모듈 2

버튼 누름 Button Press 이벤트 다루기 Event Handling

마우스 클릭, 버튼 누름, 키보드 누름, 창 열기 등 사용자의 행위에 반응하도록 코드를 작성할 수 있게 됩니다.

비주얼 컴포넌트에서는 마우스 클릭, 버튼 누름, 키보드 누름, 창 열기 등 수십 종류의 이벤트들을 만들고, 다룰 수 있다. 이것을 이벤트 핸들링event handling이라고 한다.

사용자가 마우스 버튼을 클릭하면 운영체제인 윈도우는 "마우스 버튼이 클릭되었다"는 메시지를 해당 애플리케이션에게 전달한다("윈도우 메시징"이라고 한다). 메시지를 받은 프로그램은 전달 받은 이벤트에 맞게 (만약 프로그래머가 프로그램을 해놓았다면) 반응한다.

시스템 이벤트가 발생하고 나면, 그 이벤트에 대해 반응이 수행되는 것이다.

우리는 이벤트에 반응하여 수행할 행위들의 목록을 작성한다. 이 목록은 코드 편집기Code Editor 창에서 작성한다. 마우스 버튼을 클릭할 때 프로그램이 반응하도록 만들려면 수행할 행위들을 이벤트핸들러event handler라고 하는 코드 블록 안에 적어 넣어야 한다.

코드 편집기Code Editor 창

버튼을 더블 클릭하여 코드 편집기 창을 처음 열면 창의 제목에는 파일 이름이 나타난다. 코드 편집기 창 안에 소스 코드 파일 여러 개를 열 수도 있다. 열린 파일들은 각자 자신의

소스 코드 파일 이름이 표시된 탭을 가지고 있다. 프로그램에 창이 3개 떠 있으면 유닛unit도 3개 있는 것이고, 이 3개 모두 소스 코드 편집기 창 안에 들어간다.

단순 이벤트라면 이벤트핸들러Event Handler 프로시저Procedure에게 Sender(이벤트를 발생시켜서 보내는 이)만 파라미터parameter, 매개변수로 전달한다. 복잡한 이벤트를 다루려면 이벤트핸들러 프로시저에게 Sender 이외에도 추가로 더 많은 파라미터들을 전달한다.

애플리케이션에 있는 콘트롤을 사용자가 마우스로 클릭하면, *OnClick*[온클릭] 이벤트가 발생된다. 마우스로 누르지 않고 키보드에서 콘트롤을 선택하고 엔터키를 눌러도 *OnClick* 이벤트가 발생될 수 있기 때문에 버튼 프레스Press 이벤트라고도 한다. *OnClick* 이벤트는 애플리케이션을 개발할 때 매우 자주 사용된다.

예제로, `frmMy` 폼 안에 올려놓은 `btnMy` 버튼에 프레스 이벤트핸들러를 만들어보자.

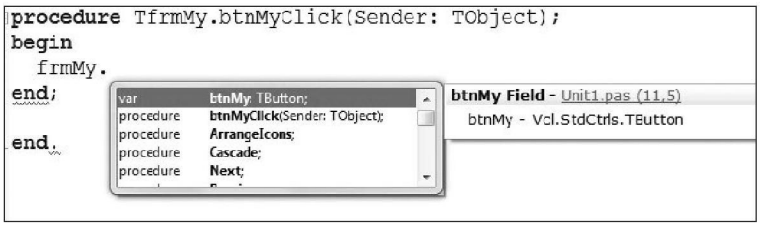
`btnMy` 버튼을 더블클릭 한다.

키보드에서 코드를 적는 일을 줄여주기 위해, 델파이가 마우스 버튼 프레스mouse button press 프로시저의 본문을 다음과 같이 자동으로 만들어 낸다.

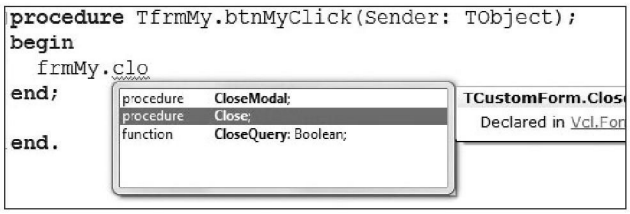
```
procedure TfrmMy.btnMyClick(Sender: TObject);
begin
end;
```

이때, 마우스 커서는 *begin* 과 *end* 키워드 사이에 있게 된다. 우리는 여기에 폼 윈도우를 닫는(창의 위쪽 오른편에 있는 X 자 종료 버튼과 완전히 같은 기능이다) 명령을 적는다. 지금 이 애플리케이션에는 이 창 하나밖에 없기 때문에, 이 창이 메인main 폼이다. 그러므로 창을 닫으면 프로그램 전체가 종료된다.

`frmMy` 라고 타이핑하고, 마침표('.')를 찍으면, 잠시 후 델파이에는 코드 완성 선택 목록이 나온다. 여기에는 `frmMy` 컴포넌트에서 사용될 수 있는 메소드method들과 프로퍼티property들이 모두 있다.



우리는 이 목록에서 알맞은 프로퍼티나 메소드를 선택한다. 나오는 항목이 꽤 많기 때문에, 보다 빠르게 찾으려면 이름의 일부 몇 자를 입력한다. 목록에 표시되는 요소들의 수가 현격하게 줄어든다.



엔터 키를 눌러서 코드 완성을 마친다.

이제 btnMy 명령버튼이 눌리면 바로 반응할 코드가 만들어졌다. 최종 코드는 다음과 같다.

```
procedure TfrmMy.btnMyClick(Sender: TObject);
begin
    frmMy.Close;
end;
```

이 프로그램을 실행시키고, btnMy 명령버튼을 누르면 프로그램이 종료된다. 위 예제에서, 우리는 *Close* 프로시저 즉, 메소드[method]를 사용해보았다. 메소드를 호출[call]하는 것은 그 컴포넌트가 무슨 일을 수행할 것인지를 말하는 것이다. 이 예제는 메소드를 호출하는 문장의 구조[syntax]를 보여준다. 컴포넌트 이름을 먼저 적고 마침표를 찍고 그 뒤에 메소드 이름을 적는다.

```
<컴포넌트 이름>.<메소드 이름>;
```

컴포넌트에는 자신들이 할 수 있는 행위 즉 메소드들이 있다. 서로 다른 컴포넌트이지만 동일한 메소드를 가지고 있는 경우도 가끔 있다.

이번에는 *Clear* 라는 메소드를 사용해보자. 텍스트박스에서 내용을 모두 지우는데 사용된다.

```
Edit1.Clear;
```

위에서도 말했지만, 컴포넌트에는 메소드method들과 프로퍼티property들이 있다.

프로퍼티란 컴포넌트가 가지는 특성들이다. 프로퍼티를 변경하거나 지정하려면 컴포넌트의 이름을 부르고, 그 다음 마침표를 찍고, 그 뒤에 프로퍼티 이름을 명시한다(델파이를 사용하면 코드 완성 목록 중에서 하나를 고르면 된다). 그리고 나서 할당 assignment 연산자operator를 적고, 마지막으로 프로퍼티에 넣을 값value을 명시한다.

```
<컴포넌트 이름>.<프로퍼티>:=<값>;
```

컴포넌트의 프로퍼티 대부분은 오브젝트 인스펙터 창 목록에 들어 들어있다. 하지만 어떤 프로퍼티들은 오브젝트 인스펙터에 나오지 않기도 한다. 이것은 차차 배우기로 하자. 지금은 명령버튼을 누르면 폼 창의 배경을 하얀색으로 채우는 프로그램을 만들어 보자. 이 프로그램에서는 Color프로퍼티를 사용하기로 한다.

```
procedure TfrmMy.btnMyClick(Sender: TObject);
begin
   frmMy.Color:=clWhite;
end;
```

우리는 컴포넌트 프로퍼티에 값을 넣을 수도 있고, 읽을 수도 있다. 그러므로 한 컴포넌트에 있는 프로퍼티 값을 다른 컴포넌트의 프로퍼티에 적용할 수도 있다.

문장 구조

```
<컴포넌트 1 이름>.<프로퍼티 1>:=<컴포넌트 2 이름>.<프로퍼티 2>;
```

예문

```
lblMy.Caption:=edtMy.Text;
```

위 문장은 텍스트박스 edtMy에 있는 글을 레이블 lblMy의 캡션에 넣도록 지시한다.

수행할 행위가 여러 개일 경우 세미콜론(;) 으로 구분한다.

예문

```
procedure TfrmMy.btnMyClick(Sender: TObject);
begin
    edit1.Clear;
    edit1.Color:=clBlue;
end;
```

이 버튼이 눌러지면 행위 두 개가 적힌 순서대로 수행된다. 텍스트박스가 비워지고, 배경색이 파랑으로 채워진다.

실습

Exercise 1.

폼을 하나 만들고 레이블 하나와 버튼 두 개를 올려둔다. 첫 번째 버튼으로는 레이블을 사용할 수 있게(enable) 하고, 두 번째 버튼은 사용하지 못하게(disable) 한다.

Exercise 2.

폼을 하나 만들고 레이블 하나, 텍스트박스 하나, 버튼 하나를 올려둔다. 버튼을 누르면 텍스트박스의 글이 레이블로 옮겨지고 텍스트박스는 비워지도록 한다.

Exercise 3.

"교통 신호등": 폼을 하나 만들고, 레이블 세 개, 버튼 세 개를 올려둔다. 각 버튼은 알맞은 레이블 하나만 사용할 수 있게(enable) 하고, 나머지 레이블들은 사용하지 못하게(disable) 한다. *Enabled* 프로퍼티를 사용하면 된다.

Exercise 4.

폼을 하나 만들고 레이블 하나를 넣고, 버튼 3개씩 4 모둠(총 12개의 버튼)을 올려둔다. 첫 번째 모둠(3개의 버튼)은 레이블의 배경색background color을 변경한다. 두 번째 모둠(3개의 버튼)은 레이블의 글꼴 색font color을, 세 번째 모둠(3개의 버튼)은 레이블의 글꼴 크기font size를, 마지막 네 번째 모둠(3개의 버튼)은 레이블의 글꼴 font family을 변경한다.