیرنده و سرویس‌دهنده استفاده از مدل چمدانی است. این کار تنها با تخصیص نام فایل بانک اطلاعاتی به خصوصیت TClientDataset.Filename انجام می‌پذیرد. با تخصیص این نام به خصوصیت Filename از جزء سازندهٔ TClientDataset ، یک کپی از بانک اطلاعاتی به صورت محلی در طرف سرویس‌گیرنده ایجاد می‌شود و از این پس تغییرات تنها بر روی این کپی و بر روی دستگاه سرویس‌گیرنده انجام می‌گیرد.

■ نکته

فایلهایی با قالب XML نیز در این خصوصیت قابل تعریف می‌باشند.

استفاده از کدهای SQL پویا

کدهای SQL به‌عنوان بخش‌هایی غیرقابل تفکیک از یک برنامهٔ بانک اطلاعاتی مطرح می‌باشند. برای استفاده از کدهای SQL به صورت پویا، ابتدا لازم است تا یک کد SQL را در خصوصیت TClientDataset.CommandText درج نمائید. سپس گزینهٔ PoAllowCommandText را به خصوصیت DatasetProvider.Options اختصاص دهید. اکنون با فراخوانی متد TClientDataset.Execute() قادر خواهید بود تا کد SQL خود را به طرف دستگاه سرویس‌دهنده ارسال و بر روی آن اعمال نمائید.

روشهای بهینه‌سازی برنامهٔ سرویس‌دهنده

روشهایی برای بهینه‌سازی برنامهٔ طرف سرویس‌دهنده، نیز وجود دارد که در زیر به آنها اشاره می‌نمائیم.

تطبیق جزئی اطلاعات

این روش به مفهوم تطبیق جزئی اطلاعات موجود با اطلاعات قبلی در یک بانک اطلاعاتی است. بدین معنی که اگر دو یا چند کاربر بر روی یک رکورد از بانک اطلاعاتی، عملیاتی را انجام دهند، پس از آن در ارسال رکورد موردنظر برای سایر کاربران، رکورد بروزرسانی شده ارسال گردد.

این کار به وسیلهٔ خصوصیت TDatasetProvider.UpdateMode صورت می‌پذیرد. نحوهٔ تنظیم

این خصوصیت به همراه پارامترهای آن در زیر ارائه گردیده است.

- upWhereAll : در هنگامی که یکی از کاربران در حال انجام عملیات بروزرسانی بر روی یک رکورد خاص می‌باشد و کاربر دیگری درخواست همان رکورد را اعلام می‌دارد، سیستم با نمایش پیغام خطای "Another user changed the Record" معتبر نبودن رکورد موردنظر را اطلاع می‌دهد.
- upWhereChanged : در این حالت دو کاربر قادرند به صورت همزمان نسبت به بروزرسانی رکوردها اقدام نمایند. به طور مثال چنانچه کاربر در حال تغییر فیلدی به نام Address از یک بانک اطلاعاتی است، همزمان با آن کاربر دیگری می‌تواند فیلد دیگری از این رکورد را تغییر دهد.
- upWhereKeyOnly : در این حالت تمامی کاربران قادرند عملیات بروزرسانی رکوردها را به صورت همزمان به انجام برسانند. در نظر داشته باشید که همیشه آخرین تغییرات اعمال شده به عنوان محتویات رکورد موردنظر بروزرسانی خواهد شد.

گزینه‌های مربوط به TDatasetProvider

TDatasetProvider.Options دارای گزینه‌های مختلفی است که هر کدام به نوعی در کارآیی برنامهٔ سرویس‌دهنده مؤثر خواهند بود. به عنوان مثال، انتخاب مقدار PoReadonly برای این خصوصیت قابلیت فقط خواندنی بودن بانک اطلاعاتی را برای سرویس‌گیرنده محیا می‌سازد. گزینه‌های PoDisableDeletes و PoDisableInserts و PoDisableEdits در جلوگیری از عملیاتهای ویرایشی کاربران دخالت دارند. از گزینه PoPrepogateChanges نیز جهت اعمال تغییرات بهنگام در طرف سرویس‌دهنده استفاده می‌شود.

یادداشت

گزینه‌های متنوعی در خصوصیت TDatasetProvider.Options وجود دارند که از هر کدام از آنها برای پیاده‌سازی یک قابلیت خاص استفاده می‌شوند. توصیه می‌شود با مراجعه به راهنمای دلفی این گزینه‌ها را بررسی نمائید. ▲

بروزرسانی بانک اطلاعاتی

همانطور که دیدید TClientDataset.ApplyUpdates متدی است که عملیات بروزرسانی جداول بانک اطلاعاتی را به انجام می‌رساند. در این قسمت نحوهٔ عملکرد این متد و متدهای مرتبط با آن را با ارائه چند مثال بررسی خواهیم نمود. لیست ۲-۱۶ انجام عملیات بروزرسانی در طرف سرویس‌گیرنده را نمایش می‌دهد.

نحوهٔ پیاده‌سازی این روال در طرف سرویس‌دهنده به صورت کد ارائه شده در لیست ۳-۱۶ می‌باشد.

متدهای فوق از لحاظ کارآیی قابل اطمینان می‌باشند اما جهت انجام عملیات بروزرسانی به صورت

لیست ۲-۱۶ عملیات بروزرسانی‌در طرف سرویس‌گیرنده

```
procedure TClientDM.ApplyUpdates;
var
  MasterVar, DetailVar: OleVariant;
begin
  Master.CheckBrowseMode;
  Detail_Proj.CheckBrowseMode;
  if Master.ChangeCount > 0 then
    MasterVar := Master.Delta else
    MasterVar := NULL;
  if Detail.ChangeCount > 0 then
    DetailVar := Detail.Delta else
    DetailVar := NULL;
  RemoteServer.AppServer.ApplyUpdates(DetailVar, MasterVar);
  { Reconcile the error datapackets. Since we allow 0 errors, only one error
    packet can contain errors. If neither packet contains errors then we
    refresh the data.}
  if not VarIsNull(DetailVar) then
    Detail.Reconcile(DetailVar) else
  if not VarIsNull(MasterVar) then
    Master.Reconcile(MasterVar) else
  begin
    Detail.Reconcile(DetailVar);
    Master.Reconcile(MasterVar);
    Detail.Refresh;
    Master.Refresh;
  end;
end;
```

لیست ۳-۱۶ عملیات بروزرسانی در طرف سرویس‌دهنده

```
procedure TServerRDM.ApplyUpdates(var DetailVar, MasterVar: OleVariant);
var
  ErrCount: Integer;
begin
Database.StartTransaction;
  try
    if not VarIsNull(MasterVar) then
    begin
      MasterVar := cdsMaster.Provider.ApplyUpdates(MasterVar, 0, ErrCount);
      if ErrCount > 0 then
        SysUtils.Abort;    // This will cause Rollback
    end;
    if not VarIsNull(DetailVar) then
   begin
     DetailVar := cdsDetail.Provider.ApplyUpdates(DetailVar, 0, ErrCount);
```

لیست ۳-۱۶ ادامه

```
      if ErrCount > 0 then
        SysUtils.Abort;    // This will cause Rollback
    end;
    Database.Commit;
  except
    Database.Rollback
  end;
end;
```

بهینه‌تر لازم است تا مراحل زیر را طی نمائید.

۱- محتویات خصوصیت Delta هر جزء سازنده را در یک آرایه نگهداری کنید.

۲- محتویات Providers مربوط به هر جزء سازنده را نیز در یک آرایه ذخیره‌سازی نمائید.

۳- تمام Delta را در یک تراکنش به انجام برسانید.

پیاده‌سازی این مراحل در یونیت ارائه شده در لیست ۴-۱۶ آورده شده است.

لیست ۴-۱۶ یونیت CDSUtil

```
unit CDSUtil;

interface

uses
  DbClient, DbTables;

function RetrieveDeltas(const cdsArray : array of TClientDataset): Variant;
function RetrieveProviders(const cdsArray : array of TClientDataset): Variant;
procedure ReconcileDeltas(const cdsArray : array of TClientDataset;
                          vDeltaArray: OleVariant);

procedure CDSApplyUpdates(ADatabase : TDatabase; var vDeltaArray: OleVariant;
                          const vProviderArray: OleVariant);

implementation

uses
  SysUtils, Provider, Midas, Variants;
type
  PArrayData = ^TArrayData;
  TArrayData = array[0..1000] of Olevariant;

{Delta is the CDS.Delta on input. On return, Delta will contain a data packet}
{containing all of the records that could not be applied to the database.}
{Remember Delphi needs the provider name, so it is passed in the first}
{element of the AProvider variant.}
procedure ApplyDelta(AProvider: OleVariant; var Delta : OleVariant);
```

لیست ۴-۱۶ ادامه

```
var
  ErrCount : integer;
  OwnerData:  OleVariant;
begin
  if not VarIsNull(Delta) then
  begin
    // ScktSrvr does not support early-binding
    Delta := (IDispatch(AProvider[0]) as IAppServer).AS_ApplyUpdates(
                AProvider[1], Delta, 0, ErrCount, OwnerData);
    if ErrCount > 0 then
      SysUtils.Abort;  // This will cause Rollback in the calling procedure
  end;
end;

{Server call}
procedure CDSApplyUpdates(ADatabase : TDatabase; var vDeltaArray: OleVariant;
  const vProviderArray: OleVariant);
var
  i : integer;
  LowArr, HighArr: integer;
  P: PArrayData;
begin
  {Wrap the updates in a transaction. If any step results in an error, raise}
  {an exception, which will Rollback the transaction.}
  ADatabase.Connected:=true;
  ADatabase.StartTransaction;
  try
    LowArr:=VarArrayLowBound(vDeltaArray,1);
    HighArr:=VarArrayHighBound(vDeltaArray,1);
    P:=VarArrayLock(vDeltaArray);
    try
      for i:=LowArr to HighArr do
        ApplyDelta(vProviderArray[i], P^[i]);
finally
      VarArrayUnlock(vDeltaArray);
    end;
   ADatabase.Commit;
 except
   ADatabase.Rollback;
 end;
end;

{Client side calls}
function RetrieveDeltas(const cdsArray : array of TClientDataset): Variant;
var
  i : integer;
  LowCDS, HighCDS : integer;
```

لیست ۴-۱۶ ادامه

```
 begin
   Result:=NULL;
   LowCDS:=Low(cdsArray);
   HighCDS:=High(cdsArray);
   for i:=LowCDS to HighCDS do
     cdsArray[i].CheckBrowseMode;

   Result:=VarArrayCreate([LowCDS, HighCDS], varVariant);
   {Setup the variant with the changes (or NULL if there are none)}
   for i:=LowCDS to HighCDS do
   begin
     if cdsArray[i].ChangeCount>0 then
       Result[i]:=cdsArray[i].Delta else
       Result[i]:=NULL;
   end;
 end;

 {If we're using Delphi 5 or greater, then we need to return the provider name
  AND the AppServer from this function. We will use ProviderName to call
  AS_ApplyUpdates in the CDSApplyUpdates function later.}
 function RetrieveProviders(const cdsArray : array of TClientDataset): Variant;
 var
   i: integer;
   LowCDS, HighCDS: integer;
 begin
   Result:=NULL;
   LowCDS:=Low(cdsArray);
   HighCDS:=High(cdsArray);

   Result:=VarArrayCreate([LowCDS, HighCDS], varVariant);
   for i:=LowCDS to HighCDS do
     Result[i]:=VarArrayOf([cdsArray[i].AppServer, cdsArray[i].ProviderName]);
 end;
procedure ReconcileDeltas(const cdsArray : array of TClientDataset;
  vDeltaArray: OleVariant);
var
  bReconcile : boolean;
  i: integer;
  LowCDS, HighCDS : integer;
begin
  LowCDS:=Low(cdsArray);
  HighCDS:=High(cdsArray);

  {If the previous step resulted in errors, Reconcile the error datapackets.}
  bReconcile:=false;
  for i:=LowCDS to HighCDS do
```

لیست ۴-۱۶ ادامه

```
    if not VarIsNull(vDeltaArray[i]) then begin
      cdsArray[i].Reconcile(vDeltaArray[i]);
      bReconcile:=true;
      break;
    end;

  {Refresh the Datasets if needed}
  if not bReconcile then
    for i:=HighCDS downto LowCDS do begin
      cdsArray[i].Reconcile(vDeltaArray[i]);
      cdsArray[i].Refresh;
    end;
end;

end.
```

اکنون برای اجرای متدهای فوق و استفاده از یونیت ارائه شده، یک فرم نمونه طراحی کرده و لیست ۵-۱۶ را در آن وارد نمائید.

لیست ۵-۱۶

```
procedure TForm1.btnApplyClick(Sender: TObject);
var
  vDelta: OleVariant;
  vProvider: OleVariant;
  arrCDS: array[0..1] of TClientDataset;
begin
  arrCDS[0]:=cdsMaster;  // Set up ClientDataset array
  arrCDS[1]:=cdsDetail;

  vDelta:=RetrieveDeltas(arrCDS);                    // Step 1
  vProvider:=RetrieveProviders(arrCDS);              // Step 2
  DCOMConnection1.ApplyUpdates(vDelta, vProvider); // Step 3
  ReconcileDeltas(arrCDS, vDelta);                   // Step 4
end;

procedure TServerRDM.ApplyUpdates(var vDelta, vProvider: OleVariant);
begin
  CDSApplyUpdates(Database1, vDelta, vProvider);  // Step 3
end;
```

برنامه‌های HTML

در فصل ۱۸ نحوهٔ ایجاد و پیاده‌سازی برنامه‌های سرویس‌دهندهٔ وب را فرا خواهید گرفت. امروزه گستردگی برنامه‌های کاربردی تحت وب، فراگیری و آموزش اینگونه برنامه‌ها را اجتناب‌ناپذیر نموده است. در فصل ۱۸ از این کتاب، مراحل مربوط به پیاده‌سازی برنامه‌های سرویس‌دهندهٔ وب بررسی شده است. در این قسمت قصد داریم تا شما را با نوشتن برنامه‌های بانک اطلاعاتی تحت وب، آشنا کنیم.

اولین گام برای این کار اضافه کردن اجزاء سازندهٔ TClientDataset و TDispatchConnection به ماژول وب برنامه‌تان می‌باشد.

از جزء سازندهٔ TDatasetTableProducer نیز برای برقراری ارتباط با TClientDataset استفاده می‌شود. لیست ۶-۱۶ جهت پیاده‌سازی عملیات ویرایش و بروزرسانی بانک‌های اطلاعاتی تحت وب ارائه شده است.

در لیست ۶-۱۶ کد HTML مربوط به این مثال را ملاحظه کردید. یونیت این برنامه نیز در لیست ۷-۱۶ ارائه گردیده است.

لیست ٦-١٦ پیاده‌سازی عملیات ویرایش و بروزرسانی اطلاعاتی تحت وب

```
<form action="<#SCRIPTNAME>/updaterecord" method="post">
<b>EmpNo: <#EMPNO></b>
<input type="hidden" name="EmpNo" value=<#EMPNO>>
<table cellspacing="2" cellpadding="2" border="0">
<tr>
  <td>Last Name:</td>
  <td><input type="text" name="LastName" value=<#LASTNAME>></td>
</tr>
<tr>
  <td>First Name:</td>
  <td><input type="text" name="FirstName" value=<#FIRSTNAME>></td>
</tr>
<tr>
  <td>Hire Date:</td>
  <td><input type="text" name="HireDate" size="8" value=<#HIREDATE>></td>
</tr>
<tr>
  <td>Salary:</td>
  <td><input type="text" name="Salary" size="8" value=<#SALARY>></td>
</tr>
<tr>
  <td>Vacation:</td>
  <td><input type="text" name="Vacation" size="4" value=<#VACATION>></td>
</tr>
</table>
<input type="submit" name="Submit" value="Apply Updates">
<input type="Reset">
</form>
```

لیست ۷-۱۶ یونیت WebMain

```
unit WebMain;

interface

uses
  Windows, Messages, SysUtils, Classes, HTTPApp, DBWeb, Db, DBClient,
  MConnect, DSProd;

type
  TWebModule1 = class(TWebModule)
    dcJoin: TDCOMConnection;
    cdsJoin: TClientDataSet;
    dstpJoin: TDataSetTableProducer;
    dsppJoin: TDataSetPageProducer;
    ppSuccess: TPageProducer;
    ppError: TPageProducer;
    procedure WebModuleBeforeDispatch(Sender: TObject;
      Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
    procedure WebModule1waListAction(Sender: TObject; Request: TWebRequest;
      Response: TWebResponse; var Handled: Boolean);
    procedure dstpJoinFormatCell(Sender: TObject; CellRow,
      CellColumn: Integer; var BgColor: THTMLBgColor;
      var Align: THTMLAlign; var VAlign: THTMLVAlign; var CustomAttrs,
      CellData: String);
    procedure WebModule1waEditAction(Sender: TObject; Request: TWebRequest;
      Response: TWebResponse; var Handled: Boolean);
    procedure dsppJoinHTMLTag(Sender: TObject; Tag: TTag;
      const TagString: String; TagParams: TStrings;
      var ReplaceText: String);
    procedure WebModule1waUpdateAction(Sender: TObject;
      Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
  private
    { Private declarations }
    DataFields : TStrings;
  public
    { Public declarations }
  end;

var
  WebModule1: TWebModule1;

implementation

{$R *.DFM}

procedure TWebModule1.WebModuleBeforeDispatch(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
```

لیست ۷-۱۶ ادامه

```
begin
  with Request do
    case MethodType of
      mtPost: DataFields:=ContentFields;
      mtGet: DataFields:=QueryFields;
    end;
end;

function LocalServerPath(sFile : string = '') : string;
var
  FN: array[0..MAX_PATH- 1] of char;
  sPath : shortstring;
begin
  SetString(sPath, FN, GetModuleFileName(hInstance, FN, SizeOf(FN)));
  Result := ExtractFilePath( sPath ) + ExtractFileName( sFile );
end;

procedure TWebModule1.WebModule1waListAction(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  cdsJoin.Open;
  Response.Content :=  dstpJoin.Content;
end;
procedure TWebModule1.dstpJoinFormatCell(Sender: TObject; CellRow,
  CellColumn: Integer; var BgColor: THTMLBgColor; var Align: THTMLAlign;
  var VAlign: THTMLVAlign; var CustomAttrs, CellData: String);
begin
  if (CellRow > 0) and (CellColumn = 0) then
    CellData := Format('<a href="%s/getrecord?empno=%s">%s</a>',
      [Request.ScriptName, CellData, CellData]);
end;

procedure TWebModule1.WebModule1waEditAction(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  dsppJoin.HTMLFile := LocalServerPath('join.htm');
  cdsJoin.Filter := 'EmpNo = ' + DataFields.Values['empno'];
  cdsJoin.Filtered := true;
  Response.Content := dsppJoin.Content;
end;

procedure TWebModule1.dsppJoinHTMLTag(Sender: TObject; Tag: TTag;
  const TagString: String; TagParams: TStrings; var ReplaceText: String);
begin
  if CompareText(TagString, 'SCRIPTNAME')=0 then
   ReplaceText:=Request.ScriptName;
end;
```

لیست ۷-۱۶ ادامه

```
procedure TWebModule1.WebModule1waUpdateAction(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
var
  EmpNo, LastName, FirstName, HireDate, Salary, Vacation:  string;
begin
  EmpNo:=DataFields.Values['EmpNo'];
  LastName:=DataFields.Values['LastName'];
  FirstName:=DataFields.Values['FirstName'];
  HireDate:=DataFields.Values['HireDate'];
  Salary:=DataFields.Values['Salary'];
  Vacation:=DataFields.Values['Vacation'];

  cdsJoin.Open;
  if cdsJoin.Locate('EMPNO', EmpNo, []) then
  begin
    cdsJoin.Edit;
    cdsJoin.FieldByName('LastName').AsString:=LastName;
    cdsJoin.FieldByName('FirstName').AsString:=FirstName;
    cdsJoin.FieldByName('HireDate').AsString:=HireDate;
    cdsJoin.FieldByName('Salary').AsString:=Salary;
    cdsJoin.FieldByName('Vacation').AsString:=Vacation;
    if cdsJoin.ApplyUpdates(0)=0 then
      Response.Content:=ppSuccess.Content else
      Response.Content:=pPError.Content;
  end;
end;

end.
```

خلاصه

در این فصل روش جدید و هیجان‌انگیزی برای مدیریت داده‌ها در سیستم‌های سرویس‌گیرنده/ سرویس‌دهنده ارائه شد. این روش سیستم Multitier نام داشت. با درک مفاهیم پایهٔ سیستم‌های Multitier و بررسی گام‌به‌گام مثال‌های کاربردی این فصل به مهارت خود در زمینه برنامه‌سازی پیشرفته بیفزائید.

فصل هفدهم

طراحی و ساخت برنامه‌های ASP

در این فصل می‌خوانید

- **شیءها و صفحات Active Server**
- **ویزارد Active Server Object**
- **اشیاء Actice Server و بانک‌های اطلاعاتی**
- **خطایابی شیءهای Active Server**

در این فصل ضمن درک مفاهیم Active Server Objects(ASO) و Active Server Pages(ASP) با ابزارهای فوق‌العادهٔ برنامه‌سازی دلفی در این خصوص آشنا خواهید شد.

شیءهای Active Server

ASP (Active Server Pages) نیز همچون برنامه‌های CGI ، ISAPI و NSAPI که به وسیلهٔ WebBroker دلفی قابل پشتیبانی می‌باشند، یک برنامهٔ کاربردی مبتنی بر وب است. بنابراین می‌توانید آنها را (Active Server Objects و Active Server Pages) را بر روی یک سرور وب قرار دهید تا چندین سرویس‌گیرنده از آنها استفاده کنند.

نسخه‌های دلفی ۵ به بعد دارای ویزاردی می‌باشند که عملیات ساخت ASO ها را به انجام می‌رساند. یک ASP با استفاده از ASO ها کدهای HTML پویا تولید می‌کند. بنابراین لازم است تا برای هر ASP چندین ASO تعریف نمائید. در این فصل ضمن آشنائی با مفاهیم ASO و ASP با نحوهٔ ارتباط آنها با CGI ، ISAPI و COM آشنا خواهید شد. ASO ها به عنوان اجزاء سازندهٔ سرویس‌دهنده مطرح بوده (بر روی سرور نصب می‌شوند) و لذا نوع سیستم عامل سرور در چگونگی نحوهٔ پیاده‌سازی آنها تأثیر دارد. به طور مثال طریقهٔ پیاده‌سازی آنها در سیستم عاملهای ویندوز ۲۰۰۰ و NT متفاوت خواهد بود.

صفحات Active Server

قبل از این که نحوهٔ ایجاد ASO را شرح دهیم، لازم است تا شما را با مفاهیم کلی ASP آشنا سازیم.

Active Server برای آن طراحی شده تا به تولید کنندگان برنامه‌های کاربردی امکان دهد که بدون هرگونه نیاز به تمرکز بر روی پیچیدگی شبکه، به طراحی و ایجاد نرم‌افزارهای اینترنت و اینترانت با زبانهای مختلف بپردازند. ASP یک تکنولوژی وب‌گراست که به وسیله مایکروسافت ارائه گردیده است. Active Server امکان فرآیندنویسی طرف سرویس‌دهنده (در مقابل طرف سرویس‌گیرنده) را فراهم می‌کند. صفحه‌های Active Server فایل‌های متنی هستند که همچون سندهای استاندارد وب نه تنها متن و tag های HTML را نگهداری می‌کنند بلکه می‌توانند دستوراتی راکه با یک زبان فرآیندنویس همچون JavaScript و VBScript نوشته شده‌اند را نیز ذخیره کنند. این کار در طرف سرویس‌دهنده به یک طراح وب امکان می‌دهد که ویژگی محاوره‌ای بودن را به سندهای خود بیفزاید و مشاهده و تحویل اطلاعات به سرویس‌گیرنده را بدون هرگونه نگرانی در رابطه با محیطی که سرویس‌گیرنده در آن در حال اجرا است، به طور دلخواه تغییر دهد.

تمامی صفحات Active Server با انشعاب asp. ذخیره شده و همچون URL های استاندارد از طریق مرورگر وبی چون Netscape Navigator یا Internet Explorer قابل دستیابی هستند.

هنگامی که یکی از این صفحات توسط یک مرورگر درخواست می‌شود، سرویس‌دهنده دستورات فرآیند موجود در صفحه را اجرا می‌کند، سپس یک صفحه HTML تولید کرده و سند را برای نمایش در کامپیوتر درخواست کننده (سرویس‌گیرنده) ارسال می‌کند.

صفحات Active Server از دو نوع شیء (که به صورت توکار تعریف شده‌اند) برای برقراری ارتباط بین سرویس‌دهنده‌ها و مرورگرها استفاده می‌کنند. وجود این دو شیء همواره به عنوان امتیازات صفحات Active Server مطرح بوده است.

این دو شیء به شرح زیر می‌باشند:

- Request : این شیء برای دریافت ورودی از کاربران پیاده‌سازی شده است. به عبارت دیگر برای دریافت عناوین ورودی و چک نمودن مقادیر آنها از این شیء استفاده می‌شود.
- Response : این شیء برای چاپ و اعلان مقادیر خروجی به کاربران پیاده‌سازی شده است. به بیان دیگر برای نمایش خروجی داده‌ها از این شیء استفاده می‌شود.

برای درک بهتر خصوصیات این اشیاء مثال ساده‌ای را ارائه می‌کنیم. کد زیر قطعه‌ای از یک اسکریپت ASP را نشان می‌دهد. این اسکریپت عملیات ساده‌ای را انجام می‌دهد، در این اسکریپت با دریافت یک متغیر ورودی تحت عنوان Name ، اگر نام وارد شده Bob باشد، پیغام "Hello,Bob" در طرف سرویس‌گیرنده نمایش داده می‌شود و در غیر این صورت پیغام "Hello,User!" به نمایش در می‌آید. این اسکریپت با استفاده از اشیاء Request و Response پیاده‌سازی شده است.

```
<%
  if Request("Name") = "Bob" then
    Response.Write("Hello, Bob!")
  else
```

```
    Response.Write("Hello, User!")
  end if
%>
```

اگر این اسکریپت ASP را با نام test.asp ذخیره کنیم، آنگاه نحوهٔ فراخوانی آن به وسیله یک صفحه HTML به صورت زیر خواهد بود.

```
<FORM ACTION="test.asp" METHOD=POST>
Name: <INPUT TYPE=text NAME=Name>
<P>
<INPUT TYPE=submit>
</FORM>
```

توجه داشته باشید که صفحات Active Server تنها به وسیلهٔ سرورهای وب (سرویس‌دهنده‌های وب) قابل اجرا می‌باشند. یعنی اینکه پس از طراحی، برای اجراء لازم است تا آنها را در یک سرویس‌دهندهٔ وب قرار داده و از یک URL برای دسترسی به آنها استفاده کنیم. به طور مثال **(file:///d:/www/cgi-bin/test.asp)**.

ویزارد Active Server Objects

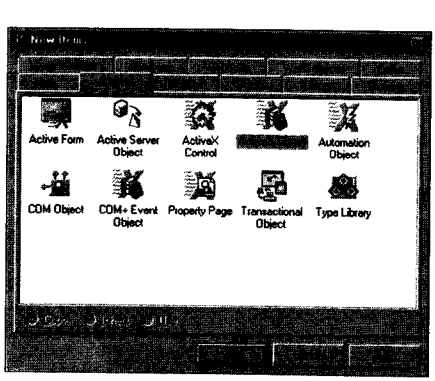
ویزاردی در نسخه Enterprise دلفی ۷ برای طراحی اشیاء Active Server ارائه شده است. این در حالی است که طراحی این اشیاء در نسخه Professional دلفی ۷ به وسیله برنامه‌نویسی صورت می‌پذیرد. آنچه که در اینجا انجام خواهیم داد مربوط به نسخهٔ Enterprise دلفی ۷ است. در Object Repository دلفی ۷ و بر روی تب ActiveX، ویزاردی جهت طراحی اشیاء Active Server وجود دارد.

حال کار با این ویزارد را با ارائهٔ یک مثال کاربردی شروع می‌کنیم و به وسیلهٔ آن اشیاء Active Server را ایجاد می‌نمائیم. ترتیب و مراحل انجام کار به شرح ذیل ارائه گردیده است:

۱- دلفی ۶ را اجرا کرده و به وسیلهٔ گزینهٔ close، تمامی فایلهای پروژهٔ موجود را ببندید.

۲- در منوی File، گزینه New و سپس نشان ActiveX Library را از تب ActiveX انتخاب نمائید (شکل ۱-۱۷).

۳- فایل پروژهٔ ActiveX Library را با عنوان D6ASP.dpr ذخیره کنید.



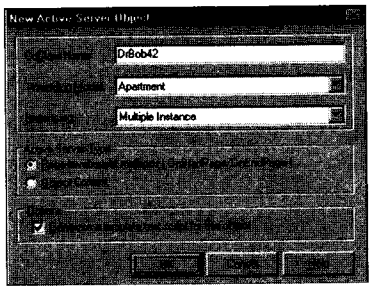
شکل ۱-۱۷ ActiveX Library که در تب ActiveX وجود دارد

نکته

توصیه می‌شود بلافاصله بعد از اجرای دلفی، فایلهای پروژه را با استفاده از گزینهٔ close بسته و غیرفعال نمائید. اگر این کار شما را خسته می‌سازد، می‌توانید از گزینهٔ np- در اجرای دلفی استفاده کنید. در این صورت عملیات بستن فایلهای پروژه به صورت خودکار انجام می‌پذیرد. به طور مثال دلفی را به صورت زیر اجرا نمایید.

```
"C:\Program Files\Borland\Delphi7\Bin\delphi32.exe" -np
```

پس از انجام مراحل فوق، جعبهٔ مکالمه‌ای همانند شکل ۲-۱۷ به نمایش در می‌آید.



شکل ۲-۱۷ جعبه مکالمه
New Active Server Object

عنوان CoClass Name در این جعبه مکالمه نمایانگر نامی برای اشیاء COM برنامه‌تان می‌باشد. شما می‌توانید هر نام دلخواهی را در این قسمت وارد کنید اما ما نام DrBob42 را برای این مثال انتخاب کرده‌ایم (با تخصیص DrBob42 به CoClassName کلاسی تحت عنوان TDrBob42 و Interface ای با نام IDrBob42 ایجاد خواهد شد).

در قسمت دیگر این جعبه مکالمه گزینه Threading Model وجود دارد. همانطور که ملاحظه می‌کنید مقدار پیش‌فرض Apartment برای این گزینه در نظر گرفته شده است. عنوان Instancing این جعبهٔ مکالمه نیز با مقدار Multiple Instance مقداردهی شده است.

برای عنوان Threading Model ، پنج انتخاب به شرح زیر وجود دارد:

- Single : در این حالت تمامی درخواستهای سرویس‌گیرنده‌ها در قالب یک thread[1] مطرح و به آنها پاسخ داده می‌شود، بدین معنی که همه سرویس‌گیرنده‌ها، درخواستهایشان را تنها در قالب یک thread ارائه می‌نمایند. در این حالت تمامی درخواستها در یک صف انتظار (برای پاسخ‌دهی) قرار می‌گیرند. بدون تردید تشکیل این صف انتظار باعث افت کارایی سیستم می‌شود.
- Apartment : در این حالت هر کدام از سرویس‌گیرنده‌ها، یک thread مخصوص به خود داشته و

۱- در برنامه‌سازی به فرآیندی گفته می‌شود که بخشی از یک فرآیند بزرگتر است.

درخواستهایش را از طریق آن ارسال می‌کند.

- Free : در این حالت نمونهٔ کلاس ایجاد شده در یک زمان در معرض دسترس چندین thread قرار می‌گیرد. با انتخاب این حالت می‌بایست اقدامات امنیتی بیشتری را برای برنامه‌تان در نظر بگیرید.
- Both : این حالت برابر با ترکیب دو حالت Apartment و Free می‌باشد.
- Neutral : این حالت مشابه با حالت Apartment بوده که تنها برای شیءهای COM و COM+ قابل پیاده‌سازی است.

اما در رابطه با عنوان Instancing این جعبهٔ مکالمه، سه انتخاب به شرح زیر وجود دارد:

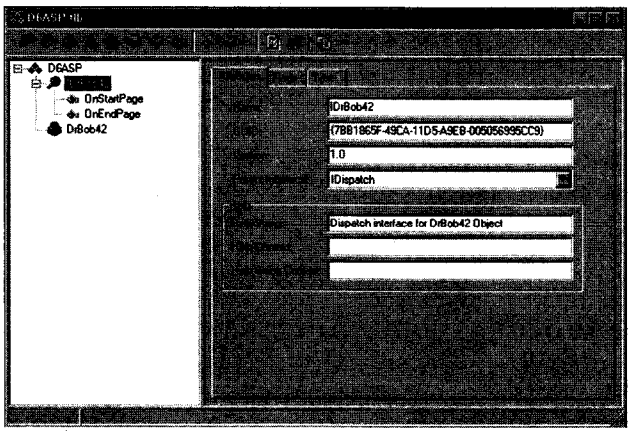
- Internal Instance : در این حالت، یک شیء COM تنها در درون DLL مربوط به خودش تعریف می‌شود.
- Single Instance : در این حالت هر برنامهٔ کاربردی از یک نمونه برنامهٔ سرویس‌گیرنده استفاده می‌نماید.
- Multiple Instance : در این حالت هر برنامهٔ کاربردی از چند نمونه برنامه سرویس‌گیرنده استفاده می‌نماید.

تعیین گزینهٔ Active Server Type بر روی این جعبهٔ مکالمه، وابسته به نسخهٔ IIS ای است که بر روی سیستم نصب گردیده است. برای IIS3 گزینهٔ page-level event methods و برای IIS5 و IIS4 گزینهٔ Object Context را انتخاب نمائید. گزینهٔ اخیر برای برنامه‌های COM+ و *MTS (Microsoft Transaction Server)* بکار برده می‌شود.

آخرین گزینهٔ موجود بر روی این جعبهٔ مکالمه مربوط به ایجاد یک اسکریپت موقت برای تست و خطایابی اشیاء ایجاد شده است.

ویراستار Type Library

برای هر شیء Active Server یک type library وجود دارد. شکل ۳-۱۷ ویراستار Type Library را برای مثال مطرح شده نمایش می‌دهد.



شکل ۳-۱۷ ویراستار Type Library برای IDrBob42

اکنون گزینهٔ Save All را از منوی File انتخاب کرده و Until برنامه را با نام DrBob42ASP.pas ذخیره نمائید (این یونیت حاوی شیءهای Active Server می‌باشد). بعد از آن سیستم از شما می‌خواهد تا فایل DrBob42.asp را ذخیره کنید. این فایل حاوی کدهای HTML برای ASP است. محتویات این فایل در زیر ارائه شده است.

```
<HTML>
<BODY>
<TITLE> Testing Delphi ASP </TITLE>
<CENTER>
<H3> You should see the results of your Delphi Active Server method below </H3>
</CENTER>
<HR>
<% Set DelphiASPObj = Server.CreateObject("D6ASP.DrBob42")
   DelphiASPObj.{Insert Method name here}
%>
<HR>
</BODY>
</HTML>
```

همچنان که مشاهده می‌کنید tag های ASP به وسیلهٔ علامت % از tag های HTML جدا شده‌اند و در این مثال این tag در برگیرندهٔ یک اسکریپت است.

در خط اول این اسکریپت یک نمونه از شیء DrBob42 ایجاد شده است و در خط دوم از این اسکریپت یک متد بدون نام را فراخوانی نموده‌ایم.

مجدداً به شکل ۳-۱۷ برمی‌گردیم. همانطور که در شکل ملاحظه می‌کنید، Interface ی IDrBob42 شامل دو متد به نامهای OnStartPage و OnEndPage می‌باشد. وجود این دو متد از آنجا ناشی می‌شود که شما در قسمت Active Server Type (شکل ۲-۱۷)، گزینهٔ Page-level event methods را انتخاب کرده بودید.

کد یونیت DrBob42.ASP (که آن را به وسیلهٔ ویزارد ایجاد و ذخیره نموده‌اید) در لیست ۱-۱۷ ارائه گردیده است.

اکنون مثال دیگری را مطرح می‌کنیم. جعبه مکالمهٔ New Active Server Object (شکل ۲-۱۷) را فراخوانی نموده و نام Micha42 را به CoClass Name اختصاص دهید. این بار گزینهٔ Object Context را برای Active Server Type انتخاب نمائید (در مقابل مثال قبل که Page-level event methods را انتخاب کرده بودید). برنامه را با عنوان Micha42ASP.pas و فایل ASP آن را تحت عنوان Micha42.asp ذخیره سازید.

کد یونیت Micha42ASP در لیست ۲-۱۷ ارائه شده است. همانطور که مشاهده می‌کنید، این یونیت تفاوت اندکی با یونیت DrBob42ASP دارد. دیگر اثری از متدهای OnStartPage و OnEndPage مشاهده نمی‌شود، ضمناً توجه داشته باشید که کلاس TDrBob42 زیرکلاسی از

لیست ۱-۱۷ کد مربوط به شیء DrBob42.ASP

```
unit DrBob42ASP;

{$WARN SYMBOL_PLATFORM OFF}

interface
uses
  ComObj, ActiveX, AspTlb, D6ASP_TLB, StdVcl;

type
  TDrBob42 = class(TASPObject, IDrBob42)
  protected
    procedure OnEndPage; safecall;
    procedure OnStartPage(const AScriptingContext: IUnknown); safecall;
  end;

implementation

uses ComServ;

procedure TDrBob42.OnEndPage;
begin
  inherited OnEndPage;
end;

procedure TDrBob42.OnStartPage(const AScriptingContext: IUnknown);
begin
  inherited OnStartPage(AScriptingContext);
end;

initialization
  TAutoObjectFactory.Create(ComServer, TDrBob42, Class_DrBob42,
    ciMultiInstance, tmApartment);
end.
```

TASPObject و کلاس TMicha42 زیرکلاسی از TASPMTSObject می‌باشند.

لیست ۲-۱۷ کد مربوط به شیء Micha42.ASP

```
unit Micha42ASP;

{$WARN SYMBOL_PLATFORM OFF}

interface
uses
```

لیست ۲-۱۷ ادامه

```
  ComObj, ActiveX, AspTlb, D6ASP_TLB, StdVcl;

type
  TMicha42 = class(TASPMTSObject, IMicha42)
  end;

implementation

uses ComServ;

initialization
  TAutoObjectFactory.Create(ComServer, TMicha42, Class_Micha42,
    ciMultiInstance, tmApartment);
end.
```

اکنون با افزودن چند متد به یونیتهای DrBob4ASP و Micha42ASP قصد داریم تا رفتار و نحوهٔ کار متدها و اشیاء Active Server را بررسی نمائیم.

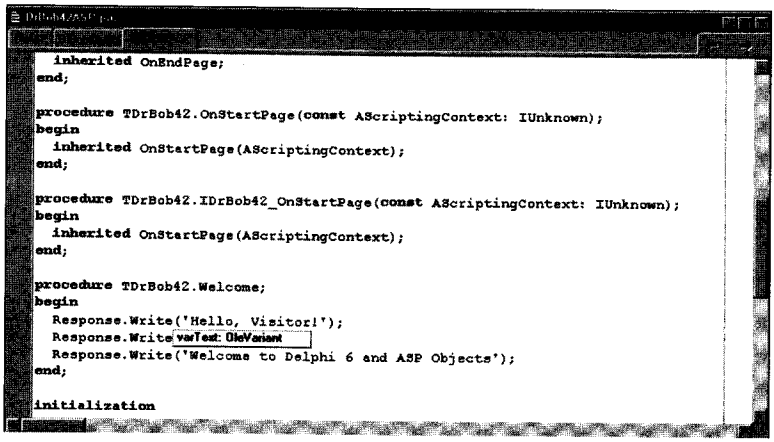
متدهای جدید

در این قسمت یک متد جدید را به IDrBob42 اضافه می‌کنیم. این متد را طوری پیاده‌سازی می‌نماییم که توسط یک صفحهٔ وب نیز قابل فراخوانی باشد، توجه داشته باشید که این متدها می‌توانند به عنوان بخشی جدا از متدهای OnStartPage و OnEndPage تعریف شوند. اکنون روش این کار را توضیح می‌دهیم. با رجوع به ویراستار Type Library برنامهٔ DrBob42 (شکل ۳-۱۷)، به سراغ Interface ی IDrBob42 رفته و دکمهٔ سمت راست ماوس را بر روی آن بفشارید. در این حالت گزینهٔ New و سپس Method را انتخاب کرده و یک متد جدید به نام Welcome را به آن بیفزائید.

قصد داریم از این متد برای نمایش یک پیام داینامیک استفاده کنیم (نمایش یک پیام welcome که قابل تغییر نیز باشد). اکنون که متد Welcome ایجاد شده است، می‌توانید کد آن را در TDrBob42.Welcome وارد نمائید. اما قبل از آن اجازه دهید تا توضیحات بیشتری را در خصوص اشیاء Response و Request ارائه نمائیم و مطمئن باشید که این توضیحات در نحوهٔ پیاده‌سازی شیءها و صفحات Active Servere مؤثر خواهد بود.

شیء Response

Responce یک شیء داخلی برای ASP می‌باشد، به این معنی که تنها توسط متدها و سایر اشیاء ASP قابل دسترسی است. از این شیء برای نمایش خروجی‌های برنامه به صورت داینامیک (مقادیر خروجی برای هر کاربر می‌تواند متفاوت باشد) استفاده می‌شود. این شیء نیز همانند سایر اشیاء دارای خصوصیات و متدهای خاص خود است که یکی از متداول‌ترین آنها، متد Write می‌باشد. این متد



شکل ۴-۱۷ ویراستار کد دلفی

دارای یک آرگومان از نوع OleVariant است که معرفی برای خروجی از نوع داینامیک می‌باشد. (شکل ۴-۱۷ را ملاحظه کنید).

اکنون کد ارائه شده در لیست ۳-۱۷ را برای متد TDrBob42.Welcome وارد نمائید.

لیست ۳-۱۷ پیاده‌سازی متد Welcome

```
procedure TDrBob42.Welcome;
begin
  Response.Write('Hello, Visitor!');
  Response.Write('<P>');
  Response.Write('Welcome to Delphi 6 and ASP Objects');
end;
```

از آنجا که نام متد (Welcome) برایتان مشخص شده است لذا لازم است تا فایل DrBob42.ASP را به صورت زیر اصلاح نمائید.

```
<% Set DelphiASPObj = Server.CreateObject("D6ASP.DrBob42")
   DelphiASPObj.Welcome
%>
```

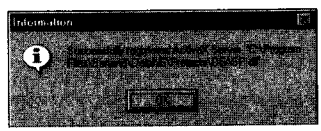
اولین اجرا

به خاطر دارید که فایل پروژه مثال ذکر شده را به عنوان D6ASP.dpr ذخیره کرده‌اید. اکنون تنها کاری که باید انجام دهید ثبت D6ASP.dll است (این کار به منزلهٔ ثبت شیء Active Server می‌باشد). دو مدل

برای ثبت اشیاء ASP وجود دارد که شما را با مفهوم این دو مدل (in-process و out-of-process) آشنا می‌سازیم. in-process به معنی بارگذاری و اجرای اشیاء ASP در سرویس‌دهنده وب (سرور وب) می‌باشد. در این صورت با shutdown کردن سرور وب می‌توان به عملیات اشیاء خاتمه داد.

در مقابل آن out-of-process به مفهوم بارگذاری و اجرای اشیاء ASP در هنگام ارسال درخواست از یک سرویس‌گیرنده است. شیءهایی که به صورت in-process تعریف می‌شوند، قابلیت اجرائی بهتری دارند و در مقابل، عملیات خطایابی آنها نسبت به شیءهای out-of-process مشکل‌تر است.

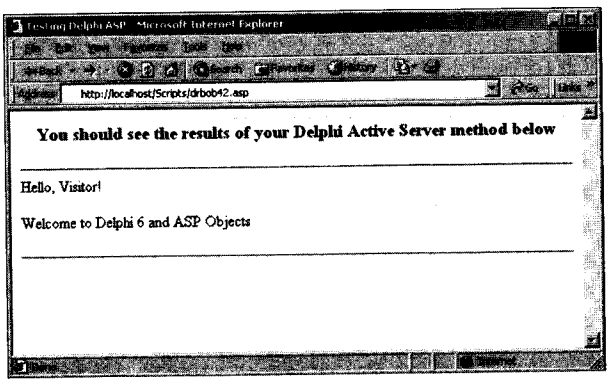
D6ASP.dll را با مدل in-process ثبت کنید. برای این کار گزینهٔ Register ActiveX Server را از منوی Run دلفی انتخاب نمائید (برای لغو عمل ثبت لازم است تا گزینهٔ Unregister Active Server را از همان منو انتخاب کنید). شکل ۵-۱۷ پیغام نمایش داده شده پس از ثبت D6ASP.dll را نشان می‌دهد.



شکل ۵-۱۷
ثبت ActiveX Server

پس از انجام عملیات ثبت، می‌بایست فایل DrBob42.asp را به فهرست اسکریپت‌های ASP انتقال دهید (WWWRoot/Scripts) . اکنون با فراخوانی URL زیر، شکلی همانند شکل ۶-۱۷ به نمایش درخواهد آمد.

http:/localhost/Scripts/DrBo42.asp



شکل ۶-۱۷ اجرای یک شیء Active Server

شیء Request

Request شیءای است که از آن جهت دریافت ورودی‌ها در صفحات Active Server استفاده می‌شود.

چنانچه ورودی‌ها مربوط به یک فرم باشند، از متد POST مربوط به این شیء برای دریافت ورودی‌ها استفاده می‌شود. برای متغیرهای URL و Cookie ها هم از متد GET استفاده خواهد شد. تمامی این متدها دارای یک خصوصیت به نام Items بوده که متغیر ورودی را در خود نگه می‌دارند. برای مثال، متد Welcome را به صورت لیست ۴-۱۷ تغییر داده‌ایم.

لیست ۴-۱۷ متد تغیر یافته Welcome

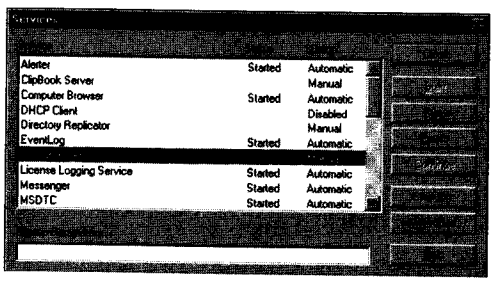
```
procedure TDrBob42.Welcome;
var
  Str: String;
begin
  Str := Request.Form.Item['Name'];
  Response.Write('Hello, '+Str+'!');
  Response.Write('<P>');
  Response.Write('Welcome to Delphi 6 and ASP Objects');
end;
```

کامپایل مجدد شیء‌های Active Server

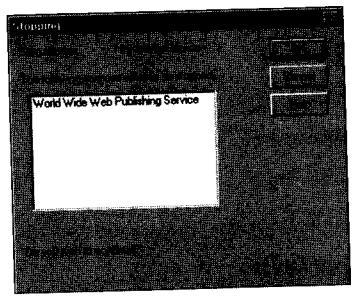
حال اگر فایل پروژهٔ D6ASP را کامپایل کنید، پیغام خطای Could not create output file D6ASP.dll برای شما نمایش داده می‌شود. این خطا به خاطر فعال بودن شیء DrBob42 در D6ASP.dll می‌باشد. برای رفع این خطا می‌بایست IIS Admin Service را shutdown کنید. (با این کار تمامی سرویسهای WWW ، FTP و ... Shutdown خواهند شد). شکل ۷-۱۷ جعبهٔ مکالمهٔ مربوط به این عملیات را نشان می‌دهد.

چنانچه عملیات Shutdown را از طریق این جعبهٔ مکالمه انجام دهید، یک جعبهٔ مکالمهٔ دیگر با عنوان Stopping برای شما نمایش داده می‌شود (شکل ۸-۱۷). این جعبهٔ مکالمه، سرویسهای مرتبط با IIS Admin Server را نمایش می‌دهد. باید توجه داشته باشید که در صورت Shutdown نمودن IIS ، سرویس‌های نمایش داده شده در این جعبهٔ مکالمه نیز Shutdown خواهند شد.

عملیات (Shutdown و Startup) از طریق سرویسهای موجود در ویندوز انجام می‌شود. علاوه بر



شکل ۷-۱۷ IIS Admin Service



شکل ۸-۱۷ جعبهٔ مکالمهٔ مربوط به Shutdown کردن Word Wide Web Publishing Service

روش ذکر شده برای Shutdown نمودن IIS و Word Wide Web Publishing Service ، روش دیگری نیز برای این کار وجود دارد. در این روش با نوشتن یک Batch فایل عملیات Shutdown و Startup را انجام می‌دهیم. محتویات این batch فایل می‌تواند به صورت زیر باشد.

```
net stop "World Wide Web Publishing Service"
net stop "IIS Admin Service"
net start "World Wide Web Publishing Service"
```

اجرای صفحات Active Server

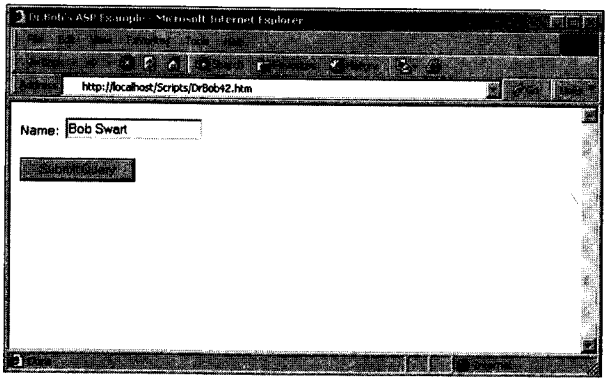
اکنون می‌خواهیم طریقهٔ اجرای یک صفحه از نوع Active Server را بررسی نمائیم. برای تشریح این عملیات از مثال قبل یعنی DrBob42 استفاده می‌کنیم.

ابتدا یک صفحه HTML برابر با محتویات زیر ایجاد کرده و سپس آن را در یک محل دلخواه بارگذاری نمائید. به عنوان مثال **(http:/localhosts/cgi-bin/drbob42.htm)** .

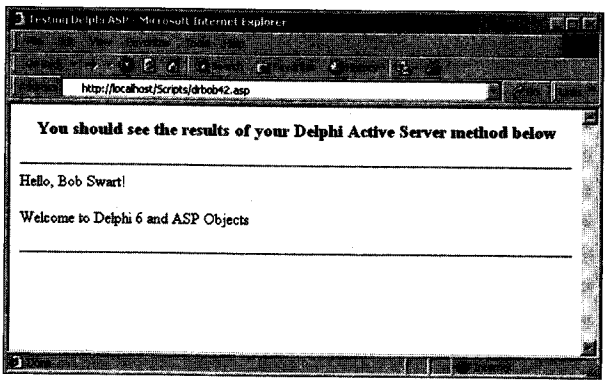
```
<HTML>
<HEAD>
<TITLE>Dr.Bob's ASP Example</TITLE>
</HEAD>
<BODY BGCOLOR=FFFFCC>
<FONT FACE="Verdana"SIZE=2>
<FORM ACTION="drbob42.asp" METHOD=POST>
Name: <INPUT TYPE=text NAME=Name>
<P>
<INPUT TYPE=submit>
</FORM>
</BODY>
</HTML>
```

اجرای این HTML ، چیزی مشابه با شکل ۹-۱۷ خواهد بود.

اما صفحهٔ Active Server تنها زمانی بارگذاری خواهد شد که شما یک نام را در بخش ویرایش Nameوارد کرده و دکمهٔ Submit Query را بفشارید. این صفحه در شکل ۱۰-۱۷ نشان داده شده است.



شکل ۹-۱۷ فراخوانی DrBob42.htm به وسیله مرورگر Internet Explorer



شکل ۱۰-۱۷ اجرای DrBob42.asp بر روی مرورگر Internet Explorer

شیءهای Application, Session و Server

تا اینجا شیءهای Response و Requset در ASP آشنا شده‌اید. ASP علاوه بر این دو شیء به اشیاء دیگری نظیر Session، Application و Server نیز دسترسی دارد. تمام این اشیاء معرف امتیازات ASP نسبت به CGIها و ISAPIها می‌باشند. این سه شیء همگی به عنوان زیرمجموعه‌ای از کلاس TASPObject مطرح بوده و همانند Response و Request فراخوانی می‌شوند (به طور مثال TASPObject.Session خصوصیتی است که از آن برای دسترسی به Session یک ASP استفاده می‌شود). اکنون با ارائهٔ یک مثال روش استفاده از این اشیاء را شرح می‌دهیم. فرض کنید می‌خواهیم نام کاربرانی که از صفحهٔ ASP مثال DrBob42 استفاده کرده‌اند را ذخیره و نگهداری نمائیم. این امر با بکارگیری شیء Session انجام‌پذیر می‌باشد. در این حالت لازم است تا ASP ذکر شده را در کد HTML

به صورت زیر اصلاح کنید.

```
<% Set DelphiASPObj = Server.CreateObject("D6ASP.DrBob42")
Session.Value("Name") = "Bob Swart"
DelphiASPObj.Welcome
%>
```

همچنین متد TDrBob42.Welcome را مطابق با لیست ۵-۱۷ اصلاح می‌کنیم.

لیست ۵-۱۷ متد جدید TDrBob42.Welcome

```
procedure TDrBob42.Welcome;
var
  Str: String;
begin
  Str := Session.Value['Name'];
  Response.Write('Hello, '+Str+'!');
  Response.Write('<P>');
  Response.Write('Welcome back to Delphi 6 and ASP Objects');
end;
```

شی‌ءهای Active Server و بانک‌های اطلاعاتی

در این قسمت، مثال کاربردی‌تری را ارائه می‌نمائیم. در این مثال به نحوهٔ کار با بانک‌های اطلاعاتی از طریق صفحات Active Server خواهیم پرداخت. بدون هیچ توضیح اضافی، مثال خود را مطرح می‌نمائیم.

در محیط دلفی، برنامهٔ مثال قبل (DrBob42) را فراخوانی کنید. سپس به منوی File رفته، گزینهٔ New و بعد از آن گزینهٔ Data Module را انتخاب کنید و از این طریق یک data module را به برنامهٔ خود بیفزایید.

اکنون مقدار DataModuleASP را به خصوصیت Name این data module اختصاص داده و یونیت جدید را تحت عنوان DataMod.pas ذخیره کنید. با انتخاب گزینهٔ Save All از منوی File، تغییرات ایجاد شده در برنامه (D6ASP.dpr) را ذخیره نمائید. در محیط IDE دلفی، بر روی تب Data Access بروید و یک جزء سازندهٔ TClientDataSet را به data module اضافه کنید. توجه داشته باشید که از این جزء سازنده برای دسترسی به مجموعه‌های داده‌ای استفاده می‌شود.

اکنون یک جدول بانک اطلاعاتی را به خصوصیت FileName جزء سازندهٔ TClientDataSet اختصاص دهید. برای این کار دکمهٔ ماوس را بر روی جزء سازندهٔ TClientDataSet فشرده و در همین حالت دکمهٔ ellipsis را بفشارید. در فهرست c:\Program Files\Common Files\Borland Shared\Data (در صورتی که دلفی ۷ را در فهرست Program Files از درایو C نصب کرده باشید) چندین جدول

وجود دارد که ما برای این مثال از جدول biolife.xml استفاده می‌کنیم.

با قرار دادن data module در برنامه، اعمال مدیریت و به اشتراک گذاشتن جداول بانک‌های اطلاعاتی بخصوص در سیستم‌هایی که از چندین thread استفاده می‌کنند، کمی مشکل به نظر می‌رسد. راه مطمئن‌تر این است که data module را در یک شیء Active Server ایجاد کنید (به طور مثال در رویداد BeginPage یا EndPage مربوط به یک شیء Active Server ، data module را فراخوانی کنید).

اکنون متد Welcome را مطابق با لیست ۶-۱۷ پیاده‌سازی نمائید.

لیست ۶-۱۷ شکل دیگر متد Welcome

```
procedure TDrBob42.Welcome;
var
  Str: String;
  DM: TDataModuleASP;
begin
  Str := Request.Form.Item['Name'];
  Response.Write('Hello, '+Str+'!');
  Response.Write('<P>');
  Response.Write('Welcome to Delphi 6 and ASP Objects');
  try
    DM := TDataModuleASP.Create(nil);
    // use DM...
  finally
    DM.Free
  end
end;
```

اکنون برای نمایش فیلدهای این جدول (biolife.xml) لازم است تا یک قطعه کد، HTML را مطابق لیست ۷-۱۷ به متد Welcome اضافه کنید.

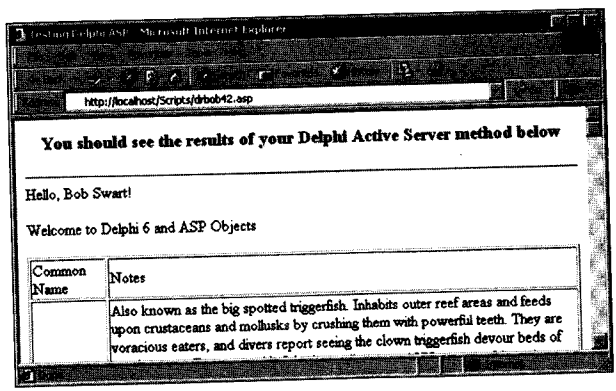
لیست ۷-۱۷ شکل دیگری از متد Welcome

```
procedure TDrBob42.Welcome;
var
  Str: String;
  DM: TDataModuleASP;
begin
  Str := Request.Form.Item['Name'];
  Response.Write('Hello, '+Str+'!');
  Response.Write('<P>');
  Response.Write('Welcome to Delphi 6 and ASP Objects');
```

لیست ۷-۱۷ ادامه

```
try
  Response.Write('<P>');
  DM := TDataModuleASP.Create(nil);
  with DM.ClientDataSet1 do
  try
    Open;
    First;
    Response.Write('<TABLE BORDER=1><TR><TD>Common_Name</TD>');
    Response.Write('<TD>Notes</TD></TR>');
      while not Eof do
      begin
        Response.Write('<TR><TD>');
        Response.Write(FieldByName('Common_Name').AsString);
        Response.Write('</TD><TD>');
        Response.Write(FieldByName('Notes').AsString);
        Response.Write('</TD></TR>');
        Next
      end;
      Close;
    finally
      Response.Write('</TABLE>')
    end;
  finally
    DM.Free
  end
end;
```

با اجرای برنامه، خروجی همانند شکل ۱۱-۱۷ را مشاهده خواهید کرد.



شکل ۱۱-۱۷ یک صفحهٔ HTML داینامیک که آن را توسط یک شیء Active Server ایجاد نموده‌اید

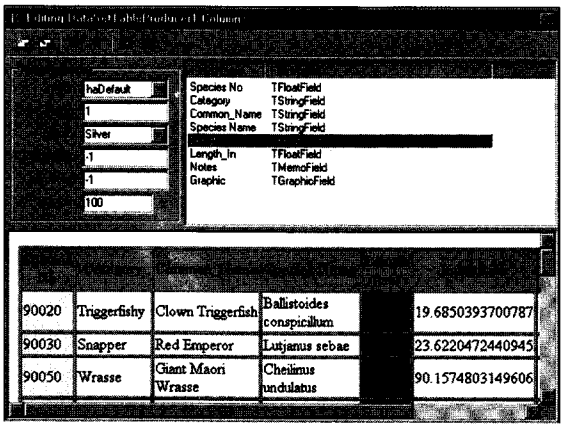
شیءهای Active Server و پشتیبانی از NetCLX

چنانچه تابحال شیءهای Active Server را با ساختار NetCLX مقایسه کرده باشید، شباهتهای زیادی را بین آن دو یافته‌اید. هر دوی آنها از شیءهای Response و Request برای برقراری ارتباط با سرویس‌گیرنده استفاده می‌کنند. NetCX توسط اجزاء سازندهٔ PageProducer و TableProducer پشتیبانی می‌شود.

از آنجاکه در فصل‌های میانی این کتاب توضیحاتی را در خصوص NetCLX ارائه داده‌ایم، لذا تنها به ارائه مثالی از آن اکتفا خواهیم کرد.

مثالمان همان DrBob42 خواهد بود. در اینجا یک جزء سازندهٔ TDataSetTableProducer را به data module اضافه نموده و خصوصیت DataSet آن را با مقدار ClientDataSet تنظیم نمائید (از جزء سازندهٔ ClientDataSet در مثال قبل استفاده کرده‌اید).

اکنون مقدار True را به خصوصیت Active از این جزء سازنده اختصاص دهید. سپس دکمهٔ سمت راست ماوس را بر روی جزء سازندهٔ DataSetTableProducer فشرده و گزینهٔ Response Editor را انتخاب نمائید. با این کار ویراستاری مشابه شکل ۱۲-۱۷ به نمایش در می‌آید (این ویراستار مربوط به خصوصیت Columns از DataSetTableProducer می‌باشد).



شکل ۱۲-۱۷ Response Editor مرتبط با جزء سازندهٔ DataSetTableProducer

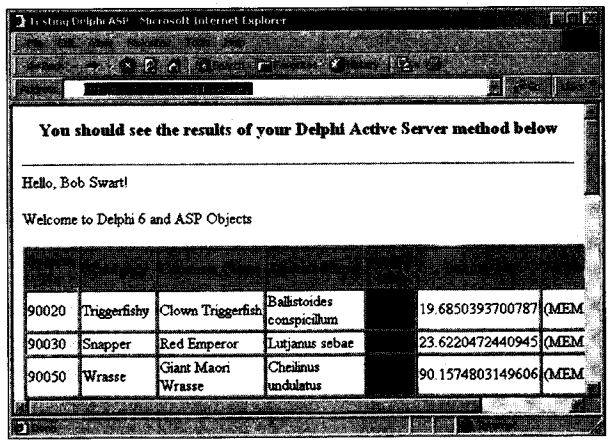
اکنون به متد Welcome باز گردید و آن را به صورت لیست ۸-۱۷ اصلاح نمائید.

پس از کامپایل مجدد این برنامه و انجام سایر عملیات لازم برابر با مثالهای قبل، خروجی مشابه با شکل ۱۳-۱۷ را مشاهده خواهید کرد.

در اینجا توصیه می‌کنیم که تمامی مثالهای قبل را یک بار دیگر اجرا کرده و این بار نحوهٔ کارآیی آنها را با هم مقایسه کنید.

لیست ۸-۱۷ شکل اصلاح شده متد Welcome

```
procedure TDrBob42.Welcome;
var
  Str: String;
  DM: TDataModuleASP;
begin
  Str := Request.Form.Item['Name'];
  Response.Write('Hello, '+Str+'!');
  Response.Write('<P>');
  Response.Write('Welcome to Delphi 6 and ASP Objects');
  try
    Response.Write('<P>');
    DM := TDataModuleASP.Create(nil);
    Response.Write(DM.DataSetTableProducer1.Content);
  finally
    DM.Free
  end
end;
```



شکل ۱۳-۱۷ خروجی NetCLX فایل DrBob42.asp با استفاده از مرورگر Internet Explorer

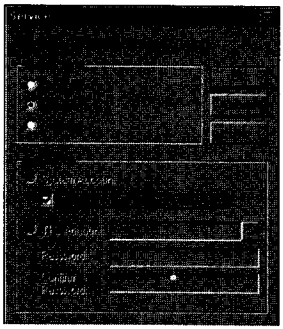
خطایابی شی‌های Active Server

همانطورکه در قسمتهای پیش ملاحظه کردید، برای غیرفعال نمودن شیءهای Active Server لازم بود تا خود سرویس‌دهندهٔ وب را shutdown نمائید (به دلیل بارگذاری این اشیاء در یک ASP.DLL که مشابه با DLL های ISAPI عمل می‌کنند). یعنی با ایجاد هر تغییری (نظیر اضافه کردن یک اسکریپت) در شیءهای Active Server برنامه‌تان، باید عمل shutdowm سرور و خود اشیاء را هم انجام دهید. اما

آیا این کار منطقی به نظر می‌رسد؟ مسلماً جواب شما خیر خواهد بود.

در دلفی ابزاری برای خطایابی شیءها و صفحات Active Server وجود دارد که عملیات شبیه‌سازی (اعمالی از قبیل نمایش پیغام‌های خطا، نمایش کاراکترهای ارسالی بین اشیاء و ...) این اشیاء و صفحات را نیز انجام می‌دهند.

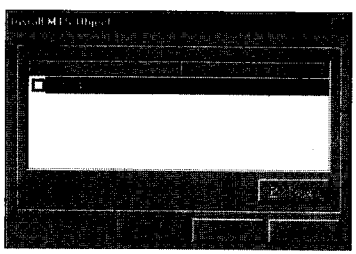
در این قسمت قصد داریم شما را با نحوهٔ انجام این کار آشنا سازیم. یکی از این روشها کنترل پیام‌ها در محیط Desktop دلفی است. جعبهٔ مکالمهٔ نمایش داده شده در شکل ۱۴-۱۷ را ملاحظه کنید، در این جعبهٔ مکالمه با انتخاب گزینهٔ Allow Service to intreract with Desktop عملیات کنترل پیام‌های اشیاء Active Server به صورت محاوره‌ای انجام می‌پذیرد. (سرویس از نوع ISS Admin Service در نظر گرفته شده است).



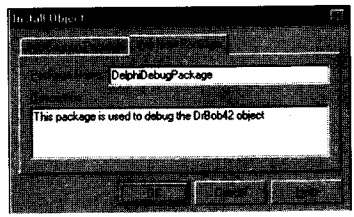
شکل ۱۴-۱۷ محاوره‌ای
نمودن ارسال پیام‌ها

خطایابی شیءهای Active Server به وسیلهٔ MTS

MTS روش آسان‌تری را برای مدیریت و خطایابی شیءهای Active Server ارائه می‌کند. اولین قدم انتخاب گزینهٔ Unregister ActiveX Server از منوی Run می‌باشد. این کار باعث خارج شدن شیء Active Server از حالت ثبت می‌شود. بعد از این مرحله لازم است تا دوباره شیء موردنظر را به عنوان یک شیء MTS ثبت نمائید. برای این کار گزینهٔ Install MTS Objects را از منوی Run انتخاب کنید. (ممکن است به جای این گزینه، گزینهٔ Install COM+ Object وجود داشته باشد). چنانچه این مراحل را برای مثال مطرح شده در این فصل (DrBob42) اجراء نمائید، جعبهٔ مکالمه‌ای همانند شکل ۱۵-۱۷ به نمایش در خواهد آمد.



شکل ۱۵-۱۷ جعبهٔ مکالمهٔ
Install MTS (COM+) Objects



شکل ۱۶-۱۷ نصب شیء در یک Package

در این جعبه مکالمه شیء DrBob42 را برگزینید. در این حالت یک جعبهٔ مکالمهٔ دیگر نظیر شکل ۱۶-۱۷ ظاهر می‌شود. در این جعبهٔ مکالمه گزینه‌ای برای وارد کردن نام Package وجود دارد. این نام معرف Package ای می‌باشد که قرار است شیء DrBob42 در آن نصب شود. چنانچه تمایلی به استفاده از نام پیش‌فرض این قسمت ندارید می‌توانید نام دیگری را برای این Package انتخاب کنید. به عنوان مثال نام DelphiDebugPackage را برگزینید.

سپس دکمهٔ OK را در این جعبهٔ مکالمه و جعبهٔ مکالمهٔ Install MTS Objects بفشارید. اینک شیء DrBob42 را به عنوان یک شیء MTS نصب کرده‌اید و خواهید توانست عملیات خطایابی و کنترل آن را از طریق سرویسهای ویندوز NT یا ۲۰۰۰ انجام دهید.

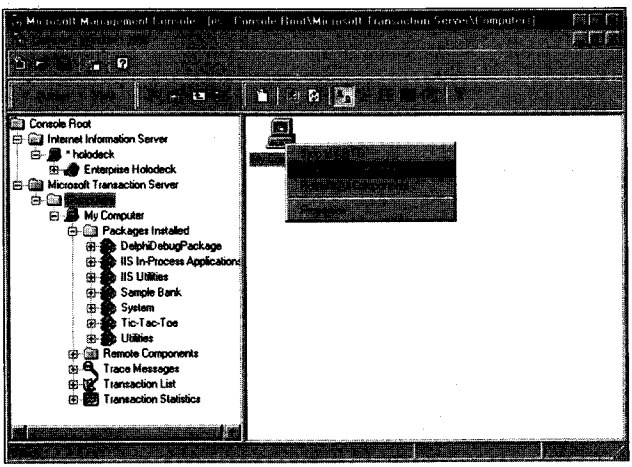
استفاده از ویندوز NT برای عملیات خطایابی

توجه داشته باشید که MTS در ویندوز NT به عنوان یک برنامهٔ میزبان مطرح است از آنجا که MTS در حالت عادی به صورت فعال می‌باشد، لازم است تا ابتدا آن را shutdown نمائید. برای انجام این کار در ویندوز NT از برنامهٔ Internet Service Manager استفاده می‌شود (Internet Service Manager یکی از برنامه‌های موجود در ویندوز NT می‌باشد). با فراخوانی این برنامه پنجره‌ای همانند شکل ۱۷-۱۷ برای شما نمایش داده می‌شود. از قسمت سمت چپ این پنجره به ترتیب گزینه‌های Microsoft Transaction Server و Computers را انتخاب کنید. با این کار نشانهٔ My Computer در قسمت سمت راست این پنجره ظاهر می‌شود. اکنون برای shutdown نمودن تمامی فرآیندهای موجود بر روی سرور، می‌بایست دکمهٔ سمت راست ماوس را بر روی My Computer فشار داده و گزینهٔ Shutdown All Server Process را انتخاب کنید (شکل ۱۷-۱۷).

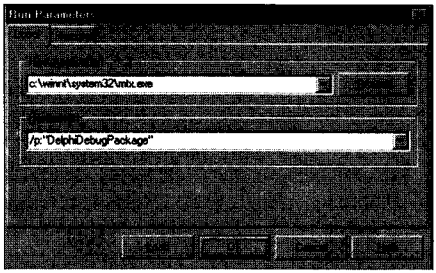
پس از انجام این کار، لیستی از فرآیندهای موجود بر روی سرور بر روی صفحه ظاهر می‌شود که حذف آنها از این لیست منوط به گرفتن تأیید از شما می‌باشد.

در مرحلهٔ بعدی باید MTS را به عنوان یک برنامهٔ میزبان تعریف کنید. این کار به وسیلهٔ جعبهٔ مکالمهٔ Run Parameters (شکل ۱۸-۱۷) انجام می‌گیرد. در این جعبهٔ مکالمه می‌بایست مسیر برنامه بر روی سرور (به عنوان مثال فایل mtx.exe که در فهرست **C:\winnt\system32** قرار داده شده است) و نام package (به عنوان مثال **"/P:"DelphiDebugPackage**) را مشخص نمائید.

اکنون با قرار دادن یک breakpoint در برنامه‌تان قادر خواهید بود تا آن را با فشردن کلیدهای F5 و



شکل ۱۷-۱۷ Shutdown نمودن فرآیندهای موجود بر روی سرور

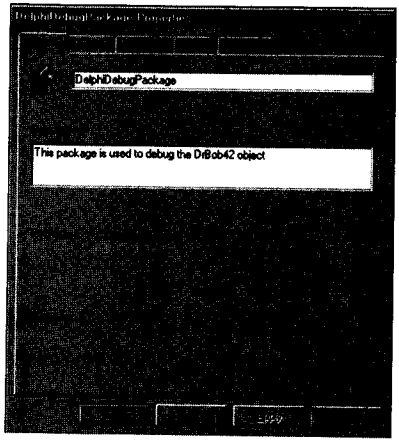


شکل ۱۸-۱۷ جعبهٔ مکالمهٔ Run Parameters

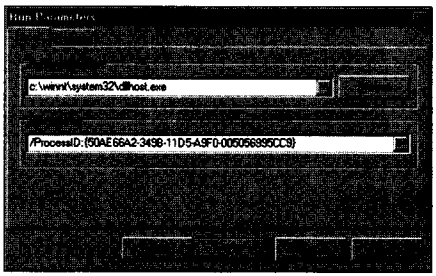
F9 ، اجرا و خطایابی نمائید. البته قبل از انجام این کار لازم است تا فایل asp خود را (برای مثال DrBob42.asp) از طریق مرورگری همانند Interner Explorer بارگذاری کنید.

استفاده از ویندوز ۲۰۰۰ برای عملیات خطایابی

نحوهٔ پیاده‌سازی MTS در ویندوز ۲۰۰۰ متفاوت با ویندوز NT است. MTS در ویندوز ۲۰۰۰ به عنوان بخشی از سیستم عامل انجام وظیفه می‌نماید. شما می‌توانید با استفاده از dllhost.exe ، شماره شناسایی یک فرآیند Active Server را بارگذاری کنید (توجه داشته باشید که این برنامه باید به عنوان یک برنامهٔ میزبان معرفی شود). این بار به جای درج نام package در قسمت Parmeters جعبهٔ مکالمهٔ Run Parameters ، تنها شمارهٔ شناسایی فرآیند Active Server وارد می‌گردد (شکل ۲۰-۱۷). این شمارهٔ شناسایی بر روی جعبهٔ مکالمهٔ Run Properties مشخص می‌شود. شکل ۱۹-۱۷ این جعبهٔ مکالمه را برای برنامهٔ DrBob42 نشان می‌دهد. همانطور که ملاحظه می‌کنید شمارهٔ شناسایی DelphDebugPackage برابر با {50AE66A2-349B-11D5-A9F0-005056995CC9} اعلام شده است.



شکل ۱۹-۱۷ شمارهٔ شناسایی
DelphiDebugPackage



شکل ۲۰-۱۷ جعبهٔ مکالمهٔ
Run Parameter برای این مثال

بقیهٔ مراحل مشابه با مراحل ذکر شده در قسمت ویندوز NT می‌باشد.

خلاصه

در این فصل با مفاهیم صفات Active Server و نحوهٔ عملکرد آنها آشنا شدید. امروزه پیاده‌سازی و اجرای این صفحات به عنوان یک مهارت برای برنامه‌نویسان وب مطرح می‌باشد. با ارائهٔ مثالهای متنوع در این فصل، سعی نمودیم تا شما را در زمرهٔ برنامه‌نویسان حرفه‌ای وب قرار دهیم.

فصل هجدهم

استفاده از WebSnap در طراحی سرویس‌دهنده‌های وب

در این فصل می‌خوانید

- **ویژگیهای WebSnap**
- **طراحی و ساخت برنامه‌های کاربردی WebSnap**
- **موضوعات پیشرفته در طراحی سرویس‌دهنده‌های وب**

در این فصل تکنیکهای لازم جهت آماده‌سازی برنامه‌های کاربردی سرویس‌دهندهٔ وب را خواهید آموخت. دلفی ۷ از ابزار جدیدی به نام WebSnap برای طراحی آسان‌تر این برنامه‌ها استفاده می‌کند. WebSnap یک ابزار *RAD*[1] می‌باشد یعنی روشهای به کار برده شده در آن بسیار قابل درک‌تر از سایر تکنیک‌هاست و شما به آسانی فشردن چندین مرتبه دکمه ماوس قادرید تا برنامه خود را طرح‌ریزی نمائید.

اگر تابحال با WebBroker دلفی کار کرده‌اید، WebSnap را ابزاری مناسب‌تر از WebBroker خواهید یافت. WebSnap مجموعه‌ای از ابزارهایی است که شما را در طراحی برنامه‌های کاربردی وب نظیر مدیریت جلسات، طراحی و ساخت اسکریپت‌ها، بانکهای اطلاعاتی تحت وب، صفحات پویا و ... یاری می‌رساند.

ویژگیهای WebSnap

WebSnap ابزاری متفاوت با WebBroker نیست بلکه قابلیتهای آن در قیاس با آن از انعطاف و کارآیی بالاتری برخوردار است. WebSnap ابزاری مشابه InternetExpress و WebBroker در دلفی است که امکانات ویژه‌ای در آن گنجانده شده است. در ادامه، اشاره مختصری به این امکانات خواهیم داشت.

۱- Rapid Application Development

امکان استفاده از چندین ماژول وب

در گونه‌های قبلی دلفی نظیر دلفی ۵، برای تولید برنامهٔ کاربردی مبتنی بر وب از یک ماژول وب استفاده می‌گردید. WebSnap ابزاری است که به کارگیری چندین ماژول وب در آن امکان‌پذیر بوده و شما را قادر می‌سازد تا چندین ماژول داده‌ای را نیز به برنامه کاربردی خود الحاق نمائید. این مزیت شما را قادر می‌سازد تا بدون تغییر کد قسمتهای مختلف برنامه‌تان، هر قسمت از آن را بنا بر ضروریات مورد نظرتان طراحی و به کار ببرید. هنگامی که ایجاد یک برنامه کاربردی وب را شروع کردید، محاسن این امر وضوح بیشتری خواهد یافت.

امکان طراحی اسکریپت

WebSnap شما را در خلق اسکریپت‌های کارآمد کمک خواهد کرد. با استفاده از این ابزار قادرید اسکریپت‌های پرقدرتی برای کدهای HTML و برنامه‌های اینترنتی خود طراحی کنید. این اسکریپت‌ها هم می‌توانند بر روی Client اجرا شوند و هم می‌توانند بر روی سرور اعمال خود را به اتمام برسانند.

اجزاء سازندهٔ TAdapter

این اجزاء سازنده رابطی بین برنامه‌های کاربردی شما با اسکریپت‌های Server-side[1] می‌باشند. اسکریپت‌ها تنها از طریق این اجزاء با برنامهٔ کاربردی شما ارتباط برقرار می‌کنند و همین امر ضریب اطمینان برنامه‌هایتان را بالا می‌برد. شما می‌توانید اجزاء سازندهٔ دیگری بنا به ضرورت کارتان تحت عنوان اجزاء سازندهٔ TAdapter ایجاد نمائید. TAdapter در برگیرندهٔ اجزاء سازنده‌ای جهت نگهداری داده‌ها و اجرای برنامه‌های کاربردی است. این بدین معنی است که اجزاء موجود در TAdapter قابلیت اجرائی بالایی دارند و کمتر نیاز به طراحی اجزاء سازندهٔ دیگری خواهید داشت. به عنوان مثال تنها با افزودن یک جزء سازنده TDataSetAdapter، عملیات مربوط به درج، حذف، ویرایش و نمایش یک مجموعه از داده‌ها و یا یک بانک اطلاعاتی امکان‌پذیر خواهد بود.

استفاده از روشهای جدید در ارسال سریع داده‌ها

مدیریت و پاسخدهی به درخواستهای HTTP در WebSnap به طرق متنوعی قابل انجام است. روشهای مختلف دسترسی و تنوع در نحوهٔ ورود به صفحات وب امکانی است که در WebSnap دلفی به عنوان یک ویژگی منحصر به فرد گنجانده شده است.

اجزاء سازندهٔ صفحات

WebBroker از جزء سازنده‌ای به نام TPageProducer برای طراحی و مدیریت صفحات HTML

۱- این اسکریپت‌ها بر روی سرور اجرا می‌گردد.

استفاده می‌کرد. در InternetExpress این جزء با اندکی تغییرات و با نام TMidasPageProducer ارائه گردید. در WebSnap این تغییرات به گونه‌ای محسوس‌تر و پویاتر اعمال شده است به گونه‌ای که حتی پردازش صفحات در قالب کدهای **XSL/XML** نیز امکان‌پذیر است. این جزء سازنده در WebSnap تحت عنوان TAdapterPageProducer نام‌گذاری شده است.

مدیریت جلسات[۱]

WebSnap از تکنیک مدیریت جلسات به صورت خودکار استفاده می‌کند. مدیریت جلسات به معنی کنترل عملیات کاربران و نحوهٔ پاسخ‌دهی به درخواستهای موجود در شبکه است معمولاً این عملیات در پروتکل **HTTP** توسط **URL** و تعداد فایل‌های کنترلی مخفی انجام می‌پذیرد. مدیریت جلسات در WebSnap توسط جزء سازندهٔ SessionService انجام می‌گیرد. اگر تابحال برنامه‌ای را برای مدیریت جلسات در شبکه طراحی کرده‌اید، حتماً به خاطر دارید که چقدر در نحوهٔ مدیریت و پردازش اعمال کاربران شما را دچار زحمت نموده است. بدون اینکه زیاده‌روی کرده باشیم، خودتان خواهید دید که WebSnap این کار را به چه سادگی انجام خواهد داد، ضمن این که ساختار داخلی آن به گونه‌ای است که هم امکان کنترل اسکریپت‌های موجود بر روی سرور وجود دارد و هم کنترل برنامه‌های کاربردی مبتنی بر وب انجام‌پذیر می‌باشد.

امنیت برای ورود به شبکه

مهمترین کار در طراحی شبکه، برنامه‌های سرویس‌دهنده و همچنین صفحات وب، برقراری یک سیستم امنیتی برای ورود کاربران می‌باشد. این طراحی باید به گونه‌ای صورت پذیرد که ضمن کنترل عملیات ورود و خروج کاربران، از ورود کاربران غیرمجاز و یا مهاجمان جلوگیری به عمل آورد. در این فصل نحوهٔ انجام این کار را نیز خواهید آموخت.

کنترل کاربران

یکی از عملیات مهم در برنامه‌های سرویس‌دهنده کنترل اعمال کاربران است. WebSnap در برگیرندهٔ اجزائی است که انجام این کار را بسیار آسان می‌نمایاند.

مدیریت صفحات HTML

همانطور که می‌دانید در برنامه‌های سرویس‌دهندهٔ وب، پاسخهای کاربران به صورت یک سند **HTML** ارائه می‌شود. از آنجا که عموماً صفحات **HTML** حجم بالایی از اطلاعات را در بر می‌گیرند، نحوهٔ مدیریت آنها کار مشکلی به نظر می‌رسد. سختی این کار در صفحات **HTML** که به صورت پویا

۱- Session

(dynamic) تعریف شده‌اند محسوس‌تر خواهد بود. باور داشته باشید که WebSnap انعطاف‌پذیری بالایی برای انجام این کار دارد.

سرویسهای انتقال فایل

با قرار دادن یک جزء سازنده از اجزاء TAdapter می‌توانید قابلیت ارسال و دریافت فایل را از طریق برنامه‌های سرویس‌دهنده به برنامهٔ کاربردی خود بیفزائید و بدین ترتیب قدرت WebSnap را در سرویسهای انتقال فایل ملاحظه کنید.

طراحی و ساخت برنامه‌های کاربردی WebSnap

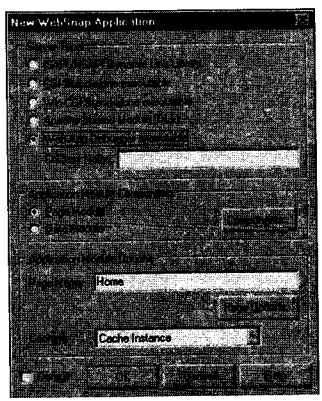
حتماً با برنامهٔ "Hello Word" آشنایی دارید. این برنامه یک مثال متداول و بسیار ساده برای نوشتن کدهای **HTML** و گاهاً اسکریپت‌هاست. در اینجا نیز قصد داریم نحوهٔ ایجاد این برنامه را به عنوان یک برنامهٔ کاربردی WebSnap شرح دهیم.

برای کار با WebSnap ابتدا می‌بایست میلهٔ ابزار آن را بر روی **IDE** محیط دلفی خود نصب کنید. برای این کار نشانگر ماوس را بر روی Speedbutton برده دکمهٔ سمت راست ماوس را فشار داده و عنوان Interner toolbar را انتخاب کنید (همانند شکل ۱-۱۸).

با این کار شما امکان استفاده از WebSnap را به **IDE** دلفی خود را اضافه کرده‌اید. با فشردن دکمهٔ ماوس بر روی Internet toolbar ، جعبهٔ مکالمه‌ای همانند شکل (۲-۱۸) بر روی صفحهٔ نمایش کامپیوتر شما نمایان خواهد شد.

این جعبهٔ مکالمه که به عنوان New WebSnap Application نمایش داده می‌شود، وظیفهٔ تنظیمات WebSnap را برای تهیهٔ برنامه‌های کاربردی به عهده دارد. (شکل ۲-۱۸).

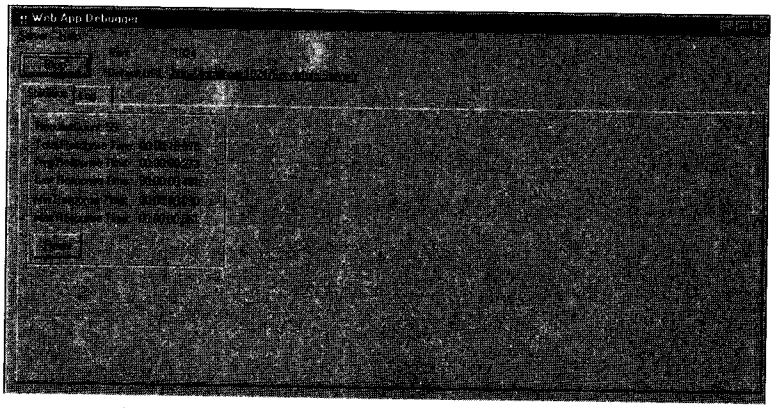
شکل ۱-۱۸ Internet toolbar



شکل ۲-۱۸ جعبهٔ مکالمهٔ New WebSnap Application

تعیین نوع سرور اولین چیزی است که در این قسمت می‌بایست تنظیم گردد. همانطورکه در شکل ۲-۱۸ می‌بینید ۵ گزینه برای تعیین نوع سرور وجود دارد که در زیر به شرح آنها خواهیم پرداخت.

- ISAPI/NSAPI Dynamic Link Library : از این گزینه جهت تولید برنامه‌های وبی استفاده می‌شود که قابلیت نصب بر روی سرورهای Netscape (IIS) را دارند. هنگام کامپایل اینگونه پروژه‌ها در دلفی، یک فایل **DLL** تولید خواهد شد که قابلیت اجرا بر روی سرورهای NetScape را داراست.
- CGI Standalone executable : از این گزینه برای تولید برنامه‌هایی وبی استفاده می‌شود که براساس استاندارد **CGI**[1] طراحی می‌شوند. امروزه اغلب سرویس‌دهنده‌های وب از اسکریپت‌های **CGI** پشتیبانی می‌کنند.
- Win-CGI Standalone executlble : این گزینه جهت تولید برنامه‌های **Win-CGI** کاربرد دارد. ارتباط این برنامه‌ها با برنامه‌های سرویس‌دهنده وب از طریق یک سری فایل‌های متنی که با عنوان **INI** شناسایی می‌شوند، برقرار می‌گردد. سرورهای **WinCGI** بسیار غیرمتداول بوده و کمتر به کار برده می‌شوند.
- Apache Shared Module (DLL) : از این گزینه برای تولید برنامه‌های سرویس‌دهنده تحت سرورهای Apache استفاده می‌شود. برای دریافت اطلاعات بیشتر در مورد این گونه سرورها به آدرس **www.apache.org** بر روی اینترنت مراجعه نمائید.
- Web APP Debugger executable : در صورت انتخاب این گزینه برنامهٔ کاربردی وب شما تحت ابزار Web App Debugger دلفی اجرا خواهد شد (شکل ۳-۱۸ را ملاحظه کنید). با انجام این کار،



شکل ۳-۱۸ Web App Debugger و تست یک برنامهٔ کاربردی وب

۱- CGI: Comman Gateway Interface استانداردی است که روش برقراری ارتباط بین سرویس‌دهنده وب و سایر برنامه‌ها را مشخص می‌سازد. CGI توانایی‌های زیادی به صفحات وب می‌بخشد و به طور مثال به کمک آن می‌توان بین صفحات بانک‌های اطلاعاتی، ارتباط برقرارکرد و یا صفحات وب را به گونه‌ای طراحی نمود که برای هرکاربر خاص ظاهر و محتوای متفاوتی داشته باشد.

Web App Debuggere کنترل و اجرای برنامهٔ کاربردی شما را به عهده خواهد گرفت. این عملیات برای تست و خطایابی برنامه‌تان کاربرد دارد. لازم به ذکر است که برای انجام این کار به مدت زمان زیادی نیاز نخواهید داشت و با صرف زمان اندکی می‌توانید برنامهٔ وب خود را تست نمائید.

یادداشت

از منوی Tools در محیط IDE دلفی می‌توانید به Web App Debugger دسترسی پیدا کنید. Web App Debugger یک برنامه بر پایهٔ COM است که با استفاده از آن می‌توانید محیطی همانند یک سرور شبیه‌سازی کرده و برنامهٔ خود را تحت آن به اجرا در آورید. وقتی گزینهٔ Web App Debugger را انتخاب می‌کنید، نمایانگر این خواهد بود که برنامهٔ شما حاوی یک فرم و یا ماژول وب است. Web App Debugger نیز با استفاده از یک مرورگر وب کار می‌کند. توجه داشته باشید که برنامه‌ای که از این طریق می‌سازید. تنها یک فایل اجرایی دلفی است و هیچ سنخیتی با فایل‌های سرویس‌دهندهٔ وب ندارد. اگر مایلید برنامهٔ شما از طریق یک مرورگر اینترنت نیز قابل دسترسی باشد ابتدا می‌بایست بر روی برچسب Default URL کلیک نمائید و بعد از آن برنامهٔ خود را برای اجرا انتخاب کنید. اکنون می‌توانید اطلاعات دقیق‌تری را در خصوص برنامه‌هایتان بر روی این فرم دریافت دارید. ضمناً می‌توانید برنامه‌هایی را که مدت زمان زیادی از آنها استفاده نشده است را از این لیست حذف کنید. به خاطر داشته باشید که اگر برنامهٔ ServerInfo را از این لیست حذف کنید، برای استفاده از آن می‌بایست دوباره آن را ثبت نمائید. این برنامه تحت عنوان ServerInfo.exe در فهرست Bin از محل نصب دلفی وجود داشته و تنها با اجرای آن ثبت می‌گردد. ▲

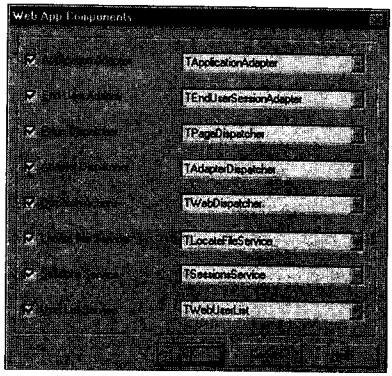
سعی کنید برای ساخت برنامه‌های کاربردی وب که در اینجا به آنها اشاره خواهیم کرد، گزینهٔ Web App Debugger را فعال سازید تا هنگام ساخت، امکان خطایابی همزمان آنها وجود داشته باشد.

همانطور که در شکل ۲-۱۸ ملاحظه می‌کنید بعد از عنوان تعیین نوع سرور، گزینه‌های مربوط به تعیین نوع ماژول وب و اجزاء مرتبط با آن قرار دارد. در صورتی که مایلید از یک صفحه در طراحی برنامهٔ وب خود بهره بگیرید می‌بایست گزینهٔ Page Module را فعال سازید. انتخاب گزینهٔ Data Module امکان به کارگیری ماژولهای داده‌ای را به برنامهٔ وب شما می‌افزاید. این ماژولها تماماً مشابه با ماژولهای داده‌ای موجود بر روی برنامه‌های سرویس‌دهنده/ سرویس‌گیرنده می‌باشند و کارآیی متناظر با آنها دارند.

اکنون گزینهٔ Page Module را انتخاب کرده و دکمهٔ Components را بفشارید. با فشردن آن جعبهٔ محاوره‌ای همانند شکل ۴-۱۸ نمایان خواهد شد.

بر روی این جعبهٔ محاوره‌ای ۸ گزینه وجود دارد که به تشریح آنها می‌پردازیم.

- Application Adapter: اگر قرار باشد یک اسکریپت بر روی سرور نصب و اجرا شود، از این گزینه برای مدیریت فیلدهای آن استفاده می‌شود.
- End User Adapter: از این گزینه جهت مدیریت اعمال کاربران استفاده می‌شود، کاربرانی که یک



شکل ۴-۱۸ این جعبهٔ محاوره‌ای امکان گنجاندن اجزائی نظیر آنچه که مشاهده می‌کنید را به برنامهٔ وب شما می‌افزاید

برنامه را فراخوانی و یا اجرا می‌کنند قاعدتاً با یک نام منحصر به فرد توسط سیستم شناسایی می‌شوند، چنانچه مایلید عملیات این کاربران را کنترل نموده و اطلاعاتی مرتبط با آنها داشته باشید، از این گزینه استفاده کنید.

- Page Dispatcher : مدیریت و پاسخ‌دهی به درخواستهای HTTP ای که توسط یک صفحه ارسال می‌گردد، با فعال کردن این گزینه انجام می‌گیرد. با این کار قادرید یک پیوند از نوع HREF ایجاد نمائید و به وسیله آن صفحات مورد نیاز خود و یا کاربران را فراخوانی نمائید.
- Adapter Dispatcher : این گزینه جهت مدیریت صفحات HTML کاربرد دارد.
- Dispatcher Actions : با انتخاب این گزینه یک شیء به نام TWebDispatcher به برنامهٔ شما اضافه می‌گردد. اگر قبلاً با WebBroker کار کرده باشید، حتماً این شیء را به خاطر خواهید آورد. این شیء وظیفه مدیریت **URL** ها را به عهده دارد.
- Locate File Service : چنانچه فایل HTML بر روی یک منبع خاص وجود دارد، بدین معنی که مسیر آن برای شما مشخص است، از این گزینه استفاده می‌شود.
- SessionsService : از این گزینه جهت فعال ساختن مدیریت جلسات برای کاربران استفاده می‌شود. با انجام این کار قادرید اطلاعات مربوط به عملیات کاربران در شبکه را ثبت نمائید و همچنین اگر کاربرانی وجود داشته باشند، که برای مدت زمان معینی هیچ فعالیتی در شبکه بروز ندهند، به صورت خودکار نسبت به حذف آنها توسط مدیریت جلسات اقدام خواهد شد. خصوصیت Session.Values نیز برای تعیین سرویسها و منابع (همانند cookie[1] یا **URL**) به کار برده می‌شود.

۱- cookie : بلوکی از داده‌ها که یک سرویس‌دهنده در پاسخ به درخواست یک سرویس‌گیرنده برای آن ارسال می‌کند. cookie برای شناسایی کاربران، ارائه اطلاعات Account برای کاربر و دیگر مقاصد مدیریتی مورد استفاده قرار می‌گیرد.

● Uer List Service : برای هر برنامه‌ای که بر روی سیستم‌های سرویس‌دهنده نصب می‌گردد، تعداد کاربر مجاز نیز تعریف می‌گردد و تنها این کاربران قادرند از این برنامه‌ها استفاده کنند. این گزینه جهت نگهداری لیست این کاربران کاربرد دارد.

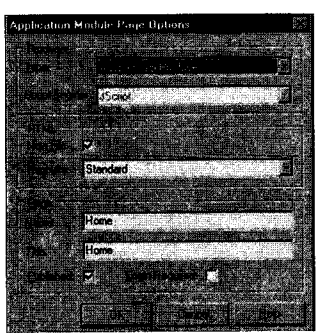
یادداشت

مقابل هر کدام از این گزینه‌ها یک جعبه drop–down وجود دارد که شما را قادر می‌سازد تا شیء موردنظر خود را انتخاب نمائید. اگر خودتان یک شیء را طراحی و به وسیلهٔ WebSnap آن را ثبت کرده باشید، خواهید دید که نام آن شیء نیز در این فهرست‌ها وجود دارد. ▲

برگردیم به مثالی که مطرح کرده بودیم یعنی همان "Hello World". اکنون تمام گزینه‌های مربوط به Web App Component را فعال نمائید و سپس گزینهٔ End User Adapter شیء TEndUserSession Adapter را انتخاب کنید. با این کار شما یک شمارهٔ شناسایی برای مدیریت جلسات کاربران خود معرفی کرده‌اید. قدم بعدی تخصیص نام به صفحهٔ موردنظر می‌باشد. عنوان Home برای نام صفحه در نظر گرفته شده است. با فشردن دکمهٔ ماوس بر روی Page Options کادر محاوره‌ای همانند شکل ۵–۱۸ نمایان می‌شود.

این کادر محاوره‌ای مربوط به جزء سازندهٔ PageProducer می‌باشد و شما در حین کار با این کادر محاوره‌ای، جزء سازندهٔ PageProducer را تنظیم خواهید نمود. این جزء سازنده متعلق به ماژول وب شماست که قطعاً با یک یا چندین صفحهٔ **HTML** ارتباط برقرار می‌کند.

تعیین نوع صفحه، اولین گزینه‌ای است که در این کادر محاوره‌ای ملاحظه می‌کنید. WebSnap از چندین نوع صفحه برابر با عناوین موجود در این گزینه پشتیبانی می‌کند (**XML** و **XSL** و پشتیبانی از صفحات **VBScript** و **JScript** نیز در WebSnap گنجانده شده است). برای مثالی که مطرح کرده‌ایم،



شکل ۵–۱۸ کادر محاوره‌ای Application Module Page Options شما را قادر می‌سازد تا صفحهٔ مربوط به ماژول وب خود را تنظیم نمائید

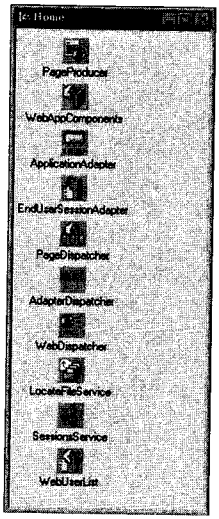
همان گزینهٔ پیش‌فرض موجود در این گزینه مناسب است. با گذر از این قسمت و بدون این که به گزینهٔ Script Engine توجهی نشان دهید مدل Standard را برای گزینهٔ Template انتخاب نمائید. همانطور که می‌دانید این کار برای تعیین نوع صفحهٔ **HTML** ضرورت دارد البته اگر انواع دیگری را توسط WebSnap ایجاد و ثبت کرده باشید، لیست آنها را هم مشاهده خواهید کرد.

Home به عنوان نام صفحه و آن هم به صورت خودکار تخصیص می‌یابد. اکنون گزینهٔ Published را فعال کنید و دقت داشته باشید که گزینهٔ Login Required غیرفعال باشد. انتخاب گزینهٔ Published موجب نمایش یک لیست از صفحات در برنامهٔ شما می‌شود. با فشردن دکمهٔ OK در این فرم و در فرم main wizard شکلی مشابه شکل ۶-۱۸ نمایش داده خواهد شد.

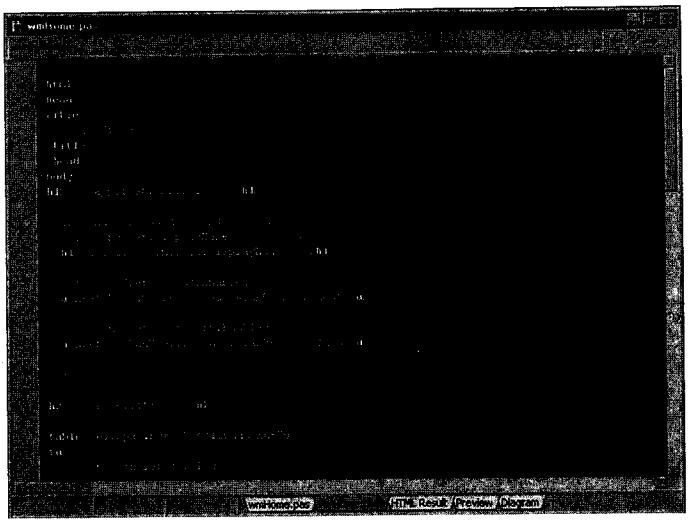
با کمی دقت در شکل ۶-۱۸ متوجه عنوان جدیدی به نام WebAPPComponents خواهید شد. این عنوان، جزء سازنده‌ای است که ارتباط بین اجزاء WebSnap را برقرار می‌سازد و خصوصیات آن نیز همانند سایر اجزاء موجود در WebSnap است.

اکنون به مرحله‌ای رسیدیم که می‌بایست برنامهٔ خود را ذخیره نمائیم. برای راحتی کار ماژول وب (Unit2) را تحت عنوان wmHome و فرم اصلی (Unit1) را هم که به عنوان یک پروژهٔ دلفی مطرح است DDG6Demo نام‌گذاری نمائید.

اگر به بخش ویرایش کد مرتبط با برنامه وارد شوید، چندین تب را در قسمت پایین این ویراستار ملاحظه خواهید کرد. با انتخاب دومین تب، کد صفحهٔ **HTML** به نمایش در خواهد آمد (به شکل ۷-۱۸ مراجعه کنید). توجه داشته باشید که برای این برنامه مدل Standard را برای صفحهٔ **HTML** خود انتخاب کرده‌اید. در این ویراستار یک کد **HTML** بصورت پیش‌فرض وجود دارد که در صورت نیاز قادرید آن را به طریق دلخواه ویرایش نمائید. لیست ۱-۱۸ این کد را نمایش می‌دهد.



شکل ۶-۱۸ ماژول وب مربوط به مثال موردنظر که به وسیلهٔ WebSnap Wizard ایجاد می‌شود



شکل ۷-۱۸ این صفحهٔ HTML با ماژول وب مثال ذکر شده ارتباط برقرار خواهد کرد

لیست ۱-۱۸ کد HTMLای که به صورت پیش‌فرض وجود دارد

```
<html>
<head>
<title>
<%= Page.Title %>
</title>
</head>
<body>
<h1><%= Application.Title %></h1>

<% if (EndUser.Logout != null) { %>
<%   if (EndUser.DisplayName != '') { %>
  <h1>Welcome <%=EndUser.DisplayName %></h1>
<%   } %>
<%   if (EndUser.Logout.Enabled) { %>
  <a href="<%=EndUser.Logout.AsHREF%>">Logout</a>
<%   } %>
<%   if (EndUser.LoginForm.Enabled) { %>
  <a href=<%=EndUser.LoginForm.AsHREF%>>Login</a>
<%   } %>
<% } %>
<h2><%= Page.Title %></h2>

<table cellspacing="0" cellpadding="0">
<td>
<%  e = new Enumerator(Pages)
      s = ''
```

لیست ۱-۱۸ ادامه

```
    c = 0
    for (; !e.atEnd(); e.moveNext())
    {
      if (e.item().Published)
      {
        if (c>0) s += ' | '
        if (Page.Name != e.item().Name)
          s += '<a href="' + e.item().HREF + '">' + e.item().Title + '</a>'
        else
          s += e.item().Title
        c++
      }
    }
    if (c>1) Response.Write(s)
%>
</td>
</table>

</body>
</html>
```

توجه داشته باشید که یک ماژول وب زیرکلاسی از ماژولهایی داده‌ای است که برای ایجاد سرویس‌دهنده‌های وب از آن استفاده می‌شود. علاوه بر ماژول وب شیءهای کنترلی دیگری نظیر TPageProducer را نیز باید به سرویس‌دهندهٔ وب خود بیفزائید. این شیءهای کنترلی قادرند تا به درخواستهایی که به صورت **URL** ارسال می‌شوند، پاسخ دهند. در صورتیکه تمایل دارید می‌توانید از ویراستار دیگری برای صفحهٔ **HTML** خود استفاده کنید که در این صورت **IDE** دلفی امکان برقراری ارتباط با آن را میسر می‌سازد.

باز هم HTML

در اینجا قصد داریم کمی بیشتر با کدهای **HTML** کار کنیم. ابتدا با ورود به Home Web module (شکل ۱۸-۶) شیء Application Adapter را انتخاب کنید. سپس عبارت

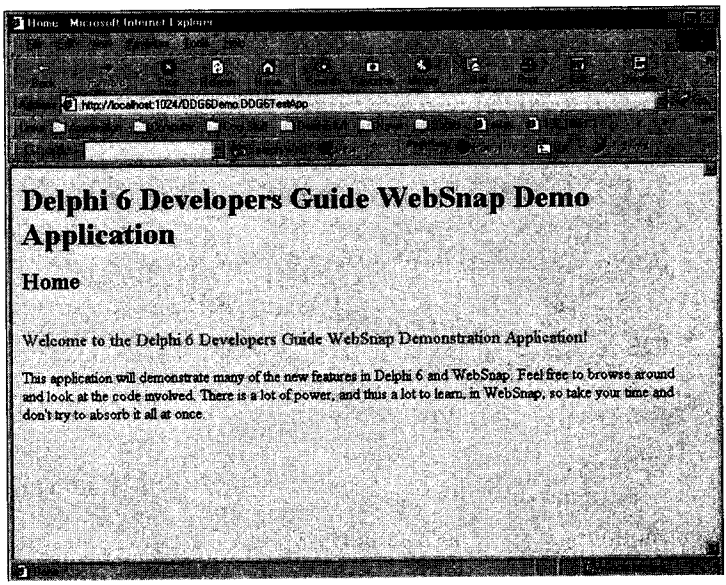
Delphi Developers Guide 7 WebSnap Demo Application

را به خصوصیت ApplicationTitle اختصاص دهید. بخش: **<h1><%= Application.Title %></h1>** که در کد **HTML** مشاهده خواهید کرد، موجب نمایش این مقدار خواهد بود. با ورود به صفحهٔ **HTML** سعی کنید قطعه کد زیر را بعد از **</table>tag** وارد نمائید.

```
<P>
<FONT SIZE="+1" COLOR="Red">Welcome to the Delphi 6 Developers Guide WebSnap
➥ Demonstration Application!</FONT>
<P>
```

```
This application will demonstrate many of the new features in Delphi 6 and
➥WebSnap. Feel free to browse around and look at the code involved. There is
➥a lot of power, and thus a lot to learn, in WebSnap, so take your time and
don't try to absorb it all at once.
<P>
```

حالا می‌توان گفت که یک برنامهٔ وب کوچک طراحی کرده‌اید. برنامه را اجرا کنید. تعجب نکنید. فرم اولیه‌ای که برای شما نمایش داده می‌شود، تنها یک فرم خالی است. از آنجا که فرم موردنظر از نوع **COM** در نظر گرفته می‌شود لازم است مراحل زیر را جهت فراخوانی درست برنامهٔ خود انجام دهید. از این برنامه خارج شوید و برنامهٔ Web App Debugger را از منوی Tools اجرا کنید. بر روی گزینهٔ Default URL hyperlink رفته و برنامهٔ خود را از این طریق انتخاب نمائید. دکمهٔ Go را بفشارید تا صفحه‌ای که ساخته‌اید به نمایش در آید (شکل ۸-۱۸). همانطور که می‌بینید به سادگی یک برنامهٔ وب را طراحی نموده‌اید.



شکل ۸-۱۸ نتیجه اجرای برنامهٔ وبی که طراحی کرده‌اید

منوی Navigation

حال می‌خواهیم بر روی صفحات دیگری نیز کار کنیم. بر روی منوی اصلی محیط **IDE** دلفی حرکت کنید و دومین ابزار منوی Internet را انتخاب نمائید. انجام این مراحل نیز همانند آنچه قبلاً آموختید باعث ایجاد یک صفحهٔ WebSnap مشابه با کادرهای محاوره‌ای می‌شود. همهٔ گزینه‌های نمایش داده شده را به صورت پیش‌فرض رها کرده و تنها در edit box نام صفحه را برابر با Simple نمائید. برنامه را

با نام wmSimple.pas ذخیره کنید.

اکنون همانند مثال قبل یک پیام دلخواه در صفحهٔ **HTML** قرار داده و سپس برنامه را کامپایل کنید. اگر با اجرای Web App Debugger هنوز صفحهٔ قبلی برای شما نمایش داده می‌شود، لازم است تا بر روی گزینهٔ Refresh در محیط مرورگر اینترنتی خود کلیک کنید.

قطعه کد زیر پیاده‌سازی یک منوی Navigation را برای برنامهٔ وب شما نشان می‌دهد. توجه داشته باشید که این کد می‌بایست بر روی دستگاه سرویس‌دهنده وب اجرا گردد.

```
<%  e = new Enumerator(Pages)
    s = ''
    c = 0
    for (; !e.atEnd(); e.moveNext())
    {
      if (e.item().Published)
      {
        if (c>0) s += ' | '
        if (Page.Name != e.item().Name)
          s += '<a href="' + e.item().HREF + '">' + e.item().Title + '</a>'
        else
          s += e.item().Title
        c++
      }
    }
    if (c>1) Response.Write(s)
%>
```

همانطور که ملاحظه می‌کنید این کد با نمایش لیست صفحات موجود بر روی سرویس‌دهندهٔ وب، امکان برقراری ارتباط را با آنها برقرار می‌سازد.

یادداشت

اگر بین دو یا چندین صفحه حرکت می‌کنید، به این نکته توجه داشته باشید که با ورود به هر صفحه، یک درخواست جدید ساخته و ارسال خواهد گردید. دلیل این کار در روش فراخوانی برنامه‌ها توسط Web App Debugger می‌باشد. Web App Debugger با این برنامه‌ها به عنوان یک شیء COM برخورد می‌کند. یعنی ابتدا برنامه اجرا شده، سپس درخواست‌های HTTP پاسخ داده خواهد شد و به محض خروج، برنامه از حالت فعال خود خارج می‌شود که این امر باعث ارسال درخواستهایی جدید به سرویس‌دهنده‌ها خواهد شد. ▲

حال فرض کنید که می‌خواهیم کاربران مشخصی را برای این برنامه‌ها معرفی کنیم، یعنی برنامه موردنظر تنها برای افرادی خاص قابل دسترسی باشد. برای این کار یک صفحهٔ جدید انتخاب کرده و به فرم اصلی برنامه بیفزائید. از روی میلهٔ ابزار Internet گزینهٔ New WebSnap Page را به وسیلهٔ ماوس برگزینید و

نام صفحه را "LoggedIn" بگذارید. قدم بعدی افزودن یک check box با عنوان LogginRequired می‌باشد که در محیط **IDE** دلفی وجود دارد. دکمهٔ **OK** را بفشارید تا به سراغ گامهای بعدی برای طراحی اینگونه صفحات برویم. بار دیگر یادآور می‌شویم که این صفحات تنها برای کاربر وارد شده به شبکه (کاربر معتبر) قابل رؤیت خواهد بود. برنامه را با نام wmLoggedIn.pas ذخیره کنید.

اکنون می‌بایست کد **HTML**‌ای مشابه با کد زیر را در صفحهٔ **HTML** برنامه وارد نمائیم.

```
<P>
<FONT COLOR="Green"><B>Congratulations! </B></FONT>
<BR>
You are successfully logged in!  Only logged in users are granted access
➥to this page. All others are sent back to the Login page.
<P>
```

در اینجا لازم است تفاوت بین صفحات نوع Simple و صفحات نوع LoggedIn را شرح دهیم. این تفاوت تنها در پارامترهای موجود در این صفحات و نحوهٔ ثبت آنهاست. هر صفحهٔ WebSnap در برگیرندهٔ بخشی جهت مقداردهی اولیه صفحات است که عملیات ثبت آنها را انجام می‌دهد. این مقداردهی اولیه که کدی مشابه با قطعه کد ذیل دارد، یک صفحه را تحت یک شیء از نوع WebRequestHandler ثبت می‌کند. این شیء وظیفهٔ مدیریت صفحات **HTML** و نحوهٔ پاسخ‌دهی به آنها را در محیط دلفی به عهده دارد. WebRequestHandler قابلیت ایجاد و حذف ماژولهای وب را داراست.

```
initialization
  if WebRequestHandler <> nil then
  WebRequestHandler.AddWebModuleFactory(TWebPageModuleFactory.Create(TLoggedIn
TWebPageInfo.Create([wpPublished, wpLoginRequired], '.html'),
➥crOnDemand, caCache));
```

با دقت در قطعه کد فوق، پارامتری را با عنوان wpLoginRequired ملاحظه خواهید کرد که وظیفهٔ فراخوانی صفحات برای کاربران مجاز را به عهده دارد. هنگامی که از New Page Wizard استفاده می‌کنید، این کد به صورت پیش‌فرض در برنامهٔ شما قرار می‌گیرد. از آنجا که شکل پیش‌فرض این کد به صورت کدهای توضیحی است (کامپایل نمی‌شود)، لازم است تا در صورت نیاز به قرار دادن کلمهٔ رمز و سایر موارد مختص به کاربران، این کد را از حالت توضیحی خارج و سپس عمل کامپایل را انجام دهید.

اجازهٔ ورود به کاربران

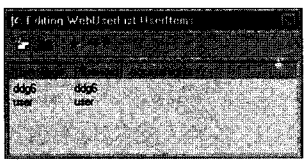
در ادامهٔ بحث قبل، قصد داریم تا مثال دیگری ارائه نمائیم.

یک صفحهٔ جدید ایجاد کرده و نام آن را Login بگذارید. گزینهٔ TAdapterPageProducer را برای انتخاب نوع صفحه برگزینید. در اینجا لازم است گزینهٔ Published را غیرفعال کنید (شکل ۵-۱۸). برنامه را با نام wmLogin ذخیره کنید. با ورود به صفحهٔ WebSnap، جزء سازندهٔ TLoginAdapter را به ماژول خود اضافه نمائید. TAdapterPageProducer در نحوهٔ نمایش و مدیریت صفحات **HTML**

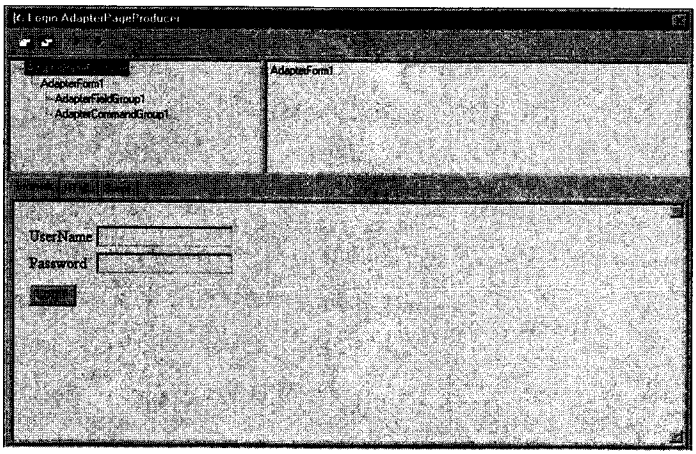
دخالت دارد. عموماً در تمام صفحات خود از TAdapterPageProducer استفاده خواهید کرد، و این به دلیل کارآئی آن است. در این مثال TAdapterPageProducer علاوه بر موارد بالا، وظیفهٔ نمایش و درخواست کلمهٔ رمز و کد کاربر را به عهده دارد. با افزودن TLoginFormAdapter به برنامهٔ خود، دیگر نیازی به کدنویسی هم نخواهید داشت.

اکنون برای مدیریت کاربران، لازم است تا چند کاربر به عنوان نمونه در سیستم تعریف کنید. ماژول وب را فراخوانی کرده (شکل ۶-۱۸) و جزء سازندهٔ WebUserList را انتخاب کنید. این جزء سازنده قادر به مدیریت کد کاربران و کلمات رمز می‌باشد. اکنون با انتخاب دکمهٔ New قادرید به سادگی تعدادی کاربر و کلمهٔ رمز تعریف نمائید. شکل ۹-۱۸ نمونه‌ای از این کار را نمایش می‌دهد.

اکنون لازم است EndUserSessionAdapter را انتخاب کرده و مقدار Login را به خصوصیت LoginPage آن اختصاص دهید. این مقدار معرف نام صفحه‌ای است که از آن برای کنترل ورود کاربران استفاده می‌شود. یک گام به عقب برگردید و بر روی جزء سازندهٔ TAdapterPageProducer دوبار کلیک کنید تا پنجره‌ای همانند شکل ۱۰-۱۸ برای شما نمایش داده شود. AdapterPageProducer را از قسمت بالای این پنجره برگزیده و دکمهٔ New Component را بفشارید. AdapterForm را از منوی ظاهر شده انتخاب کرده و OK را بفشارید. اکنون AdapterForm1 (شکل ۹-۱۸) را انتخاب کرده، مجدداً دکمهٔ New



شکل ۹-۱۸ ویراستار مرتبط با جزء سازندهٔ WebUserList. در این ویراستار دو کاربر و کلمهٔ رمز آنها تعریف شده است



شکل ۱۰-۱۸ پنجرهٔ مربوط به TAdapterPageProducer به همراه جزء سازندهٔ LoginFormAdapter

Component را فشرده و AdapterErrorList را انتخاب کنید. روال فوق را برای AdapterFieldGroup و AdapterCommandGroup نیز انجام دهید. خصوصیت Adpter هر سه شیء را برابر با LoginFormAdapter1 قرار دهید. اکنون لازم است تا با انتخاب AdapterFieldGroup ، دو شیء از نوع AdapterDisplayField را به آن اضافه نمائید. مقدار UserName و Password را برای دوبار متوالی به خصوصیت FiledName اختصاص دهید. کارمان هنوز تمام نشده است. اکنون AdapterCommandGroup را انتخاب کرده و مقدار AdapterFieldGroup1 را به خصوصیت DisplayComponent آن اختصاص دهید، حالا می‌بایست دقیقاً پنجره‌ای متناظر با شکل ۱۰-۱۸ برای شما به نمایش درآمده باشد. حالا با ورود به ویرایشگر دلفی می‌توانید کدهایی برای اعمال کنترل کاربران پیاده‌سازی نمائید.

با ورود کد کاربر و کلمهٔ رمز دلخواه شما، از این پس تنها صفحات مختص شما برایتان به نمایش در خواهد آمد و در صورت معتبر نبودن مجدداً پنجرهٔ مربوط به درخواست ورود برای دریافت کد کاربر و کلمهٔ رمز به نمایش در خواهد آمد.

اگر مایلید نام کاربر در هنگام استفاده از صفحات **HTML** بر روی همان صفحات مشخص باشد، می‌بایست کد **HTML**ای نظیر کد ذیل ماژول وب خود بیفزائید.

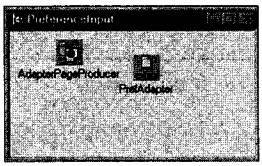
```
<% if (EndUser.Logout != null) { %>
<%   if (EndUser.DisplayName != '') { %>
  <h1>Welcome <%=EndUser.DisplayName %></h1>
<%   } %>
```

مدیریت محیط اختصاص داده شده به هر کاربر

مورد دیگری که شما ممکن است به طراحی آن علاقمند باشید امکان اختصاص محیطهای مجزا به کاربرانتان می‌باشد، در این صورت هر کاربر قادر به انجام تغییرات مورد علاقهٔ خود در این محیط اختصاص داده شده است. به عنوان مثال کاربر قادر به تغییر رنگ پس‌زمینه، نشانه‌ها، ... و یا افزودن ابزارهای موردنظر در محیط مربوط به خود است. این کار قطعاً مشکل به نظر می‌رسد در حالی که WebSnap دلفی این کار را بسیار آسان ساخته است با نوشتن چند کد شما قادر به طراحی این مورد خواهید بود.

اکنون می‌خواهیم این کار را شروع کنیم، در ابتدای امر صفحهٔ جدیدی را به برنامهٔ خود اضافه کنید. اجزاء سازندهٔ TAdapterPageProducer و TAdapter را به صفحهٔ خود اضافه نموده و نام جزء سازندهٔ TAdapter را از TAdapter1 به PrefAdapter تغییر دهید (همانند شکل ۱۱-۱۸). سپس برنامه را با نام wmPreferenceInput ذخیره کنید.

اکنون، اشاره‌گر ماوس را بر روی PerfAdapter برده، آن را دوبار بفشارید و در این حالت دو شیء از نوع AdapterFields و یک شیء از نوع AdapterBooleanField به آن اضافه نمائید. نام اشیاء AdapterFields را به ترتیب FavoriteMovie و PasswordHint بگذارید. نام



شکل ۱۱-۱۸ ماژول وب PereferenceInput که مدیریت فضای اختصاص داده شده به کاربرها را امکان‌پذیر می‌سازد

AdapterBooleanField را نیز برابر با LikesChocolate قرار دهید. (توجه کنید که در این حالت مقادیر DisplayLable و FieldName نیز متعاقباً تغییر خواهند کرد).

جزء سازندهٔ PrefAdapter که از طریق صفحات دیگر نیز قابل دسترسی است، وظیفهٔ نگهداری مقادیر این Prefrence را به عهده دارد.

از اجزاء سازندهٔ TAdapter نیز برای مدیریت، نگهداری و ویرایش اطلاعات و ... استفاده می‌شود. بعد از این مراحل، برای دسترسی و دریافت اطلاعات، لازم است تا قطعه کدی برای رویداد OnGetValue مربوط به AdapterField ها تعریف گردد. شما می‌توانید این اطلاعات را در خصوصیت Session.Values که یک متغیر از نوع رشته‌ای است ذخیره نمائید. شایان ذکر است که این خصوصیت انعطاف بالایی در پذیرش رشته‌ها دارد به نوعی که هر کاراکتری را می‌توان در آن درج نمود. البته تنها در زمان فعال بودن Session عملیات ذخیره‌سازی اطلاعات در آن انجام می‌گیرد.

توجه داشته باشید که با استفاده از کلاس Tadapter قادرید اعمالی را بر روی داده‌های موجود در صفحات نظیر **HTML** ها انجام دهید. به طور مثال فرض کنید که یک صفحهٔ **HTML** دارید که قرار است یک سری اطلاعات در آن درج شود و سپس به وسیلهٔ فشردن دکمه‌ای نظیر Submit اطلاعات آن به سرویس‌دهندهٔ مشخصی ارسال و پاسخ مناسبی دریافت گردد. برای پیاده‌سازی چنین عملیاتی لازم است تا مراحل ذیل را بر روی همین صفحه‌ای که ساخته‌اید (شکل ۱۱-۱۸) ادامه دهید.

با انتخاب PrefAdapter به محیط Object Inspector دلفی وارد شده و بر روی خصوصیت Actions اشاره‌گر ماوس را دوبار بفشارید. یک single action به آن اضافه نموده و نام آن را SubmitAction بگذارید. خصوصیت DispalyLabel را به SubmitInformation تغییر نام دهید. حال به صفحهٔ رویدادهای موجود بر روی Object Inspector مراجعه کرده و کد ارائه شده در لیست ۲-۱۸ را جهت رویداد OnExectue درج نمائید.

از این کد جهت بازیابی مقادیر وارد شده در فیلدهای یک صفحهٔ **HTML** استفاده می‌گردد. این کار با فشردن دکمهٔ Submit اتفاق می‌افتد. در حین انجام این کار، یک نسخه از اطلاعات موجود به وسیلهٔ AdapterFields بر روی session قرار می‌گیرد.

مسلماً هنگامی که برای اولین بار مقادیر این فیلدها جایگزین می‌گردد، باید امکان برگشت به مقدار قبلی حداقل برای یک گام به عقب وجود داشته باشید (Undo) ، پس می‌بایست تصویری از آنها در SessionsService نیز قرار گیرد. برای این کار لازم است تا کد ارائه شده در لیست ۳-۱۸ را به رویداد OnGetValue اضافه نمود.

لیست ۱۸-۲ رویداد OnExecute

```
procedure TPreferenceInput.SubmitActionExecute(Sender: TObject;
  Params: TStrings);
var
  Value: IActionFieldValue;
begin

  Value := FavoriteMovieField.ActionValue;
  if Value.ValueCount > 0 then
  begin
    Session.Values[sFavoriteMovie] := Value.Values[0];
  end;

  Value := PasswordHintField.ActionValue;
  if Value.ValueCount > 0 then
  begin
    Session.Values[sPasswordHint] := Value.Values[0];
  end;

  Value := LikesChocolateField.ActionValue;
  if Value <> nil then
  begin
    if Value.ValueCount > 0 then
    begin
      Session.Values[sLikesChocolate] := Value.Values[0];
    end;
  end else
  begin
    Session.Values[sLikesChocolate] := 'false';
  end;;
end;
```

لیست ۱۸-۳ رویداد OnGetValue

```
...
const
  sFavoriteMovie = 'FavoriteMovie';
  sPasswordHint = 'PasswordHint';
  sLikesChocolate = 'LikesChocolate';
  sIniFileName = 'DDG6Demo.ini';
...

procedure TPreferenceInput.LikesChocolateFieldGetValue(Sender: TObject;
  var Value: Boolean);
var
  S: string;
begin
```

لیست ۳-۱۸ ادامه

```
  S := Session.Values[sLikesChocolate];
  Value := S = 'true';
end;

procedure TPreferenceInput.FavoriteMovieFieldGetValue(Sender: TObject;
  var Value: Variant);
begin
  Value := Session.Values[sFavoriteMovie];
end;

procedure TPreferenceInput.PasswordHintFieldGetValue(Sender: TObject;
  var Value: Variant);
begin
  Value := Session.Values[sPasswordHint];
end;
```

فراموش نکنید، که در صورت دریافت اطلاعات از کاربر و ارسال آن بایستی این عملیات به اطلاع او نیز برسد، بدین معنی‌که کاربر از ارسال اطلاعات خود اطمینان حاصل‌کند، این‌کار به‌وسیلهٔ TAdapterPageProducer انجام می‌پذیرد. با دوبار فشردن اشاره‌گر ماوس بر روی TAdapterPageProducer ، صفحه‌ای همانند مثالهای قبل به نمایش در خواهد آمد. یک AdapterForm جدید ایجاد کرده و اجزاء AdapterFieldGroup و AdapterCommandGroup را به آن بیفزائید. مقدار PrefAdapter را به خصوصیت Adapter متعلق به AdapterFieldGroup و مقدار AdapterFieldGroup را به خصوصیت DisplayComponent از AdapterCommandGroup اختصاص دهید، سپس اشاره‌گر ماوس را بر روی AdapterFieldGroup برده و ضمن فشردن دکمهٔ سمت راست ماوس گزینهٔ All Fields Add را انتخاب کرده و بعد از آن خصصویت FieldName تمامی فیلدها را با مقادیر متناظرشان جایگزین کنید.

این بار اشاره‌گر ماوس را بر روی AdapterCommandGroup برده و با فشردن دکمهٔ سمت راست ماوس، گزینهٔ Add All Commands را انتخاب نمائید.

حالا می‌بایست مقدار SubmitAction را به خصوصیت ActionName از AdapterActionButton.PageName اختصاص دهید. در پایان لازم است تا PrefrencesPage را به خصوصیت AdapterActionButton.PageName اختصاص دهید.

در صورتی که در انجام اعمال فوق اشتباهی رخ داده باشد و یا خطاهای دیگری در برنامه پیش آید پیامی مرتبط با خطایی که روی داده است در تب مربوط به Browser نمایش داده خواهد شد. مسلماً ساختار این پیام‌ها به گونه‌ای است که شما را در نحوهٔ رفع خطاها و اشکال‌زدایی برنامه یاری می‌نماید.

با اجرای این برنامه و با ورود به آن، صفحهٔ مربوط به ورود اطلاعات اولیه را مشاهده خواهید کرد. بدون این که دکمهٔ Submit را بفشارید (به دلیل آن که هیچ صفحه‌ای مرتبط با آن طراحی نکرده‌اید). به وسیلهٔ toolbar یک صفحه برای نمایش اطلاعات کاربران ایجاد کرده و نام آن را PreferencesPage

بگذارید. یونیت جدید را با نام wmPreferences ذخیره کنید. حال به بخش **HTML** این صفحه وارد شده و کد زیر را در محل مناسبی از آن درج نمائید.

```
<P>
Favorite Movie: <%= Modules.PreferenceInput.PrefAdapter.FavoriteMovieField.
➥Value %>
<BR>
Password Hint:  <%= Modules.PreferenceInput.PrefAdapter.PasswordHintField.
➥Value %>

<BR>
<% s = ''
  if (Modules.PreferenceInput.PrefAdapter.LikesChocolateField.Value)
     s = 'You like Chocolate'
   else
     s = 'You do not like chocolate'
   Response.Write(s);
%>
```

اکنون برنامه را اجرا کرده و به دقت تمام مراحل آن را بررسی نمائید.

مطمئن هستیم که با اجرای این برنامه، نکات تجربی بیشتری کسب خواهید کرد.

یادداشت

برای هر صفحه‌ای که به وسیلهٔ WenSnap طراحی و ایجاد می‌شود، یک فایل HTML مرتبط نیز وجود دارد. از آنجا که این فایل‌ها، فایل‌هایی خارج و جدا از برنامه هستند، انجام تغییرات در آنها از طریق یک browser امکان‌پذیر بوده و به خود برنامه و نحوهٔ کامپایل آن مربوط نخواهد بود. ▲

ذخیره اطلاعات کاربر در بین جلسات

حال تنها یک مشکل وجود دارد و آن ذخیره تغییراتی است که کاربر در محیط اختصاص داده شده به خود در بین جلسات اعمال می‌کند. این تغییرات پس از خارج شدن کاربر از شبکه بدون این که در محلی ذخیره شود از بین می‌رود و بنابراین در مرحلهٔ ورود دوبارهٔ وی به شبکه قابل دسترسی نخواهد بود. راه‌حل این مسأله ثبت تغییرات اعمال شده توسط کاربر در هنگام خروج وی از شبکه و اعمال این تغییرات در هنگام ورود به شبکه است،که مطابق لیست ۴-۱۸ تغییرات در هنگام رویداد ورود به شبکه از LoginFormAdapter فراخوانی شده و در هنگام رویداد خروج از شبکه در SessionService ذخیره می‌شود.

همانطور که ملاحظه می‌کنید، زمانی که این رویدادها رخ دهد، داده‌های موجود در یک فایل از نوع INI ذخیره خواهند شد. توجه داشته باشید که شما قادرید این اطلاعات را در الگوهای دیگری نظیر یک بانک اطلاعاتی نیز ذخیره نمائید و قصد ما از ذخیره‌سازی اطلاعات در یک فایل INI تنها ارائهٔ مثال بوده است.

متغیر Lock که ذیلاً معرفی خواهد شد، یک متغیر عمومی از نوع TMultiReadExclusiveWriteSynchronizer

لیست ۴-۱۸ رویدادهای OnEndSession و OnLogin

```
procedure TLogin.LoginFormAdapter1Login(Sender: TObject; UserID: Variant);
var
  IniFile: TIniFile;
  TempName: string;
begin
  // Grab session data here
  TempName := Home.WebUserList.UserItems.FindUserID(UserId).UserName;
➥  //WebContext.EndUser.DisplayName;
  Home.CurrentUserName := TempName;

  Lock.BeginRead;
  try
    IniFile := TIniFile.Create(IniFileName);
    try
      Session.Values[sFavoriteMovie] := IniFile.ReadString(TempName,
      ➥sFavoriteMovie, '');
      Session.Values[sPasswordHint] := IniFile.ReadString(TempName,
      ➥sPasswordHint, '');
      Session.Values[sLikesChocolate] := IniFile.ReadString(TempName,
      ➥sLikesChocolate, 'false');
    finally
      IniFile.Free;
    end;
  finally
    Lock.EndRead;
  end;
end;

procedure THome.SessionsServiceEndSession(ASender: TObject;
  ASession: TAbstractWebSession; AReason: TEndSessionReason);
var
  IniFile: TIniFile;
begin
  //Save out the preferences here
  Lock.BeginWrite;
  if FCurrentUserName <> '' then
  begin
    try
      IniFile := TIniFile.Create(IniFileName);
      try
        IniFile.WriteString(FCurrentUserName, sFavoriteMovie,
        ➥ASession.Values[sFavoriteMovie]);
        IniFile.WriteString(FCurrentUserName, sPassWordHint,
        ➥ASession.Values[sPasswordHint]);
        IniFile.WriteString(FCurrentUserName, sLikesChocolate,
        ➥ASession.Values[sLikesChocolate]);
      finally
        IniFile.Free;
```

لیست ۴-۱۸ ادامه

```
      end;
    finally
      Lock.EndWrite
    end;
  end;
end;
```

بوده که به وسیلهٔ بخش مقداردهی اولیه صفحهٔ اصلی برنامه ایجاد می‌شود. از آنجاکه عملیات خواندن و نوشتن در فایل مذکور به صورت چندگانه انجام می‌شود، می‌بایست در حین عمل نوشتن و یا خواندن به مسألهٔ امنیت داده‌ها در استفاده مشترک از آنها توجه شود. استفاده از این متغیر در افزایش ضریب اطمینان عملیات مؤثر است. بنابراین لازم است که تعریف متغیر زیر را در بخش interface یونیت wmhome وارد نمائید.

```
var
    Lock: TMultiReadExclusiveWriteSynchronizer;
```

سپس بخش‌های initialization و finalization را به صورت زیر پیاده‌سازی کنید:

```
Initialization
  ...
  Lock := TMultiReadExclusiveWriteSynchronizer.Create;
finalization
  Lock.Free;
```

حال با افزودن قطعه کد زیر به یونیت wmHOME برنامهٔ خود راکامل نموده‌اید.

```
const
      sIniFileName = 'DDG6Demo.ini';

...
function IniFileName: string;
begin
  Result := ExtractFilePath(GetModuleName(HInstance)) + sIniFileName;
end;
```

گرافیک و انیمیشن در وب

به کارگیری جلوه‌های گرافیکی در صفحات وب در چگونگی عملکرد آنها تأثیر بسزایی داشته است. شما نیز به عنوان یک طراح و یا برنامه‌نویس لازم است تا با شیوهٔ نمایش تصاویر، پردازش و مدیریت الگوهای گرافیکی در صفحات وب آشنا شوید.

دلفی ابزارهای قدرتمندی را برای ایجاد آسان این نوع برنامه‌های کاربردی در اختیارتان می‌گذارد و

طبیعتاً WebSnap یکی از این ابزارها خواهید بود. WebSnap علاوه بر امتیازهای منحصر به فردی که دارد، از تمام الگوهای تصویری معتبر نیز پشتیبانی می‌کند. در اینجا با ارائه یک مثال شما را در شناخت تکنیک‌های دلفی برای این گونه برنامه‌ها راهنمایی خواهیم کرد.

با استفاده از میله ابزار Internet ، یک صفحهٔ جدید از نوع TAdapterPageProducer به برنامهٔ خود اضافه نمائید. (گزینهٔ Published در حالت فعال خود باشد).

نام صفحه را Images قرار داده و برنامه را با نام wmImages ذخیره کنید. حال به قسمت Image Web module رفته و با قرار دادن یک TAdapter در ماژول موجود، نام آن را به ImageAdapter تغییر دهید. نهایتاً با دوبار فشردن اشاره‌گر ماوس بر روی ImageAdapter دو فیلد از نوع TAdapterImageField را به آن اضافه نمائید.

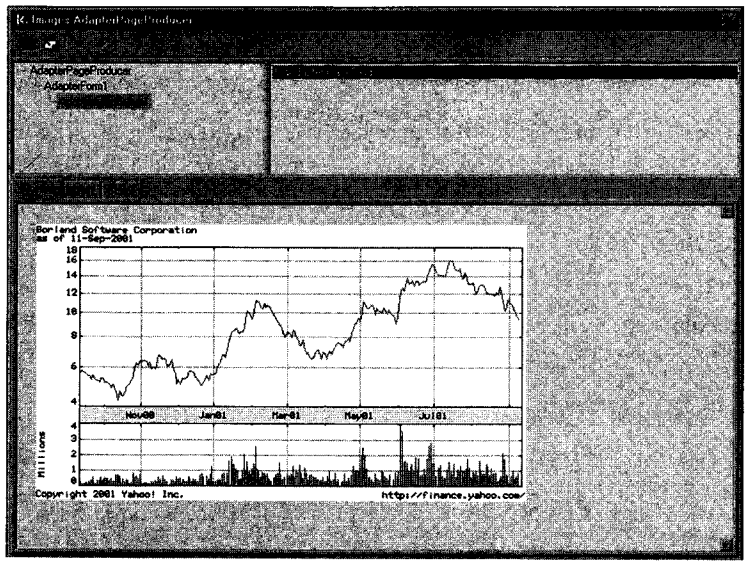
قرار است هر کدام از این فیلدها را به منظورهای متفاوتی به کار گیریم. ابتدا قصد داریم تا تصویری مرتبط با یک **URL** را به نمایش در آوریم. برای این کار، یکی از AdapterImageField ها را انتخاب کرده و آدرس تصویری موجود در **URL** دلخواه خود را به خصوصیت HREF آن اختصاص دهید. برای مثال خصوصیت HREF را برابر با **http://chart.yahoo.com/c/1y/b/borl.gif** قرار دهید.

اشاره‌گر ماوس را بر روی TAdapterPageProducer برده و با دوبار فشردن آن، یک AdapterForm به آن بیفزائید. سپس به AdapterForm یک AdapterFieldGroup اضافه کرده و مقدار ImageAdapter را به خصوصیت adapter آن اختصاص دهید. حال دوبار اشاره‌گر ماوس را بر روی AdapterFieldGroup برده و با فشردن دکمهٔ سمت راست ماوس، گزینهٔ Add All Fields را انتخاب کنید. بعد از آن فیلد ReadOnly مربوط به AdapterImageField را برابر Ture نموده تا امکان نمایش تصویر بر روی صفحه فراهم شود. در صورتی که مقدار این فیلد برابر با False باشد، در هنگام اجرا یک جعبه ویرایش برای جستجوی نام فایل، نمایش داده خواهد شد. البته عموماً این فیلد در حالت True تنظیم می‌گردد. اکنون اگر از همان آدرس بالا (آدرس http ذکر شده در بالا) استفاده کرده باشید پنجره‌ای همانند شکل ۱۲-۱۸ برای شما به نمایش در می‌آید.

تا اینجا نحوهٔ نمایش تصاویر موجود در یک URL را آموختید. اما گاهی اوقات لازم است تا تصاویر از یک stream فراخوانی شده و به نمایش در آید. برای این کار نیز می‌بایست از جزء سازندهٔ AdapterImageField استفاده کنید.

اکنون بر روی همان صفحهٔ قبل، دومین AdapterImageField را انتخاب کنید. با رفتن به روی Object Inspector به صفحهٔ رویدادهای آن وارد شده و با فراخوانی رویداد OnGetImage یک فایل JPG را در آن قرار دهید. این قسمت باید کدی مشابه با کد زیر داشته باشد.

```
procedure TImages.AdapterImageField2GetImage(Sender: TObject;
  Params: TStrings; var MimeType: String; var Image: TStream;
  var Owned: Boolean);
begin
  MimeType := 'image\jpg';
  Image := TFileStream.Create('athena.jpg', fmOpenRead);
end;
```



شکل ۱۲-۱۸ درج یک تصویر در یک برنامهٔ کاربردی وب

از آنجا که همهٔ تصویر از نوع فایل‌های JPG نمی‌باشند، برای فراخوانی سایر فایل‌های تصویری، می‌بایست نوع آن را در پارامتر MimeType قرار دهید. اکنون با اجرای برنامه نحوهٔ فراخوانی این تصویر را مشاهده کنید.

نمایش داده در برنامه‌های وب

می‌خواهیم یک برنامهٔ وب طراحی کنیم تا در دستیابی، پردازش و مدیریت حجم بالایی از داده‌ها ما را یاری رساند. آیا دلفی روش آسانی را برای انجام این کار ارائه می‌دهد؟ پاسخ این پرسش مثبت است. WebSnap دلفی این مکانیزم را فراهم می‌کند.

تاکنون با ویژگیهای زیادی از WebSnap آشنا شده‌اید. اکنون قصد داریم قدرت آن را در نحوهٔ مدیریت بانک‌های اطلاعاتی تحت وب شرح دهیم. بهترین کار ارائهٔ یک مثال کاربردی در این خصوص است.

با انتخاب گزینهٔ سوم (WebDataModule) بر روی میلهٔ ابزار Internet کارمان را شروع می‌کنیم. اکنون از روی تب مربوط به WebSnap یک جزء سازندهٔ TDatesetAdapter و از روی تب مربوط به BDE یک جزء سازندهٔ TTable را با بانک اطلاعاتی DBDemos و جدول BioLife مرتبط سازید. حال مقدار خصوصیت DatasetAdapter1 را برابر Table1 قرار دهید. برای باز کردن جدول BioLife توسط این برنامه می‌بایست مقدار Ture را به خصوصیت Table1.Active اختصاص دهید.

نام WebDatamodule را BioLife data گذاشته و برنامه را با نام wdmBioLife ذخیره کنید.

یادداشت

به خاطر داشته باشید که روش استفاده از TTable در مثال فوق برای تمام برنامه‌های وب عمومیت ندارد. اگر در برنامهٔ کاربردی شما از sessionها هم پشتیبانی می‌شود، به کارگیری TTable به دور از اشکال خواهد بود و در غیر این صورت توصیه می‌کنیم از اجزاء سازندهٔ دیگری در این رابطه استفاده کنید. ▲

اکنون نوبت DatasetAdapter است. DatasetAdapter را انتخاب کرده و بر روی Actions node دکمهٔ راست ماوس را بفشارید. از منوی ظاهر شده گزینهٔ Add All Actions را انتخاب کنید. TTable را از طریق خصوصیت Dataset با TAdapterDataSet مرتبط سازید. با فشردن دکمهٔ سمت راست ماوس بر روی Fields node گزینهٔ Add All Fileds را برگزینید. همین روال را برای TTable انجام داده، تمام فیلدهای موجود در Dataset را انتخاب و به WebDatamodule اضافه نمائید.

بی‌تردید این شیوهٔ استفاده از بانک‌های اطلاعاتی هم، از قوانین حاکم بر ساختار آنها پیروی می‌کند. منظور متناظر بودن نحوهٔ مدیریت و پردازش بانک‌های اطلاعاتی موجود بر روی سرویس‌دهنده‌های وب و بانک‌های اطلاعاتی عادی است.

یکی از ابتدایی‌ترین عملیات در مدیریت بانک‌های اطلاعاتی تعریف یک کلید اصلی برای بانک‌های اطلاعاتی است. کلید اصلی به فیلد کلیدی گفته می‌شود که در تمام سطرهای جدول بانک اطلاعاتی یک مقدار منحصر به فرد است و یا به بیان دیگر هیچ دو سطری را نمی‌توان یافت که دارای کلیدهای اصلی یکسان باشند. در WebSnap نیز این کار به صورت خودکار و پس از انتخاب فیلد Species_No در Object Treeview و تخصیص همزمان مقدار pfInKey به خصوصیت ProviderFlags انجام می‌گیرد.

دوباره صفحهٔ دیگری را به برنامهٔ خود اضافه کنید. یک جزء سازندهٔ TAdapterPageProducer به صفحهٔ برنامهٔ خود بیفزایید و نام صفحه را BioLife بگذارید. یونیت ساخته شده را تحت عنوان wmBioLife ذخیره کنید. از آنجا که قرار است داده‌های بانک اطلاعاتی در این جدول نمایش داده شود لازم است در قسمت uses یونیت wmBioLife، یونیت wdmBioLife را نیز قرار دهید.

اگر کمی صبر داشته باشید تا لحظاتی دیگر، برنامهٔ موردنظر خود را پیاده‌سازی کرده‌اید. با رفتن به روی ماژول BioLife، دکمهٔ سمت راست ماوس را بر روی AdapterPageProducer بفشارید. حال بر روی WebPageItems رفته و گزینهٔ New Component را برگزیده، سپس AdapterForm را انتخاب کنید. با فشردن دکمهٔ سمت راست ماوس بر روی AdapterForm اجزاء سازندهٔ AdapterErrorList و AdapterGrid را به آن اضافه نمائید. خصوصیت Adapter هر دو جزء سازندهٔ ذکر شده را برابر با DatasetAdapter مقداردهی کنید. اشاره‌گر ماوس را بر روی AdapterGrid برده و با فشردن دکمهٔ سمت راست ماوس، گزینهٔ Add All Columns را انتخاب کنید. گزینهٔ Add All Actions را از روی Action node متعلق به DatasetAdapter برگزینید. با رفتن به روی Field node، گزینهٔ Add All Fields را انتخاب کنید. تا اینجا مراحل لازم برای نمایش داده‌ها بر روی یک ماژول وب را

طی کرده‌اید. اما هنوز چند گام دیگر تا تکمیل نهائی این برنامه باقی است.

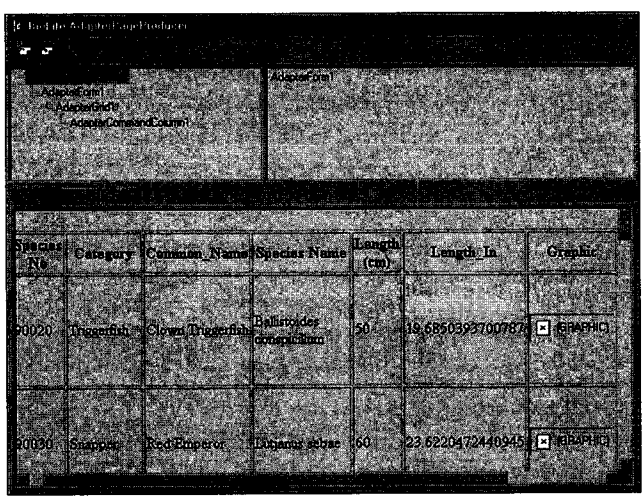
به ماژول وب BioLife رفته و دکمهٔ ماوس را بر روی AdapterPageProducer دوبار بفشارید. در این حالت می‌بایست صفحه‌ای حاوی داده‌های بانک اطلاعاتی برای شما به نمایش درآید. اگر چنین اتفاقی نیفتاد، لازم است تا مجدداً ارتباط بین اشیاء و جدول بانک اطلاعاتی را بررسی کنید. از آنجا که فیلدهای این بانک اطلاعاتی زیاد است، لازم است تا گام بعدی را جهت نمایش بهتر این صفحه به انجام برسانید، بر روی صفحهٔ ظاهر شده گزینهٔ AdapterGrid را که در سمت چپ بالای صفحه قرار دارد، انتخاب کرده و جزء سازندهٔ ColNotes که در سمت راست صفحه نمایش داده شده است را حذف نمائید. در حال حاضر می‌بایست پنجره‌ای مشابه با شکل ۱۳-۱۸ ایجاد شده باشد.

هم اینک نخستین برنامهٔ بانک اطلاعاتی تحت وب خود را در دلفی ایجاد کرده‌اید. این برنامه نمایشی از اطلاعات جدول BioLife را در اختیار کاربران قرار می‌دهد.

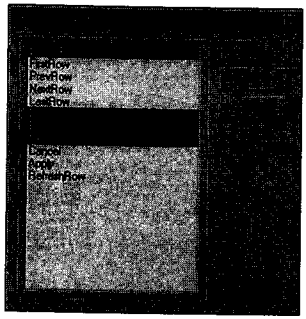
حال که بانک اطلاعاتی در کار است، به روشی نیاز دارید که با استفاده از آن رکوردها را پیمایش کرده و اعمال نظیر درج، حذف و ویرایش را بر روی داده‌ها به انجام رسانید. این کار را با استفاده از برنامه‌سازی نیز می‌توانید انجام دهید، اما روش آسان‌تری نیز وجود دارد و آن هم استفاده از WebSnap است.

به محیط Web Surface Designer رفته و با انتخاب AdapterGride، دکمهٔ سمت راست ماوس را بر روی آن بفشارید و یک جزء سازندهٔ AdapterCommandColumn را به آن اضافه کنید. با فشردن دکمهٔ سمت راست ماوس بر روی AdapterCommandColumn منوی همانند شکل ۱۴-۱۸ ظاهر خواهد شد. از این منو، گزینه‌های BrowseRow و EditRow و DeleteRow و NewRow را انتخاب کنید.

دکمهٔ OK را به وسیلهٔ ماوس انتخاب کنید. اکنون مقدار ۱ را به خصوصیت Display Columns از جزء سازندهٔ AdapterCommandColumn اختصاص دهید. این مقدار معرف نحوهٔ نمایش دکمه‌های



شکل ۱۳-۱۸ نمایش جدول BioLife در یک برنامهٔ وب



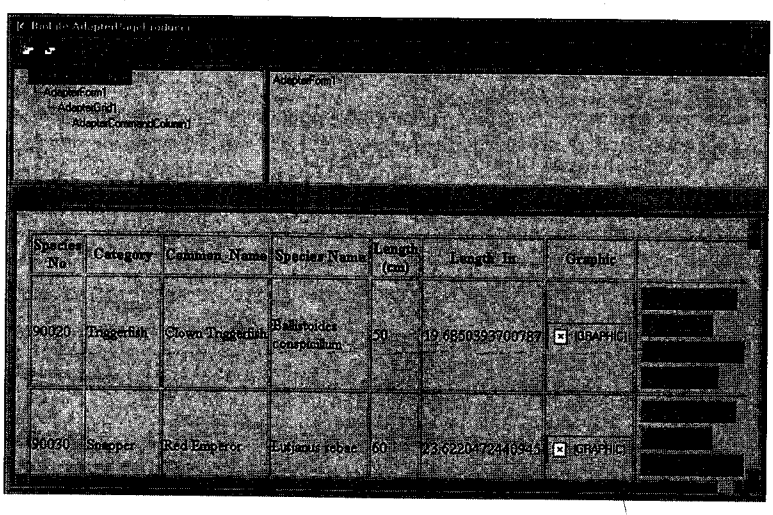
شکل ۱۴-۱۸ منوی Add Commands برای انتخاب عملیات قابل انجام بر روی بانک‌های اطلاعاتی تحت وب

فرامین مرتبط با بانک اطلاعاتی می‌باشد. (شکل ۱۵-۱۸ را ملاحظه کنید).

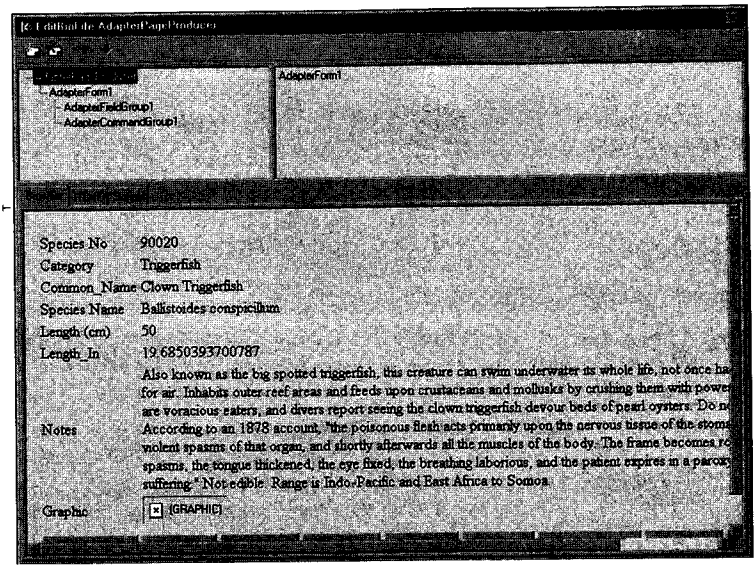
چند گام دیگر تا فعال کردن این دکمه‌ها باقی است.

از TAdapterPageProducer برای افزودن یک صفحهٔ دیگر به برنامهٔ خود استفاده کنید. نام صفحه را BioLifeEdit گذاشته و یونیت ایجاد شده را تحت عنوان wmBioLifeEdit ذخیره سازید. wdmBioLife را به قسمت uses این یونیت اضافه نمائید.

با دوبار فشردن دکمهٔ ماوس بر روی TAdapterPageProducer (در ماژول جدید)، اجزاء سازندهٔ AdapterForm، AdapterErrorList، AdapterFieldGroup و AdapterCommandGroup را به آن بیفزائید. بر روی AdapterFieldGroup دکمهٔ ماوس را فشرده و سپس تمام فیلدها و فرامین را به AdapterCommandGroup اضافه کنید. آنچه که به نمایش در می‌آید، مشابه با شکل ۱۶-۱۸ خواهد بود.



شکل ۱۵-۱۸ برنامهٔ طراحی شده به همراه فرامین کار بانک اطلاعاتی



شکل ۱۶-۱۸ صفحهٔ برنامهٔ BioLifeEdit به همراه فیلدهای موجود در بانک اطلاعاتی و فرامینی که در زیرصفحه مشاهده می‌گردد

همانطور که ملاحظه می‌کنید، قادرید با انتخاب دکمه‌های فرامین موجود در روی صفحه اعمالی نظیر ویرایش، حذف و یا نمایش رکوردها را انجام دهید. هر کدام از این دکمه‌ها یک عمل متفاوت انجام‌می‌دهد. در صورتی که دکمهٔ Browse را انتخاب کنید، رکوردهای موجود به شکل یک متن ساده نمایش داده خواهند شد. با انتخاب دکمهٔ Edit یک جعبه ویرایش برای شما نمایش داده می‌شود و بدین ترتیب با جایگذاری این دکمه‌ها، کاربران برنامهٔ خود را در مدیریت و پردازش بانک‌های اطلاعاتی تحت وب یاری کرده‌اید.

تبدیل برنامه‌های کاربردی به یک ISAPI DLL

یکی از قدیمی‌ترین ویژگیهای این گونه برنامه‌ها در نگهداری اطلاعات Sessionها و نحوهٔ مدیریت آنهاست. اما برای این که این برنامه‌ها، دارای چنین امکاناتی باشند می‌بایست آنها را به برنامه‌های مقیم در حافظهٔ ویژه‌ای تحت عنوان ISAPI DLL تبدیل نمود. هر برنامهٔ کاربردی ISAPI یک DLL است که در فضای پردازشی سرویس‌دهندهٔ وب اجرا می‌شود.

روش کار بسیار ساده و مختصر است. برای این کار ابتدا می‌بایست یک پروژه از نوع ISAPI ایجاد کرده و تمام یونیت‌های موجود در آن را حذف نمائید. سپس تمام یونیت‌های موجود در Web App به جز یونیت مربوط به فرم اصلی را به آن اضافه کرده و عمل کامپایل را انجام دهید. هم اینک فرآیند تبدیل انجام شده است.

یادداشت

توجه داشته باشید که اگر به هنگام آزمایش یک ISAPI DLL تغییراتی در آن اعمال کنید، در آن صورت تا زمان توقف و شروع مجدد سرویس وب، قادر به کپی مجدد DLL به سرویس‌دهندهٔ وب نخواهید بود. ▲

موضوعات پیشرفته

تاکنون، شیوهٔ ایجاد برنامه‌های کاربردی وب، مدیریت کاربران و حتی کار بانک‌های اطلاعاتی مبتنی بر سیستم‌های سرویس‌گیرنده/ سرویس‌دهنده را فراگرفتید. در ادامه قصد داریم مطالب پیشرفته‌تری را در رابطه با این گونه برنامه‌ها و همچنین WebSnap شرح دهیم.

LocateFileServies

همانطور که تاکنون ملاحظه کرده‌اید هنگام استفاده از WebSnap نیاز به فراخوانی سایر منابع داده‌ای نیز دارید. این بدان معنی است که WebSnap به همراه منابع داده‌ای دیگر نظیر کدهای **HTML**، کدهای دلفی، بانک‌های اطلاعاتی، تصاویر، اسکریپت‌ها و ... امکان طراحی و پیاده‌سازی برنامه‌های کاربردی وب را میسر می‌سازد. درست به همین خاطر است که WebSnap از تمام منابع داده‌ای فوق، به خصوص کدهای **HTML** و صفحات وب پویا پشتیبانی می‌کند.

WebSnap امکان فراخوانی کدهای **HTML** مرتبط با منابع مختلف را برقرار می‌سازد. برای فراخوانی و یا دسترسی به فایل‌های **HTML** موجود بر روی منابع مختلف و یا TSrteam ها از جزء سازندهٔ LocateFilesService استفاده می‌شود.

فراخوانی یک کد **HTML** از طریق TStream ها به مفهوم وجود حجم بالایی از اطلاعات در یک فایل **HTML** و ارتباط آن با صفحات دیگر است. در اینجا با ارائهٔ یک مثال نحوهٔ فراخوانی اینگونه صفحات توسعه یافته که با دیگر منابع داده‌ای مرتبط می‌باشند را بررسی می‌کنیم.

به وسیلهٔ Notepad ویندوز و یا ویراستار متنی دیگری، یک فایل متنی با نام HTML.RC ایجاد نمائید. این فایل را در همان فهرست برنامهٔ کاربردی خود ذخیره و آن را با برنامهٔ خود مرتبط سازید. سپس قطعه کد زیر را در این فایل درج نمائید.

```
#define HTML 23  // HTML resource identifier
EMBEDDEDHTML HTML embed.html
```

هنگام کامپایل، دلفی یک فایل RES را تولید خواهد کرد. اکنون به وسیلهٔ TPageProducer یک صفحهٔ جدید را به برنامهٔ خود بیفزائید و آن را Embedded بنامید. فایل را با نام wmEmbedded ذخیره سازید. اکنون از منوی Home (شکل ۶-۱۸) جزء سازندهٔ رویداد OnFindStream را فعال کنید. با انجام این کار کدی مشابه با کد زیر نمایان می‌شود.

```
procedure THome.LocateFileServiceFindStream(ASender: TObject;
  AComponent: TComponent; const AFileName: String;
  var AFoundStream: TStream; var AOwned, AHandled: Boolean);
begin

end;
```

حال این قطعه کد را مشابه کد زیر اصلاح نمائید.

```
procedure THome.LocateFileServiceFindStream(ASender: TObject;
  AComponent: TComponent; const AFileName: String;
  var AFoundStream: TStream; var AOwned, AHandled: Boolean);
begin
  // we are hunting up the Embedded file
  if Pos('EMBEDDED', UpperCase(AFileName)) > 0 then   begin
    AFoundStream := TResourceStream.Create(hInstance, 'EMBEDDED', 'HTML');
    AHandled := True; // no need to look further
  end;
end;
```

پارامتر AFileName معرف نام فایل **HTML** می‌باشد. AFoundStream نیز وظیفهٔ فراخوانی فایل **HTML** را به عهده دارد. به جملات توضیحی موجود در این کد نیز دقت داشته باشید.

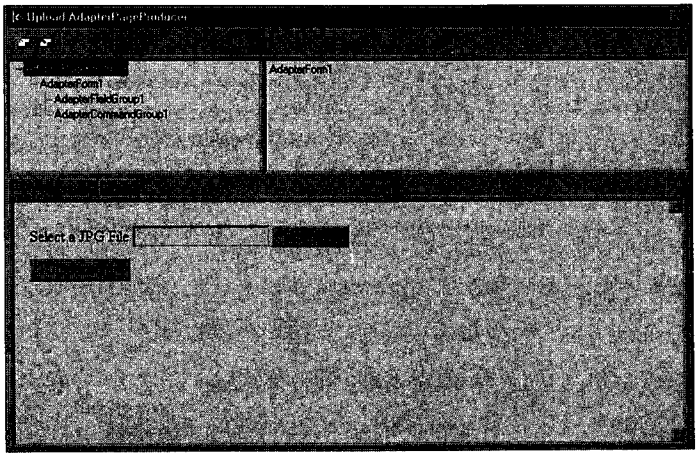
اکنون برنامه را اجرا کرده و نحوهٔ کارآیی آن را بررسی نمائید.

سرویس‌های انتقال فایل

یکی از متداول‌ترین عملیات برنامه‌های کاربردی وب، ارائه سرویسهای انتقال فایل می‌باشد، WebSnap دلفی از این سرویسها نیز پشتیبانی کرده و افزودن آنها را به برنامه‌هایتان به آسانی ممکن ساخته است. با ارائهٔ یک مثال، روش طراحی اینگونه برنامه‌ها را بررسی خواهیم کرد.

باز هم یک صفحهٔ جدید به برنامهٔ خود اضافه کنید. نام صفحه را Upload گذاشته و یک جزء سازندهٔ TAdapterPageProducer در آن قرار دهید. برنامه را با نام wmUpload ذخیره کنید. با فشردن دکمهٔ ماوس بر روی TAdapter یک AdapterFileField به آن اختصاص دهید. مدیریت عملیات انتقال فایل توسط این فیلد انجام می‌شود. یک single action به جزء سازندهٔ Adapter تخصیص دهید و نام آن را UploadAction بگذارید.

بعد از تکمیل مراحل بالا، اجزاء سازندهٔ AdapterForm ، AdapterErrorList ، AdapterFieldGroup و AdapterCommandGroup را به برنامه بیفزائید. اجزاء سازندهٔ AdapterForm و AdapterErrorList را به Adapter1 و جزء سازندهٔ AdapterCommandGroup را به AdapterFieldGroup مرتبط سازید. سپس با انتخاب گزینهٔ add all actions برای AdapterCpmmandGroup کار را ادامه دهید. caption مربوطه را به Upload File تغییر نام دهید. هم اینک می‌بایست شکل ۱۷-۱۸ برای شما نمایش داده شود.



شکل ۱۷-۱۸ نحوهٔ نمایش صفحهٔ Upload و Web Surface Designer و دکمهٔ Browse که به صورت خودکار نصب گردیده‌است.

هنوز برنامه تکمیل نشده است. برای تکمیل این برنامهٔ کاربردی بایستی دو قطعه کد در قسمت‌های مختلف برنامه درج شود. ابتدا کد ارائه شده در لیست ۵-۱۸ را برای رویداد OnFileUpload از Adapter1.AdapterFileField وارد نمائید.

لیست ۵-۱۸ رویداد OnFileUpload

```
procedure TUpload.AdapterFileField1UploadFiles(Sender: TObject;
  Files: TUpdateFileList);
var
  i: integer;
  CurrentDir: string;
  Filename: string;
  FS: TFileStream;
begin
  // Upload file here
    if Files.Count <= 0 then
    begin
      raise Exception.Create('You have not selected any files to be uploaded')
    end;
    for i := 0 to Files.Count - 1 do
    begin
    // Make sure that the file is a .jpg or .jpeg
    if (CompareText(ExtractFileExt(Files.Files[I].FileName), '.jpg') <> 0)
        and (CompareText(ExtractFileExt(Files.Files[I].FileName), '.jpeg')
        ➥<> 0) then
    begin
```

لیست ۵-۱۸ ادامه

```
      Adapter1.Errors.AddError('You must select a JPG or JPEG file to upload');
    end else
    begin
      CurrentDir := ExtractFilePath(GetModuleName(HInstance)) + 'JPEGFiles';
      ForceDirectories(CurrentDir);
      FileName := CurrentDir + '\' + ExtractFileName(Files.Files[I].FileName);
      FS := TFileStream.Create(Filename, fmCreate or fmShareDenyWrite);
      try
        FS.CopyFrom(Files.Files[I].Stream, 0); // 0 = copy all from start
      finally
        FS.Free;
      end;
    end;
  end;
end;
```

اکنون آنچه که در رویداد OnFileUpload و با توجه به کد بالا انجام می‌پذیرد را بررسی می‌کنیم. ابتدا یک فایل انتخاب شده و سپس عملیات بررسی نوع آن (فایل JPEG باشد) انجام می‌گیرد. در صورت صحت انجام موارد ذکر شده، فایل موردنظر در TFileStream قرار داده می‌شود. عملیات نهایی که همان مدیریت **HTTP** برای انتقال فایل از یک سرویس‌گیرنده به یک سرویس‌دهنده می‌باشد نیز توسط کلاس TUpdateFileList صورت می‌پذیرد.

حال کد زیر را برای رویداد OnExecute از Adapter1.UploadAction وارد کنید.

```
procedure TUpload.UploadActionExecute(Sender: TObject; Params: TStrings);
begin
  Adapter1.UpdateRecords;
end;
```

همانطور که متوجه شده‌اید عملیات دریافت فایل به وسیلهٔ کد فوق انجام خواهد پذیرفت.

افزودن صفحات HTML اختیاری

در ابتدای این فصل، هنگامی که New Page Wizard را برای ایجاد صفحات به کار می‌گرفتیم، ملاحظه کردید که تنها دو نوع انتخاب برای صفحات **HTML** وجود داشت. یکی از این دو نوع standard template و نوع دیگر blank template بود.

standard template یک مدل ابتدایی برای طراحی برنامه‌های کاربردی وب می‌باشد (به مثال‌های قبل مراجعه کنید) و این درست بدین معنی است که برای طراحی صفحات قدرتمندتر می‌بایست مدل دیگری را به کار گرفت. اگر تمایل دارید مدل‌های صفحات **HTML** خود را در دلفی افزایش دهید می‌بایست برنامهٔ TemplateRes.dpk را از فهرست <Delphi>\Demos\WebSnap\Util\ کامپایل و نصب نمائید.

ساخت اجزاء سازندهٔ جدید در TAdapterPageProducer

حتماً تاکنون متوجه شده‌اید که جزء سازندهٔ TAdapterPageProducer ، عنصر کلیدی طراحی برنامه‌های کاربردی وب در دلفی است.

تفاوت تکنیک‌های برنامه‌نویسان در ارائهٔ اینگونه برنامه‌ها، وجود اجزاء سازندهٔ دیگری را ضروری می‌نماید که هر کدام وابسته به شرایط و ویژگیهای خاصی خواهند بود. دلفی از این قابلیت نیز پشتیبانی می‌کند. برای تعریف یک جزء سازنده در کلاس TAdapterPageProducer می‌بایست از کلاس سطح بالایی به نام TWebContainedComponent[1] و یک insterface با نام IWebContent استفاده کرد. لیست ۶-۱۸ یک کلاس انتزاعی را نمایش می‌دهد.

لیست ۶-۱۸ کلاس انتزاعی TWebContainedComponent

```
type

Tddg6BaseWebSnapComponent = class(TWebContainedComponent, IWebContent)
  protected
    { IWebContent }
    function Content(Options: TWebContentOptions; ParentLayout: TLayout):
    ➥string;
    function GetHTML: string; virtual; abstract;
  end;
```

This class is implemented like so:

```
function Tddg6BaseWebSnapComponent.Content(Options: TWebContentOptions;
  ParentLayout: TLayout): string;
var
  Intf: ILayoutWebContent;
begin
  if Supports(ParentLayout, ILayoutWebContent, Intf) then
    Result := Intf.LayoutField(GetHTML, nil)
  else
    Result := GetHTML;
end;
```

دلیل پیاده‌سازی تابع Content در این کلاس انتزاعی فقط به خاطر تابع GetHTML و نوع آن می‌باشد. تابع GetHTML نیز به صورت انتزاعی تعریف گردیده است. چنانچه جزء سازندهٔ طراحی شده جزو LayoutGroup باشد، از تابع Content برای درج آن در LayoutGroup استفاده خواهد شد و در غیر این صورت با فراخوانی تابع GetHTML کد **HTML**ای برای ثبت در TAdapterPageProducer

---

۱- این کلاس، دربرگیرندهٔ شکل تخصصی‌تری از کلاس TAdapterPageProducer می‌باشد.

ارجاع خواهد گردید.

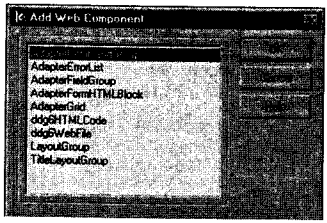
دو جزء سازنده برای برنامه‌های کاربردی وب در CD اختیاری این کتاب گنجانده شده این اجزاء وظیفهٔ درج یک **HTML** (کد **HTML** و یا یک فایل **HTML**) در TAdapterPageProducer را به عهده دارند. یکی از این اجزاء Tddg6HTMLCode نام دارد، کد ارائه شده در لیست ۷-۱۸ نحوهٔ پیاده‌سازی آن را نمایش می‌دهد.

لیست ۷-۱۸ جزء سازندهٔ Tddg6HTMLCode

```
  Tddg6HTMLCode = class(Tddg6BaseWebSnapComponent)
  private
    FHTML: TStrings;
    procedure SetHTML(const Value: TStrings);
  protected
    function GetHTML: string; override;
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
  published
    property HTML: TStrings read FHTML write SetHTML;
  end;

constructor Tddg6HTMLCode.Create(AOwner: TComponent);
begin
  inherited;
  FHTML := TStringList.Create;
end;

destructor Tddg6HTMLCode.Destroy;
begin
  FHTML.Free;
  inherited;
end;

function Tddg6HTMLCode.GetHTML: string;
begin
  Result := FHTML.Text;
end;

procedure Tddg6HTMLCode.SetHTML(const Value: TStrings);
begin
  FHTML.Assign(Value);
end;
```

همانطور که ملاحظه می‌کنید، در اینجا یک کد **HTML** که به صورت رشته‌ای پذیرفته می‌شود برای TAdapterPageProducer فرستاده خواهد شد، تابع GetHTML نیز وظیفهٔ تبدیل کد **HTML** به رشته‌های کاراکتری را به عهده دارد. برای استفاده از این جزء سازنده می‌بایست آن را در Package مربوط

به design–time نصب نمائید. در آن صورت اجزاء سازندهٔ TAdapterPageProducer همانند شکل ۱۸-۱۸ نمایان خواهد شد.



شکل ۱۸-۱۸ اجزاء سازندهٔ مرتبط با TAdapterPageProducer

خلاصه

فرم‌های وب برای جمع‌آوری اطلاعات از کاربران مورد استفاده قرار می‌گیرند. همانطور که ملاحظه کردید، یکی از آسان‌ترین روشهای ایجاد برنامه‌های سرویس‌دهندهٔ وب در دلفی، استفاده از ویژگیهای WebSnap بود.

WebSnap یک چارچوب کاری منظم و قدرتمند را برای طراحی سرویس‌دهنده‌های وب در اختیار کاربران می‌گذارد. حرف آخر این که دلفی یکی از بهترین ابزارهای موجود برای تولید برنامه‌های کاربردی وب سرویس‌دهنده/ سرویس‌گیرنده است.

فصل نوزدهم

طراحی Wireless در دلفی

در این فصل می‌خوانید

- **تاریخچهٔ تکنولوژی اطلاعات**
- **ابزارهای Wireless**
- **به‌کارگیری Wireless در دلفی**

رشد و گسترش فزایندهٔ تکنولوژی اطلاعات، تحولی شگرف در دنیای امروز ایجاد نموده است. تحولات اخیر به ویژه مجوعه پیشرفتهایی که در جهان ارتباطات رخ داده است نوید آن را می‌دهند که جامعهٔ بشری وارد عصر نوینی به نام جهانی شدن اطلاعات و یا دهکدهٔ جهانی شده است.

امروزه، کمتر کسی را می‌توان یافت که با مفهوم اینترنت و یا وسیله‌ای همچون موبایل آشنایی نداشته باشد. هر چند که برخی اینترنت را بستری ناامن در امر ارتباطات می‌دانند، اما کاربرد و نحوه به کارگیری این پدیده در مسائل روزمره زندگی نظیر خرید و فروش، مکاتبه، آموزش، تفریح، درمان و ... آنچنان بوده است که تفکر از دست دادن آن هر انسانی را به وحشت می‌اندازد.

تکنولوژی موبایل نیز همسو با پدیدهٔ اینترنت تحولات عظیمی را در عرصهٔ ارتباطات ایجاد نموده است. در این فصل به تشریح زیربناهای سخت‌افزاری و نرم‌افزاری استفاده از این دو پدیده خواهیم پرداخت و با ارائهٔ برنامه‌های کاربردی در عمل نیز همراه شما خواهیم بود.

به‌کارگیری تکنولوژی‌های ذکر شده، انعطاف‌پذیری و جذابیت حقیقی را برای تولیدکنندگان نرم‌افزار و سخت‌افزار به همراه دارد، اما به لحاظ گسترش و تنوع ابزارهای موجود ممکن است تولیدکنندگان را دچار تردید نماید. تکنولوژی بهینه کدام است؟ کدام یک برای انتقال اطلاعات امن‌تر می‌باشد؟ مزیت هر کدام از آنها چیست، اینها سؤالاتی است که باید به دنبال پاسخی مناسب برای آنها بود.

مسلماً پاسخگویی به تمام جوانب این سؤالها، کمی مشکل خواهد بود، اما با توجه به مفاهیم

Wireless در دلفی دو جنبهٔ مهم‌تر آنها را بررسی خواهیم نمود. ابتدا شما را با تکنولوژی‌های سخت‌افزاری و نرم‌افزاری این کار آشنا نموده و بعد از آن نحوهٔ بکارگیری آنها را در دلفی آموزش خواهیم داد.

تکنولوژی اطلاعات

در این قسمت تکنولوژی اطلاعات از دیدگاه تاریخی آن مرور خواهد شد. از آنجا که داشتن اطلاعات در این زمینه کمکی به ارائه یک دید کلی نسبت به موقعیت این تکنولوژی خواهد بود، در اینجا مروری بر نحوهٔ تکامل آن خواهیم داشت.

اوایل دهه ۱۹۸۰

قبل از سال ۱۹۸۰، مقولهٔ تکنولوژی اطلاعات محدود به سیستم‌های MainFrame می‌شد. مشکل عمدهٔ این سیستم‌ها، هزینه بسیار بالای آنها و عدم کارآیی متناسب با این هزینه‌ها بود. این مشکلات طراحان را بر آن داشت تا روش بهینه‌تری را برای انتقال اطلاعات برگزینند.

اواخر دههٔ ۱۹۸۰

اواخر دههٔ ۱۹۸۰ ، سالهای ارائهٔ بسته‌های نرم‌افزاری بود. این بسته‌ها که هر کدام برای منظوری خاص تهیه می‌شد، روز به روز در گونه‌های متنوع‌تر آن به بازار عرضه می‌گردید. نرم‌افزارهایی همچون dBASE ، Paradox ، Foxpro هر روز تکنیک‌های جدیدی را برای طراحان بانک‌های اطلاعاتی به ارمغان می‌آورند. ظهور زبانهای نسل سوم نظیر C ، Pascal و Basic به عنوان یک انقلاب در دنیای نرم‌افزار مطرح می‌گردید. در این سالها تنها، شبکه‌های محلی برای فعالیتهای تجاری مورد استفاده قرار می‌گرفتند.

اوایل دههٔ ۱۹۹۰

حضور مدل سرویس‌گیرنده/ سرویس‌دهنده در اوایل دههٔ ۱۹۹۰ تحول دیگری را در زمینه تکنولوژی اطلاعات رقم زد. تفکر ارائه این تکنولوژی توسط کمپانی‌های قدرتمندی همچون Oracle ، Sybase ، Informix به کاربران این امکان را می‌داد که در محل کار خود به یک شبکه مرتبط شده و عملیات موردنظر خود را به انجام برسانند. حضور زبانهای نسل چهارم نظیر Delphi و Visual Basic در این سالها کار طراحی بسته‌های نرم‌افزاری را راحت‌تر و ضریب کارآئی بانک‌های اطلاعاتی را ارتقاء بخشید.

اواخر دههٔ ۱۹۹۰

مشکل عمدهٔ مدل سرویس‌گیرنده/ سرویس‌دهنده، محل نصب و توزیع و به کارگیری آن بود وهمیشه این سؤال مطرح می‌شد که دستگاه سرویس‌گیرنده محل مناسبی برای این کار است یا دستگاه

سرویس‌دهنده؟ این مشکل با ارائه سیستم Multitier که سرویس‌گیرنده و سرویس‌دهنده را به صورت منطقی و فیزیکی از هم جدا می‌نمود، مرتفع گردید. ابزارهایی همچون CORBA ، COM ، EJB (که در فصلهای ۱۳ و ۱۴ به آنها اشاره شد) نیز تحولات دیگری را در طراحی بسته‌های نرم‌افزاری موجب گردیدند. این ابزارها در نحوهٔ مدیریت، پردازش و انتقال اطلاعات در اینترنت کاربرد داشتند.

اوایل دههٔ ۲۰۰۰ ، سالهای ظهور ابزارهای Wireless

حجم روزافزون اطلاعات در اینترنت نگران کننده به نظر می‌رسید. تنوع اطلاعات و درخواست آنها به همراه افزایش تعداد کاربران، طراحان را بر آن داشت تا اطلاعات را بنا بر جنبه‌های مختلف آن اختصاصی نمایند. با ارائه تکنولوژی موبایل و PDA[1] ، امکان برقراری ارتباط خارج از محیط کار و منزل نیز برای کاربران مهیا شد.

ابزارهای Wireless

امروزه تمام کسانی که از موبایل، PDA و یا Pager های کوچک استفاده می‌کنند، قادرند تا ارتباطی پیوسته با محیط کار، منزل و ... خودشان داشته باشند و از این راه به کنترل عملیاتهای روزمره خود بپردازند.

Wireless یک مبحث جدید است و دلفی از WAP[2] پشتیبانی می‌کند. این بدان معنی است که دلفی امکان ایجاد برنامه‌های کاربردی سرویس‌دهنده‌ای را فراهم می‌سازد که قادر به سرویس‌دهی به صفحات WML[3] می‌باشند. ابزارهای Wireless همچون موبایل‌ها دارای پهنای کوچک (حدود ۹/۶ تا ۱۴/۴ کیلو بایت) می‌باشند. (با پیشرفت تکنولوژی ممکن است این پهنا تا حدود ۲ مگابایت افزایش یابد).

ابزارهای PalmOS

سیستم عامل PalmOS که همزمان با PDA ها به بازار عرضه گردید، یکی دیگر از سیستم‌هایی است که از Wireless پشتیبانی می‌نمود.این ارتباط از طریق مودم و یا گوشی‌های موبایلی که قابلیت اتصال به سیستم Palm را داشتند، میسر می‌گردید.

PC های جیبی

چندی پیش کمپانی‌های Compaq ، HP ، Caiso یک PDA جدید تحت سیستم عامل‌های PC ارائه نمودند.

هر چند که این ابزارها همچون ابزارهای قبلی قابلیت کاربردی ندارند، اما از Wireless پشتیبانی

۱- Personal Digital Assistance

۲- Wireless Application Protocol

۳- Wireless Markup Language

می‌کنند. نکته حائز اهمیت در این PCها و یا مزیت آنها نسبت به سایر ابزارها، به کارگیری کارتهای استاندارد سخت‌افزاری است که قابلیت اتصال به کامپیوتر را دارند. این توانائی انعطاف‌پذیری بیشتری را در زمینه‌های شبکه به همراه دارد.

RIM BlackBerry

RIM BlackBerry عموماً در سرویس‌های پست الکترونییک موبایل کاربرد دارد.

تکنولوژی رادیویی

این تکنیک‌ها، ارتباط بین موبایل‌ها و اینترنت را فراهم می‌آورند. البته ممکن است از آنها جهت ارتباط با شبکه‌های محلی (LAN) نیز استفاده گردد.

GSM, CDMA, TDMA

موبایل‌ها از تکنولوژی‌های بسیار ابتدایی جهت برقراری ارتباط استفاده می‌کنند. استاندارد این ارتباطات در ایالات متحده براساس TDMA ، CDMA بوده و در سایر نقاط جهان بر اساس GSM پایه‌ریزی می‌گردد. جزئیات این تکنولوژی از دیدگاه توسعه‌دهندگان نرم‌افزار چندان اهمیتی ندارد. سرعت انتقال اطلاعات در این نوع از شبکه‌ها بین ۹/۶ تا ۱۴/۴ کیلو بایت می‌باشد.

CDPD

یکی دیگر از تکنیک‌هایی که جهت انتقال داده‌ها کاربرد دارد، CDPD می‌باشد. سرعت آن حدود ۱۹/۲ کیلو بایت است.

3G

3G یا نسل سوم شبکه‌های موبایل، برای دسترسی راحت‌تر به داده‌ها مورد استفاده قرار می‌گیرد. سرعت این شبکه‌ها بین ۳۸۴ کیلو بایت تا ۲ مگا بایت می‌باشد.

GPRS

GPRS در حقیقت یک جهش از 2G به 3G است و برخی از کارشناسان آن را با نام 2.5G می‌شناسند. GPRS ترافیک کمتری در شبکه خود دارد.

Bluetooth

کاربرد عمده Bluetooth در انجام عملیات همزمانی بین داده‌های کامپیوتری و PDAهاست. این یک پروتکل رادیویی است که مورد استفاده کمتری دارد. مزیت Bluetooth در هزینهٔ پایین آن است. برخی

از آن به عنوان جایگزینی برای پورتهای USB و شبیه‌سازی آنها در کنترل شبکه نام می‌برند.

802.11

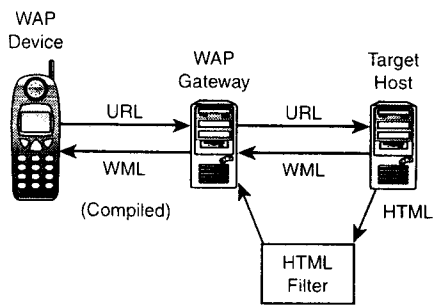
802.11 از پروتکلهایی است که در شبکه‌های محلی (LAN) مورد استفاده قرار می‌گیرند. نسل جدید این پروتکل 802.11b یا WiFi بوده که پهنای باندی برابر با ۱۱ مگا بایت دارد. قدرت این پروتکل به مراتب بیشتر از پروتکل Bluetooth می‌باشد. ضمناً شما قادر خواهید بود که همزمان از دو پروتکل 802.11 و Bluetooth استفاده نمائید.

دلفی و Wireless

دلفی به چه طریقی از Wireless استفاده می‌کند؟ برنامه‌های کاربردی WAP قادرند داده‌ها را از طریق URL درخواست نمایند. وقتی برنامه‌های کاربردی Wireless ایجاد می‌کنید با زبانهایی همچون HTML ، XML و WML نیز سروکار خواهید یافت. تشابه زیادی بین tag های این زبانها وجود دارد. در اینجا می‌خواهیم شما را با WAP بیشتر آشنا سازیم.

کاربرد پروتکل WAP که دلفی از آنها بهره می‌گیرد، در راستای دسترسی به اطلاعات اینترنتی است. البته باید بدانید که WAP دارای مشکلاتی نظیر محدودیت در نحوهٔ نمایش و روشهای ورود داده‌ها نیز می‌باشد. منتهی این معایب به گونه‌ای جزئی است که چیزی از مزایای این پروتکل نمی‌کاهد.

در شکل ۱-۱۹ ، معماری و ساختار داخلی یک WAP نمایش داده شده است.



شکل ۱-۱۹ نمونه‌ای از ساختار یک WAP

WML زبانی برای پروتکل WAP

همانطور که قبلاً اشاره کردیم، اطلاعات در WAP با استفاده از WML دستکاری و پردازش می‌شوند. WML شباهت زیادی به HTML و XML دارد. اگر با HTML یا XML آشنائی دارید در آن صورت کار با WML برایتان بسیار ساده خواهد بود. هر سند WML حاوی مجموعه‌ای از کارت‌ها می‌باشد. هر کارت معادل یک صفحه از داده‌هاست و یا به عبارت دیگر هر کارت معادل یک صفحهٔ HTML است.

همانطور که قبلاً نیز متذکر شدیم به دلیل محدودیت صفحهٔ نمایش WAP ، حجم و نوع محتوایی که برای هر کارت تعریف می‌شود به طور قابل ملاحظه‌ای کمتر از میانگین سندهای HTML است. هر سند WML دارای یک مقدمه، دو tag آغازین و پایانی و حداقل یک جفت <card></card> است.

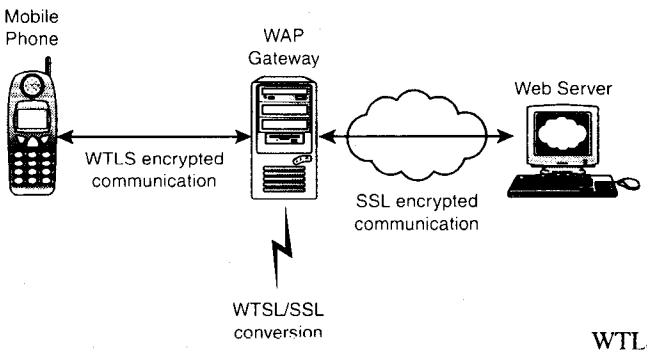
فهرست ذیل یک کد WML را نشان می‌دهد.

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.WAPforum.org/DTD/wml_1.1.xml">
<wml>
  <card>
    <do type="accept">
      <go href="#hello"/>
    </do>
    <p>Punch the Button</p>
  </card>
  <card id="hello">
    <p>Hello from WAP!</p>
  </card>
</wml>
```

اگر تمایل دارید اطلاعات بیشتری در خصوص WML بدست آورید می‌توانید به سایتهای **http://www.WAPforum.org** و **http://www.openwave.com** مراجعه نمائید.

امنیت در WAP

پروتکل WAP از سرویس WTLS[1] برای تضمین امنیت خود استفاده می‌کند. WTLS بین یک درگاه WAP و سیستم گیرنده قرار گرفته و عملیات مربوط به بسته‌بندی داده‌ها و همچنین خصوصی‌سازی آنها را انجام می‌دهد. امنیت شبکه از درگاه WAP تا سرویس‌دهندهٔ اینترنت توسط سرویس SSL برقرار می‌شود.



شکل ۲-۱۹ ساختار WTLS

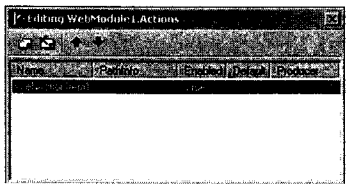
۱- Wireless Transport Layer Security

طراحی یک WAP

ساخت و طراحی WAP کمی با طراحی سرویس‌دهنده‌های وب در دلفی تفاوت دارد. به عبارت دیگر طراحی WAP در دلفی ساده‌تر از ساخت سرویس‌دهنده‌های وب می‌باشد. یک برنامهٔ کاربردی WAP در دلفی را می‌توان با استفاده از اجزاء ساخت WebBroker ایجاد نمود. (این کار را در فصل ۱۸ آموخته‌اید).

در اینجا یک مثال کاربردی از WAP را با استفاده از WebBroker ارائه می‌نمائیم.

همانطور که در شکل ۳-۱۹ ملاحظه می‌کنید فیلد Default این ماژول وب علامت زده شده است.



شکل ۳-۱۹ یک WAP کاربردی

کد ارائه شده در لیست ۱-۱۹ مربوط به یونیت اصلی این ماژول می‌باشد که توسط رویداد OnAction فراخوانی خواهد شد.

لیست ۱-۱۹ یونیت اصلی پروژه SimpWap

```
unit Main;

interface

uses
  SysUtils, Classes, HTTPApp;

type
  TWebModule1 = class(TWebModule)
    procedure WebModule1WebActionItem1Action(Sender: TObject;
      Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  WebModule1: TWebModule1;

implementation

{$R *.DFM}
```

لیست ۱-۱۹ ادامه

```
const
  SWMLContent = 'text/vnd.wap.wml';
  SWMLDeck =
    '<?xml version="1.0"?>'#13#10 +
    '<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"'#13#10 +
    '"http://www.WAPforum.org/DTD/wml_1.1.xml">'#13#10 +
    '<wml>'#13#10 +
    '  <card>'#13#10 +
    '    <do type="accept">'#13#10 +
    '      <go href="#hello"/>'#13#10 +
    '    </do>'#13#10 +
    '    <p>Punch the Button</p>'#13#10 +
    '  </card>'#13#10 +
    '  <card id="hello">'#13#10 +
    '    <p>Hello from WAP!</p>'#13#10 +
    '  </card>'#13#10 +
    '</wml>'#13#10;


procedure TWebModule1.WebModule1WebActionItem1Action(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  Response.ContentType := SWMLContent;
  Response.Content := SWMLDeck;
end;

end.
```

همانطور که ملاحظه می‌کنید خصوصیت ContentType و Content حاوی رشته‌هایی از نوع WML می‌باشند.

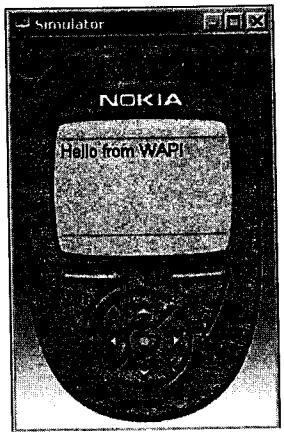
یادداشت

به خاطر داشته باشید که محتوای ContentType رشته‌هایی از نوع MIME[1] می‌باشند. این محتویات مربوط به HTTP است و هر گونه رشتهٔ نامعتبر در این خصوص دلالت بر نقص ابزارهای ارتباطی دارد. برخی از این رشته‌ها در اینجا ارائه شده است.

- text/vnd.wap.wml نمایانگر قطعه کدهای WML خواهد بود.
- text/vnd.wap.wmlscript نمایانگر کدهای مربوط به یک اسکریپت WML می‌باشد.
- image/vnd.wap.wbmp نمایانگر تصاویر Wireless خواهد بود. ▲

۱- Multi Purpose Internet Mail Extension

شکل ۴-۱۹ یک WAP کاربردی را در هنگام اجرای آن نشان می‌دهد.



شکل ۴-۱۹ یک WAP کاربردی

نحوه گزارش خطاها

اطلاعات مربوط به خطاهای اجرایی یک برنامهٔ کاربردی WebSnap توسط یک استثناء و در قالب یک پیام HTML ارسال می‌گردد. البته اغلب ابزارهای WAP قادر به درک پیام‌های HTML نیستند و لذا باید توجه داشته باشید که این پیام‌ها در قالب WML ارسال گردد. تبدیل پیام‌های نوع HTML به WML توسط فراخوانی رویدادی نظیر OnAction انجام می‌پذیرد. مراحلی از انجام این عملیات را در لیست ۲-۱۹ مشاهده خواهید نمود.

لیست ۲-۱۹ نحوهٔ بکارگیری wbmp در برنامه‌های WAP کاربردی

```
unit Main;

interface

uses
  SysUtils, Classes, HTTPApp;

type
  TWebModule1 = class(TWebModule)
    procedure WebModule1WebActionItem1Action(Sender: TObject;
      Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
    procedure WebModule1GraphActionAction(Sender: TObject;
      Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
  private
    procedure CreateWirelessBitmap(MemStrm: TMemoryStream);
```

لیست ۲-۱۹ ادامه

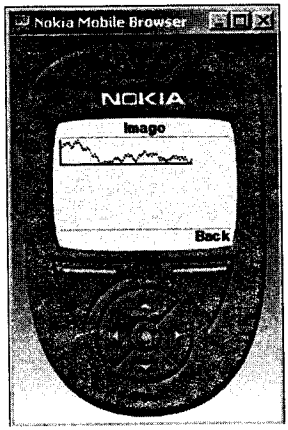
```
    procedure HandleException(e: Exception; Response: TWebResponse);
  end;

var
  WebModule1: TWebModule1;

implementation

{$R *.DFM}
const
  SWMLContent = 'text/vnd.wap.wml';
  SWBMPContent = 'image/vnd.wap.wbmp';
  SWMLDeck =
    '<?xml version="1.0"?>'#13#10 +
    '<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"'#13#10 +
    '"http://www.WAPforum.org/DTD/wml_1.1.xml">'#13#10 +
    '<wml>'#13#10 +
    '  <card>'#13#10 +
    '    <do type="accept">'#13#10 +
    '      <go href="#hello"/>'#13#10 +
    '    </do>'#13#10 +
    '    <p>Punch the Button</p>'#13#10 +
    '  </card>'#13#10 +
    '  <card id="hello">'#13#10 +
    '    <p>Hello from WAP!</p>'#13#10 +
    '  </card>'#13#10 +
    '</wml>'#13#10;

  SWMLError =
    '<?xml version="1.0"?>'#13#10 +
    '<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"'#13#10 +
    '"http://www.wapforum.org/DTD/wml_1.1.xml">'#13#10 +
    '<wml>'#13#10 +
    '  <card id="error" title="SimpWAP">'#13#10 +
    '    <p>Error: %s'#13#10 +
    '      <do type="prev" label="Back">'#13#10 +
    '        <prev/>'#13#10 +
    '      </do>'#13#10 +
    '    </p>'#13#10 +
    '  </card>'#13#10 +
    '</wml>'#13#10;



procedure TWebModule1.HandleException(e: Exception; Response: TWebResponse);
begin
  Response.ContentType := SWMLContent;
  Response.Content := Format(SWMLError, [e.Message]);
end;
```

لیست ۲-۱۹ ادامه

```
procedure TWebModule1.WebModule1WebActionItem1Action(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  try
    Response.ContentType := SWMLContent;
    Response.Content := SWMLDeck;
  except
    on e: Exception do
      HandleException(e, Response);
  end;
end;

procedure TWebModule1.WebModule1GraphActionAction(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
var
  MemStream: TMemoryStream;
begin
  try
    MemStream := TMemoryStream.Create;
    try
      CreateWirelessBitmap(MemStream);
      MemStream.Position := 0;
      with Response do
      begin
        ContentType := SWBMPContent;
        ContentStream := MemStream;
        SendResponse;
      end;
    finally
      MemStream.Free;
    end;
  except
    on e: Exception do
      HandleException(e, Response);
  end;
end;

procedure TWebModule1.CreateWirelessBitmap(MemStrm: TMemoryStream);
const
  Header : Array[0..3] of Char = #0#0#104#20;
var
  Bmp: array[1..104,1..20] of Boolean;
  X, Y, Dir, Bit: Integer;
  B: Byte;
```

لیست ۲-۱۹ ادامه

```
  begin
    { clear the bitmap out }
    FillChar(Bmp,SizeOf(Bmp),0);
    { draw X and Y axis }
    for X := 1 to 104 do Bmp[X, 20] := True;
    for Y := 1 to 20 do Bmp[1, Y] := True;
    { draw random data }
    Y := Random(20) + 1;
    Dir := Random(10);
    for X := 1 to 104 do
    begin
      Bmp[X,Y] := True;
      if (Dir > 4) then Y := Y+Random(2)+1
      else Y := Y - Random(2) - 1;
      if (Y > 20) then Y := 20;
      if (Y < 1) then Y := 1;
      Dir := Random(10);
    end;
    { create WBMP data }
    MemStrm.Write(Header, SizeOf(Header));
    Bit := 7;
    B := 0;
    for Y := 1 to 20 do
    begin
      for X := 1 to 104 do
      begin
        if Bmp[X,Y] = True then
          B := B or (1 shl Bit);
        Dec(Bit);
        if (Bit < 0) then begin
          B := not B;
          MemStrm.Write(B, SizeOf(B));
          Bit := 7;
          B := 0;
        end;
      end;
    end;
  end;

initialization
  Randomize;
end.
```



شکل ۵-۱۹ نحوهٔ نمایش یک تصویر wbmp

تصاویر BMP و Wrieless

امکان به کارگیری فایل‌های تصویری از نوع JPEG و GIF در WAP وجود ندارد و WAP تنها از فایل‌های تک رنگ BMP در قالب فایل‌های wbmp[1] پشتیبانی می‌کند. بخشی از لیست ۲-۱۹ مربوط به نحوهٔ به کارگیری این تصاویر است. با اجرای آن خواهید دید که تصاویر به صورت تصادفی بر روی صفحهٔ نمایش ظاهر می‌شوند.

شکل ۵-۱۹ یک تصویر wbmp را بر روی صفحه نمایش یک تلفن نشان می‌دهد.

یادداشت

در نظر داشته باشید که تصاویر wbmp تنها توسط برخی از گونه‌های WAP پشتیبانی می‌شوند و ممکن است بعضی از آنها امکان به کارگیری wbmpها را نداشته باشند.

I-mode

یکی دیگر از تکنولوژی‌های به کار برده شده برای ارتباطات اینترنتی و موبایل‌ها، I-mode می‌باشد. شرکت NTTDoCoMo طراح I-mode معتقد است که این تکنولوژی در ژاپن قریب به بیست میلیون مشترک دارد. این تکنولوژی همخوان با پروتکل TCP/IP می‌باشد و قادر به پشتیبانی از فایل‌های تصویری تا ۲۵۶ رنگ می‌باشد. برای دریافت اطلاعات بیشتر در خصوص این تکنولوژی به آدرس **http://www.nttdocomo.com** مراجعه نمائید.

۱- wireless bitmaps

PQA[1]

PQA وجه تشابه زیادی با صفحات HTML دارد و از آن جهت نمایش صفحاتی مشابه با HTML ها بر روی سرورهای PalmOS استفاده می‌شود. ابزارهای Wireless که توسط سیستم عاملی همچون PalmOS پشتیبانی می‌شوند، تنها در آمریکای شمالی وجود داشته و مورد استفاده قرار می‌گیرد. در صورتی که به خلق صفحات PQA علاقه دارید می‌توانید به آدرس **http://www.palmos.com/dev/tech/docs** مراجعه نمائید.

همانطور که گفتیم کد PQA همانند کد HTML است ولی PQA قابلیت به کارگیری applet را ندارد. همچنین tag های LINK ، SUB ، SUP ، ISINDEX و VSPACE نیز که در HTML وجود دارند، در PQA غیرمعتبر می‌باشند.

PQA بر روی client

اولین قدم برای ایجاد یک صفحهٔ PQA ، تهیهٔ یک قطعه کد است که قابلیت ذخیره شدن بر روی ابزارهای client را داشته باشد. این کار تنها با ایجاد یک سند HTML و کامپایل آن با نرم‌افزار PQA Builder امکان‌پذیر است. لیست ۳-۱۹ یک سند HTML را برای صفحهٔ PQA نمایش می‌دهد.

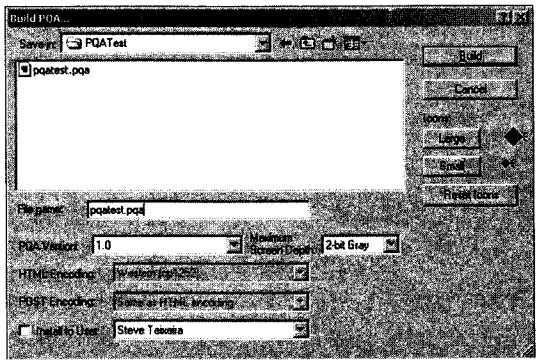
لیست ۳-۱۹ سند HTML برای صفحهٔ PQA

```
<html>
<head>
<title>DDG PQA Test</title>
<meta name="palmcomputingplatform" content="true">
</head>
<body>
<p>This is a sample PQA for DDG</p>
<img src="image.gif">
<form method="post" action="http://128.64.162.164/scripts/pqatest.dll">
<input type="hidden" value="%zipcode" name="zip">
<input type="hidden" value="%deviceid" name="id">
<input type="submit">
</form>
</body>
```

نحوهٔ کامپایل کردن این قطعه کد در شکل ۶-۱۹ نمایش داده شده است.

هنگامی که یک سند PQA را کامپایل می‌کنید، یک فایل با پسوند pqa ایجاد خواهد شد. این فایل به عنوان یک سند HTML در نظر گرفته می‌شود که در آن ارجاعاتی به یک سری تصاویر وجود دارد. این

۱- Palm Query Application



شکل ۶-۱۹ کامپایل کردن کد PQA با استفاده از PQA Builder

فایل می‌تواند بر روی یک سیستم که از PalmOS استفاده می‌کند نصب گردد.

PQA بر روی سرور

بر روی سرور که همان WAP می‌باشد، یک برنامهٔ کاربردی وب عملیات درخواستی از clientها را پاسخ می‌دهد. لیست ۴-۱۹ یونیت موجود بر روی سرور راکه با استفاده از WebBroker ایجاد شده است نمایش می‌دهد.

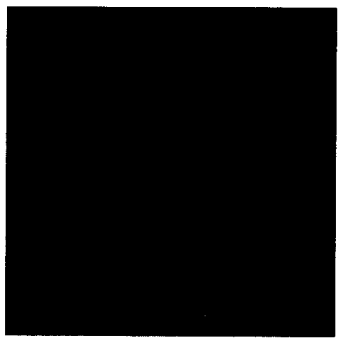
لیست ۴-۱۹ یک برنامهٔ کاربردی به نام PQATest

```
unit Main;

interface

uses
  SysUtils, Classes, HTTPApp;

type
  TWebModule1 = class(TWebModule)
    procedure WebModule1WebActionItem1Action(Sender: TObject;
      Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  WebModule1: TWebModule1;

implementation
```

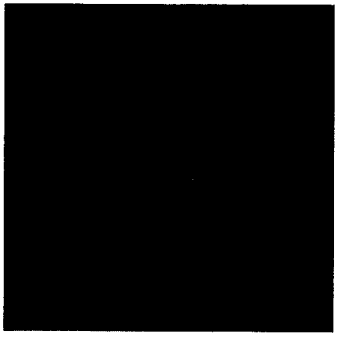
لیست ۴-۱۹ ادامه

```
{$R *.DFM}

const
  SPQAResp =
    '<html><head><meta name="palmcomputingplatform" content="true"></head>'+
    #13#10 +
    '<body>Hello from a Delphi server<br>Your zipcode is: %s<br>'#13#10 +
    'Your device ID is: %s<br><img src="file:pqatest.pqa/image.gif"></body>'+
    '</html>';

procedure TWebModule1.WebModule1WebActionItem1Action(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  Response.Content := Format(SPQAResp, [Request.ContentFields.Values['zip'],
    Request.ContentFields.Values['id']]);
end;

end.
```

عملیات پاسخ‌دهی به درخواست‌های client توسط یک zipکد و یک شمارهٔ شناسایی ابزار یا همان ID انجام می‌گیرد. در کد بالا یک تصویر نیز فراخوانی شده است که بر روی فرم موردنظر نمایش داده می‌شود. شکل ۷-۱۹ طریقهٔ اجرای این کد را نمایش می‌دهد. با فشردن دکمهٔ submit بر روی فرم، شکلی همانند شکل ۸-۱۹ نمایان خواهد کرد.



شکل ۷-۱۹ اجرای برنامهٔ PQATest بر روی یک سیستم PalmOS



شکل ۸-۱۹ PQATest پس از فشردن دکمهٔ Submit

کاربران سیستم‌های Wireless

تجربیات و نظرات کاربران در نحوهٔ استفاده از سیستم‌های Wireless ، مهمترین عامل جهت تشخیص

کارآیی آنهاست. در دنیای امروز، کاربران مهمترین عامل برای طراحی و ساخت ابزارهای کاربردی می‌باشند و توسعهٔ تکنولوژی Wireless هدفی جز راحتی کاربران و توسعهٔ کارآیی سیستم‌هایشان ندارد. در اینجا لازم است نکته‌ای را جهت کارآیی بهتر ابزارهای WAP یادآور شویم. از آنجاکه در ابزار WAP با پهنای باند محدود سروکار داریم لذا پاسخ به هر درخواست متناسب با این پهنای باند صورت می‌گیرد.

شبکه‌های Circuit–Switched و Packet–Switched

در شبکه‌های Circuit–Switched ارتباط از طریق سیستم تلفن و به وسیلهٔ ایجاد یک پیوند فیزیکی بین طرف گیرنده و فرستنده برقرار می‌شود. این روش عموماً در شبکه‌های تلفنی که در آنها اتصال از طریق شماره‌گیری انجام می‌شود مورد استفاده قرار می‌گیرد. شبکه‌های Packet–Swithed اطلاعات را در واحدهای کوچک مدیریت می‌کند، یعنی پیامها را پیش از ارسال به چندین بسته تجزیه می‌کند که این بسته‌ها با استفاده از ایستگاههای موجود در شبکه‌های کامپیوتری از طریق بهترین مسیر قابل دسترسی، در بین مبدأ و مقصد ارسال می‌شود. اینترنت نمونه‌ای از این شبکه‌ها است.

خلاصه

در این فصل آموختید که چگونه دلفی شما را در آماده‌سازی راه‌حلهای پویای وب برای ابزارهای Wireless یاری می‌رساند. با ارائه تفاوتهای بین برنامه‌های کاربردی سرویس‌دهندهٔ WAP و برنامه‌های کاربردی سرویس‌دهندهٔ وب در این فصل سعی نمودیم تا شما را در انتخاب بهترین روش پیاده‌سازی برنامه‌های سرویس‌دهندهٔ وب کمک نمائیم.

ضمیمهٔ الف

NET. و دلفی ۷

NET. جدیدترین محیط توسعه Microsoft است. چارچوب کاری NET. تشکیل شده از CLR[1] یا اصطلاحاً موتوری است که سیستم را هدایت می‌کند. CLR یک سیستم استاندارد از داده‌ها و اشیاء فراهم می‌نماید و بدین ترتیب امکان توسعه مستقل از زبان و مستقل از سیستم را میسر می‌سازد. لایهٔ میانی NET. دربرگیرندهٔ سرویس‌هایی نظیر ADO ، ASP و XML است و از این طریق قابل دسترس بودن آنها را در تمامی زبان‌ها به صورت یکسان در می‌آورد.

بالاترین لایهٔ NET. دربرگیرندهٔ کاربران و رابطهای برنامه‌هاست. سرویس‌های وب یکی دیگر از مهمترین ویژگی‌های NET. است. این سرویس‌ها، متدهایی را برای دسترسی به اشیاء راه دور از طریق اینترنت فراهم می‌کنند.

کلاس‌های NET.

کلاس‌های NET. توابعی را پیاده‌سازی می‌کنند که در زبان‌های سازگار با NET. از آنها استفاده می‌شود. این کلاس‌ها مجموعهٔ گسترده‌ای از نیازها را برآورده می‌سازند. نمونه‌هایی از امکانات قابل ارائه به وسیلهٔ این کلاس‌ها در زیر ارائه شده است.

- ایجاد توابع استاندارد زبان.
- دسترسی داده‌ای با قدرت بالا.
- مدیریت امنیت برنامه‌ها.
- ایجاد رابطهای Windows .
- کار با سرویس‌های Windows ، زمان‌سنج‌ها و پیام‌ها.
- پشتیبانی پروتکل‌های شبکه‌ای.

NET. و دلفی ۷

دلفی ۷ با پشتیبانی از NET. ، جنبه‌های پیشرفته سایر زبان‌های برنامه‌نویسی را به دست آورده است.

۱- Common Language Runtime

این کار به سادگی و با به کارگیری نوع CoClass در ویراستار Type Library دلفی ۷ امکان‌پذیر است. در خصوص نوع CoClass و نحوهٔ به کارگیری و ایجاد آن در فصل‌های ۱۳ و ۱۴ توضیحات مفصلی ارائه شده است.

با استفاده از کلاس‌های NET. در دلفی ۷ قادر خواهید بود سرویس‌های Windows را نیز طراحی نمائید. منظور از سرویس‌های Windows ، نرم‌افزارهایی است که اعمال پشت پرده را در سیستم‌های Windows NT ، Windows 2000 یا Windows XP انجام می‌دهند. این سرویس‌ها به شما امکان می‌دهند تا به سادگی برنامه‌هایی نظیر برنامه‌های سرویس‌دهنده اینترنت طراحی نمائید.

ضمیمهٔ ب

آماده‌سازی و ایجاد فایل‌های DLL

برنامه‌های کاربردی مبتنی بر ویندوز به دو صورت عرضه می‌شوند. یکی از آنها فایل‌های اجرایی با پسوند exe است که به صورت مستقل قابل اجرا می‌باشند. شکل دیگر برنامه‌های کاربردی تحت ویندوز، فایل‌هایی با پسوند DLL[1] می‌باشد که توسط برنامه‌های کاربردی دیگر به کار گرفته و اجرا می‌شوند. فایل‌های DLL تنها به هنگام نیاز در یک برنامه بارگذاری می‌شوند. این فایل‌ها چندین مزیت دارند. نخست این که تا زمانی که به کار برده نشوند، حافظه‌ای مصرف نمی‌کنند. دوم این که، چون هر DLL یک فایل جداگانه است، برنامه‌نویس می‌تواند بدون هرگونه تأثیر بر عملکرد برنامهٔ فراخوان یا فایل‌های دیگر، تنها همان فایل تصحیح یا تغییر دهد و در نهایت این که یک برنامه‌نویس می‌تواند فایل‌های DLL را در برنامه‌های دیگر نیز به کار ببرد.

ایجاد فایل‌های DLL

برای ایجاد فایل‌های DLL می‌توانید از امکانات و ابزارهای ارائه شده در دلفی ۷ استفاده نمائید. این کار به سادگی و به وسیلهٔ انتخاب DLL wizard از برگهٔ New منوی File امکان‌پذیر است. با انجام این کار متن یک فایل DLL به شما نمایش داده خواهد شد. نمونه‌ای از یک فایل DLL در زیر ارائه شده است و همانطور که مشاهده می‌کنید به جای عبارت Program در ابتدای برنامه، عبارت Library وجود دارد. این عبارت به کامپایلر فرمان می‌دهد که برنامهٔ کاربردی را به صورت یک DLL بسازد. قطعه کد زیر شیوهٔ ایجاد یک DLL را نمایش می‌دهد.

```
Libary TestDLL;
Uses ShareMem, Sysutils, Dialogs, Classes;
{$R * .RES}
Procedure Test;
begin
```

۱- Dynamic Link Libray

```
        ShowMessage ('Test');
end;
exports
   Test;
begin
end.
```

دستور exports در قطعه کد فوق، مشخص کنندهٔ روال‌های مربوط به رابط DLL می‌باشد. یعنی روال‌هایی که برنامه‌های کاربردی دیگر قادر به استفاده از آنها خواهند بود.

آزمایش و اشکال‌زدایی DLLها

آزمایش DLLها نیز همانند برنامه‌های کاربردی دیگر انجام می‌پذیرد. پیش از گنجایش DLL در یک پروژه نرم‌افزاری، بهتر است تا یک برنامهٔ آزمایشی را برای بررسی قابلیت‌های فایل‌های DLL ایجاد نمائید.

هنگامی که DLLهای خود را اشکال‌زدایی می‌کنید، تمامی خطاهای زمان اجراء در برگهٔ Complie صفحهٔ Project Options را فعال نمائید.

این خطاها شامل بررسی ورودی/ خروجی، Overflow و ... می‌شود.

اکنون با استفاده از سوئیچ D+ و رهنمود کامپایلر {$IFOPT} {$ENDIF} می‌توانید دستورات اشکال‌زدایی را در برنامهٔ خود قرار دهید.

پیاده‌سازی DLLها

برای پیاده‌سازی DLLها طی کردن سه گام اساسی به شرح زیر لازم است.

۱- ساختن برنامه‌ای (یکسری دستورات) که DLL با آن ایجاد شود.

۲- تعریف روال‌هایی که قابل استفاده برای سایر برنامه‌های کاربران باشد، بدین معنی که قابل صادر کردن باشد.

۳- تهیه یک برنامهٔ کاربردی آزمایشی که روال‌های ایجاد شده را آزمایش کند.

سطح
مبتدی ✓
متوسط ✓
پیشرفته

Delphi 7 Professional Release 2003

The Complete Reference

Delphi 7 Studio Professional

ISBN 964-377-041-9

انتشارات ناقوس