액티브X 컨트롤, 액티브 폼의 제작

(Creating ActiveX Controls, ActiveForms)

델파이의 VCL 컴포넌트와 액티브 X 컴포넌트는 사실 의미 상으로 많은 부분이 통하지만, 실제 구현 방법은 많은 차이가 있기 때문에 VCL 컨트롤을 액티브 X 컨트롤로 전환개발하려면 사실 많은 단순 작업을 해주어야 한다. 델파이 4 에서는 이런 작업을 단순화 시키는 레이어를 제공하는데 이것이 바로 액티브 X 컨트롤 위저드이다. 마찬가지로 컴포넌트가 추가된 폼을 하나의 액티브 X 컨트롤처럼 사용할 수 있는데, 이를 액티브 폼이라고 하며 액티브 폼 역시 델파이 4 에서 제공되는 액티브 폼 위저드를 이용해서 쉽게 작성할 수 있다.
액티브 X 컨트롤에 대한 내용의 경우 필자가 가장 많이 참고한 자료는 97 년도에 볼랜드가 개최한 컨퍼런스에서 Conrad Herrman 이 발표한 컨퍼런스 자료이다. Conrad Herrman 은 inprise 에서 운영하는 뉴스 그룹에서도 액티브 X 분야에서 가장 활발한 활동을 하고 있는 사람으로 inprise 와 관련이 없는데에도 불구하고, 어려운 질문에도 즉각 즉각 답변을 해주는 사람이다. 아마도 이 글을 읽을 기회는 없겠지만 이 자리를 빌어 감사의 뜻을 전하고 싶다.

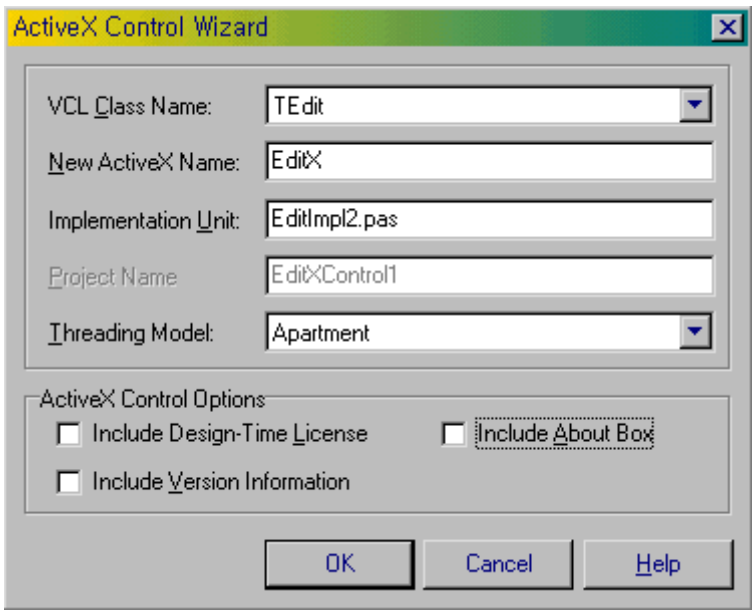
액티브X 컨트롤 위저드의 이용

사용하는 것은 간단하지만, 위저드가 내부적으로 동작하는 방법을 간단하게 설명하겠다.
먼저 자동화 인터페이스와 이벤트 인터페이스를 지정하고, 구현할 객체의 클래스 ID 를 생성한다. 그리고 나서 이 객체를 액티브 X 서버 라이브러리에서 사용할 수 있도록 포장해 준다. 이 과정에서 VCL 컴포넌트의 프로퍼티, 메소드, 이벤트를 OLE 스타일로 변환시켜주는 짧은 어댑터 루틴이 필요하다.
실제로 액티브X 컨트롤 위저드를 사용해 보자.

1. TWinControl 에 기초한 정상적인 델파이 컨트롤을 빌드하고, 이를 컨트롤 팔레트에 인스톨한다. 여기에서는 델파이 TEdit 를 이용하겠다. 이 컴포넌트를 액티브 X 컨트롤로 만들면 다른 개발도구에서도 유용하게 사용할 수 있을 것이다.
2. File|New... 메뉴를 선택하고, 여기에서 ActiveX 페이지를 선택하면 액티브 X 제작과 관련한 여러가지 마법사를 볼 수 있는데, ActiveX Control 아이템을 선택하면 다음 그림과 같은 대화상자가 나타난다.
3. VCL Class Name 항목에는 액티브 X 컨트롤로 전환시킬 VCL 클래스를 선택하면 자동으로 아래의 3 항목이 채워진다. 이를 그대로 사용해도 좋고, 다른 이름으로 바꿀 수도 있다. 참고로 각각의 항목은 새로운 액티브 X 컨트롤 클래스의 이름, 컨트롤을 구현하는

Unit 파일, 액티브 X 서버 라이브러리 프로젝트 파일을 지정한다.

4. 적절한 쓰레딩 모델을 선택한다. 보통은 Apartment 모델을 사용한다. 앞 장에서 여기에 대해서는 자세히 설명하였으므로, 이를 참조하기 바란다.
5. 대화상자 하단의 옵션 체크 박스는 나중에 설명하기로 하고, 여기서 OK 를 누르면 위저드는 액티브 X 컨트롤을 구현하는 코드와 이를 담을 액티브 X 서버 라이브러리 프로젝트 파일을 자동으로 생성하거나 수정한다.
6. 프로젝트를 빌드하면 액티브 X 컨트롤이 만들어진다.
7. 마지막으로 만들어진 컨트롤을 시스템에 등록해야 하는데, Run|Register ActiveX Sever 메뉴를 선택하면 등록이 완료된다.



액티브 X 컨트롤 위저드 대화상자

TEdit 의 경우 액티브 X 컨트롤 위저드에 의해 EditXControl1.dpr 파일과 액티브 X 서버를 구현하는 EditXImpl1.pas 파일, 프로젝트에 import 되는 타입 라이브러리 파일(.tlb)과 타입 라이브러리의 파스칼 버전이 생성된다.

액티브 X 프로젝트 파일

위저드에 의해 생성되는 프로젝트 파일의 내용을 살펴보자. 소스 코드는 다음과 같다. 이해를 돕기 위해 필자가 주석을 달았다.

```
library EditXControl1;
        //EditXControl1 이라는 DLL 파일을 생성하는 프로젝트임을 나타냄
uses
```

```
  ComServ,
  EditXControl1_TLB in 'EditXControl1_TLB.pas',
  EditImpl1 in 'EditImpl1.pas' {EditX: CoClass};
        //EditX 라는 액티브 X 클래스를 구현하는 unit 가 EditXImpl1.pas 파일에 구현되어 있음을 나
        타내는 줄

{$E ocx}            //링커에게 output 파일의 확장자가 '.ocx'임을 알려준다.

exports
  DllGetClassObject,
  DllCanUnloadNow,
  DllRegisterServer,
  DllUnregisterServer;
         //표준 액티브 X 서버 함수를 export.
         이 함수들은 ComServ unit 에 구현되어 있으므로, 따로 구현할 필요가 없다.

{$R *.TLB}          //링커에게 타입 라이브러리 파일을 DLL 의 리소스로 포함할 것을 요구한다.
{$R *.RES}          //링커에게 프로젝트의 리소스를 포함할 것을 요구한다. 여기에는 툴바 비트맵과
                    버전 정보 리소스가 포함된다.

begin
end.
```

액티브 X 컨트롤러 객체의 선언

실제로 액티브 X 컨트롤의 구현부분은 'EditXImpl1.pas' 파일에 담겨져 있다. 이 파일에는 액티브 X 컨트롤의 액티브 X 컨트롤러 객체가 구현되어 있으며, 액티브 X 컨트롤러 객체에 의해 자동화 인터페이스가 정의되고, OLE 자동화 스타일의 프로퍼티, 메소드, 이벤트가 구현된다. 주요 소스를 살펴 보자.

```
unit EditImpl1;

interface
uses
  Windows, ActiveX, Classes, Controls, Graphics, Menus, Forms, StdCtrls,
  ComServ, StdVCL, AXCtrls, EditXControl1_TLB;
```

EditXControl1_TLB.pas unit 은 파스칼 버전의 타입 라이브러리로 여기에 인터페이스가 정의되어 있다. 위저드에 의해 생성된다.

```
type
  TEditX = class(TActiveXControl, IEditX)
  //타입 라이브러리에 정의된 IEditX 인터페이스를 구현하는 컨트롤러 객체인
    TEditX 클래스를 TActiveXControl 클래스에 기초하여 선언
  private
    { Private declarations }
    FDelphiControl: TEdit;
    //VCL 컨트롤을 가리키는 필드. InitializeControl 메소드에 의해 초기화 된다. 아래에 열거되는 멤
      버들을 이용하여 프로퍼티, 메소드 등에 접근한다.
    FEvents: IEditXEvents;
    //컨테이너의 이벤트 싱커에 대한 포인터이다. 일종의 dispinterface 로 IDispatch 포인터를 저장
      한다. 이 값은 컨트롤이 컨테이너에 삽입되거나 제거될 때 EventSinkChanged 메소드가 호출되
      면 설정된다. 이 값이 nil 일 수도 있는데, 이는 컨테이너 애플리케이션이 이벤트에 대한 처리를
      하지 않는 경우이다.

    procedure ChangeEvent(Sender: TObject);
    procedure ClickEvent(Sender: TObject);
    procedure DblClickEvent(Sender: TObject);
    procedure KeyPressEvent(Sender: TObject; var Key: Char);
```

이벤트 핸들러 프록시에 대한 선언 부분이다.

```
  protected
    { Protected declarations }
    procedure InitializeControl; override;
    procedure EventSinkChanged(const EventSink: IUnknown); override;
    procedure DefinePropertyPages(DefinePropertyPage: TDefinePropertyPage); override;

   ... (중략)
```

앞의 세가지 메소드는 TActiveXControl 에 선언된 가상 메소드를 오버라이드한 것으로, 다음에 자세히 설명한다.

```
    function Get_AutoSelect: WordBool; safecall;
    function Get_AutoSize: WordBool; safecall;
    function Get_BevelInner: TxBevelCut; safecall;

    ... (중략)

    //이들은 프로퍼티의 getter 메소드로, 이들은 IEditX 인터페이스에 정의되어 있다. 이들은 모두
      safecall 호출규칙(calling convention)을 사용하는데, 이것은 오브젝트 파스칼에서 자동화 메소
      드에 호환되는 듀얼 인터페이스를 선언할 때 사용하는 것으로, 예외가 발생하면 OLE 호출규칙
      에 의거하여 OLE 에러 값을 반환한다.

    procedure Set_AutoSelect(Value: WordBool); safecall;
    procedure Set_AutoSize(Value: WordBool); safecall;
    procedure Set_BevelInner(Value: TxBevelCut); safecall;

    ... (중략)

    //이들은 프로퍼티의 setter 메소드이다. 이들 역시 IEditX 인터페이스에 정의되어 있다.

end;

    //참고로 TEditX 컴포넌트에는 액티브 X 컨트롤로 전환될 때 사용될 수 있는 public 메소드가 없
      는 관계로, 메소드에 대한 선언이 빠져 있다.

implementation

uses ComObj;

{ TEditX }

procedure TEditX.InitializeControl;
begin
  FDelphiControl := Control as TEdit;
  //Control 은 TActiveXControl 에 선언되어 있는 프로퍼티로, TWinControl 에서 상속받은 VCL 컨트롤
    을 지정한다. FDelphiControl 필드에 TEdit VCL 객체의 포인터가 저장된다.
```

```
  FDelphiControl.OnChange := ChangeEvent;
  FDelphiControl.OnClick := ClickEvent;
  ... (중략)
  //컨트롤의 VCL 이벤트를 컨트롤러 객체의 이벤트 핸들러 프록시 메소드에 매핑하는 코드이다. 이
    렇게 함으로써 VCL 컨트롤이 이벤트를 발생시키면, 컨트롤러 객체가 이벤트를 받게 된다.
end;
```

InitializeControl 메소드는 컨트롤이 생성되고, 컨테이너에 삽입되기 전에 호출되는 메소드로, COM 컨트롤러 객체와 VCL 객체 간의 커넥션을 만든다. 보다 자세하게 설명하면, 컨트롤러 객체가 VCL 객체의 포인터를 얻은 다음, 이벤트 프록시를 VCL 객체에 연결해 준다.

```
procedure TEditX.EventSinkChanged(const EventSink: IUnknown);
begin
  FEvents := EventSink as IEditXEvents;
end;
```

컨테이너가 제공하는 이벤트 싱크를 저장한다. FEvents 필드는 컨테이너 객체에 이벤트를 발생시킬 때 사용된다. IEditXEvents 는 컨트롤의 이벤트 dispinterface 로 타입 라이브러리에 디폴트 소스 인터페이스로 선언되어 있다.

```
procedure TEditX.DefinePropertyPages(DefinePropertyPage: TDefinePropertyPage);
begin

end;
```

이 메소드는 개발자가 나중에 프로퍼티 페이지를 추가할 때 사용한다. 프로퍼티 페이지를 추가하는 방법에 대해서는 나중에 설명한다.

프로퍼티의 Get, Set 메소드의 구현 방법

VCL 컨트롤의 프로퍼티는 위저드에 의해 생성되는 컨트롤러 객체의 Get, Set 메소드에 의해 접근할 수 있다. 이들이 실제로 어떻게 구현되어 있는지 살펴 보자.
가장 전형적인 Get, Set 메소드는 다음과 같다.

```
function TEditX.Get_AutoSelect: WordBool;
begin
```

```
  Result := FDelphiControl.AutoSelect;
end;

procedure TEditX.Set_AutoSelect(Value: WordBool);
begin
  FDelphiControl.AutoSelect := Value;
end;
```

대부분의 경우는 이와 같이 간단하게 구현이 되어 있지만, 프로퍼티의 데이터 형이 호환되지 않을 때에는 약간의 조작이 필요하다. 대부분의 경우 위저드가 자동으로 구현해 주지만, 지원하지 않는 데이터 형을 사용하는 public 프로퍼티는 사용할 수 없게 될 수도 있으며, 이를 구현하기 위해서 약간의 코딩이 필요할 수도 있다.
가장 복잡한 경우가 폰트, 그림, string list 등의 경우이다. 예를 들어 폰트의 경우 독립적인 dispatch 인터페이스를 가지는 독립적인 객체이기 때문에, 적절한 접근 방법이 있어야 한다. 이를 해결하려면 Get_Font 메소드는 폰트의 프로퍼티를 OLE 프로퍼티로 노출(expose)시킬 수 있는 OLE 객체를 생성하고, 이를 반환해야 하며, 반대로 Set_Font 메소드는 OLE 폰트 객체의 값을 VCL 프로퍼티에 대입할 수 있어야 한다.
다행히 이를 위해, DAX 라이브러리에서는 TFont, TPicture 클래스를 위한 함수를 제공한다. 위저드에서 자동으로 생성한 Get_Font, Set_Font 메소드는 다음과 같다.

```
function TEditX.Get_Font: IFontDisp;
begin
  GetOleFont(FDelphiControl.Font, Result);
end;

procedure TEditX._Set_Font(const Value: IFontDisp);
begin
  SetOleFont(FDelphiControl.Font, Value);
end;
```

위저드에서 일부의 프로퍼티는 생성하지 않는데, 이들의 예를 들면 Height, Left 등과 같이 Extended 프로퍼티를 사용해야 하는 경우나, ParentFont, Hints 등과 같이 액티브X 컨테이너가 지원할 수 없는 경우, PopUpMenu 와 같이 OLE 프로퍼티 데이터 형이 도저히 지원할 수 없는 경우 등이 있다.

이벤트의 처리

기본적으로 InitializeObject 메소드에 의해 VCL 컨트롤과 이벤트 핸들러가 연결된다는 것은 위에서 설명했다. 그럼 실제 이벤트를 처리하는 부분을 살펴 보자.

```
procedure TEditX.ChangeEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnChange;
end;
```

이를 분석하면, FEvents dispatch 인터페이스가 설치되어 있으면(즉, IDispatch 포인터가 할당되어 있으면) 단순히 이벤트를 컨테이너의 이벤트 싱크에 넘겨주는 것으로 되어 있다. FEvents 는 위에서 설명한 EventSinkChanged 메소드에 의해 설정된다.
KeyPressEvent 는 OnClick, OnChange 이벤트에 비해 다소 복잡하게 구현되어 있는데, 이는 OLE 이벤트에서 요구하는 파라미터가 델파이 이벤트의 파라미터와 다르기 때문에 이를 변환하는 코드가 생성되기 때문이다. 실제 코드를 살펴보자.

```
procedure TEditX.KeyPressEvent(Sender: TObject; var Key: Char);
var
  TempKey: Smallint;
begin
  TempKey := Smallint(Key);
  if FEvents <> nil then FEvents.OnKeyPress(TempKey);
  Key := Char(TempKey);
end;
```

이 경우 이벤트 핸들러 프록시는 이벤트를 컨테이너에 발생시키기 전에 파라미터를 조작하게 된다. OnKeyPress 이벤트는 SmallInt 에 대한 포인터를 컨테이너에 넘기지만, VCL 컨트롤은 Char 에 대한 포인터를 이벤트 핸들러에게 넘기므로 이를 SmallInt 와 Char 로 타입 캐스팅하는 부분이 필요한 것이다.
참고로, TDateTimePicker 와 같이 다소 복잡한 컨트롤의 경우에는 더 복잡한 처리방법을 가지는 경우도 있다. 이때에는 VCL 컨트롤에서 사용하는 TDateTime, String, Boolean 데이터 type 을 OLE 이벤트의 WideString, WordBool 등으로 타입 캐스팅할 필요가 있다. 이런 부분을 모두 위저드가 자동으로 코딩해준다.

클래스 팩토리를 통한 인스턴스의 생성

마지막으로 라이브러리가 메모리 상에 로드되었을 때, 클래스 팩토리를 생성하는 부분을 살펴보자.

```
initialization
  TActiveXControlFactory.Create(ComServer, TEditX, TEdit, Class_EditX, 1,
    '', 0, tmApartment);
```

ComServer 는 라이브러리를 나타내는 전역변수로 라이브러리에 의해 생성된 클래스 팩토리의 리스트를 포함하고 있다.
2, 3 번째 파라미터는 각각 액티브 X 를 구현하는 클래스, VCL 컨트롤 클래스이며, Class_EditX 는 EditXControl1_TLB.pas unit 에 선언되어 있는 객체의 ClassID 이다. 5, 6, 7 번째 파라미터는 각각 ToolBarBitmapID, LicenseString, MiscControl flag 등을 나타내며, 마지막 파라미터에는 쓰레딩 모델을 지정한다.

타입 라이브러리

일단 액티브 X 라이브러리를 컴파일하면 타입 라이브러리는 DLL 에 리소스로 복사된다.
액티브 X 컨트롤 위저드는 처음 액티브 X 컨트롤을 델파이 VCL 에서 생성할 때 타입 라이브러리를 만들며 이를 .TLB 파일로 저장한다. 이때 위저드는 프로퍼티와 파라미터를 OLE 에 호환되는 타입으로 전환한다. 컨트롤에 나열형 프로퍼티나 파라미터가 포함되어 있으면 이에 대한 타입 선언부를 만들어 주며, 데이터 타입이 TFont, TPicture, TStrings 일 경우에는 프로퍼티, 파라미터를 IFont, IPicture, IStrings 로 대입하고, 이들에 대한 어댑터(adapter) 코드를 생성해준다. 만약 VCL 컨트롤에 COM 객체 타입이 될 수 없는 프로퍼티나 파라미터가 있을 경우에는 위저드는 이들을 생략해 버린다.

사용자 정의 객체 스트리밍

DAX 객체에 대한 기본적인 스트리밍(streaming)은 VCL 의 포맷을 따르게 된다. 그러나, 개발자가 기본적인 정보 외에 추가적인 정보를 지속적 스트림(persistence stream)에 저장하려 한다면 LoadFromStream, SaveToStream 메소드를 오버라이딩하면 된다. 이때 컨트롤의 프로퍼티를 제대로 읽고, 저장하려면 inherited 메소드를 호출해야 한다.
이들 메소드는 다음과 같이 정의되어 있다.

```
procedure LoadFromStream(const Stream: IStream);
```

```
procedure SaveToStream(const Stream: IStream)
```

델파이의 표준 스트리밍 클래스는 TStream 이다. 이 클래스에는 Read, Write, Seek 메소드가 있으며, TPersistent 객체에서 TStream 을 이용해 스트리밍을 하게 된다. 대부분의 델파이 객체가 TPersistent 객체에서 상속받으므로, 객체의 내용을 저장할 때 TStream 을 사용한다. OLE 에서는 스트리밍 객체는 IStream 이라는 인터페이스를 제공하며, 여기에 역시 Read, Write, Seek 메소드가 정의되어 있다. 델파이 4 에서는 TOleStream 이라는 클래스를 제공하는데 이를 통해 TStream 을 IStream 으로 노출(expose)시킬 수 있다.
TActiveXControl 이 SaveToStream 메소드를 이용해서 객체의 상태를 스트림에 저장할 때 파라미터로 IStream 을 가지는 것이다. 만약 개발자가 추가 데이터를 스트림에 저장하고자 한다면 스트림 어댑터를 이용해 TPersistent 객체가 그 내용을 IStream 에 저장하게 된다. 다음 코드는 TOleStream 을 이용해 추가적인 정보를 문자열 리스트 (string list)에 담아서 저장하는 예이다. 또한, InitializeControl 메소드에서 ExtraInfo 문자열 리스트 객체를 생성하고, destructor 인 Destroy 메소드를 오버라이드하여 사용한 ExtraInfo 문자열 객체를 해제재 주어야 한다.

```
... (전략)

protected
      procedure InitializeControl; override;
      procedure EventSinkChanged(const EventSink: IUnknown); override;
      procedure LoadFromStream(const Stream: IStream); override; //추가
      procedure SaveToStream(const Stream: IStream); override;    //추가

... (중략)

var
   ExtraInfo: TStringList;

procedure TEditX.InitializeControl;
begin
   ...
   ExtraInfo := TStringList.Create;
end;

destructor TEditX.Destroy;
```

```
begin
  ExtraInfo.Free;
end;

procedure TEditX.SaveToStream(const Stream: IStream);
var
  TempStream: TStream;
begin
  inherited;
  TempStream := TOleStream.Create(Stream);
  try
    ExtraInfo.SaveToStream(TempStream);
  finally
    TempStream.Free;
  end;
end;

procedure TEditX.LoadFromStream(const Stream: IStream);
var
  TempStream: TStream;
begin
  inherited;
  TempStream := TOleStream.Create(Stream);
  try
    ExtraInfo.LoadFromStream(TempStream);
  finally
    TempStream.Free;
  end;
end;
```

컨트롤에 verb 추가하기

Verb 는 사용자가 발생시키는 액션을 말한다. 예를 들어, 메뉴 아이템에서 copy, paste, run 등을 선택하면 객체가 특정한 동작을 한다면 이들이 좋은 verb 의 예가 된다. 그럼 이러한 verb 를 우리가 제작하는 컨트롤에 추가시키는 방법을 배워보자.
Verb 를 컨트롤에 추가할 때 필요한 것은 verb 를 등록하는 코드와 verb 를 실행하는 코드

이다.
Verb 를 등록하는 것은 객체의 클래스 팩토리를 이용해서 라이브러리가 인스톨될 때 verb information 을 시스템 레지스트리에 복사하는 것을 말한다. 이런 작업은 팩토리 객체의 AddVerb 메소드를 이용하면 된다. 다음의 코드를 살펴보자.

```
const
  VERB_TEXT = 100;

initialization
  with TActiveXControlFactory.Create(ComServer, TEditX, TEdit, Class_EditX,
          1, '', 0, tmApartment) do
  begin
    AddVerb(VERB_CLICK, '&Text');
  end;
end.
```

이 코드에 의해서 'Text'라는 verb 가 컨트롤에 추가되며 사용자가 컨테이너의 메뉴 아이템에서 'Text'를 선택하면 에디트 컨트롤의 Text 프로퍼티를 변경하도록 설정하도록 하자.
컨테이너는 객체에 verb 가 있을 때 이를 나타낼 책임을 가지고 있으며, 사용자가 이를 선택했을 때 액티브 X 컨트롤을 호출하여야 한다. DAX 에서는 실제로 verb 를 실행할 때 PerformVerb 메소드를 호출한다.
앞의 Text verb 에 대응하기 위한 PerformVerb 메소드는 다음과 같은 식으로 작성하면 된다. 먼저 interface 의 protected 섹션에 메소드를 선언하고, 이를 구현한다. InputBox 함수는 Dialogs.pas 유닛에 선언되어 있으므로 uses 절에 Dialogs.pas 유닛을 추가할 필요가 있다.

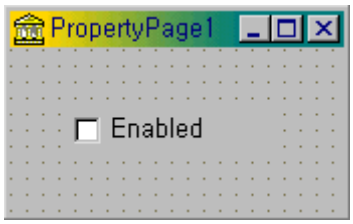
```
procedure PerformVerb(Verb: Integer); override;

procedure TEditX.PerformVerb(Verb: Integer);
var
  InputText: string;
begin
  case Verb of
  VERB_TEXT:
    begin
      InputText := InputBox('텍스트', '문자열을 입력하세요 !', '');
```

```
        FDelphiControl.Text := InputText;
      end;
    else
      Inherited PerformVerb(Verb);
    end;
end;
```

프로퍼티 페이지의 제작

프로퍼티 페이지는 사용자에게 컨트롤의 프로퍼티를 편집할 수 있게 하기 위해 제공되는 폼이다. 프로퍼티 페이지를 반드시 제공할 필요는 없는데, 이때에는 각 개발 툴의 프로퍼티 인스펙터에서 프로퍼티를 변경하게 된다. 그러나, 프로퍼티를 제공하면 사용자가 보다 쉽고 직관적인 인터페이스를 가지고 프로퍼티를 편집할 수 있게 된다.
프로퍼티 페이지 역시 OLE 객체로 액티브 X 라이브러리에 포장되고 시스템 레지스트리에 등록되어야 한다. 그러나, 이를 사용하는 컨트롤과 같은 라이브러리에 포함되어야 하는 것은 아니기 때문에 다른 컨트롤에서 등록된 같은 프로퍼티 페이지를 사용하는 것도 가능하다. 델파이에서는 폰트와 컬러, 그림과 문자열에 대한 4 가지의 기본적인 프로퍼티 페이지를 제공한다. 이들의 클래스 ID 는 각각 Class_DColorPropPage, Class_DFontPropPage, Class_DPicturePropPage, Class_DStringPropPage 이다.
프로퍼티 페이지를 추가하려면 객체 저장소나 File|New.. 메뉴의 액티브 X 페이지에서 액티브 X 프로퍼티 페이지 위저드를 실행하면 된다. 위저드를 실행하면 폼하나와 unit 파일 하나가 자동으로 생성되는데, 다음 그림은 TEdit 의 Enabled 프로퍼티를 체크하는 체크 박스를 프로퍼티 페이지에 올려 놓은 그림이며, 생성되는 코드는 다음과 같다.




```
unit EditXProp;

... (중략)

type
  TPropertyPage1 = class(TPropertyPage)
    CheckBox1: TCheckBox;
```

```
    private
    protected
    public
      procedure UpdatePropertyPage; override;
      procedure UpdateObject; override;
    end;
```

여기에서 프로퍼티 페이지 클래스는 TPropertyPage 클래스에서 상속받음을 알 수 있으며, 폼에 삽입한 CheckBox 컨트롤이 있다. TPropertyPage 클래스는 TCustomForm 클래스에서 상속받은 것으로 OleObject 프로퍼티를 이용해 프로퍼티 페이지와 객체의 편집을 연결하며, UpdatePropertyPage, UpdateObject 메소드를 제공하는데, 이를 오버라이드하여 실제로 OLE 객체를 편집하게 된다. 이들 메소드를 다음과 같이 구현하였다.

```
procedure TPropertyPage1.UpdatePropertyPage;
begin
  CheckBox1.Checked := OleObject.Enabled;   //이 부분을 실제로 추가해 주어야 함
end;

procedure TPropertyPage1.UpdateObject;
begin
  OleObject.Enabled := CheckBox1.Checked;   //이 부분을 실제로 추가해 주어야 함
end;
```

UpdatePropertyPage 메소드는 OLE 객체의 프로퍼티를 프로퍼티 페이지에 반영하는 역할을 하며, UpdateObject 메소드는 프로퍼티 페이지의 내용을 OLE 객체에 반영한다.

```
initialization
  TActiveXPropertyPageFactory.Create(ComServer, TPropertyPage1, Class_PropertyPage1);
end.
```

이 코드는 프로퍼티를 COM 객체로 등록하는 것으로, TActiveXPropertyPageFactory 의 메소드를 사용하게 된다. ComServer 는 액티브 X 라이브러리를 대표하는 전역 변수이며, TpropertyPage1 은 폼의 클래스, Class_PropertyPage1 은 프로퍼티 페이지 객체의 클래스 ID 이다. 이렇게 제작한 프로퍼티를 실제로 액티브 X 컨트롤에 적용하려면 DefinePropertyPages 메소드를 변경하면 되는데, 처음 만들어진 코드에는 이 부분이 주석으로 채워져 있다. 이를 다음과 같은 코드로 바꾸어 주면 된다.

```
procedure TdateTimePickerX.DefinePropertyPages(DefinePropertyPage: TdefinePropertyPage);
begin
  DefinePropertyPage(Class_PropertyPage1);      //추가된 부분
end;
```

여기에 여러 개의 프로퍼티 페이지를 DefineProperyPage 메소드를 이용해서 컨트롤이 여러 개의 등록된 프로퍼티 페이지를 이용하도록 할 수 있다.

Ambient 프로퍼티의 이용

Ambient 프로퍼티는 컨트롤의 컨테이너에 의해 제공되는 프로퍼티를 말한다. 일단 컨트롤이 컨테이너에 삽입되면, 컨테이너의 ambient 프로퍼티의 정보를 물어보게 된다. 그러므로, 컨테이너는 노출시키려는 ambient 프로퍼티를 정의하고 있어야 한다. 액티브 X 는 BackColor, DisplayName 등의 표준 ambient 프로퍼티를 정의하고 있다. 물론 이들을 모두 컨테이너가 노출할 필요는 없지만, 마이크로소프트는 이들을 사용하기 위해 각각에 대한 dispID 를 정의하고 있다. 그러므로, 액티브 X 에서 ambient 프로퍼티에 접근할 때에는 사이트의 IDispatch 인터페이스를 사용한다. 델파이에서는 IAmbientDispatch 인터페이스를 제공하는데, 이 인터페이스를 이용하여 표준 인터페이스에 접근이 가능하다.
IAmbientDispatch 인터페이스는 어디까지나 dispinterface 이므로 단지 IDispatch 의 포인터일 뿐이며, 다른 dispinterface 로 형변환이 가능하다. IAmbientDispatch 인터페이스의 선언부는 다음과 같다.

```
IAmbientDispatch = dispinterface
  ['{00020400-0000-0000-C000-000000000046}']
  property BackColor: Integer dispid DISPID_AMBIENT_BACKCOLOR;
  property DisplayName: WideString dispid DISPID_AMBIENT_DISPLAYNAME;
  property Font: IFontDisp dispid DISPID_AMBIENT_FONT;
  property ForeColor: Integer dispid DISPID_AMBIENT_FORECOLOR;
  property LocaleID: Integer dispid DISPID_AMBIENT_LOCALEID;
  property MessageReflect: WordBool dispid DISPID_AMBIENT_MESSAGEREFLECT;
  property ScaleUnits: WideString dispid DISPID_AMBIENT_SCALEUNITS;
  property TextAlign: Smallint dispid DISPID_AMBIENT_TEXTALIGN;
  property UserMode: WordBool dispid DISPID_AMBIENT_USERMODE;
  property UIDead: WordBool dispid DISPID_AMBIENT_UIDEAD;
  property ShowGrabHandles: WordBool dispid DISPID_AMBIENT_SHOWGRABHANDLES;
```

```
  property ShowHatching: WordBool dispid DISPID_AMBIENT_SHOWHATCHING;
  property DisplayAsDefault: WordBool dispid DISPID_AMBIENT_DISPLAYASDEFAULT;
  property SupportsMnemonics: WordBool dispid DISPID_AMBIENT_SUPPORTSMNEMONICS;
  property AutoClip: WordBool dispid DISPID_AMBIENT_AUTOCLIP;
end;
```

다음의 코드는 TEditX 컨트롤을 클릭할 때 컨테이너에서의 컨트롤의 이름을 메시지 박스로 나타내도록 한 것이다. 이를 위해 DisplayName ambient 프로퍼티를 이용하였다.

```
procedure TEditX.ClickEvent(Sender: TObject);
var
  Site: IOleClientSite;
  Ambients: IDispatch;
begin
  GetClientSite(Site);
  if Site <> nil then
    Site.QueryInterface(IDispatch, Ambients);
  if Ambients <> nil then
  begin
    ShowMessage(IAmbientDispatch(Ambients).DisplayName);
  end;
  if FEvents <> nil then FEvents.OnClick;
end;
```

이렇게 하면, 앞으로는 에디트 박스를 클릭할 때마다 폼에서의 에디트 박스의 Name 프로퍼티의 내용이 메시지 박스로 뜰 것이다.
Ambient 프로퍼티를 가장 유용하게 사용되는 경우는 컨테이너의 색상 변화에 따라 컨트롤의 색상 변화를 유도하고 싶다거나, 글꼴 변화에 따른 컨트롤의 변화와 같이 사용자 인터페이스 측면에서 반드시 사용해야할 경우 들이 있다. 이럴 때에는 ambient 프로퍼티의 이용이 유일한 해결책이 될 수 있다.
이를 위해서 컨테이너가 ambient 프로퍼티의 값을 변경했을 때, 이를 알아챌 수 있어야 하는데 이때에는 컨테이너 객체의 OnAmbientPropertyChange 메소드를 호출하게 된다. 그러므로 델파이에서 이를 이용하려면 OnAmbientPropertyChange 메소드를 오버라이드하고, IOleControl 인터페이스를 다시 구현하면 된다.

사용자 정의 레지스트리 엔트리의 추가

액티브 X 컨트롤을 작성한 뒤에는 레지스트리에 컨트롤에 대한 정보를 추가하고 싶을 경우가 있다. 보통 액티브 X 컨트롤이 등록될 때 이런 작업이 병행된다면 좋을 텐데, 이를 해결하는 방법이 없는지 궁금하지 않은가 ?
이를 위해서는 델파이 COM 팩토리의 UpdateRegistry 메소드를 이용하면 된다. 이 메소드는 라이브러리가 등록되거나 해제될 때 호출되기 때문에 여기에다가 레지스트리에 각종 정보를 추가하는 코드를 삽입하면 된다. 그러므로, UpdateRegistry 메소드를 오버라이드해야 하므로 TActiveXControlFactory 클래스를 상속받은 새로운 클래스 팩토리 클래스를 정의하고, 이 클래스 팩토리를 이용하여 액티브 X 라이브러리를 등록하도록 수정해야 한다.
다음의 클래스는 클래스 팩토리의 constructor 에 레지스트리에 등록할 문자열을 미리 파라미터로 받아서 이를 레지스트리에 등록하도록 수정한 클래스 팩토리 클래스를 구현한 것이다.

```
type
  TRegistryFactory = class(TActiveXControlFactory)
  public
    constructor Create(ComServer: TComServerObject; ActiveXControlClass: TActiveXControlClass;
      WinControlClass: TWinControlClass; const ClassID: TGUID; ToolboxBitmapID: Integer;
      const LicStr: string; MiscStatus: Integer; ThreadingModel: TThreadingModel = tmSingle;
      SpecialKeyValue: string); override;
    procedure UpdateRegistry(Register: Boolean); override;
  protected
    FSpecialKeyValue: string;
  end;

constructor TRegistryFactory.Create(ComServer: TComServerObject;
  ActiveXControlClass: TActiveXControlClass; WinControlClass: TWinControlClass;
  const ClassID: TGUID; ToolboxBitmapID: Integer; const LicStr: string; MiscStatus: Integer;
 ThreadingModel: TThreadingModel = tmSingle; SpecialKeyValue: string);
var
  TypeAttr: PTypeAttr;
begin
  FSpecialKeyValue := SpecialKeyValue;
  inherited Create(ComServer, ActiveXControlClass, WinControlClass, ClassID,
    ToolboxBitmapID, LicStr, MiscStatus, ThreadingModel);
```

```
end;

procedure TRegistryFactory.UpdateRegistry(Register: Boolean);
var
  ClassKey: string;
begin
  ClassKey := 'CLSID₩' + GUIDToString(ClassID);
  if Register then
  begin
    inherited UpdateRegistry(Register);
    CreateRegKey(ClassKey + '₩SpecialKey', '', FSpecialKeyValue);
  end
  else
  begin
    DeleteRegKey(ClassKey + '₩SpecialKey');
    inherited UpdateRegistry(Register);
  end;
end;
```

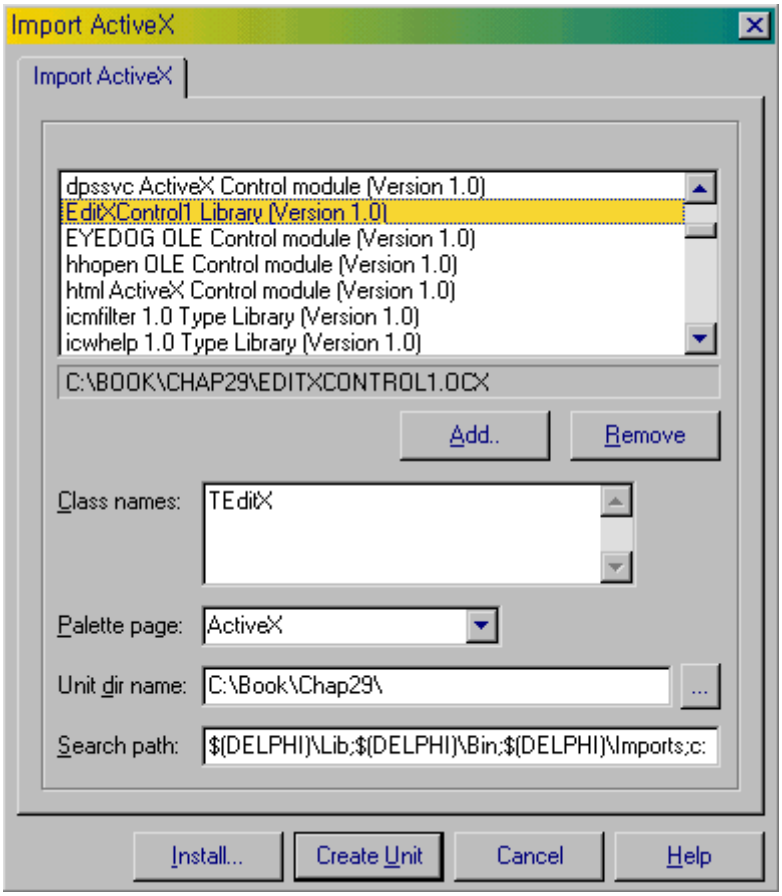
이 클래스 팩토리를 이용하기 위해서는 액티브 X 라이브러리의 initialization 섹션을 다음과 같이 수정하면 된다. 다음의 경우 추가적인 레지스트리 키 값을 'Sample'로 설정하는 경우이다.

```
initialization
  TRegistryFactory.Create(ComServer, TEditX, TEdit, Class_EditX, 1, '', 0, tmApartment, 'Sample');
end.
```

액티브 X 컨트롤의 등록과 이용

그러면, 이렇게 작성한 액티브 X 컨트롤을 등록하고 이용하는 방법에 대해서 알아보자. 여기에서 설명하는 내용은 인터넷에서 쉽게 구할 수 있는 여러가지 다른 액티브 X 컨트롤에 대해서도 공통적으로 적용된다고 할 수 있다.
델파이에서 액티브 X 컨트롤을 이용하려면 먼저 .ocx 파일을 레지스트리에 등록시켜야 한다. 델파이를 이용해서 작성한 경우에는 컴파일하고, 간단히 Run|Register ActiveX Server 메뉴를 이용하여 등록이 가능하지만, 인터넷에서 구한 경우에는 RegSvr.EXE 와 같은 등록 프로그램을 이용하거나 .REG 파일을 작성하여 등록해야 한다.

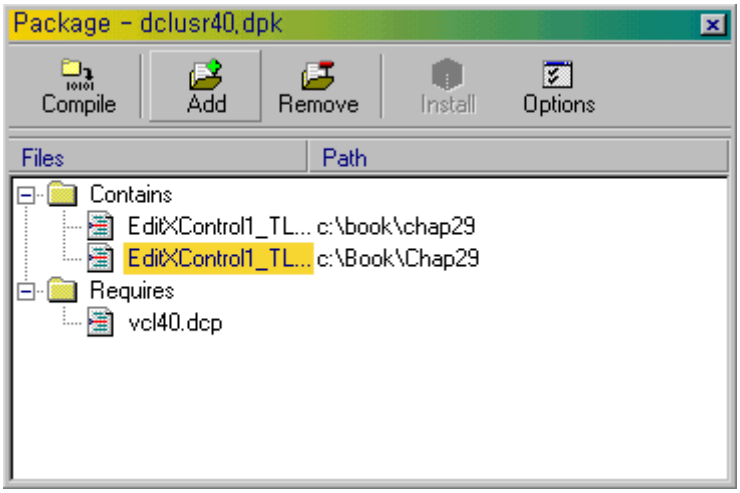
EditX 컨트롤은 간단히 컴파일하고 Run|Register ActiveX Server 메뉴를 선택하여 레지스트리에 등록할 수 있다. 이렇게 등록한 액티브 X 컨트롤을 이용하려면 델파이의 컴포넌트 팔레트에 추가해야 한다. 이를 위해서는 Component|Import ActiveX Control 메뉴를 선택하거나, Component|Install Packages 메뉴를 선택하여 패키지를 먼저 선택한 뒤에 Edit 버튼을 클릭하고 액티브 X 컨트롤을 추가할 수 있다.



이 대화 상자에서 팔레트 페이지의 이름과 생성될 _TLB.pas 파일의 위치를 지정할 수 있다. Search Path 에디트 박스에는 액티브 X 서버 파일의 위치를 검색할 디폴트 패스를 나열하게 된다. 추가가 가능하며, 여기서 지정한 패스가 아닌 곳에 위치한 액티브 X 서버는 이 대화 상자에 나타나지 않는다. 이런 경우에는 Add 버튼을 클릭하여 액티브 X 서버 파일의 위치를 지정할 수 있다.

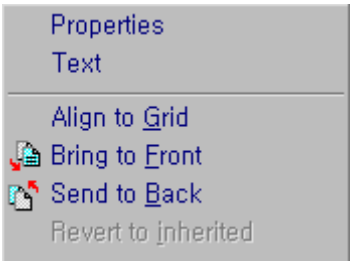
Install.. 버튼을 클릭하면 설치할 패키지를 지정하라는 대화 상자가 나타날 것이다. 여기에서 적절한 패키지 파일을 선택하거나, 새로운 패키지를 지정하여 설치할 수 있다.

이렇게 하면, 다음과 같은 패키지 관리자가 나타나면서 설치가 될 것이다.

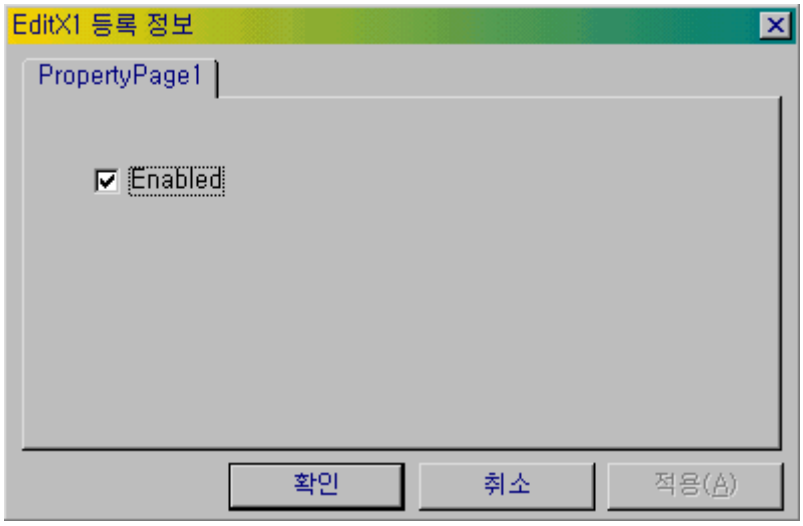


변경 사항이 있는 경우에는 Compile 버튼을 누르면 컴포넌트 팔레트에 변경된 내용이 즉시 반영된다. 여기에서 더 추가하거나 제거하고 싶은 컴포넌트나 액티브 X 컨트롤이 있으면 Add 버튼을 클릭하여 추가나 제거가 가능하다.

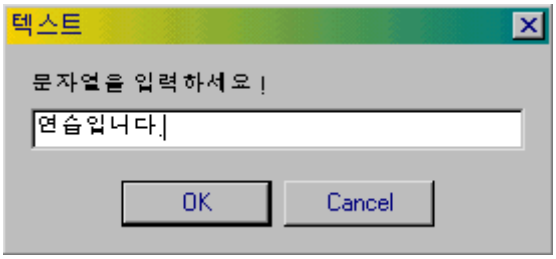
이제 새로운 어플리케이션을 시작하고, EditX 컨트롤을 사용해보자. 컴포넌트 팔레트의 ActiveX 탭에서 EditX 컨트롤을 폼에 올려 놓도록 하자. 아마도 TEdit 컴포넌트를 올려 놓는 것과 거의 같다는 것을 느낄 수 있다. 사용법 역시 동일하다. TEditX 컨트롤 위에서 오른쪽 버튼을 클릭하여 팝업 메뉴를 띄워 보자. 아마도 기본적인 Properties 메뉴와 추가한 Verb 메뉴인 Text 메뉴가 Verb 메뉴로 다음과 같이 나타날 것이다.



여기서 Properties 메뉴를 선택하여 우리가 추가한 프로퍼티 페이지가 다음과 같이 나타날 것이다.



여기서 체크 박스의 내용을 이용하여 Enabled 프로퍼티를 조절할 수 있다.
그러면, 이번에는 Text verb 를 실행해 보도록 하자. 그러면, 우리가 코딩한 대로 다음과 같은 입력 박스가 나타날 것이다. 여기에서 아무 값이나 입력하면 에디트 컨트롤의 내용인 입력한 값으로 변경될 것이다.



나머지 사용 방법은 TEdit 컴포넌트와 거의 유사하게 사용할 수 있다. 다른 액티브 X 컨트롤도 이와 비슷한 방법으로 쉽게 사용할 수 있다.

액티브 폼(ActiveForm)의 제작

액티브 폼은 VCL 폼에 기반을 둔 액티브 X 컨트롤이라고 생각하면 된다. 그러니까, 일종의 액티브 X 컨트롤인 셈이다. 그렇지만, 델파이 3 에서 발표된 액티브 폼은 인트라넷 개발 환경을 지원하기 위해서 여러모로 활용되면서 나름대로 독자적인 입지를 굳힌 듯하다. 비록 인터넷에서 범용 액티브 X 컨트롤로 사용하기에는 다소 무리가 따르지만 일정 정도의 네트워크 속도가 보장되는 인트라넷 환경에서는 델파이에서 일반적으로 개발하는 폼에 기반한 개발방법을 그대로 적용해서 하나의 컨트롤로 만들어낼 수 있다는 것은 나름대로의 매력이 있다고 생각할 수 있다.
액티브 폼은 델파이에서 제공하는 액티브 폼 위저드를 이용하여 작성한다. 먼저 File|New

메뉴의 ActiveX 탭에서 ActiveX Library 아이콘을 더블 클릭한다. 액티브 폼 역시 액티브 X 컨트롤이기 때문에 .ocx 파일 형태의 in-process 액티브 X 라이브러리로 등록되어 사용된다.
이렇게 하면, 델파이에서 비어있는 프로젝트가 하나 생성될 것이다. 그 다음에는 폼을 생성할 차례이다. 마찬가지로 File|New 메뉴의 ActiveX 탭에서 이번에는 ActiveForm 아이콘을 더블 클릭하면 액티브 X 컨트롤 위저드를 생성할 때와 거의 동일한 대화상자가 나타날 것이다. 여기에서 적당한 이름과 모델 들을 선택하고 OK 버튼을 클릭하면, 흔히 보는 비어있는 폼이 하나 만들어질 것이다.
여기까지 별로 한 것이 없어 보이지만, 델파이가 내부적으로 많은 코딩을 해 놓은 상태이다. 개발자가 할 일은 이 폼을 이용해서 마음대로 사용자 인터페이스를 디자인하고 필요한 코딩을 해주면 된다.
개발하는 방법은 일반적인 폼을 이용해서 개발하는 것과 완전히 동일하다. 컴포넌트를 올려 놓고, 거기에 대한 이벤트 핸들러를 작성한다.
단지 다른 점이 있다면, 컴파일을 하고 나서 이 폼을 실제로 실행할 수 있는 것은 HTML 페이지 위에서만 할 수 있다는 것이다. 다음에 설명하는 웹 배포 방법을 이용해서 액티브 폼을 HTML 파일로 변경하고, 이 파일을 열어 보면 액티브 폼이 제대로 실행되는지 알아볼 수 있다.
그러면 만들어진 빈 폼에 다음과 같이 버튼을 하나 올려 놓고 Caption 프로퍼티를 'OK!'로 설정하고, OnClick 이벤트 핸들러를 다음과 같이 작성한다.

```
procedure TActiveFormX.Button1Click(Sender: TObject);
begin
  ShowMessage('연습입니다 !');
end;
```

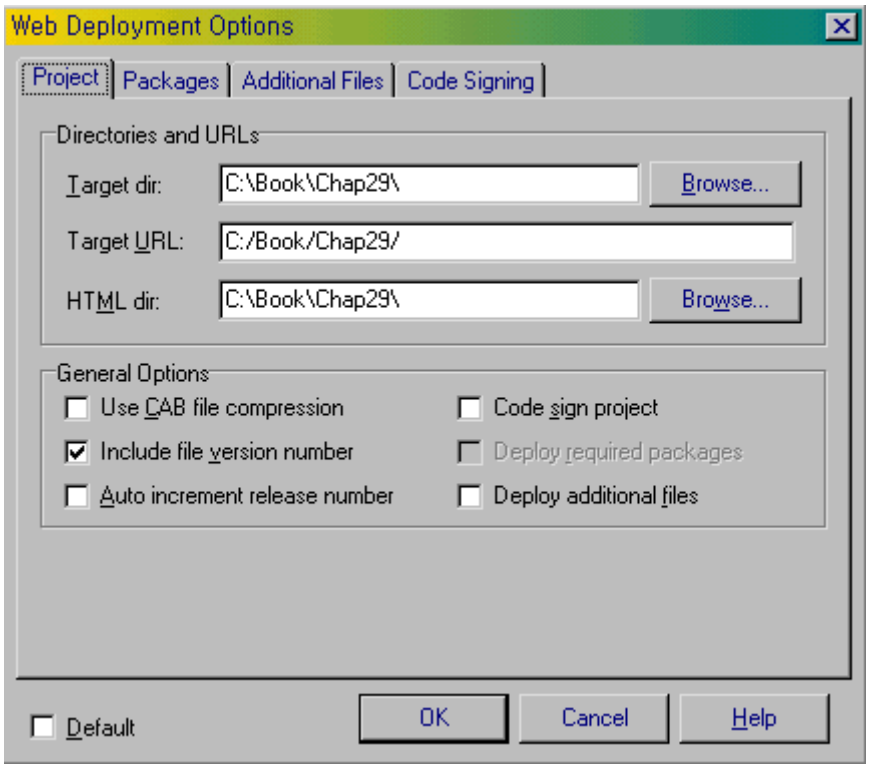
그리고, 컴파일을 하면 프로젝트 파일에 대한 .ocx 파일이 생성될 것이다. 앞으로 할 일은 웹을 이용해 배포할 수 있도록 옵션을 지정하고, 실제로 배포를 하면 완료된다.

웹 배포 (Web Deployment)

델파이로 제작한 액티브 X 컨트롤을 웹 사이트에 배포할 때에는 다음과 같은 몇 가지를 지정해 주어야 한다. 웹 배포의 옵션은 Project|Web Deployment Options 메뉴를 선택하면 설정할 수 있다. 이 메뉴를 선택하면 웹 배포 옵션을 위한 대화 상자가 나타나는데, 여기에서 가장 중요한 부분은 다음에 설명하는 Target dir, Target URL, HTML dir 에 대한 내용이다.

1. Target dir: 델파이가 복사할 코드베이스 파일이 위치할 디렉토리를 지정한다.
2. Target URL: 바이너리 코드베이스가 존재하는 서버를 지정한다. 이는 URL 의 형태인데 예를 들어 'http://www.sample.com/code/Sample.ocx' 등과 같은 형태를 가진다.
3. HTML dir: 델파이가 복사할 HTML 파일이 위치할 디렉토리를 지정한다. HTTP 서버가 로컬 머신에 있다면 'c:₩httpsW₩codebase₩' 과 같은 패스 이름이 된다.

일단 이들 옵션을 지정하면, Project|Web Deploy 메뉴를 선택하여 코드를 서버로 복사할 수 있다. Web Deploy 를 선택하면 델파이는 컨트롤이 포함될 웹 페이지를 HTML 디렉토리에 복사하고, 컨트롤을 코드베이스 디렉토리에 복사한다.
그러면 앞에서 간단하게 만들어본 액티브 폼 .ocx 파일을 배포해 보도록 하자.
Project|Web Deployment Options 메뉴를 선택하고, 대화 상자의 내용을 다음과 같이 설정한다.

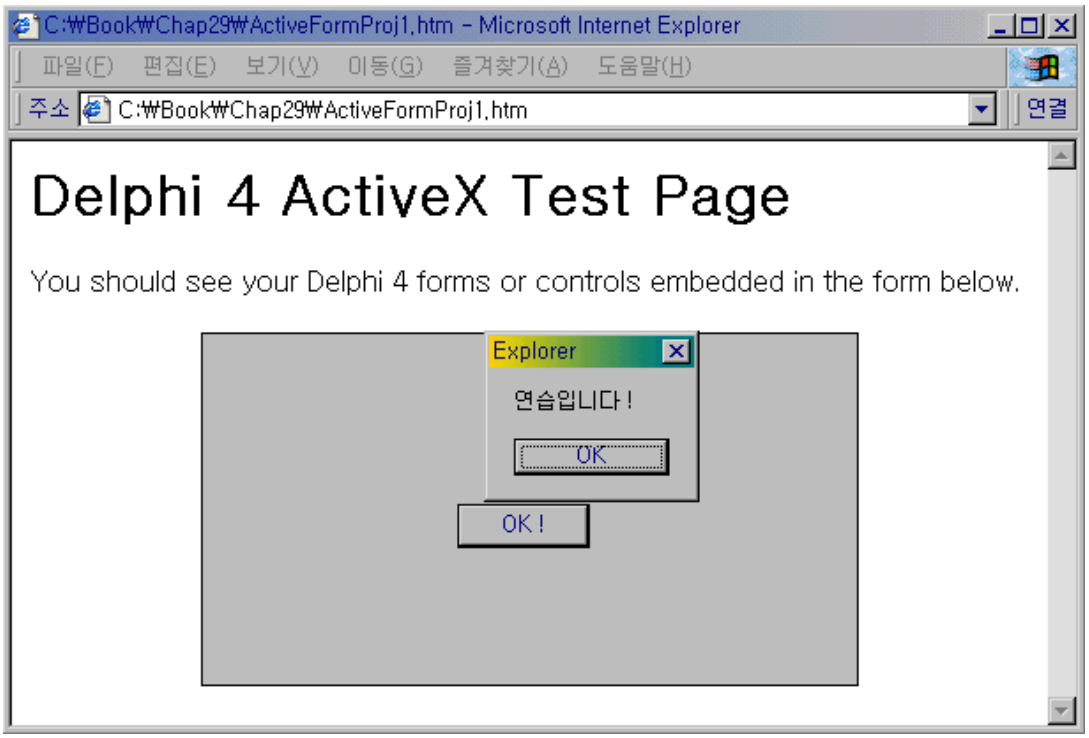


물론 여기에서 보여준 디렉토리 정보는 개발자의 컴퓨터의 .ocx 파일의 위치에 해당되는 것이므로 모두들 다를 것이다. 아마도 제공되는 CD-ROM 의 HTML 파일의 내용도 이렇게 설정되어 있으므로 액티브 폼의 디렉토리가 'C:₩Book₩Chap29'가 아니면 액티브 폼의 내용을 IE 로 볼 수 없을 것이다. 그럴 때에는 HTML 파일을 열어서 자신의 디렉토리에 맞게 변경하기만 하면 된다.
여기서 URL 을 'c:/Book/Chap29'로 설정하였는데, 물론 앞에서 간단히 설명했듯이 웹 서버

의 위치를 알면 웹 서버의 URL 이름을 이용하여 디렉토리까지만 설정하면 된다. 여기서는 개발자가 일단 웹 서버가 없어도 컨트롤을 쉽게 보고 디버깅할 수 있도록 개발자의 컴퓨터의 물리적인 디렉토리를 그대로 지정하였다. 여기서 주의할 것은 URL 명명 규칙상 '₩'가 아닌 '/'을 사용한다는 것이다. 그리고, 만약에 서버가 리모트 웹 서버이면, 로컬 컴퓨터에 HTML 파일과 코드 베이스 파일을 저장하도록 지정한 후, FTP 를 이용해서 웹 페이지에 배포해야 한다. OK 버튼을 클릭하고, Project|Web Deploy 메뉴를 선택하면 지정된 디렉토리에 .htm 파일이 생성될 것이다. 필자의 HTML 파일의 코드는 다음과 같다.

```
<HTML>
<H1> Delphi 4 ActiveX Test Page </H1><p>
You should see your Delphi 4 forms or controls embedded in the form below.
<center><P>
<OBJECT
        classid="clsid:761B37B2-31E6-11D2-9774-0000E838052E"
        codebase="C:/Book/Chap29/ActiveFormProj1.ocx"#version=1,0,0,0
        width=372
        height=200
        align=center
        hspace=0
        vspace=0
>
</OBJECT>
</HTML>
```

물론, HTML 태그를 이용하여 <OBJECT>, </OBJECT> 사이의 width, height 등의 내용을 변경하여 폼의 크기도 변경할 수 있고, 그 밖의 다른 내용도 에디터를 이용하여 쉽게 변경이 가능하다. 그러면, IE 를 띄우고 만들어진 HTML 파일을 더블 클릭하여 실제 액티브 폼을 띄워 보자. 이때 다음에 설명할 코드 사인을 하지 않은 .ocx 파일이 포함되지 않은 액티브 폼이므로 IE 의 보기|인터넷 옵션 메뉴를 선택하고 보안 레벨을 '낮음'으로 선택해야 액티브 폼을 볼 수 있다. 보안 레벨을 '보통' 이상으로 설정한 경우에는 코드 사인이 된 액티브 X 컨트롤만 볼 수 있다.



코드 사인 (Code Signing)

코드 사인의 가장 중요한 요소는 인증을 부여할 수 있는 업체에게서 코드키(code key)를 부여 받는 것이다. 마이크로소프트에서는 테스트의 목적으로 키를 생성할 수 있는 방법을 제공하고 있다. 마이크로소프트의 웹 사이트에 가면 이에 대한 정보를 얻을 수 있으며, 'Authenticode 2.0'에 대한 글을 읽어 보면 도움이 될 것이다.

델파이 4 는 프로젝트 옵션 페이지에서 code signature 정보를 지정할 수 있다. Project|Web Deployment Options 대화 상자의 Project 페이지에서 Code Sign Project checkbox 를 체크하고, Code Signing page 에서 배포할 라이센스 파일의 이름과 private key 를 지정하면 끝난다.

Application name, optional company URL 필드는 컨트롤이 다운로드될 때 이곳에 회사의 정보를 적어 넣으면 된다.

또한, 암호화 알고리즘을 선택할 수 있는데 디폴트로는 가장 흔히 사용되는 MD5 가 선택되어 있을 것이다. MD5 는 RSA Data Security 에 의해 개발된 해쉥 알고리즘으로 128 비트 해쉬(hash) 값을 만들어 낸다. 다른 것으로 SHA 1 을 선택할 수 있는데, 이것은 NIST(National Institute of Standards and Technology)와 NSA(National Security Agency)에 의해 개발된 해쉥 알고리즘으로 160 비트 해쉬 값을 만들어 낸다.

웹 보안 (Web Security)

액티브 X 컨트롤은 강력하고 편리하지만 사용자에게 심각한 위험을 초래할 수도 있다. 액티브 X 컨트롤을 만드는 사람이면 꼭 고려해야 할 사항이 있어서 여기에 몇가지 원칙을 나열하였다.

1. 당연한 말이지만 해로운 액티브 X 컨트롤을 만들면 안된다.
2. 만들어진 컨트롤이 함부로 변형될 수 있으면 안된다.
3. 아무나 함부로 사용할 수 있도록 해서는 안된다. 가능하면 인증을 받고 패스워드에 신경을 쓰도록 한다.
4. 가능하면 어떤 사람이, 어디서 컨트롤을 다운로드 받았는지 파악할 수 있도록 서버를 설정한다.

액티브 X 컨트롤은 인터넷 보다는 사실 인트라넷 환경에 적절하다고 할 수 있다. 인트라넷에 액티브 X 컨트롤을 배포하는 사람은 다음과 같은 사항에 신경을 쓰는 것이 좋다.

1. Code signature 는 완벽한 보안이 되지 못한다. 그러므로 과신은 금물이다.
2. 아주 믿을만하지 못한 사람이 만든 HTML 파일에 포함된 액티브 X 컨트롤을 받아들이지 않는 것이 좋다.

추가적인 배포 옵션

그 밖에도 웹 배포를 할 때 지정할 수 있는 옵션 들이 많은데, 여기에 대해서 조금 더 알아보도록 하자.
CAB 파일을 설정하면 윈도우 95 에서 사용되던 캐비넷이라는 압축 파일로 파일 라이브러리를 관리할 수 있도록 할 수 있다. 캐비넷 압축은 다운로드 속도를 많이 줄여 주므로, 인터넷 환경에서는 매우 유용하게 사용될 수 있다. 설치 도중에 브라우저가 캐비넷에 저장된 파일의 압축을 해제해서 저장하므로, 수행 성능의 저하나 비효율성은 거의 없다고 생각해도 된다.
보통 액티브 X 컨트롤이나 액티브 폼을 델파이에서 개발하여 배포할 때에는 처음에 런타임 패키지를 배포하고, 작아진 컨트롤을 업그레이드로 배포하는 것이 효율적이다. 이렇게 하기 위해서 처음에 패키지에 대한 옵션을 설정할 수 있다. 참고로, 델파이에 의해 제공되는 모든 패키지는 기본적으로 볼랜드에서 코드 사인한 것이므로 쉽게 CAB 파일로 통합해서 배포할 수 있다.
이렇게 액티브 X 컨트롤을 배포할 때 패키지나 다른 추가적인 파일과 함께 배포되어야 한다면, .INF 파일이 자동으로 생성된다. 이 파일은 다운로드할 파일과 액티브 X 라이브러리를

설정하는 방법 등에 대한 내용을 지정하게 된다.

- 옵션의 조합

다음 테이블은 패키지와 CAB 파일 압축, 코드 사인에 대한 정보를 옵션의 체크 박스에서 선택할 때의 결과에 대해서 정리한 것이다.

패키지/ 추가 파일	CAB 압축	코드 사인	결 과
X	X	X	.OCX 파일 단독
X	X	O	.OCX 파일 단독
X	O	X	.OCX 파일을 포함한 CAB 파일
X	O	O	.OCX 파일을 포함한 CAB 파일
O	X	X	.INF 파일과 .OCX 파일 그리고 추가적인 파일과 패키지
O	X	O	.INF 파일과 .OCX 파일 그리고 추가적인 파일과 패키지
O	O	X	.INF 파일, .OCX 파일을 포함한 .CAB 파일, 그리고 추가적인 파일과 패키지를 포함한 CAB 파일
O	O	O	.INF 파일을 포함한 CAB 파일, .OCX 파일을 포함한 CAB 파일, 추가적인 파일과 패키지를 포함한 CAB 파일

- Packages, Additional Files 탭

웹 배포 옵션의 Packages, Additional Files 탭에서 어떤 패키지를 배포할 것인지 지정할 수 있다. 특정 패키지의 설정을 변경하려면 Packages used by project 리스트 박스에 나열된 패키지 중에서 변경할 패키지를 선택하면 된다. 기본적으로 이들 탭의 옵션 내용은 동일하다.
CAB 옵션에서 Compress in a separate CAB 옵션은 각 패키지 별로 분리된 .CAB 파일을 생성한다. 이 옵션이 디폴트로 설정되어 있다. Compress in a project CAB 옵션은 프로젝트 .CAB 파일에서의 패키지를 포함하게 하는 옵션이다.
Output 옵션은 패키지가 버전 정보를 가지고 있거나 코드 사인에 대한 내용을 설정하는 옵션으로 Use file VersionInfo 옵션은 패키지 파일의 리소스에 저장된 버전 정보를 .INF 파일에서 사용할 수 있도록 한다. Code sign file 옵션은 패키지나 .CAB 파일의 코드 사인을 한다.
Directory and URL options 옵션에서 Target URL 에디트 박스는 URL 형태로서의 웹 서버의 패키지 위치를 지정한다. 이 에디트 박스가 비어 있으면, 목적 기계에 이미 파일이 존재하는 것으로 간주한다. 즉, 패키지가 지정된 위치에 존재하지 않으면, 액티브 X 컨트롤

의 다운로드가 실패한다. Target directory 에디트 박스에는 웹 서버에서 패키지의 패스를 지정한다.

정 리 (Summary)

이번 장을 마치기 전에 관심있는 독자들을 위해 델파이와 COM, 액티브 X 컨트롤에 대해 공부할 거리를 제시하고자 한다. 기본적으로는 TActiveXControl 이 지원하는 각종 인터페이스의 정의와 설명을 마이크로소프트의 SDK 등의 자료를 이용해서 숙지하는 것이 필요하며, 그 밖에 앰비언트 프로퍼티(컨테이너에서 제공되는 프로퍼티), 커스텀 레지스트리 엔트리를 추가하는 법 등의 정보를 알아보는 것이 좋겠다.
가장 좋은 공부법은 다소 어렵더라도 Inprise 에서 제공하는 뉴스 그룹을 참조하는 것이 가장 큰 도움이 될 것으로 생각되는데, borland.public.delphi.activex.controls.writing 과 borland.public.delphi.oleautomation 의 2 사이트가 가장 많은 정보를 가지고 있으며, 이를 읽다 보면 좋은 사이트가 많이 소개되어 있으니 한번씩 들러보는 것이 도움이 될 것이다.