26장 델파이의 Open Tools API 활용

* Open Tools의 인터페이스들
* Open Tools API 사용법
* 폼 전문가

델파이가 대단한 물건임은 틀림없지만, 개발자에 필요한 모든 것들을 제공하지는 않는다. 델파이는 현대적인 프로그래밍 도구답게 IDE 자체를 확장할 수 있는 수단을 제공하는데, 그것이 바로 Open Tools API이다. 델파이의 Open Tools API를 사용하면 델파이의 IDE를 입맛에 맞게 확장/개선시킬 수 있다. 이번 장에서는 Open Tools API를 구성하는 여러 가지 인터페이스들을 설명하고, 그 인터페이스를 이용해서 쓸 만한 델파이 '전문가(expert)'를 만드는 방법을 제시하겠다.

Open Tools의 인터페이스들

Open Tools API는 여덟 개의 유닛들로 구성되어 있으며, 각 유닛마다 IDE의 여러 가지 기능에 관련된 인터페이스나 객체들이 정의되어 있다. 이 인터페이스들을 적절히 구현하기만 하면 델파이 전문가나 버전 관리자, 그리고 델파이 IDE를 확장/개선하는 데 관련된 사용자 컴포넌트나 속성 편집기를 만들 수 있다.

Open Tools API는 COM을 지원하며(COM에 대해서는 23장을 참고하기 바란다), 컴포넌트나 속성 편집기를 위한 일부 인터페이스들을 제외하고는 모두 가상 인터페이스들만을 제공한다. 다시 말하면 개발자는 인터페이스의 가상 메소드들만을 구현할 수 있을 뿐이며, Open Tools API 인터페이스 객체들의 데이터 필드나 속성, 정적 메소드들을 직접 사용할 수는 없다. Open Tools API가 COM으로 만들어져 있다는 말은 델파이뿐만 아니라 COM을 지원하는 다른 언어로도 델파이 IDE 확장용 소프트웨어를 만들 수 있다는 뜻도 된다. 이번 장에서는 델파이만을 이용해서 Open Tools API를 다루지만, C++로도 Open Tools API를 이용한 추가 기능(Add-on)을 만들 수 있는 것이다.



델파이 프로페셔널과 엔터프라이즈 에디션에는 완전한 Open Tools API가 포함되어 있지만, 델파이 스탠다드 에디션으로는 Open Tools API로 만든 추가 기능을 사용할 수만 있고, 직접 만들 수는 없다. Open Tools 인터페이스들의 소스 코드는 \Delphi 5\Source\ToolsAPI 디렉토리에 있다.

표 26.1은 Open Tools API를 구성하는 유닛들과 그들이 제공하는 인터페이스들이다. 그런데 엄밀히 말하면 표에 나온 모든 인터페이스들이 진짜 COM 인터페이스(즉, 델파이의 interface형)는 아니다. Open Tools API는 델파이가 COM 인터페이스를 지원하기 전에도 존재했었으며, Open Tools API는 기본적으로 가상/추상 메소드들로 이루어진 델파이 고유의 클래스들로 만들어졌었다. 볼랜드(인프라이즈) 사는 현재 COM 인터페이스들로 기존의 클래스들을 대체하는 중이다. 델파이 5의 경우, IDesigner와 IFormDesigner 인터페이스 등이 진짜 COM 인터페이스이다. 새로운 델파이에서는 보다 더 많은 인터페이스들이 추가될 것이다.



표 26.1 Open Tools API를 구성하는 유닛들

유닛 이름	용 도
VirtInft	다른 인터페이스들의 기반 클래스인 TInterface 클래스를 정의한다. 또한, VCL TStream을 캡슐화하는 TIStream 클래스도 정의한다.
IStreams	TIstream의 후손들인 TIMemoryStream, TIFileStream, TIVirtualStream 클래스들을 정의한다. 이 인터페이스들은 IDE의 스트림화 메커니즘을 가로채는 데 쓰인다.
ToolIntf	델파이 IDE의 메뉴를 다루는 데 관련된 TIMainMenuInft와 TIMenuItemIntf 클래스를 정의한다. 또한, IDE와 추가 기능 사이의 이벤트 전달에 관련된 TIAddInNotifier 클래스도 정의한다. 그리고 이 유닛이 정의하는 가장 중요한 클래스는 TIToolServices인데, 이 클래스는 델파이 IDE의 여러 요소들(컴포넌트 라이브러리, 코드 편집기, 폼 디자이너, 파일 시스템 등)을 다루기 위한 인터페이스를 제공한다.
VCSIntf	델파이 IDE와 버전 관리 소프트웨어 사이의 통신에 관련된 TIVCSClient 클래스를 정의한다.
FileIntf	파일 관리에 관련된 TIVirtualFileSystem 클래스를 정의한다. 이 클래스를 이용하면 델파이 고유의 파일 시스템 관리 작업을 가로채는 전문가, 버전 관리자, 속성/컴포넌트 편집기를 만들 수 있다.
EditIntf	델파이 코드 편집기와 폼 디자이너를 다루는 데 필요한 여러 클래스들을 정의한다. TIEditReader 클래스는 편집 버퍼를 읽는 데 관련된 것이고, TIEditWriter 클래스는 편집 버퍼에 뭔가를 쓰는 데 관련된 것이다. TIEditView는 편집 버퍼의 개별 뷰를 뜻한다. TIEditInterface는 앞에서 이야기한 다른 인터페이스를 얻을 때 사용하는 기반 인터페이스이다. TIComponentInterface는 설계 시점에서 폼 위에 놓여진 개별 컴포넌트들에 대한 인터페이스이다. TIFormInterface는 설계 시점의 폼이나 데이터 모듈에 대한 기반 인터페이스이다. TIResourceEntry는 프로젝트의 리소스 파일(*.res) 안에 있는 데이터에 대한 인터페이스이다. TIResourceFile은 프로젝트 리소스 파일에 대한 고수준 인터페이스이다. TIModuleNotifier는 특정 모듈에 대해 벌어지는 여러 가지 이벤트들을 감지하는데 필요한 클래스이다. 마지막으로, TIMdouleInterface는 IDE 안에 열려진 모든 종류의 파일이나 모듈에 대한 인터페이스이다.

유닛 이름	용 도
ExptIntf DsgnIntf	모든 전문가 클래스들이 상속하는 기반 클래스인 TIExpert를 정의한다. 사용자 속성/컴포넌트 편집기를 만들 때 사용하는 TFormDesigner, TPropertyEditor, TComponentEditor 클래스들을 정의한다.



위에서 말한 여러 클래스들을 델파이 도움말에서 찾기는 힘들 것이다. 각 클래스들에 대한 자세한 설명은 개별적인 문서가 아니라 유닛의 소스 코드에 들어 있다. 이 책에서는 소스 코드에 나오는 설명들을 되풀이하지 않으니, 자세한 정보는 소스 코드를 참고하기 바란다.

Open Tools API 사용법

인터페이스들에 대해서 대략적인 설명을 마쳤으니, 이제 실제 코드를 좀 보자. 이 장에서는 Open Tools API를 이용해서 전문가(expert)를 만드는 방법에 초점을 둔다. 버전 관리 시스템은 Open Tools API에 관련된 부분만으로는 모두 설명할 수 없는 방대한 주제이므로 제외하겠다. 그리고 속성이나 컴포넌트 편집기에 대해서는 21장, “델파이 사용자 컴포넌트 만들기”과 22장, “고급 컴포넌트 기법”에서 이미 자세히 설명했다.

간단한 전문가 예제

전문가(expert)를 만드는 방법을 설명하기 위해서 ‘바보’ 전문가를 하나 만들어 보자. 별로 쓸모는 없지만, 전문가 작성 방법을 이해하기에는 충분할 것이다. 전문가를 만들기 위해서는 TIExpert를 상속하는 클래스를 만들고, 메소드들을 적절히 재정의해야 한다. TIExpert의 정의는 다음과 같다.

```
type
  TIExpert = class(TInterface)
  public
    { 전문가의 UI 문자열들 }
    function GetName: string; virtual; stdcall; abstract;
    function GetAuthor: string; virtual; stdcall; abstract;
    function GetComment: string; virtual; stdcall; abstract;
    function GetPage: string; virtual; stdcall; abstract;
    function GetGlyph: HICON; virtual; stdcall; abstract;

    function GetStyle: TExpertStyle; virtual; stdcall; abstract;
    function GetState: TExpertState; virtual; stdcall; abstract;
    function GetIDString: string; virtual; stdcall; abstract;
    function GetMenuText: string; virtual; stdcall; abstract;

    { 전문가를 띄우는 메소드 }
    procedure Execute; virtual; stdcall; abstract;
  end;
```

GetXXX() 메소드들은 전문가에 대한 여러 가지 정보를 IDE에 제공하기 위한 함수들이다. 각 메소드들에 대해서는 예제 전문가들을 설명할 때 함께 설명하겠다. TIExpert.Excute()는 사용자가 메인 메뉴나 New Items 대화상자에서 전문가를 선택할 때 IDE가 호출한다. 따라서, Excute()에는 전문가를 생성하고 띄우는 코드가 들어가야 한다.

그럼 이번 예제 전문가인 TDumbExpert를 담고 있는 DumbExpt.pas 유닛의 소스 코드를 살펴보자.

목록 26.1 Ch26\DumbExpt\DumbExpt.pas

```
unit DumbExpt;
interface

uses
  ShareMem, dialogs,SysUtils, Windows, VirtIntf, ExptIntf, ToolIntf;

type
```

```
  TDumbExpert = class(TIExpert)
    function GetName: string; override;
    function GetStyle: TExpertStyle; override;
    function GetState: TExpertState; override;
    function GetIDString: string; override;
    function GetMenuText: string; override;
    procedure Execute; override;
  end;

procedure Register;

implementation

function TDumbExpert.GetName: string;
begin
  Result := '바보 전문가';
end;

function TDumbExpert.GetStyle: TExpertStyle;
begin
  Result := esStandard;
end;

function TDumbExpert.GetState: TExpertState;
begin
  Result := [esEnabled];
end;

function TDumbExpert.GetIDString: String;
begin
  Result := 'DDG.DumbExpert';
end;

function TDumbExpert.GetMenuText: string;
begin
  Result := '바보 전문가';
end;

procedure TDumbExpert.Execute;
```

```
begin
  MessageDlg('이것은 델파이 5로 작성한 바보 전문가입니다.', mtInformation,
  [mbOk], 0);
end;

procedure Register;
begin
  RegisterLibraryExpert(TDumbExpert.Create);
end;

end.
```

GetName() 메소드는 전문가의 이름을 돌려주도록 재정의해야 한다. GetStyle()은 전문가의 종류를 TExpertStyle형의 값으로 돌려줘야 하는데, TExpertStyle의 정의는 다음과 같다.

```
type
  TExpertStyle = (esStandard, esForm, esProject, esAddIn);
```

다음은 각 열거값들의 의미이다.

- esStandard는 IDE의 Help 메뉴를 통해서 이 전문가를 조작한다는 뜻이다. 이런 형태의 전문가가 가장 작성하기 쉽다.
- esForm은 이 전문가가 폼 전문가라는 뜻이다. 폼 전문가는 New Items 대화상자(File | New 메뉴)의 Forms 페이지를 통해서 띄우게 된다. 특정한 종류의 폼을 쉽게 만들 수 있는 전문가를 작성할 때 사용한다.
- esProject는 이 전문가가 프로젝트 전문가라는 뜻이다. 프로젝트 전문가는 New Items 대화상자의 Projects 페이지를 통해서 띄우게 된다. 특정한 종류의 프로젝트를 쉽게 만들 수 있는 전문가를 작성할 때 사용한다.
- esAddIn은 이 전문가가 TIToolServices 인터페이스를 통해서 IDE와 통신을 한다는 뜻이다. 이러한 전문가는 자신만의 사용자 인터페이스를 가지게 된다. 앞의 세 스타일에 속하지 않는 독특하고 고유한 작업을 하는 전문가라면 이 스타일을 사용하면 된다.

Tip

esForm과 esProject 형태의 전문가에 대한 예제를 찾는다면 \Delphi 5\Demos\Experts 디렉토리의 ExptDemo.dpr를 보기 바란다. 그리고 같은 디렉토리의 ProjExpl.dpr 프로젝트는 esAddIn 형태의 예제 전문가이다.

esStandard 전문가의 경우, GetState()를 통해서 자신의 메뉴 상태를 IDE에 돌려준다. GetState() 메소드의 리턴값은 esEnabled와 esChecked의 조합인데, 두 값은 각각 메뉴 항목의 활성화 여부와 체크 표시 여부를 뜻한다. IDE는 이 메소드의 리턴값을 통해서 메인 메뉴의 전문가 메뉴 항목을 어떻게 표시할 것인지를 결정한다.

GetIDString()은 전문가를 식별하기 위한 고유한 문자열을 돌려줘야 한다. 이 메소드의 리턴값으로는 다음과 같은 형식의 문자열을 이용하는 것이 관례이다.

```
제작사 이름.전문가 이름
```

GetMenuText()는 메인 메뉴에 나타날 메뉴 문자열을 돌려준다. 사용자가 Help 메뉴를 클릭할 때마다 이 메소드가 호출되므로 전문가가 실행되는 도중에도 메뉴 항목의 문자열이 동적으로 변경될 수 있다.

Execute()는 전문가를 띄우는 역할을 한다. 목록 26.1의 TDumbExpert는 그냥 메시지 상자 하나만을 띄우지만, 진짜 전문가라면 좀더 복잡한 일을 수행해야 할 것이다. 나중에 정말로 쓸 만한 전문가 예제가 나오니 기대하기 바란다.

그럼 전문가의 등록 과정을 살펴보자. Register() 프로시저의 RegisterLibraryExpert() 부분을 주목하기 바란다. 컴포넌트 라이브러리에 컴포넌트, 컴포넌트 편집기, 속성 편집기 등록용 프로시저들과 거의 비슷한 문법을 사용한다는 점을 알 수 있을 것이다. 이런 종류의 전문가는 컴포넌트 라이브러리의 일부로 패키지에 저장되기 때문에 등록하는 방식도 비슷하다. 전문가는 독립적인 DLL 파일에 저장할 수도 있는데, 그에 대해서는 다음 예제에서 설명하겠다.

이 전문가를 설치하는 방법은 컴포넌트를 등록하는 것과 동일하다. 메인 메뉴에서 Components | Install Component를 선택한 다음, 유닛을 선택하면 된다. 일단 설치가 되면 Help 메뉴 아래에 이 전문가에 대한 메뉴 항목이 나타난다(그림 26.1). 그 메뉴 항목을 선택하면 전문가가 뜬다(그림 26.2)

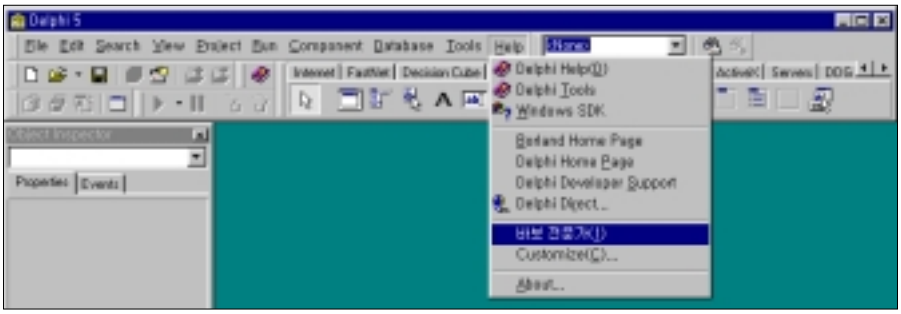


그림 26.1
메인 메뉴에 추가된 Dumb Expert

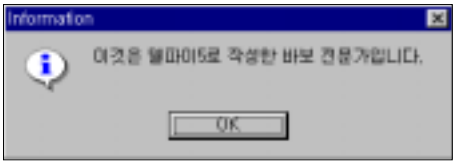


그림 26.2
Dumb 전문가의 실행 모습

DLL 기반의 전문가 작성

DLL 기반의 전문가를 만들려면 좀더 해야 할 일이 많다(컴포넌트 라이브러리 기반의 전문가에 비해). DLL 기반의 전문가 작성을 설명하기 위해서 Expert라는 예제 전문가를 살펴보자. 이번 예제를 통해서 DLL 전문가와 레지스트리와의 관계, 그리고 하나의 소스 코드를 기반으로 EXE와 DLL 전문가를 모두 만족시키는 방법 등을 배울 수 있을 것이다.



DLL에 대해서는 9장, "DLL의 활용"을 참고하기 바란다.

Tip

전문가를 컴포넌트 라이브러리 안의 패키지로 만들 것인지 아니면 하나의 DLL로 만들 것인지를 판단하는 결정적인 기준은 사실 없다고 할 수 있다. 전문가를 사용하는 입장에서 볼 때 가장 큰 차이는 설치 방법이다. 컴포넌트 라이브러리 기반의 전문가는 패키지나 컴포넌트를 설치하는 방식대로 설치해야 하는 반면, DLL 기반의 전문가는 레지스트리에 등록을 해야 하며, 델파이를 한 번 종료한 후 다시 시작해야 전문가를 사용할 수 있다. 전문가 개발자의 입장에서 본다면 컴포넌트 라이브러리 기반의 전문가쪽이 좀더 다루기 편하다. 특히, 델파이 컴포넌트 기반이라는 점은 전문가 안에서 발생한 예외가 자연스럽게 IDE로 넘어간다는 장점을 제공하며, 메모리 관리를 위한 sharemem.dll이나 기타 DLL의 전역 변수를 다루는 데 필요한 특별한 기법이 필요 없다는 점도 매력적이다. 또한, 풍선 도움말이나 마우스 들어옴/나감 메시지 같은 것들도 DLL에 비해 완벽히 수행된다.
설치는 DLL쪽이 편하다고 느낄 수도 있겠다. 어쨌든 위에서 말한 점들을 고려해서 DLL 기반인지 컴포넌트 라이브러리 기반인지를 결정하기 바란다.

델파이가 DLL 전문가를 인식하려면 시스템 레지스트리의 HKEY_CURRENT_USER\Software\Borland\Delphi\5.0\Experts키 아래에 전문가에 대한 항목을 추가해야 한다. 그림 26.3은 전문가에 대한 항목이 표시된 윈도우즈 레지스트리 편집기(RegEdit.exe)의 모습이다.

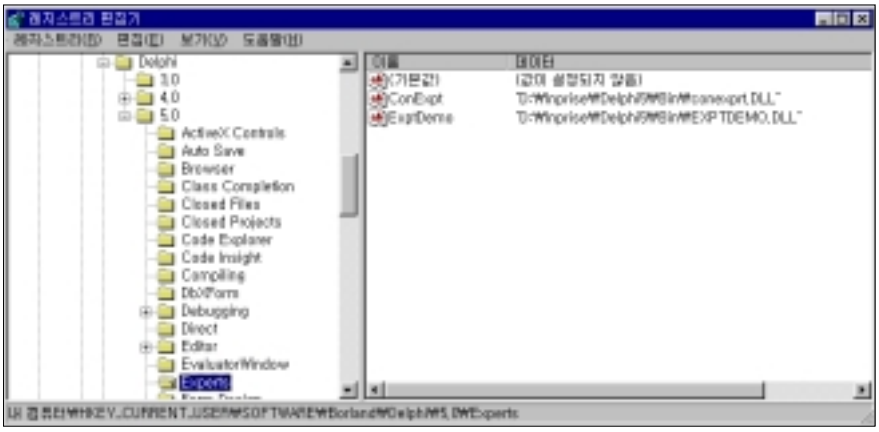


그림 26.3
RegEdit에 표시된 델파이 전문가 항목

Expert 예제의 인터페이스

예제 전문가인 Expert의 목적은 RegEdit를 거치지 않고 DLL 전문가의 레지스트리 항목을 추가, 수정, 삭제하기 위한 것이다. 그럼 전문가의 인터페이스를 담고 있는 유닛인 IntExpt.pas의 소스 코드를 살펴보자(목록 26.2).


목록 26.2 Ch26\InitExpt\IntExpt.pas

```
unit InitExpt;

interface

uses
  SysUtils, Windows, VirtIntf, ExptIntf, ToolIntf;

procedure HandleException;

type
  TExpertExpert = class(TIExpert)
    function GetName: string; override;
    function GetComment: string; override;
    function GetGlyph: HICON; override;
    function GetStyle: TExpertStyle; override;
    function GetState: TExpertState; override;
    function GetIDString: string; override;
    function GetMenuText: string; override;
    procedure Execute; override;
  end;

function InitExpert(AToolServices: TIToolServices;
  RegisterProc: TExpertRegisterProc;
  var Terminate: TExpertTerminateProc): Boolean; stdcall;

var
  { 레지스트리키 }
  SDelphiKey: string;

implementation
```

```
uses Forms, Controls, Main;


procedure HandleException;
{ IDE의 문맥하에서 예외를 발생시킨다. }
begin
  ToolServices.RaiseException(ReleaseException);
end;

function TExpertExpert.GetName: string;
{ 전문가 이름을 돌려준다. }
begin
  Result := 'ExpertExpert';
end;

function TExpertExpert.GetComment: string;
{ 리퍼지토리에서 사용할 설명을 돌려준다. exStandard 전문가에는 필요 없다. }
begin
  Result := '';
end;

function TExpertExpert.GetGlyph: HICON;
{ 전문가에 관련된 비트맵을 돌려준다. exStandard 전문가에는 필요 없다. }
begin
  Result := 0;
end;

function TExpertExpert.GetStyle: TExpertStyle;
{ 전문가의 종류를 돌려준다. 이 전문가는 표준형 전문가이다. }
begin
  Result := esStandard;
end;

function TExpertExpert.GetState: TExpertState;
{ 항상 활성화되어 있도록 한다. }
begin
  Result := [esEnabled];
end;
```

```
function TExpertExpert.GetIDString: String;
{ "제작사.전문가이름" 형태의 문자열을 전문가 ID로 돌려준다. }
begin
  Result := 'DDG.ExpertExpert';
end;

function TExpertExpert.GetMenuText: string;
{ 전문가의 메뉴에 나타날 문자열 }
begin
  Result := 'Expert 전문가';
end;

procedure TExpertExpert.Execute;
{ 사용자가 메인 메뉴에서 전문가를 선택하면 이 메소드가 실행된다.
 이 메소드는 메인 폼을 생성하고, 표시하고, 해제한다. }
begin
  try
    MainForm := TMainForm.Create(Application);
    try
      MainForm.ShowModal;
    finally
      MainForm.Free;
    end;
  except
    HandleException;
  end;
end;

function InitExpert(AToolServices: TIToolServices;
  RegisterProc: TExpertRegisterProc;
  var Terminate: TExpertTerminateProc): Boolean; stdcall;
begin
  Result := AToolServices <> nil;
  if Result then
    try
      ToolServices := AToolServices;
      Application.Handle := ToolServices.GetParentHandle;
      RegisterProc(TExpertExpert.Create);
    except
```

```
      HandleException;
    end;
  end;

  end.
```

앞에서 만들었던 컴포넌트 라이브러리 기반의 Dumb 예제와는 몇 가지 다른 점들이 눈에 뜨일 것이다. 무엇보다도 전문가의 초기화 메소드, 즉 전문가 DLL의 진입점 역할을 할 함수가 필요한데, 이 예제의 InitExpert()가 바로 DLL의 진입점이다. 이 함수는 TExpertInitProc형의 함수인데, 다음과 같은 초기화 작업들을 수행한다.

- 나중에 사용하기 위해서 ToolServices 포인터를 보존한다.
- DLL의 Application 객체의 핸들에 ToolServices.GetParentHandle()의 리턴값을 설정한다. GetParentHandle()은 전문가가 만들어내는 모든 최상위 윈도우들의 부모 역할을 할 윈도우의 핸들을 돌려준다.
- 전문가의 인스턴스를 생성하고 IDE에 등록한다. RegisterProc() 메소드에 전문가의 인스턴스를 넣어서 호출하면 전문가 인스턴스가 IDE에 등록된다.
- TExpertTerminateProc형의 인자인 Terminate에는 아무런 작업도 수행하지 않지만, 만일 전문가 DLL의 종료 처리가 필요하다면 그 인자에 종료 처리 프로시저를 넣어 주면 된다. IDE는 전문가를 언로드하기 전에 Terminate에 설정된 프로시저를 호출한다. Terminate 인자의 기본값은 nil이므로 종료 처리 프로시저가 필요 없다면 그냥 그대로 놔두면 된다.



전문가의 초기화 메소드는 반드시 stdcall 호출 규약을 사용해야 한다.

HandleException() 프로시저 역시 이전 예제에는 없었던 것이다. 이 프로시저는 ToolServices.RaiseException()을 호출하는데, 그러면 IDE의 문맥하에서 예외가 하나 발생된다. 전문가의 Execute()가 호출되기 전에 발생하는 예외는 모두 이런 식으로 IDE에 넘겨줘야 한다. 그런데 ToolServices.ReaiseException()은 스택을 정리하지 않으며,

이 메소드가 호출된 다음에는 전문가의 코드로 다시 실행 제어권이 넘어오지 않는다는 점을 주의해야 한다.



문자열 인자가 포함된 Open Tools API 함수(예를 들어서 ToolServices.OpenFile() 등)를 호출하는 DLL 전문가는 반드시 uses절에 ShareMem 유닛을 포함시켜야 한다. 그렇게 하지 않으면 전문가 인스턴스가 해제될 때 델파이가 접근 위반(access violation) 예외를 발생시킨다.

Expert 예제의 사용자 인터페이스

Excute() 메소드는 전문가의 메인 폼(MainForm) 인스턴스를 생성해서 모달로 표시한 다음, 폼이 닫히면 인스턴스를 해제한다. 그림 26.4는 메인 폼의 모습이며, 목록 26.3은 MainForm이 정의되어 있는 Main.pas 유닛의 소스 코드이다.

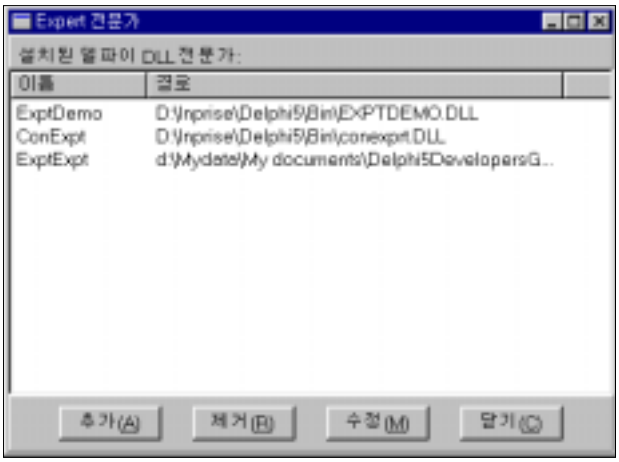


그림 26.4
Expert 전문가의 메인 폼

목록 26.3 Ch26\ExptExpt\Main.pas

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls, HeadList, Registry, AddModU, ComCtrls,
  Menus;

type
  TMainForm = class(TForm)
    TopPanel: TPanel;
    Label1: TLabel;
    BottomPanel: TPanel;
    ButtonPanel: TPanel;
    AddBtn: TButton;
    RemoveBtn: TButton;
    ModifyBtn: TButton;
    CloseBtn: TButton;
    ExptList: TListView;
    PopupMenu1: TPopupMenu;
    Add1: TMenuItem;
    Remove1: TMenuItem;
    Modify1: TMenuItem;
    procedure FormResize(Sender: TObject);
    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 InitExpt;

{$R *.DFM}

var
  DelReg: TRegistry;

const
  { 델파이 5.0 전문가들이 등록된 레지스트리 키, 이 키를 사용하지 않을 경우
    수동으로 입력해야만 한다.}
  SDelphiKey = '\Software\Borland\Delphi\5.0\Experts';

procedure TMainForm.FormResize(Sender: TObject);
{ 메인 폼의 OnResize 이벤트 처리 메소드 }
begin
  { 버튼들을 폼의 중앙에 배치한다. }
  ButtonPanel.Left := (BottomPanel.Width div 2) -
      (ButtonPanel.Width div 2);
end;

procedure TMainForm.RemoveBtnClick(Sender: TObject);
{ Remove 버튼의 클릭 이벤트 처리 메소드. 선택된 항목을 레지스트리에서
제거한다.}
var
  Item: TListItem;
begin
  Item := ExptList.Selected;
  if Item <> nil then
  begin
    if MessageDlg(Format('제거 아이템- "%s"', [Item.Caption]),
          mtConfirmation, [mbYes, mbNo], 0) = mrYes then
       DelReg.DeleteValue(Item.Caption);
    RefreshReg;
  end;
end;
```

```
procedure TMainForm.CloseBtnClick(Sender: TObject);
{ Handler 버튼의 클릭 이벤트 처리 메소드. 메인 폼을 닫는다. }
begin
  Close;
end;

procedure TMainForm.DoAddMod(Action: TAddModAction);
{ 레지스트리에 새 전문가 항목을 추가하거나 기존의 것을 수정한다. }
var
  OrigName, ExpName, ExpPath: String;
  Item: TListItem;
begin
  if Action = amaModify then          // 수정 작업이면...
  begin
    Item := ExptList.Selected;
    if Item = nil then Exit;          // 항목이 선택되어 있는지 확인
    ExpName := Item.Caption;          // 제목을 갱신
    if Item.SubItems.Count > 0 then
      ExpPath := Item.SubItems[0];
    OrigName := ExpName;              // 원래 이름을 보존한다.
  end;
 { 항목을 추가하거나 수정할 대화상자를 띄운다. }
  if AddModExpert(Action, ExpName, ExpPath) then
  begin
     { 수정 작업이고 이름이 바뀌었으면 레지스트리의 이름을 변경한다. }
    if (Action = amaModify) and (OrigName <> ExpName) then
      DelReg.RenameValue(OrigName, ExpName);
    DelReg.WriteString(ExpName, ExpPath);  // 새 값을 쓴다.
  end;
  RefreshReg;                             // 목록 상자를 갱신한다.
end;

procedure TMainForm.AddBtnClick(Sender: TObject);
{ Add 버튼의 클릭 이벤트 처리 메소드 }
begin
  DoAddMod(amaAdd);
end;
```

```
procedure TMainForm.ModifyBtnClick(Sender: TObject);
{ Modify 버튼의 클릭 이벤트 처리 메소드 }
begin
  DoAddMod(amaModify);
end;

procedure TMainForm.RefreshReg;
{ 목록 상자를 레지스트리의 내용으로 채운다. }
var
  i: integer;
  TempList: TStringList;
  Item: TListItem;
begin
  ExptList.Items.Clear;
  TempList := TStringList.Create;
  try
    { 레지스트리로부터 전문가 이름들을 얻는다. }
    DelReg.GetValueNames(TempList);
    { 각 전문가들의 경로를 얻는다. }
    for i := 0 to TempList.Count - 1 do
    begin
      Item := ExptList.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;          // 레지스트리 객체를 생성한다.
  DelReg.RootKey := HKEY_CURRENT_USER; // 루트키를 설정한다.
  DelReg.OpenKey(SDelphiKey, True);    // 델파이 전문가키를 열거나 생성한다.
```

```
finalization
  Delreg.Free;                          // 레지스트리 객체를 해제한다.
end.
```

이 유닛에는 레지스트리에 DLL 전문가 항목을 추가, 제거, 수정하기 위한 대화상자가 담겨 있다. 이 유닛의 initialization부에서 DelReg라는 TRegistry 객체가 생성된다. DelReg의 RootKey 속성을 HKEY_CURRENT_USER로 설정한 다음, OpenKey() 메소드를 이용해서 DLL 전문가에 대한 항목들이 담겨 있는 \Software\Borland\Delphi\5.0\Experts키를 연다.

메인 폼이 뜨면 레지스트리에 있는 전문가 정보들이 TListView 목록 상자인 ExptList에 채워진다. 일단 DelReg.GetValueNames()를 호출해서 레지스트리 항목 이름들을 TStringList에 채운다. 그런 다음, 그 TStringList를 루프로 돌려서 각 문자열마다 하나의 TListItem을 ExptList에 추가한다. 그리고 DelReg.ReadString() 메소드를 이용해서 각 항목의 값들을 읽은 후 그 값들을 TListItem의 SubItems 목록에 추가한다.

레지스트리 항목의 제거, 추가, 수정은 RemoveBtnClick()과 DoAddMod() 메소드에서 일어난다. RemoveBtnClick()은 목록 상자에서 선택된 항목에 해당하는 레지스트리 항목을 제거하는 메소드이다. 우선, 선택된 항목이 있는지 확인한 후 확인용 메시지 상자를 띄운다. 그런 다음, DelReg.DeleteValue() 메소드에 CuttentItem을 넣고 호출해서 목록 상자에서 현재 선택된 항목을 레지스트리에서 제거한다.

DoAddMod()는 TAddModAction형의 인자를 받는다. TAddModAction의 정의는 다음과 같다.

```
type
  TAddModAction = (amaAdd, amaModify);
```

DoAddMod()는 Action 인자로 넘어온 값을 이용해서 레지스트리에 새로운 항목을 추가할 것인지 아니면 기존의 항목을 수정할 것인지 판단한다. Action 인자의 값이 amaModify이고 목록 상자에 선택된 항목이 있으면 현재 선택된 전문가의 이름과 경로를 지역 변수 ExpName과 ExpPath에 복사한 다음, AddModExpert() 함수에 두 값을 넘긴다. 이 함수는 목록 26.4에 나온 AddModU 유닛에 정의되어 있다. 이 함수는 전문가의 이름이나 경로를 수정/추가할 수 있는 대화상자를 띄우는데(그림 26.5), 사용자가 확인 버튼을 클릭하면 True를 돌려준다. 리턴값이 True이면 레지스트리를 수정하는데, Action 인자가 amaModify이고(즉, 현재 작업이 수정이고) 이름이 변경되었으

면 DelReg.RenameValue()를 이용해서 기존의 항목 이름을 변경한다. 그리고 DelReg.WriteString()을 이용해서 새 값 또는 수정된 값을 레지스트리에 기록한다.

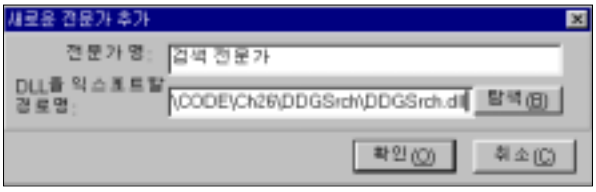


그림 26.5
Expert 예제의 AddModForm

목록 26.4 Ch26\ExptExpt\AddModU.pas

```
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
```

```
    { Private 선언부 }
  public
    { Public 선언부 }
  end;

function AddModExpert(AAction: TAddModAction; var ExpName, ExpPath:
 String): Boolean;

implementation

{$R *.DFM}

function AddModExpert(AAction: TAddModAction; var ExpName, ExpPath:
String): Boolean;
{ 대화상자를 띄우고 추가/수정할 레지스트리 항목의 내용을 입력받는다. }
const
  CaptionArray: array[TAddModAction] of string[31] =
    ('새로운 전문가 추가', '전문가 수정');
begin
  with TAddModForm.Create(Application) do   // 대화상자를 생성한다.
  begin
    Caption := CaptionArray[AAction];       // 제목을 설정한다.
    if AAction = amaModify then             // 수정 작업이면
    begin
      NameEd.Text := ExpName;               // 전문가 이름과
      PathEd.Text := ExpPath;               // 경로를 설정한다.
    end;
    Result := ShowModal = mrOk;             // 대화상자를 띄운다.
    if Result then                          // OK 버튼을 클릭했으면
    begin
      ExpName := NameEd.Text;               // 이름과 경로를
      ExpPath := PathEd.Text;               // 설정한다.
    end;
    Free;
  end;
end;

procedure TAddModForm.BrowseBtnClick(Sender: TObject);
begin
```

```
  if OpenDialog.Execute then
    PathEd.Text := OpenDialog.FileName;
end;


end.
```

EXE와 DLL을 모두 대상으로 하는 소스 코드 관리

앞에서도 말했지만 프로젝트가 DLL 전문가와 독립적인 실행 파일을 둘 다 생성해 낼 수 있도록 소스 코드를 만드는 것이 가능하다. 프로젝트 파일에 여러 가지 컴파일러 지시자들을 적절히 추가하면 된다. 목록 26.5는 이번 예제 프로젝트의 프로젝트 소스 파일인 ExptExpt.dpr의 소스 코드이다.

목록 26.5 Ch26\ExptExpt\ExptExpt.dpr

```
{$ifdef BUILD_EXE}
program ExptExpt;     // EXE로 빌드한다.
{$else}
library ExptExpt;     // DLL로 빌드한다.
{$endif}

uses
{$ifndef BUILD_EXE}
  ShareMem,           // DLL인 경우 ShareMem이 필요하다.
  InitExpt,           // Expert 관련 유닛
{$endif}
  ExptIntf,
  Forms,
  Main in 'Main.pas' {MainForm},
  AddModU in 'AddModU.pas' {AddModForm};

{$ifdef BUILD_EXE}
{$R *.RES}                            // EXE에 필요한 것
{$else}
exports                               // DLL에 필요한 것들
  InitExpert name ExpertEntryPoint;     // DLL의 진입점
```

```
{$endif}

begin
{$ifdef BUILD_EXE}                    // EXE에 필요한 것들
  Application.Initialize;
  Application.CreateForm(TMainForm, MainForm);
  Application.Run;
{$endif}
end.
```

BUILD_EXE 조건 지시자가 정의되어 있으면 이 프로젝트는 실행 파일로 빌드된다. 그렇지 않으면 DLL 기반의 전문가로 빌드된다. 조건 지시자는 Project | Options 대화상자의 Directories/Conditionals 페이지에 있는 Conditional Defines에서 정의하면 된다(그림 26.6).

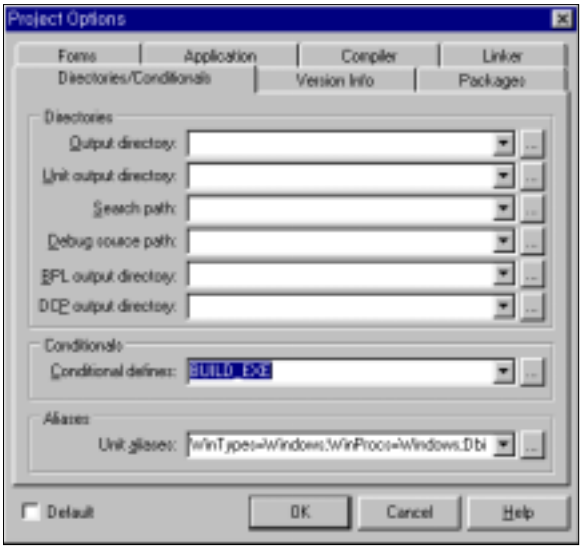


그림 26.6
Project Options 대화상자

그리고 DLL의 경우에는 진입점 프로시저가 필요한데, 프로젝트 파일을 보면 IntExpt 유닛의 InitExpert() 함수가 exports절에 포함되어 있다. 이 함수의 이름을 ExptIntf 유닛에 정의되어 있는 ExpertEntryPoint로 해야 한다는 점을 기억하기 바란다.

DDG Search 예제

11장, "멀티스레드 애플리케이션의 작성"에서 만든 Delphi Search라는 예제 프로그램을 기억할 것이다. 델파이 소스를 공부할 때 아주 유용하게 써먹을 수 있는 애플리케이션이다. 그럼 그 애플리케이션을 조금 손을 봐서 델파이 IDE에 통합되는 델파이 전문가로 변환시켜 보자. 전문가 이름은 DDG Search로 하겠다.

우선, DDG Search를 IDE에 통합시키기 위한 인터페이스 유닛이 필요한데, 목록 26.6에 나온 InitExpt.pas가 그런 유닛이다. 소스 코드를 보면 앞에서 나온 전문가 예제들의 인터페이스 유닛과 매우 흡사하다는 점을 알 수 있을 것이다. 사실, 이전의 인터페이스 유닛을 복사해 붙인 다음, 전문가 이름이 나오는 부분과 Excute() 메소드만 조금 고쳤을 뿐이다. 아주 원시적인 "코드 재사용 기법"인 셈이다. 그럼 코드를 보자.

목록 26.6 Ch26\DDGSrch\InitExpt.pas

```
unit InitExpt;

interface

uses
  SysUtils, Windows, Dialogs, VirtIntf, ExptIntf, ToolIntf;

procedure HandleException;

type
  TSearchExpert = class(TIExpert)
    function GetName: string; override;
    function GetStyle: TExpertStyle; override;
    function GetState: TExpertState; override;
    function GetIDString: string; override;
    function GetMenuText: string; override;
    procedure Execute; override;
  end;

function InitExpert(AToolServices: TIToolServices;
  RegisterProc: TExpertRegisterProc;
  var Terminate: TExpertTerminateProc): Boolean; stdcall;
```

```
implementation

uses Forms, Controls, Main, PriU;

procedure HandleException;
{ IDE의 문맥하에서 예외를 발생시킨다. }
begin
  ToolServices.RaiseException(ReleaseException);
end;

function TSearchExpert.GetName: string;
{ 전문가 이름을 돌려준다. }
begin
  Result := 'DDG Search';
end;

function TSearchExpert.GetStyle: TExpertStyle;
{ 전문가의 종류를 돌려준다. 이 전문가는 표준형 전문가이다. }
begin
  Result := esStandard;
end;

function TSearchExpert.GetState: TExpertState;
{ 항상 활성화되어 있도록 한다. }
begin
  Result := [esEnabled];
end;

function TSearchExpert.GetIDString: String;
begin
{ "제작사.전문가 이름" 형태의 문자열을 전문가 ID로 돌려준다. }
  Result := 'DDG.DDGSearch';
end;

function TSearchExpert.GetMenuText: string;
{ 전문가의 메뉴에 나타날 문자열 }
begin
  Result := 'DDG 검색 전문가';
end;
```

```
procedure TSearchExpert.Execute;
{ 사용자가 메인 메뉴에서 전문가를 선택하면 이 메소드가 실행된다.
 이 메소드는 전문가 메인 폼을 띄운다. }
begin
  try
    // 메인 폼이 생성되어 있지 않으면 생성하고 표시한다.
    if MainForm = nil then
    begin
      MainForm := TMainForm.Create(Application);
      ThreadPriWin := TThreadPriWin.Create(Application);
      MainForm.Show;
    end
    else
    // 메인 폼이 생성되어 있으면 위치와 크기를 복구하고 표시한다.
      with MainForm do
      begin
        if not Visible then Show;
        if WindowState = wsMinimized then WindowState := wsNormal;
        SetFocus;
      end;
  except
    HandleException;
  end;
end;

function InitExpert(AToolServices: TIToolServices;
  RegisterProc: TExpertRegisterProc;
  var Terminate: TExpertTerminateProc): Boolean; stdcall;
begin
  Result := AToolServices <> nil;
  if Result then
    try
      ToolServices := AToolServices;
      Application.Handle := ToolServices.GetParentHandle;
      RegisterProc(TSearchExpert.Create);
    except
      HandleException;
    end;
```

```
  end;

  end.
```

이 전문가의 Execute() 메소드는 이전에 나왔던 것들과는 조금 다르다. 전문가의 메인 폼인 MainForm은 모들리스로 표시되며, 따라서 전문가 메뉴가 선택될 때마다 메인 폼을 생성하고 해제하던 이전 예제들과는 다른 방식으로 메인 폼을 관리해야 한다. MainForm이 nil이면 전문가가 활성화되어 있지 않은 것이므로 MainForm을 생성하지만, 그렇지 않으면 이미 메인 폼이 생성되어 있다는 것이므로 다시 생성하면 안 된다. if문들을 자세히 살펴보기 바란다.

그럼 프로젝트 파일의 소스 코드를 살펴보자. 목록 26.7은 이번 예제의 프로젝트 파일인 DDGSrch.dpr의 소스 코드이다.

목록 26.7 Ch26\DDGSrch\DDGSrch.dpr

```
library DDGSrch;

uses
  ShareMem,
  ExptIntf,
  Main in 'MAIN.PAS' {MainForm},
  SrchIni in 'SrchIni.pas',
  SrchU in 'SrchU.pas',
  PriU in 'PriU.pas' {ThreadPriWin},
  InitExpt in 'InitExpt.pas',
  MemMap in '..\..\Utils\MemMap.pas',
  StrUtils in '..\..\Utils\StrUtils.pas';

{$R *.RES}

exports
  { 델파이 IDE가 호출하는 진입점 }
  InitExpert name ExpertEntryPoint;

begin
end.
```

매우 간단한 파일이다. DLL 프로젝트이므로 library로 시작한다는 점과 exports절에 델파이 IDE가 이 전문가를 초기화할 때 사용할 InitExpert() 함수가 포함되어 있다는 점 이외에는 별로 할 말이 없다.

그럼 실제로 검색 작업을 수행하는 메인 유닛을 살펴보자. 11장의 원래 소스 코드 중 몇 군데를 수정해서 다시 만들었다. 앞에서도 말했지만, MainForm이 nil이면 전문가가 아직 활성화되지 않은 것이다. 2장, "오브젝트 파스칼 언어"에서도 배웠듯이 애플리케이션이 시작될 때 객체 변수는 자동적으로 nil값을 가진다. MainForm도 애플리케이션의 초기에는 nil값을 가지므로 문제가 없다. 고쳐야 할 곳은 Close 버튼의 OnClick 이벤트이다. 원래 코드에서는 그냥 애플리케이션을 종료시켰지만, 이번 예제에서는 MainForm을 해제하고 nil을 넣도록 고친다. 새로 만든 코드는 다음과 같다.

```
procedure TMainForm.CloseButtonClick(Sender: TObject);
{ 스레드를 중지시키거나 메인 폼을 해제한다. }
begin
  // 스레드가 실행중이면 스레드를 종료시킨다.
  if Running then SearchThread.Terminate
  // 아니면 메인 폼을 해제한다.
  else begin
    MainForm.Release;
    MainForm := nil;
  end;
end;
```

원래 예제에서는 목록 상자의 파일 이름을 더블클릭하면 메모장이나 기타 애플리케이션으로 그 파일을 열도록 했지만, 이번에는 델파이 IDE의 코드 편집기에서 그 파일을 열도록 한다. FileLBDblClick()은 목록 상자의 더블클릭을 처리하는 메소드인데, 다음과 같다.

```
procedure TMainForm.FileLBDblClick(Sender: TObject);
{ 사용자가 목록 상자의 항목을 더블클릭하면 호출된다. 선택된 파일을
  IDE 코드 편집기로 여는 작업을 수행한다. }
var
  FileName: string;
  Len: Integer;
  ModInt: TIModuleInterface;
begin
  FileName := FileLB.Items[FileLB.ItemIndex];
  { 파일 이름을 클릭했는지 확인 }
  if (FileName <> '') and (Pos('File ', FileName) = 1) then
  begin
    { 문자열에서 "File "과 ":"를 잘라낸다. }
    FileName := Copy(FileName, 6, Length(FileName));
    Len := Length(FileName);
    if FileName[Len] = ':' then SetLength(FileName, Len - 1);
    { 프로젝트 파일이나 유닛 파일을 연다. }
    if CompareText(ExtractFileExt(FileName), '.DPR') = 0 then
      ToolServices.OpenProject(FileName)
    else
      ToolServices.OpenFile(FileName);
    { 코드 모듈에 대한 인터페이스를 얻는다. }
    ModInt := ToolServices.GetModuleInterface(FileName);
    try
      ModInt.ShowSource;           // 코드 모듈에 포커스를 설정
    finally
      ModInt.Release;              // 모듈 인터페이스를 해제한다.
    end;
  end;
end;
```

이 메소드는 ToolServices의 OpenFile()이나 OpenProject() 메소드를 이용해서 선택된 파일을 코드 편집기로 연다. 또, ToolServices.GetModuleInterface()로 IModuleInterface 인터페이스에 대한 참조를 얻어서 ModInt 변수에 설정한 다음, ModInt를 이용해서 코드 편집기로 방금 연 파일에 포커스를 설정한다.

주의

ToolServices로부터 얻은 인터페이스를 사용한 후에는 반드시 Release()를 호출해서 인터페이스를 해제시켜야 한다.

목록 26.8은 DDGSrch 프로젝트의 Main 유닛의 전체 소스 코드이고, 그림 26.7은 IDE 안에서 DDG Search 전문가가 실행중인 모습이다.

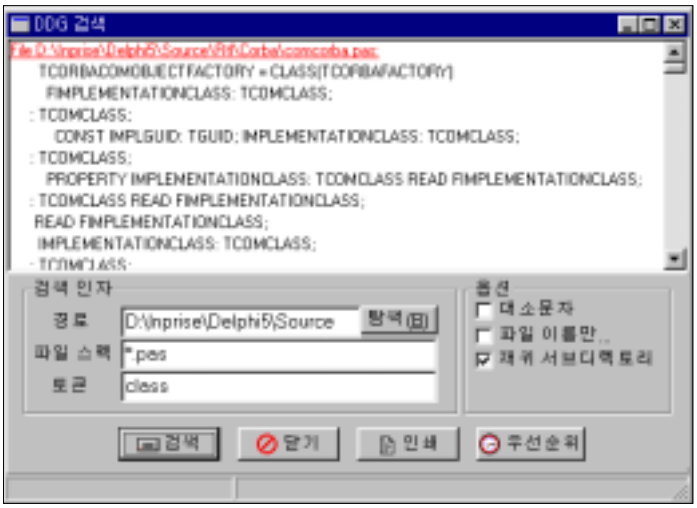


그림 26.7
DDG Search 전문가의 실행 모습

목록 26.8 Ch26\DDGSrch\Main.pas

```
unit Main;

interface

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;
    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);
  private
    procedure ReadIni;
    procedure WriteIni;
    procedure DoShowHint(var HintStr: string; var CanShow: Boolean;
      var HintInfo: THintInfo);
    procedure WMGetMinMaxInfo(var M: TWMGetMinMaxInfo);
          message WM_GETMINMAXINFO;
  public
    Running: Boolean;
    SearchPri: integer;
    SearchThread: TSearchThread;
    procedure EnableSearchControls(Enable: Boolean);
  end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

uses Printers, ShellAPI, MemMap, FileCtrl, PriU, ExptIntf, ToolIntf,
  EditIntf;

procedure PrintStrings(Strings: TStrings);
{ 이 프로시저는 Strings 인자의 모든 문자열들을 인쇄한다. }
var
  Prn: TextFile;
  i: word;
begin
  if Strings.Count = 0 then // 문자열이 하나도 없으면
  begin
    MessageDlg('인쇄할 텍스트가 없습니다!', mtInformation, [mbOk], 0);
```

```
    Exit;
  end;
  AssignPrn(Prn);                          // Prn에 프린터를 배정한다.
  try
    Rewrite(Prn);                          // 프린터를 연다.
    try
      for I := 0 to Strings.Count - 1 do   // 루프를 돌려서 모든 문자열들을
        WriteLn(Prn, Strings.Strings[I]);  // 프린터로 인쇄한다.
    finally
      CloseFile(Prn);                      // 프린터를 닫는다.
    end;
  except
    on EInOutError do
      MessageDlg('텍스트 인쇄 에러.', mtError, [mbOk], 0);
  end;
end;

procedure TMainForm.WMGetMinMaxInfo(var M: TWMGetMinMaxInfo);
begin
  inherited;
  // 사용자가 폼을 너무 작게 줄이지 못하도록 한다.
  with M.MinMaxInfo^ do
  begin
    ptMinTrackSize.x := OptionsGB.Left + OptionsGB.Width - ParamsGB.Left +
          10;
    ptMinTrackSize.y := 200;
  end;
end;

procedure TMainForm.EnableSearchControls(Enable: Boolean);
{ 검색 진행 상태에 기반해서 컨트롤들을 적절히 활성화/비활성화시킨다. }
begin
 SearchButton.Enabled := Enable;        // 컨트롤들을 적절히 활성화,
                      // 비활성화시킨다.
  cbRecurse.Enabled := Enable;
  cbFileNamesOnly.Enabled := Enable;
  cbCaseSensitive.Enabled := Enable;
  PathButton.Enabled := Enable;
  EPathName.Enabled := Enable;
```

```
  EFileSpec.Enabled := Enable;
  EToken.Enabled := Enable;
  Running := not Enable;                // Running 플래그를 설정
  ETokenChange(nil);
  with btnClose do
  begin
    if Enable then
    begin                 // Close/Stop 버튼의 속성들을 설정
      Caption := '닫기';
      Hint := '애플리케이션 닫기';
    end
    else begin
      Caption := '정지';
      Hint := '검색 정지';
    end;
  end;
end;

procedure TMainForm.SearchButtonClick(Sender: TObject);
{ Search 버튼을 클릭하면 이 프로시저가 실행된다. 이 프로시저는
 검색 스레드를 띄운다. }

begin
  EnableSearchControls(False);      // 컨트롤들을 비활성화시킨다.
  FileLB.Clear;                     // 목록 상자를 비운다.
  { 스레드를 시작한다. }
  SearchThread := TSearchThread.Create(cbCaseSensitive.Checked,
    cbFileNamesOnly.Checked, cbRecurse.Checked, EToken.Text,
    EPathName.Text, EFileSpec.Text);
end;

procedure TMainForm.ETokenChange(Sender: TObject);
begin
  SearchButton.Enabled := not Running and (EToken.Text <> '');
end;

procedure TMainForm.PathButtonClick(Sender: TObject);
{ Path 버튼이 클릭되면 실행된다. 사용자에게 새 경로를 선택받는다. }
var
```

```
  ShowDir: string;
begin
  ShowDir := EPathName.Text;
  if SelectDirectory(ShowDir, [], 0) then
    EPathName.Text := ShowDir;
end;

procedure TMainForm.FileLBDrawItem(Control: TWinControl;
  Index: Integer; Rect: TRect; State: TOwnerDrawState);
{ 오너 드로 목록 상자를 그린다. }
var
  CurStr: string;
begin
  with FileLB do
  begin
    CurStr := Items.Strings[Index];
    Canvas.FillRect(Rect);                  // 영역을 지운다.
    if not cbFileNamesOnly.Checked then     // File names Only 체크상자가
                                            // 체크되어 있지 않으면
      { 현재 줄이 파일 이름이면 }
      if (Pos('File ', CurStr) = 1) and
        (CurStr[Length(CurStr)] = ':') then
      begin
        Canvas.Font.Style := [fsUnderline]; // 글꼴 스타일은 밑줄
        Canvas.Font.Color := clRed;         // 글꼴 색은 빨강
      end
    else
        Rect.Left := Rect.Left + 15;        // 파일 이름이 아니면 들여 쓰기
    DrawText(Canvas.Handle, PChar(CurStr), Length(CurStr), Rect,
          dt_SingleLine);
  end;
end;

procedure TMainForm.Font1Click(Sender: TObject);
{ 목록 상자의 글꼴을 선택받는다. }
begin
  { 글꼴 대화상자를 띄워서 글꼴을 선택하게 한다. }
  if FontDialog1.Execute then
    FileLB.Font := FontDialog1.Font;
```

```
end;


procedure TMainForm.FormDestroy(Sender: TObject);
{ 폼의 OnDestroy 이벤트 처리 메소드 }
begin
  WriteIni;
end;

procedure TMainForm.FormCreate(Sender: TObject);
{ 폼의 OnCreate 이벤트 처리 메소드 }
begin
  Application.HintPause := 0;           // 힌트가 즉시 뜨도록 한다.
  Application.OnShowHint := DoShowHint; // 힌트 처리 프로시저를 설정
  ReadIni;                              // 레지스트리 설정을 읽는다.

end;

procedure TMainForm.DoShowHint(var HintStr: string; var CanShow: Boolean;
  var HintInfo: THintInfo);
{ 애플리케이션의 OnHint 이벤트 처리 메소드 }
begin
  { 상태 표시줄에 애플리케이션의 힌트(풍선 도움말)를 표시한다. }
  StatusBar.Panels[0].Text := HintStr;
  { 힌트 자체는 나타나지 않도록 한다. }
  CanShow := False;
end;

procedure TMainForm.PrintButtonClick(Sender: TObject);
{ Print 버튼을 클릭하면 이 프로시저가 실행된다. }
begin
  if MessageDlg('이 결과를 프린터로 보낼까요?', mtConfirmation,
    [mbYes, mbNo], 0) = mrYes then
    PrintStrings(FileLB.Items);
end;

procedure TMainForm.CloseButtonClick(Sender: TObject);
{ 스레드를 중지시키거나 메인 폼을 해제한다. }
begin
```

```
  // 스레드가 실행중이면 스레드를 종료시킨다.
  if Running then SearchThread.Terminate
  // 아니면 메인 폼을 해제한다.
  else begin
    MainForm.Release;
    MainForm := nil;
  end;
end;

procedure TMainForm.FileLBDblClick(Sender: TObject);
{ 사용자가 목록 상자의 항목을 더블클릭하면 호출된다. 선택된 파일을 IDE 코드
  편집기로 여는 작업을 수행한다. }
var
  FileName: string;
  Len: Integer;
  ModInt: TIModuleInterface;
begin
  FileName := FileLB.Items[FileLB.ItemIndex];
  { 파일 이름을 클릭했는지 확인 }
  if (FileName <> '') and (Pos('File ', FileName) = 1) then
  begin
    { 문자열에서 "File "과 ":"를 잘라낸다. }
    FileName := Copy(FileName, 6, Length(FileName));
    Len := Length(FileName);
    if FileName[Len] = ':' then SetLength(FileName, Len - 1);
    { 프로젝트 파일이나 유닛 파일을 연다. }
    if CompareText(ExtractFileExt(FileName), '.DPR') = 0 then
      ToolServices.OpenProject(FileName)
    else
      ToolServices.OpenFile(FileName);
    { 코드 모듈에 대한 인터페이스를 얻는다. }
    ModInt := ToolServices.GetModuleInterface(FileName);
    try
      ModInt.ShowSource;           // 코드 모듈에 포커스를 설정
    finally
      ModInt.Release;              // 모듈 인터페이스를 해제한다.
    end;
  end;
end;
```

```
procedure TMainForm.FormResize(Sender: TObject);
{ OnResize 이벤트 처리 메소드. 상태 표시줄의 크기를 조정한다. }
begin
 { 상태 표시줄의 두 패널을 1:2 비율로 조정한다. }
  with StatusBar do
  begin
    Panels[0].Width := Width div 3;
    Panels[1].Width := Width * 2 div 3;
  end;
  { 컨트롤들이 있는 패널을 폼의 중앙에 위치시킨다. }
  ControlPanel.Left := (AlignPanel.Width div 2) - (ControlPanel.Width div 2);
end;

procedure TMainForm.PriorityButtonClick(Sender: TObject);
{ 스레드 우선순위 설정 폼을 띄운다.}
begin
  ThreadPriWin.Show;
end;

procedure TMainForm.ReadIni;
{ 레지스트리로부터 기본 설정들을 읽어온다. }
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);
    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;
```

```
{ 현재 설정들을 레지스트리에 기록한다. }
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;

end.
```

폼 전문가

Open Tools API로 만들 수 있는 전문가의 종류 중에는 폼 전문가(Form Expert)라는 것이 있다. 폼 전문가는 New Items 대화상자로 띄우며, 주로 새 폼과 유닛들을 생성하는 역할을 한다. 이번에 제시할 예제 전문가는 24장, "윈도우즈 쉘의 확장"에서 앱바(AppBar) 예제를 만들 때 사용했던 것이다.

폼 전문가를 만드는 과정도 그리 복잡하지 않다. 기본적으로, 다음과 같은 과정을 거치게 된다.

1. TIExpert를 상속하는 클래스를 만든다. 스타일은 esForm으로 한다. 그리고 다음 세 단계의 작업을 수행하도록 Execute() 메소드를 재정의한다.

2. ToolServices.GetNewModuleName()을 호출해서 유닛의 새 이름을 설정한다.

3. TIMemoryStreams를 생성하고 거기에 유닛의 소스 코드와 폼 이미지를 집어 넣는다.

4. ToolServices.CreateModule()을 호출해서 새 유닛과 폼을 생성한다.

목록 26.9는 AppBar 전문가를 정의하는 ABExpt.pas의 소스 코드이다.

목록 26.9 Ch26\AppBar\ABExpt.pas

```
unit ABExpt;

interface

uses Windows, ExptIntf, ToolIntf, Classes;

type
  TAppBarExpert = class(TIExpert)
  private
    function GetFormStream(const AName: string): TMemoryStream;
    function GetSourceStream(const AUnitName, AFormName: string):
          TMemoryStream;
  public
    function GetName: string; override; stdcall;
    function GetAuthor: string; override; stdcall;
    function GetComment: string; override; stdcall;
    function GetPage: string; override; stdcall;
    function GetGlyph: HICON; override; stdcall;
    function GetStyle: TExpertStyle; override; stdcall;
    function GetState: TExpertState; override; stdcall;
    function GetIDString: string; override; stdcall;
    procedure Execute; override; stdcall;
  end;

implementation

uses Forms, AppBars, SysUtils, IStreams, Proxies, Dialogs;
```

```
// 다음 RES 파일에는 포맷되지 않은 소스 코드가 들어 있다.
{$R CodeGen.res}

{ TAppBarExpert }

function TAppBarExpert.GetGlyph: HICON;
begin
  Result := 0; {!!}
end;

function TAppBarExpert.GetIDString: string;
begin
  Result := 'DDG.AppBarExpert';
end;

function TAppBarExpert.GetName: string;
begin
  Result := 'DDG AppBar 전문가';
end;

function TAppBarExpert.GetPage: string;
begin
  Result := 'DDG';
end;

function TAppBarExpert.GetState: TExpertState;
begin
  Result := [esEnabled];
end;

function TAppBarExpert.GetStyle: TExpertStyle;
begin
  Result := esForm;
end;

function TAppBarExpert.GetAuthor: string;
begin
  Result := 'Delphi 5 개발자 가이드';
end;
```

```
function TAppBarExpert.GetComment: string;
begin
  Result := '새 AppBar 폼 생성.'
end;

function TAppBarExpert.GetFormStream(const AName: string): TMemoryStream;
var
  AB: TAppBar;
begin
  AB := TAppBar.CreateNew(nil);
  try
    Proxies.CreateSubClass(AB, 'T' + AName, TAppBar);
    AB.Name := AName;
    Result := TMemoryStream.Create;
    Result.WriteComponentRes(AName, AB);
    Result.Position := 0;
  finally
    AB.Free;
  end;
end;

function TAppBarExpert.GetSourceStream(const AUnitName, AFormName: string):
TMemoryStream;
var
  Text: string;
  ResInstance: THandle;
begin
  ResInstance := FindResourceHInstance(HInstance);
  Text := PChar(LockResource(LoadResource(ResInstance,
    FindResource(ResInstance, 'CODEGEN', RT_RCDATA))));
  Text := Format(Text, [AUnitName, AFormName]);
  Result := TMemoryStream.Create;
  Result.Write(Pointer(Text)^, Length(Text));
  Result.Position := 0;
end;

procedure TAppBarExpert.Execute;
var
```

```
  UnitName, FileName, FormName: string;
  SourceStream, FormStream: TIMemoryStream;
begin
  if (ToolServices <> nil) and
     ToolServices.GetNewModuleName(UnitName, FileName) then
  begin
    UnitName := LowerCase(UnitName);
    UnitName[1] := UpCase(UnitName[1]);
    FormName := 'AppBar' + Copy(UnitName, 5, Length(UnitName));
    FormStream := TIMemoryStream.Create(GetFormStream(FormName), soOwned);
    SourceStream := TIMemoryStream.Create(GetSourceStream(UnitName,
              FormName), soOwned);
    ToolServices.CreateModule(FileName, SourceStream, FormStream,
              [cmAddToProject, cmShowSource, cmShowForm, cmUnNamed,
              cmMarkModified]);
  end;
end;

end.
```

전문가가 생성할 소스 코드를 미리 RES 파일에 담아 두었기 때문에 이 유닛의 소스 코드는 생각보다 그리 길지 않다. $R 지시자로 링크하는 RES 파일에는 포맷되지 않은 소스 코드가 들어 있다. 이렇게 하면 유닛 자체를 고치지 않고도 생성될 소스 코드를 수정할 수 있다. 그 RES 파일은 RCDATA 리소스로 CODEGEN.TXT를 지정한 RC 파일(CODEGEN.RC)을 BRCC32.EXE로 컴파일한 것이다.

목록 26.10은 CODEGET.TXT의 내용이고, 목록 26.11은 CODEGEN.RC의 내용이다.

목록 26.10 Ch26\AppBar\CODEGEN.TXT

```
unit %O:s;

interface

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

```
type
  T%1:s = class(TAppBar)
  private
    { Private 선언부 }
  public
    { Public 선언부 }
  end;

var
  %1:s: T%1:s;

implementation

{$R *.DFM}

end.
```

목록 26.11 Ch26\AppBar\CODEGEN.RC

```
CODGEN RCDATA CODEGEN.TXT
```

요약

이 장에서는 델파이 Open Tools API에 관련된 여러 유닛들과 인터페이스들에 대해서 설명했다. 특히, IDE에 통합되는 전문가를 만드는 방법에 관련된 부분을 중점적으로 이야기하였다. 다음 장은 이 책의 4부인 "데이터베이스 개발"의 첫 번째 장인 "데스크탑 데이터베이스 애플리케이션 작성"이다. 데스크탑 데이터베이스 애플리케이션에 관한 여러 가지 개념과 기법들을 배울 수 있을 것이다.

쉬어가는 페이지