

---

## Delphi XE: Delphi 7 이후의 새로운 언어 기능들

Nick Hodges, 박지훈.임프

2010 년 9 월

---

---

**Americas Headquarters**  
100 California Street, 12th  
Floor  
San Francisco, California  
94111

**EMEA Headquarters**  
York House  
18 York Road  
Maidenhead, Berkshire  
SL6 1SF, United Kingdom

**Devgear**  
서울특별시 반포 1 동 743-14  
4 층 ㈜데브기어  
(T) 02.595. 4288

<b>DLL 지연 로딩</b>	<p>Delphi 2010 에서는 DLL 의 로딩 방법으로서 정적(static) 로딩, 동적(dynamic) 로딩 외에 지연(delayed) 로딩이 추가되었습니다. 지연 로딩은 정적 로딩과 기본적으로 같은 구문을 사용하지만 extern 선언에서 delayed 지시어를 추가하는 것입니다.</p> <pre><b>procedure</b> ProcInDll; <b>external</b> 'DLL_name.dll' <b>delayed</b>;</pre> <p>정적 로딩 구문에 단지 delayed 지시어를 추가한 것 뿐이지만, 이것만으로도 해당 DLL 함수가 실제로 호출될 때까지 해당 DLL 을 로드하지 않습니다.</p> <p>이전의 정적 로딩을 사용하면 코딩이 단순하고 편리한 반면 해당 DLL 이나 함수가 존재하지 않을 경우 프로그램 시작 자체가 실패하는 치명적인 문제가 있습니다. DLL 이 이런 우려가 있을 경우에 지연 로딩을 사용하면 깔끔하게 문제가 해결됩니다. 특히 Windows 7 이나 Windows Vista 등 최신 버전의 Win32 에서 추가된 API 들을 호출할 필요가 있으면서 동시에 Windows P, 2000 등 구버전과도 호환성이 필요한 경우 더욱 유용합니다.</p> <p>또한 지연 로딩을 사용하면 실제 호출할 때까지 DLL 을 로드하지 않으므로 메모리 소모도 줄어드는 효과도 있습니다.</p>
<b>RTTI 강화와 어트리뷰트</b>	<p>델파이 2010 에 이르러 델파이의 RTTI 가 대폭 강화되었습니다. 이제 RTTI 가 메소드, 필드, 속성을 지원하게 되어 동적 호출과 다른 메타 프로그래밍 방식들을 지원하게 되었습니다. 새로 추가된 RTTI.pas 유닛을 통해 강력한 RTTI 루틴들을 뒷받침합니다.</p> <p>어트리뷰트(attribute)는 이 새로운 RTTI 를 기반으로 한 문법으로서, 클래스/레코드나 클래스의 멤버에 추가로 지정되는 정보이며, 실행 중에 읽어질 수 있습니다. 어트리뷰트를 활용하면 일반화된 프레임워크를 개발할 때 아주 유용합니다.</p>
<b>제네릭 (Generics)</b>	<p>제네릭을 이용하면 명시적 타입을 일일이 명시하지 않고도 타입을 사용하는 코드를 작성할 수 있습니다. 제네릭은 흔히 파라미터 타입이라고 불리는 일반적인, 즉 제네릭(generic)한 클래스를 작성하며 구체적으로 명시하지 않은 타입에서 작동합니다. 클래스에서 제네릭을 이용하는 한 예는 리스트입니다. 리스트 클래스의 코드를 작성하는 시점에서 리스트에 포함될 아이템들의 타입을 지정할 필요가 없습니다.</p> <pre> 10  . type .   . .   . TControlReporter&lt;T: TControl, constructor&gt; = class .   . private -   .   FInternalType: T; .   . public .   .   constructor Create; .   .   procedure ProcessComponent; .   .   procedure SetType(aT: T); 20  . end; .   . </pre>



	<p>UnicodeString 은 기존의 WideString 과는 달리 AnsiString 과 유사한 레퍼런스 카운팅(Reference Counting) 구조로 설계되어 훨씬 메모리 효율적이며 속도도 빠릅니다.</p> <p>스트링을 버퍼로 사용해왔던 관행에 대한 대안으로 RawByteString 타입도 추가되었습니다.</p>
<b>RTTI 강화</b>	<p>RTTI (Runtime Type Information: 런타임 타입 정보)는 실행 중에 얻을 수 있는 타입 정보에 대한 프로그래밍 패러다임입니다. RTTI 생성이 활성화된 상태에서는 결과 바이너리는 타입들(예를 들면, 클래스 조상, 선언된 필드들, annotation 된 속성들)에 대한 정보를 가지는 특수한 메타데이터를 포함하게 됩니다. RTTI 유닛에서 제공되는 기능을 이용하면 실행 시에 이 정보를 얻어낼 수 있습니다. 이런 기능들을 이용하면, RTTI 로 노출시킨 모든 타입을 조작할 수 있는 더 추상적이고 일반화된 프레임워크를 만들어낼 수 있게 됩니다.</p>
<b>Exit 프로시저의 인자</b>	<p>이제 델파이의 Exit 에도 C/C++의 return 문처럼 인자를 넘길 수 있게 되었습니다. 기존에</p> <p>Result := 값; Exit; 라고 써야 했던 것이 이제 Exit(값); 으로 줄여 쓸 수 있습니다.</p>
<b>인라이닝</b>	<p>이제 루틴들을 inline 지시자로 표시할 수 있습니다. 이 지시자는 컴파일러에게 해당 루틴을 실제로 호출하는 대신 호출하는 측에서 해당 루틴을 포함하는 코드를 생성하도록 합니다.</p>
<b>연산자 오버로딩</b>	<p>레코드 선언 내에서 특정 함수나 연산자를 오버로드하는 것을 허용합니다.</p> <pre> TMyClass = class   class operator Add(a, b: TMyClass): TMyClass;   // TMyClass 타입 두 피연산자를 더함    class operator Subtract(a, b: TMyClass): TMyClass;   // TMyClass 타입의 빼기 연산    class operator Implicit(a: Integer): TMyClass;   // 정수형을 TMyClass 타입으로 암시적으로 변환    class operator Implicit(a: TMyClass): Integer;   // TMyClass 타입을 정수로 암시적으로 변환    class operator Explicit(a: Double): TMyClass;   // Double 타입을 TMyClass 타입으로 명시적으로 변환  end;  // Add 클래스 연산자의 구현 예시 TMyClass.Add(a, b: TMyClass): TMyClass; begin   ... end;  var   x, y: TMyClass begin </pre>

```
x := 12;    // 정수로부터의 암시적인 변환
y := x + x; // TMyClass.Add(a, b: TMyClass): TMyClass 호출
b := b + 100; // TMyClass.Add(b, TMyClass.Implicit(100)) 호출
end;
```

## 클래스 헬퍼

클래스 헬퍼는 다른 클래스와 연관되어 해당 클래스의 이름으로 사용할 수 있는 추가 메소드 이름과 속성을 도입합니다. 클래스 헬퍼는 상속을 이용하지 않고 클래스를 확장할 수 있는 방법입니다. 클래스 헬퍼를 선언하려면 헬퍼 이름과 그 헬퍼로 확장할 클래스의 이름을 지정하면 됩니다. 확장되는 클래스를 사용할 수 있는 어떤 곳이든 클래스 헬퍼를 사용할 수 있습니다. 컴파일러의 식별 영역(resolution scope)은 원래의 클래스에 클래스 헬퍼를 더한 만큼이 됩니다. 클래스 헬퍼는 클래스를 확장하는 한 방법을 제공하지만, 새로운 코드를 개발할 때 설계 방법으로서 취급되어서는 안 됩니다. 클래스 헬퍼는 오직 의도된 목적으로만 사용되어야 하며, 그것은 언어와 플랫폼 RTL의 바인딩을 위한 것입니다.

```
type
  TMyClass = class
    procedure MyProc;
    function MyFunc: Integer;
  end;

procedure TMyClass.MyProc;
var
  X: Integer;
begin
  X := MyFunc;
end;

function TMyClass.MyFunc: Integer;
begin
  ...
end;

type
  TMyClassHelper = class helper for TMyClass
    procedure HelloWorld;
    function MyFunc: Integer;
  end;

procedure TMyClassHelper.HelloWorld;
begin
  WriteLn(Self.ClassName);
  // 여기서 self는 TMyClassHelper가 아니라 TMyClass 객체.
end;

function TMyClassHelper.MyFunc: Integer;
begin
  ...
end;

var
  X: TMyClass;
begin
  X := TMyClass.Create;
  X.MyProc;    // TMyClass.MyProc 호출
  X.HelloWorld; // TMyClassHelper.HelloWorld 호출
  X.MyFunc;    // TMyClassHelper.MyFunc 호출
end;
```

<b>strict private</b>	private 키워드는 실제로는 동일한 유닛에 포함된 클래스들 사이에서는 "친구 관계"를 허용합니다. strict private 선언은 진정한 private 필드를 만들 수 있게 해주는 것으로, 동일 유닛의 클래스들을 포함해서 다른 클래스에서는 보이지 않습니다.
<b>strict protected</b>	strict private 선언과 비슷한 것으로, strict protected 는 진정한 protected 멤버를 만들 수 있게 해주는 것으로, 선언된 클래스와 그 상속 클래스들에서만 보여집니다.
<b>record 에 메소드 포함 가능</b>	<p>record 선언에서 클래스처럼 필드 외에도 속성, 메소드(생성자도 포함됨), 클래스 속성, 클래스 메소드, 클래스 필드, 네스티드 타입을 가질 수 있게 되었습니다.</p> <pre data-bbox="432 757 1390 1406"> type   TMyRecord = record     type       TInnerColorType = Integer;     var       Red: Integer;     class var       Blue: Integer;     procedure printRed();     constructor Create(val: Integer);     property RedProperty: TInnerColorType read Red write Red;     class property BlueProp: TInnerColorType read Blue write Blue;     end;  constructor TMyRecord.Create(val: Integer); begin   Red := val; end;  procedure TMyRecord.printRed; begin   writeln('Red: ', Red); end; </pre>
<b>class abstract</b>	<p>메소드 뿐만 아니라 클래스도 abstract 로 선언될 수 있습니다.</p> <pre data-bbox="432 1480 1390 1592"> type   TAbstractClass = class abstract     procedure SomeProcedure;   end; </pre>
<b>class sealed</b>	<p>필요한 경우 상속이 불가능하도록 sealed 로 지정할 수 있습니다.</p> <pre data-bbox="432 1666 1390 1771"> type   TAbstractClass = class sealed     procedure SomeProcedure;   end; </pre>
<b>class 상수</b>	<p>이제 클래스는 클래스 상수를 가질 수 있습니다. 이것은 클래스의 객체가 아니라 클래스 자체와 연관된 상수입니다.</p> <pre data-bbox="432 1899 1390 2029"> type   TClassWithConstant = class     public       const SomeConst = '이것은 클래스 상수입니다.';   end; </pre>

	<pre>procedure TForm1.FormCreate(Sender: TObject); begin   ShowMessage(TClassWithConstant.SomeConst); end;</pre>
<p><b>클래스 타입</b></p>	<p>이제 클래스는 해당 클래스 내에서만 사용 가능한 타입 선언을 포함할 수 있습니다.</p> <pre>type   TClassWithClassType = class   private     type       TRecordWithinAClass = record         SomeField: string;       end;   public     class var       RecordWithinAClass: TRecordWithinAClass;     end;  procedure TForm1.FormCreate(Sender: TObject); begin   TClassWithClassType.RecordWithinAClass.SomeField := '이것은 클래스 타입으로 선언된 필드입니다.';   ShowMessage(TClassWithClassType.RecordWithinAClass.SomeField); end;</pre>
<p><b>class var</b></p>	<p>클래스는 클래스 변수도 가질 수 있는데, 이것은 클래스의 객체가 아닌 클래스 자체에 적용되는 변수입니다. 예제는 위의 "클래스 타입"을 참고하십시오.</p>
<p><b>클래스 속성</b></p>	<p>클래스 속성은 클래스의 객체가 아닌 클래스 자체에 적용되는 속성입니다. 아래의 "static class methods" 예제를 참고하십시오.</p>
<p><b>중첩된 클래스</b></p>	<p>클래스 선언 내에서 타입 선언이 포함될 수 있습니다. 이 방법으로 개념적으로 관계가 있는 타입들을 같이 둘 수 있으며, 이름 충돌도 피할 수 있습니다.</p> <pre>type   TOuterClass = class   strict private     MyField: Integer;   public     type       TInnerClass = class         public           MyInnerField: Integer;           procedure InnerProc;         end;     procedure OuterProc;   end;  procedure TOuterClass.TInnerClass.InnerProc; begin   ... end;</pre>
<p><b>final 메소드</b></p>	<p>오버라이드한 버추얼 메소드를 final 로 표시하여 상속되는 클래스에서 해당 메소드를 더 이상 오버라이드하지 못하도록 막을 수 있습니다.</p> <pre>TAbstractClass = <b>class abstract</b> <b>public</b>   <b>procedure</b> Bar; <b>virtual;</b></pre>

	<pre> end;  TSealedClass = class sealed(TAbstractClass) public   procedure Bar; override; end;  TFinalMethodClass = class(TAbstractClass) public   procedure Bar; override; final; end; </pre>
<b>sealed 메소드</b>	sealed 로 표시된 클래스는 더 이상 상속이 불가능합니다. '파이널 메소드'의 예제를 참조하십시오.
<b>class static 메소드</b>	<p>클래스에 스테틱 클래스 메소드를 추가할 수 있는데, 이것은 클래스 타입으로부터 호출할 수 있는 메소드입니다. 클래스 스테틱 메소드는 객체에 대한 참조 없이도 사용이 가능합니다. 일반적인 클래스 메소드와는 달리, 클래스 스테틱 메소드는 Self 파라미터를 가지지 않습니다. 또한 객체의 멤버들을 액세스할 수도 없습니다. (클래스 필드, 클래스 속성, 클래스 메소드는 액세스할 수 있습니다) 또한 클래스 메소드와 달리 클래스 스테틱 메소드는 virtual 로 선언될 수 없습니다.</p> <pre> type   TMyClass = class     strict private        class var         FX: Integer;     strict protected        // 노트: 클래스 속성을 액세스하려면 클래스 스테틱으로 선언되어야 합니다.        class function GetX: Integer; static;       class procedure SetX(val: Integer); static;     public        class property X: Integer read GetX write SetX;       class procedure StatProc(s: String); static;     end;  TMyClass.X := 17; TMyClass.StatProc('Hello'); </pre>
<b>for-in 루프</b>	<p>이제 델파이에서 컨테이너에 대해 for-요소-in-컬렉션 스타일의 반복자를 지원합니다. 컴파일러는 다음의 컨테이너 반복 패턴을 인식합니다.</p> <pre> for Element in ArrayExpr do Stmt; for Element in StringExpr do Stmt; for Element in SetExpr do Stmt; for Element in CollectionExpr do Stmt; </pre>



Embarcadero Technologies Inc.는 애플리케이션 개발자 및 데이터베이스 전문가가 자신이 선택한 환경에서 소프트웨어 애플리케이션을 설계, 빌드 및 실행하는 도구를 사용할 수 있도록 합니다. 전 세계 3 백만 이상의 커뮤니티와 Fortune 지 선정 100 대 기업 중 90 개 기업이 Embarcadero 의 CodeGear™ 및 DatabaseGear™ 제품군을 기반으로 하여 생산성을 향상시키고 개방적인 협업 및 자유로운 혁신을 추구하고 있습니다. Embarcadero 는 1993 년에 설립되어 캘리포니아 샌프란시스코에 본사가 있으며 전 세계에 사무소를 두고 있습니다. Embarcadero 의 온라인 주소는 [www.embarcadero.com](http://www.embarcadero.com) 입니다.



데브기어는 미국 Embarcadero Technologies Inc.와 기존의 코드기어 한국 지사의 협력으로 전략적으로 설립된 엠바카데로 솔루션 전문 공급 기업입니다. 데브기어는 Delphi, C++Builder, JBuilder, Delphi Prism 등 개발툴 제품들과 ER/Studio, PowerSQL, DB Artisan, EA/Studio 등의 데이터베이스 툴 제품들에 대한 한국 시장에 공급은 물론 기술지원 및 교육 등의 기술 서비스를 제공합니다. 데브기어 웹 사이트는 <http://www.devgear.co.kr/> 이며 제품에 대한 문의는 [ask@embarcadero.kr](mailto:ask@embarcadero.kr) 로 하면 됩니다.