델파이 4 프로그래밍의 이해

(Understandings of Delphi 4 Programming)

오브젝트 파스칼과 심도있는 델파이 프로그래밍의 세계로 들어가기 전에, 이번 장에서는 델파이를 사용하여 첫번째 윈도우 어플리케이션을 제작하고 전반적인 델파이의 환경에 대해서 알아볼 것이다. 내용의 수준이 높지는 않겠지만, 흔히 알고 있었던 내용이라고 하더라도 별 생각 없이 넘어갔던 것들도 많을 것이다.
그러면, 델파이 4 의 세계의 역사적인 첫 발을 들여놓도록 하자 !

첫번째 어플리케이션

델파이를 일단 시작하면 기본적으로 새로운 프로젝트가 하나 시작되며, 비어 있는 폼이 나타난다. 아마도 오브젝트 인스펙터에는 현재 가리킬 수 있는 컴포넌트가 Form1 뿐일 것이므로 Form1 의 프로퍼티 값들을 표시하고 있을 것이다. 오브젝트 인스펙터의 사용법을 익히기 위해 먼저 폼의 캡션을 바꾸어 보자.
오브젝트 인스펙터의 Caption 프로퍼티를 'Form1' 에서 'Hello'로 한번 바꾸어 보자. 이 간단한 동작만으로 폼의 타이틀 바의 이름이 바뀌는 것을 관찰할 수 있을 것이다. 이제 Run 명령을 선택하거나 화살표 모양의 스피트 버튼을 클릭하면 이 어플리케이션이 다음과 같이 실행되는 것을 관찰할 수 있을 것이다.

이제 델파이 환경에서 실행된 어플리케이션을 종료하고 다시 폼 디자이너로 돌아오자.
많은 작업을 하지는 않았지만, 시스템 메뉴와 기본적으로 제공되는 전체 화면 표시 버튼과 최소화 표시 버튼, 닫기 버튼을 가지는 완전한 어플리케이션을 방금 하나 만든 것이다. 이 폼을 마우스를 이용해 크기를 조절할 수도 있고, 전체 화면으로 보거나 최소화할 수도 있다.

어플리케이션의 저장과 파일의 종류

이제 이렇게 만든 어플리케이션 소스를 저장해보자. File 메뉴에서 Save All 을 선택하면, 델파이는 폼과 관련된 소스 코드와 프로젝트 파일에 이름을 붙여 저장하게 된다. 먼저 파스칼 소스 코드인 .pas 파일의 이름을 물어오는데, 여기에는 U1_Exam1.pas 라고 명명하고 적당한 디렉토리에 저장한다. 마찬가지로 프로젝트 파일인 .dpr 파일은 Exam1.dpr 로 명명한다.

> 참고:
> 이 책에서 소스 코드의 이름을 명명할 때에는 Exam 이라는 문자열에다가 그 장에서 제작하는 예제가 몇 번째 것인지에 따라 일련 번호를 붙여서 사용한다. 이번 장과 같은 경우 3 장의 첫번째 예제이므로 Exam1.dpr 이 되며, 각 유닛 파일에는 접두어로 유닛의 개수에 따라 U1, U2 ... 등의 문자열을 사용한다. 각 장의 예제는 다른 디렉토리에 저장된다.

프로젝트 파일에 붙인 이름은 실행 시에 디폴트로 어플리케이션의 제목으로 사용되어 윈도우의 작업 표시줄에 나타나게 된다. 그러므로, 만약 프로젝트의 이름이 메인 폼의 제목과 같으면 이 이름은 작업 표시줄의 이름과도 일치하게 된다.
그러면, 델파이에서 사용하는 파일의 종류와 이들이 어떤 파일 들인지에 대해서 알아보도록 하자. 델파이 프로젝트는 폼, 유닛, 옵션 설정과 리소스에 대한 파일 들로 구성된다. 이런 모든 정보가 각각 다른 파일로 저장되며, 델파이에서 어플리케이션을 제작할 때에 생성된다. 다음 표는 델파이 4 에서 사용되는 파일 들의 종류와 이들의 역할에 대해서 나열한 것이다.

파일의 종류	설 명
프로젝트 그룹 파일 (.bpg)	델파이 4 에서 새롭게 추가된 프로젝트 그룹에 대한 파일로, 여러 프로젝트를 관리할 수 있는 프로젝트 그룹의 내용을 담고 있는 파일이다.
프로젝트 파일 (.dpr)	폼, 유닛 등에 대한 정보를 저장하는데 사용되는 파일이다.
유닛 파일 (.pas)	코드를 저장하는데 사용되는 파일로, 폼과 연관되기도 하며 어떤 것들은 함수와 프로시저만을 저장하기도 한다.
폼 파일 (.dfm)	폼에 대한 정보를 저장하기 위해 생성되는 이진 파일이다. 각 폼 파일은 유닛 파일과 연관되어 있다. 예를 들어, mine.pas 유닛 파일은 mine.dfm 이라는 폼 파일을 가진다.

프로젝트 옵션 파일 (.dfo)	프로젝트의 옵션 설정이 이 파일에 저장된다.
패키지 정보 파일 (.dfr)	델파이를 패키지와 함께 사용하는 이진 파일이다.
리소스 파일 (.res)	프로젝트에서 사용하는 각종 리소스를 저장하는 파일이다. 이 파일은 개발자가 생성하거나 변경하는 것이 아니고 델파이가 계속적으로 고치거나 다시 생성한다.
백업 파일 (.~dp, ~df, ~pa)	프로젝트, 폼, 유닛 파일 들에 대한 백업 파일이다. 여러 번 고친 경우에는 확장자의 앞글자 2 자 뒤에 고칠 때마다 하나씩 증가하는 정수로 명명된다. 예를 들어 유닛 파일의 경우 .pa1, .pa2, .pa3 등이다.
실행 파일 (.exe)	어플리케이션의 실행 파일로, 단독 실행이 가능하다.
유닛 객체 파일 (.dcu)	유닛 파일의 컴파일된 형태로, 최종 실행 파일에 링크된다.
타입 라이브러리 (.tlb)	액티브 X/COM 에서 사용되는 타입 라이브러리 파일

델파이 프로젝트

일반적으로 프로젝트란 어플리케이션에 있는 모든 객체를 포함하는 최상위 컨테이너를 말한다. 즉, 어플리케이션을 생성하는 각각의 파일들을 서로 연결한다. 하나의 프로젝트는 모든 객체의 저장소(repository)와 같은 역할을 하게 된다.
델파이의 프로젝트 역시 하나의 어플리케이션을 구성하는 모든 파일 들의 목록을 가지고 있다. 그런데, 다른 개발 툴과는 달리 그 자체의 기능을 가지는 프로그램 소스 코드를 포함하는데, 이를 볼 수도 있고 필요에 따라서 수정해서 쓸 수도 있다. 그렇지만, 프로젝트 파일의 코드는 대부분 델파이가 알아서 코딩을 해주기 때문에 건드릴 필요가 거의 없다.

- 프로젝트 파일의 이해

별로 건드릴 필요가 없는 파일이더라도, 어떻게 프로젝트를 이루는 코드가 되어 있는지는 알아야 할 것이다. 그렇다면, 프로젝트 파일의 소스 코드를 이해해 보도록 하자. 프로젝트 파일의 소스 코드를 보기 위해서는 Project|View Source 명령을 선택하면 된다. Exam1.dpr 프로젝트의 소스 코드는 다음과 같다.

```
program Exam1;

uses
  Forms,
  U1_Exam1 in 'U1_Exam1.pas' {Form1};

{$R *.RES}
```

```
begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

- program 키워드
  컴파일러에게 이 파일은 실행파일이 된다는 것을 알려준다. DLL 이거나 유닛일 경우에는 library 나 unit 키워드를 사용하게 된다.

- Uses 구문
  Uses 구문은 델파이가 실행파일을 만들 때 링크할 오브젝트 파스칼 유닛 들을 나열할 때 사용한다.

- $R 지시자
  $R 컴파일러 지시자는 컴파일러에게 지정된 윈도우 리소스를 사용하라고 알려주는 역할을 한다. $R 뒤에 나오는 별표(asterisk '*')는 리소스 파일이 프로젝트와 같은 이름을 사용한다는 것을 나타낸다. 프로젝트를 빌드하면 델파이는 프로젝트 자체와 각 폼에 대한 리소스 파일을 생성하게 된다.

- Application.CreateForm
  Application.CreateForm 구문은 프로젝트의 폼을 메모리로 읽어들인다. 일반적으로 프로젝트의 모든 폼은 여기에 나열된다. Option|Project 메뉴를 이용해 폼을 자동으로생성할 것인지 여부를 지정해줄 수 있는데, 각각의 폼은 각 폼의 인스턴스 변수(예를 들어 Form1)에 저장되며 이들은 각 폼 유닛의 interface 섹션에 정의되어 있다. Application.CreateForm 구문은 지정된 폼을 메모리로 읽어들이고, 인스턴스 변수에 그 폼에 대한 포인터를 저장한다.
  프로젝트 파일에서의 Application.CreateForm 구문의 순서는 실제로 폼이 생성되는 순서이며, 첫번째로 생성되는 폼이 메인 폼이 된다. 이 순서를 바꾸려면 Project|Option 메뉴의 Application 탭을 이용하면 된다.

- Application.Run
  이 구문에 의해 애플리케이션이 동작하게 된다.

● 유닛 파일의 이해

델파이의 각 폼은 그에 해당한 유닛 파일을 하나씩 가지고 있다. 여기에는 폼의 모양을 나타내는 클래스 정의가 포함된다. 새로운 컴포넌트를 폼에 추가할 때마다 델파이의 폼 디자이너는 이를 반영하기 위해 폼의 클래스 선언부분을 변경한다. 또한, 이벤트 핸들러를 추가할 때마다 여기에 해당되는 코드가 유닛 파일에 저장된다.
유닛 파일은 기본적으로 interface, implementation, initialization, finalization 섹션으로 구분된다. 현재의 폼에 대한 유닛 파일인 U1_Exam1.pas 파일의 소스 코드는 다음과 같다.

```
unit U1_Exam1;

interface

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

type
  TForm1 = class(TForm)
    btnOK: TButton;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

end.
```

- interface 섹션
  여기에는 유닛의 헤더 정보가 기록된다. 즉, 각종 함수, 프로시져의 선언부와 유닛의

외부에서 접근할 수 있는 변수, 상수, 타입의 정의가 위치하게 된다. 과거에 C 를 써본 경험이 있는 사람이라면 이 부분의 C 의 헤더 파일과 비슷한 역할을 한다고 생각하면 된다. C 와는 달리 이 섹션을 분리된 소스 코드 파일로 저장하지 않지만, 다른 모듈들이 이를 참조할 수 있다. 컴파일된 유닛은 인터페이스 정보를 헤더 부분에 저장하기 때문에 다른 모듈이 Uses 구문을 이용해서 유닛을 참조하면 델파이는 소스 코드를 직접 찾는 것이 아니라, 컴파일된 유닛(dcu 파일)의 헤더의 정보를 사용한다. 그렇기 때문에 델파이의 유닛을 컴파일된 dcu 파일(오브젝트 코드)로 배포할 수 있는 것이다.

- implementation 섹션
  implementation 섹션은 유닛의 실제 프로그래밍 코드가 담겨 있는 부분이다. 여기에 적힌 코드는 interface 섹션에서 나열된 부분만 외부에서 볼 수 있게 되며, 보통은 interface 섹션에서 나열한 부분을 구현하는 코드가 존재한다.

참고: 폼 인스턴스 변수 (form instance variable)
각 유닛의 interface 섹션에 보면 폼 인스턴스 변수를 선언한 부분이 있다.

```
var
  Form1: TForm1;
```

즉, TForm1 이라는 타입(type)의 Form1 이라는 변수를 선언하는 구문인데, 이때 TForm1 은 TForm 클래스를 상속받아서 델파이 폼 디자이너에 의해 생성된 새로운 클래스이다.
이렇게 선언된 폼 인스턴스 변수는 프로젝트 파일의 Application.CreateForm 이 호출될 때 초기화 되며, 유닛의 interface 섹션에 선언되어 있으므로 다른 모듈에서 Uses 구문을 사용해서 이를 사용하여 폼에 접근할 수 있게 된다.

- initialization, finalization 섹션
  앞의 소스에는 존재하지 않지만, 필요할 때 선언해서 사용하면 된다. 유닛이 처음 로드될 때 실행해야 하는 코드가 있으면, 이를 initialization 섹션에 위치시키면 된다. 또한, 어플리케이션이 메모리에서 해제될 때 마지막으로 실행해야 하는 코드가 있으면, 이는 finalization 섹션에 위치시키면 된다. 보통 유닛에서 사용한 리소스 등을 해제하는 코드 등을 여기에 사용한다.

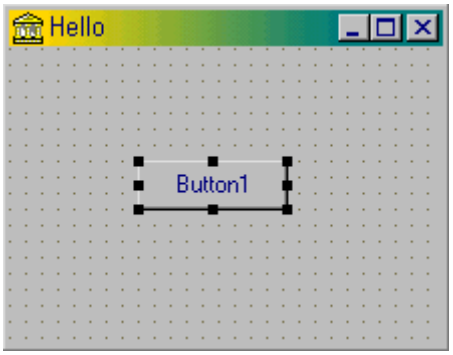
● 델파이 폼 파일의 이해

델파이의 폼 디자이너에서 작업한 내용들은 폼 파일에 저장된다. 그러므로, 폼 디자이너에서 비주얼한 환경으로 폼을 다룰 수도 있지만, 텍스트 파일을 직접 변경하는 것도 가능하다

(마치 HTML 문서를 만들 때, 텍스트 에디터를 쓸 수도 있고 프론트 페이지 같은 비주얼 툴을 사용할 수도 있는 것 처럼).

컴포넌트의 사용

폼이 델파이 신전을 이루는 기둥이라고 하면, 컴포넌트는 기둥을 이루는 벽돌이라고 할 수 있다. 이러한 컴포넌트 들은 오브젝트 파스칼 소스 코드로 만들어져 있다.
그러면 벽돌을 이용해서 집을 지어보도록 하자. 폼에 컴포넌트를 추가하는 방법은 컴포넌트 팔레트에서 컴포넌트를 클릭하고, 마우스 커서를 폼으로 이동시킨 후, 왼쪽 버튼을 눌러서 컴포넌트의 좌상부 꼭지점을 지정하고 계속 버튼을 누른 채로 컴포넌트의 우하부 꼭지점까지 드래그 하면 컴포넌트의 크기가 알맞게 정해진다. 또는, 그냥 컴포넌트를 선택하고 폼의 적당한 위치에 클릭하면 디폴트 크기의 컴포넌트가 생성되며, 컴포넌트 팔레트에서 컴포넌트를 더블 클릭하면 폼의 한 가운데에 컴포넌트가 생성된다.
Standard 페이지에서 버튼 컴포넌트 하나를 선택해서 폼에 올려 놓도록 하자.



- 프로퍼티 편집하기

앞에서 폼의 캡션을 변경한 것과 마찬가지로, 다른 컴포넌트들도 오브젝트 인스펙터를 이용해서 프로퍼티를 편집할 수 있다. 버튼의 캡션을 바꾸기 위해서는 먼저 마우스로 버튼 객체를 선택하고 오브젝트 인스펙터의 Caption 프로퍼티를 변경하면 된다. 참고로, 컴포넌트의 캡션 프로퍼티는 보통 컴포넌트의 Name 프로퍼티에 우선적으로 영향을 받는다. 물론 이름과 캡션의 문자열은 달라도 되지만, Name 프로퍼티를 변경하면 Caption 이 처음에 자동으로 설정된다. 보통 컴포넌트의 이름을 붙일 때에는 일반적으로 따르는 관습이 있는데, 많은 경우에 영어 자음으로된 소문자 2~3 자를 붙이는 이름 규칙이 가장 많이 쓰인다. 예를 들어 버튼의 경우 'btn' 이라는 문자열을 접두어로 사용하는 경우가 많다.

참고:
Name 과 Caption 프로퍼티는 처음으로 델파이를 접하는 사람들이 혼돈스러워 하는 것 중에 하나이

다. 분명히 말해서 Name 프로퍼티는 어디까지나 내부에서 사용되는 것으로 컴포넌트를 나타내는 변수의 이름이라고 생각하면 된다. 그러므로, Name 프로퍼티는 기본적으로 파스칼의 변수의 이름 규칙을 따라야 한다. 이에 비해 Caption 프로퍼티는 바깥에 보이는 부분으로 내부적인 컴포넌트의 이름과는 전혀 상관이 없다.

어쨌든 Button1 의 Name 프로퍼티를 'btnOK'로 설정하고 Caption 은 'OK' 라고 정하자. 이런 형식으로 모든 델파이 컴포넌트의 속성을 정할 수 있게 된다.

- 이벤트 핸들러의 작성

폼이나 컴포넌트에서 마우스 버튼을 누르면, 윈도우는 어플리케이션에 메시지를 보내서 그 이벤트를 알려준다. 델파이에서의 이벤트는 크게 2 가지 의미로 생각할 수 있다. 하나는 각 컴포넌트 별로 윈도우의 메시지를 포장한 것이다. 즉, 이벤트의 발생이라는 측면에서 접근하면 어디까지나 윈도우 메시지와 동일하다는 것이다. 예를 들어, 마우스 키를 누르면 해당 윈도우에는 WM_MOUSEDOWN 이라는 메시지가 전송되고, 델파이의 컴포넌트는 이를 OnMouseDown 이라는 이벤트로 발생시킨다는 것이다. 이보다 조금 언어적인 측면에서 접근하면 델파이의 이벤트는 개발자가 작성한 함수나 프로시저의 주소를 윈도우 메시지가 발생했을 때 연결해주는 함수 포인터이다. 즉, 오브젝트 인스펙터에서 객체의 published 프로퍼티로 선언된 프로시저형 데이터로 눈으로 볼 때에는 오브젝트 인스펙터의 이벤트 탭에서 나열된 메소드를 선택하여 이벤트와 메소드를 단순히 연결하는 것으로 생각할 수 있지만, 내부적으로는 호환 가능한 프로시저형의 메소드 포인터를 이벤트 탭에서 열거하면, 이를 선택하는 것으로 함수 포인터와 같은 역할을 하는 것이다.
다소 설명이 어려웠을 지도 모르지만, C 를 공부했던 독자라면 조금은 쉽게 이해했을 것으로 믿는다. 그리고, 잘 이해가 되지 않더라도 델파이로 여러 프로그램을 만들다 보면 이해가 될 날이 올 것이다.
어쨌든 이번 장의 목적은 가장 단순한 형태의 어플리케이션을 한 번 제작해보는 것이므로, 실전에 들어가 보도록 하자.
먼저 폼에 올려 놓은 버튼을 선택하고, 오브젝트 인스펙터의 이벤트(Event) 탭을 선택한다. 버튼을 클릭할 때의 이벤트 핸들러를 작성하려면 OnClick 이벤트 오른쪽 옆의 하얀 부분을 더블 클릭하거나, 여기에 새로운 메소드의 이름을 입력하고 Enter 를 치면 이벤트 핸들러를 입력할 수 있는 화면이 다음과 같이 생성된다.

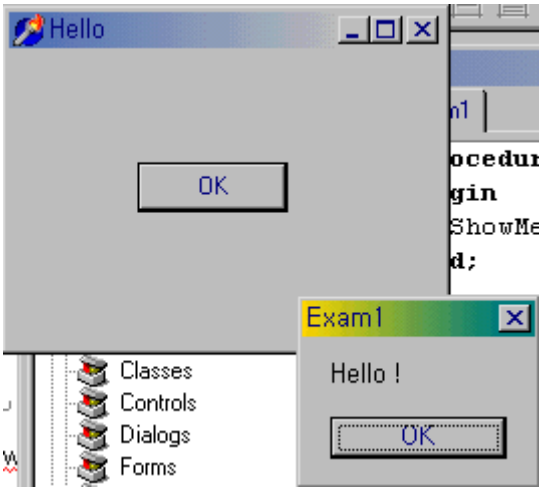
```
procedure TForm1.btnOKClick(Sender: TObject);
begin

end;
```

이제 이벤트 핸들러의 코드를 begin~end; 사이에 집어 넣으면 된다. 오브젝트 파스칼에 대한 문법을 모르더라도 걱정하지 말고, 'Hello' 라는 메시지를 출력하기 위해 다음과 같이 코드를 입력하도록 하자.

```
procedure TForm1.btnOKClick(Sender: TObject);
begin
  ShowMessage('Hello !');
end;
```

ShowMessage 는 메시지 박스를 보여주는 것으로, 파라미터로 넘어간 문자열을 단순히 보여주는 역할을 하는 것이다.
그러면, 화살표 스피드 버튼을 누르거나 Run 메뉴를 선택하여 어플리케이션을 컴파일하고 실행해보자. 버튼을 누르면 다음과 같은 화면이 나타날 것이다.



어플리케이션을 실행하면, 델파이 내부에서는 폼을 구성하는 파스칼 소스 코드를 컴파일 하고, 프로젝트 파일을 컴파일 한 후 해당되는 라이브러리와 링크할 실행 파일을 만들게 된다. 그리고 나서, 디버그 모드에서 실행 파일을 실행하는 것이다.

정 리

델파이를 이용해서 프로그래밍을 하는 방법을 대단히 간단히 알아보았다. 물론 우리가 이번 장에서 제작한 프로그램은 그렇게 큰 의미가 있는 것이 아니다. 하지만, 이 프로그램이 실제로 어떻게 구성되고 기본적인 작동 방법은 어떻게 하는 것인지를 익히는 데에는 충분했을 것이다.

이 책의 목표가 초급자에게 델파이에 대해서 아주 자세하게 가르치려는 것이 아니기 때문에, 아마도 다소 내용이 상세하지 못하다고 불평할지도 모르겠다. 하지만, 이런 상세한 내용들은 차차 프로그래밍을 해가면서, 그리고 도움말을 찾아가면서 익히면 되는 것이므로 많은 연습을 통해 배우도록 하자.
다음 장에서는 오브젝트 파스칼의 가장 핵심적이고 기초적인 문법에 대해서 알아보도록 한다.