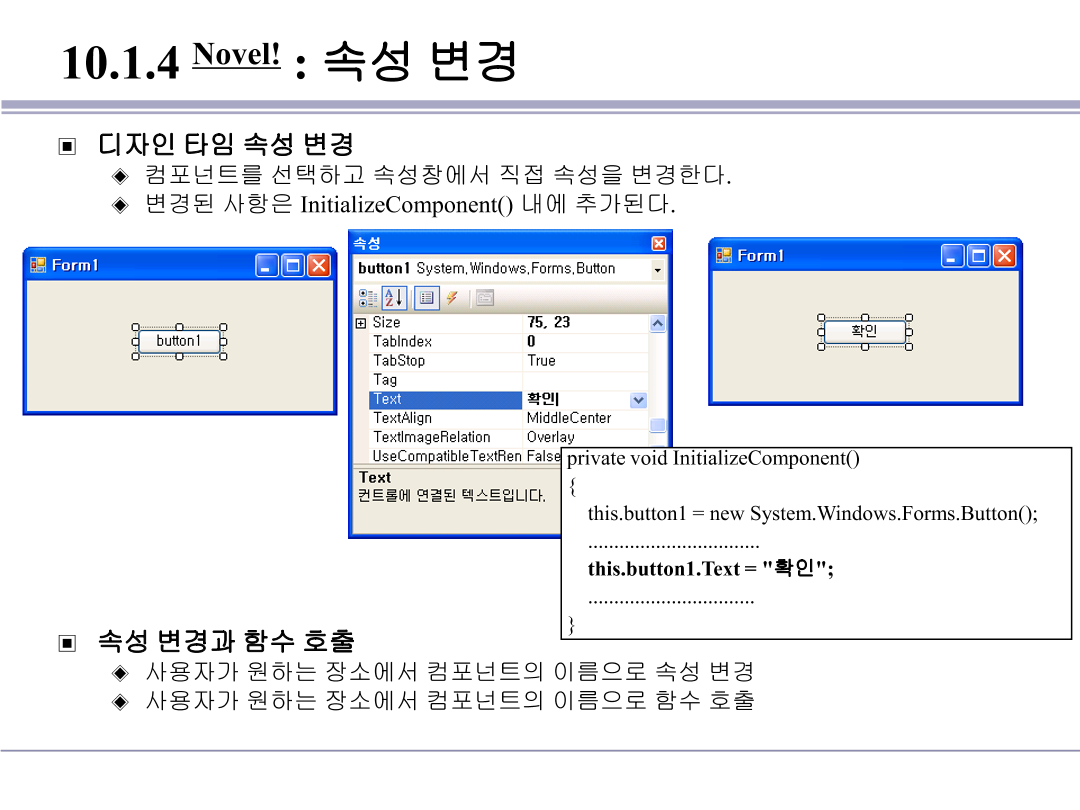
**10.1.2 윈도우 프로그래밍의 절차**  
  
일반적인 윈도우 프로그램의 절차에 대해서 알아보자. 대부분 다음과 같은 절차에 의해서 프로그램을 하게 된다.  
  
**▒ 윈도우 프로그래밍의 절차**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4 | 1. 컴포넌트 생성  2. 속성 변경  3. 이벤트 추가  4. 이벤트 구현 |

제일 먼저 우리는 폼에 컴포넌트를 배치한다. 도구상자를 이용해서 컴포넌트를 배치하게 되면 XXX.Designer.cs 파일 내에 객체 생성구문이 자동으로 추가된다.  
  
**□ 컴포넌트 생성**  
◇ 도구상자를 이용해서 객체 생성  
◇ XXX.Designer.cs 파일 내에 객체 생성구문이 자동으로 추가된다.  
  
컴포넌트 배치가 완료되었다면 그 다음으로 각각의 컴포넌트에 대한 속성을 변경하게 된다. 컴포넌트의 속성은 컴포넌트를 선택하고 속성창에서 속성을 변경하면 된다. 이렇게 변경된 사항은 하나도 빠짐없이 InitializeComponent() 내에 추가된다.  
  
**□ 속성 변경**  
◇ 컴포넌트를 선택하고 속성창에서 직접 속성을 변경한다.  
◇ 변경된 사항은 InitializeComponent() 내에 추가된다.  
  
속성 변경이 완료되었다면 이벤트를 추가하게 된다. 이벤트는 속성창에서 이벤트(번개모양)를 선택하고 원하는 이벤트를 생성하면 된다. 이벤트를 생성하는 방법은 이벤트를 선택한 후 이벤트 처리 루틴의 이름을 직접 입력하면 된다. 또 다른 방법으로 이벤트를 더블 클릭하면 자동으로 이벤트 처리 루틴의 이름이 생성된다. 그리고 이벤트 처리 루틴의 구현부로 이동하게 된다.   
  
**□ 이벤트 처리**  
◇ 이벤트 처리 루틴 생성은 속성창에서 이벤트를 선택한 후 자동으로 생성할 수 있다.  
  
마지막으로 이벤트 처리 루틴에 사용자가 직접 원하는 코딩을 작성하면 된다.

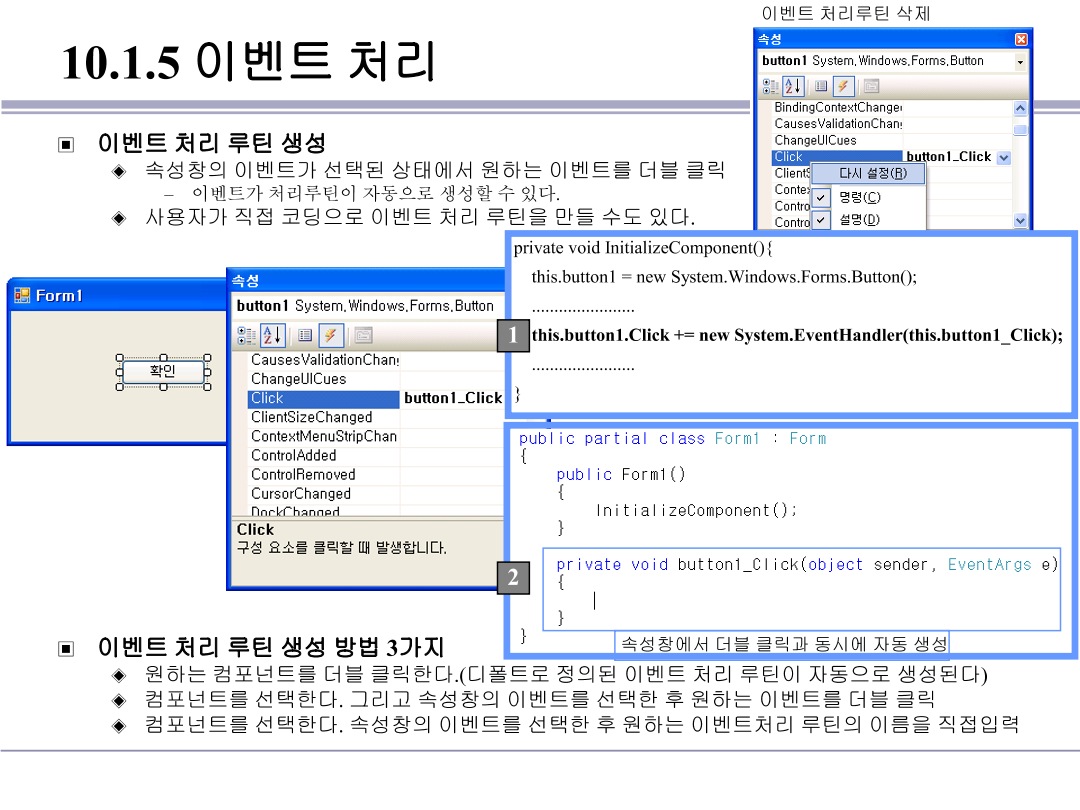
**10.1.3 컴포넌트를 생성**  
  
C#의 컴포넌트는 대부분 도구상자에 들어 있다. 도구상자에 있는 컴포넌트를 폼에 생성하는 방법은 다음과 같다.  
  
**□ 컴포넌트 생성 방법 3가지**  
◇ 도구상자에서 더블 클릭  
◇ 도구상자에서 선택한 후 폼으로 끌어넣기  
◇ 도구상자에서 선택 후 폼에서 원하는 크기로 드래그하기  
  
보통의 경우 프로그램을 실행하면 눈에 보이는 컴포넌트만 도구상자에 있기 마련이다. 하지만 C#에서는 보이지 않는 컴포넌트도 도구상자에 들어 있다. 보이는 컴포넌트는 도구 상자에서 선택한 후 폼에 배치하면 된다. 같은 방법으로 보이지 않는 컴포넌트를 배치하면 디자인 폼의 아래쪽에 생성된 순서대로 만들어진다.  
  
**□ C#의 컴포넌트**   
◇ 보이는 컴포넌트 : 도구 상자에서 선택한 후 폼에 배치된다.  
◇ 보이지 않는 컴포넌트도 : 디자인 폼의 아래쪽에 생성된 순서대로 배치된다.  
  
도구 상자를 이용해서 작업한 내용은 XXX.Designer.cs 파일 내에 자동으로 생성된다.



**10.1.4 Novel! : 속성 변경**  
  
컴포넌트 배치가 완료되었다면 컴포넌트를 선택하고 속성창에서 속성을 변경하게 된다. 변경된 사항은 하나도 빠짐없이 InitializeComponent() 내에 추가된다.  
  
**□ 속성 변경**  
◇ 컴포넌트를 선택하고 속성창에서 직접 속성을 변경한다.  
◇ 변경된 사항은 InitializeComponent() 내에 추가된다.  
  
앞에서 우리는 폼에 보이는 컴포넌트와 보이지 않는 컴포넌트가 있다고 배웠다. 속성과 관련 지어서 이것에 대해서 알아보자.   
  
사실 눈에 보이지 않는 컴포넌트의 경우에는 사용자가 직접 new를 이용해서 코드 형식으로 작성하면 된다. 예를 들어 PrintDialog의 경우 다음과 같이 사용자가 직접 만들어 사용해도 된다.  
  
**□ PrintDialog 객체 생성하는 방법**  
◇ 1. 코드로 작성하기  
― PrintDialog printDialog1 = new PrintDialog();  
◇ 2. 도구상자의 [인쇄]영역에서 PrintDialog 끌어다 놓기  
― printDialog1의 이름으로 폼 디자인 영역 아래쪽에 생성된다.  
  
그렇다면 다음과 같은 질문이 나올 수 있다. 두개의 차이점이 뭡니까? PrintDialog를 만들어서 사용한다는 측면에서는 똑같다. 그렇다면 왜 도구상자를 이용합니까? 이와 같은 질문으로 반문할 것이다. 이것의 해답은 다음과 같다.  
  
**□ PrintDialog를 코드로 생성해도 되는데 도구상자에서 생성하는 이유**  
◇ 도구상자에서 생성하면 PrintDialog의 속성을 속성창에서 수정할 수 있다.  
  
이것은 C#의 마인드와 관련이 있다. C#에서는 툴에서 속성을 변경할 수 있기 때문에, 그리고 보면서 훨씬 편하게 속성을 편집할 수 있기 때문에 보이든 보이지 않든 간에 대부분의 컴포넌트를 도구상자에 넣어 둔 것이다.  
  
**☞ 속성창의 속성들**

|  |
| --- |
| 속성창에 보면 많은 속성이 있다. 컴포넌트를 선택하게 되면 컴포넌트 내에 사용할 수 있는 모든 속성 목록이 속성창에 나온다. 그렇기 때문에 속성창에서 어떤 속성들이 있는지 확인하는 버릇을 들여야 한다. 이 많은 속성을 한꺼번에 익힐 수는 없다. 프로그램을 해나가면서 익숙해지는 방법밖에는 별다른 방법이 없다. 이것은 이벤트도 마찬가지이다. |

도구상자를 이용해서 폼에 컴포넌트를 생성한 내용과 속성창에서 변경한 모든 내용은 XXX.Designer.cs 파일 내에 들어간다.   
  
**□ 컴포넌트 생성과 속성 변경 작업의 내역**  
◇ 폼에 컴포넌트를 생성한 내용과 속성창에서 변경한 모든 내용은 XXX.Designer.cs 파일 내에 들어간다.  
  
속성에 관련된 내용은 XXX.Designer.cs 파일의 InitializeComponent() 함수 내에 하나도 빠짐없이 들어가게 된다.

****

**10.1.5 이벤트 처리**  
  
이벤트 처리루틴은 사용자가 직접 코딩으로 만들 수 있다. 그리고 Visual Studio 툴에서는 속성창에서 이벤트를 선택한 후 원하는 이벤트를 더블 클릭하면 이벤트 처리 루틴이 자동으로 생성된다.  
  
**□ 이벤트 처리 루틴 생성**  
◇ 속성창의 이벤트가 선택된 상태에서 원하는 이벤트를 더블 클릭  
― 이벤트가 처리루틴이 자동으로 생성할 수 있다.  
◇ 사용자가 직접 코딩으로 이벤트 처리 루틴을 만들 수도 있다.  
  
이벤트에서 중요한 것은 이벤트 처리 루틴을 어떻게 만드는지가 아니다. 정말 중요한 것은 어떤 이벤트가 언제 발생하느냐이다. 적절한 곳에 적절한 코딩을 넣는 것이 이벤트에서 가장 중요한 것이다.  
  
**□ 이벤트에서 제일 중요한 것**  
◇ 어떤 이벤트가 언제 발생하느냐를 알아야 한다.  
  
사실 이벤트가 언제 처리되는지를 몰라서 사용을 못하는 경우가 더 많다. 항상 이벤트를 보면 "이 이벤트는 언제 발생하지, 언제 사용하지"와 같은 질문으로 시작해야 한다.  
  
이벤트 처리 루틴을 생성하는 방법은 다음과 같다.  
  
**□ 이벤트 처리 루틴 생성 방법 3가지**  
◇ 원하는 컴포넌트를 더블 클릭한다.(디폴트로 정의된 이벤트 처리 루틴이 자동으로 생성된다)  
◇ 컴포넌트를 선택한다. 속성창의 이벤트를 선택한 후 원하는 이벤트를 더블 클릭(자동생성)  
◇ 컴포넌트를 선택한다. 속성창의 이벤트를 선택한 후 원하는 이벤트처리 루틴의 이름을 직접입력  
  
**☞ 참고**

|  |
| --- |
| 프로그램을 작성하다 모르는 이벤트가 나오면 반드시 찾아보라! 한꺼번에 확인할 수는 없지만 하나씩이라도 반드시 찾아보라. |

**10.1.6 메시지 박스 띄우기**  
  
윈도우 프로그래밍에서 메시지 박스를 띄우는 것을 모르는 것은 콘솔 프로그래밍에서 Console.WriteLine()을 모르는 것이나 마찬가지이다. 다음은 가장 간단하게 메시지 박스를 띄우는 방법이다.  
  
**▒ 가장 간단한 메시지 박스**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1 | MessageBox.Show("Hello World!"); |

타이틀이 없는 단순한 형태의 메시지 박스가 뜰 것이다. 만약 메시지 박스에 타이틀을 넣고자 한다면 다음과 같은 형태로 사용하면 된다.  
  
**▒ 타이틀이 포함된 메시지 박스**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

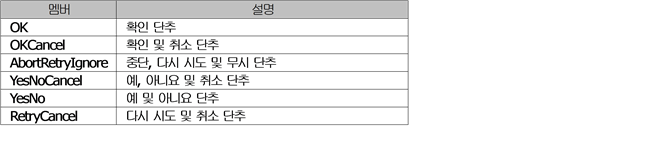
|  |  |
| --- | --- |
| 1 | MessageBox.Show("Hello World?", "Hello Tilte"); |

현재에는 버튼이 하나밖에 없다. 이 버튼 또한 사용자가 선택적으로 지정할 수 있다.   
  
**▒ 타이틀과 예(Yes) 아니오(No) 버튼이 포함된 메시지 박스**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

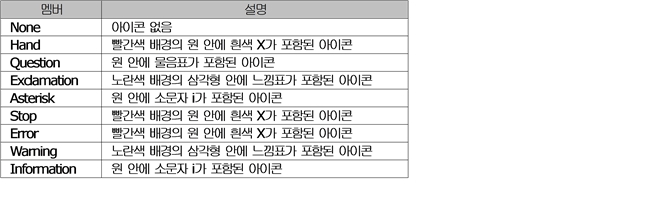
|  |  |
| --- | --- |
| 1 | MessageBox.Show("Hello World?", "Hello Tilte", MessageBoxButtons.YesNo); |

메시지 박스에 버튼의 형태를 위해서 MessageBoxButtons 열거형을 사용하면 된다.

  
[표 10\_1] MessageBoxButtons 열거형 멤버  
  
현재 위의 메시지 박스는 아이콘이 없는 상태이다. 다음과 같은 형태로 사용하면 아이콘을 메시지 박스에 표시할 수 있다.  
  
**▒ 아이콘이 포함된 메시지 박스**

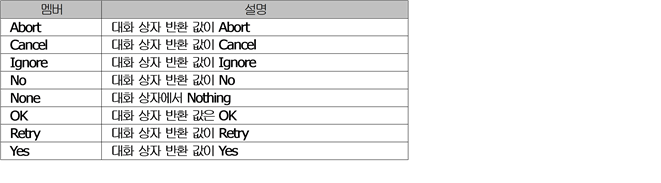
[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4  5 | DialogResult result = MessageBox.Show("Hello World?", "Hello Tilte",              MessageBoxButtons.YesNo, //메시지 박스의 버튼              MessageBoxIcon.Question, //메시지 박스의 아이콘 모양              MessageBoxDefaultButton.Button1);              //메시지 박스의 디폴트 선택 버튼, Button1, Button2, Button3가 존재한다. |

메시지 박스에 아이콘을 표시하기 위해서는 MessageBoxIcon 열거형을 사용하면 된다. 위에서는 MessageBoxIcon.Question를 사용했기 때문에 물음표 아이콘이 표시된다.   
  
  
[표 10\_2] MessageBoxIcon 열거형  
  
메시지 박스에는 버튼이 여러 개 있을 수 있다. 이 버튼 중 어떤 버튼이 눌러졌는지 확인하기 위해서 MessageBox.Show()의 리턴값을 확인하면 된다.   
  
**▒ 메시지 박스의 리턴값 얻어내기**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

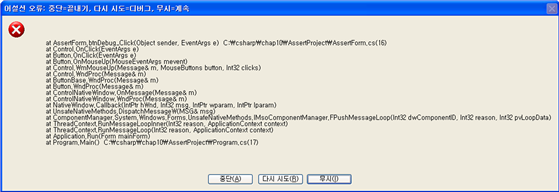
|  |  |
| --- | --- |
| 1  2  3  4 | DialogResult result = MessageBox.Show(............. );  if(result == DialogResult.Yes) {     //  } |

이 때 리턴값은 DialogResult 열거형의 형태로 리턴된다. 다음은 DialogResult 열거형의 멤버를 보여주고 있다.  
  
  
[표 10\_3] DialogResult 열거형  
  
각각의 버튼에 따라서 각각의 DialogResult 멤버가 존재하는 것을 알 수 있다.

**§ chap10\MessageBoxHelloWorld\Form1.cs**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | sing System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Text;  using System.Windows.Forms;    namespace MessageBoxHelloWorld{      public partial class Form1 : Form{          public Form1(){              InitializeComponent();          }          private void button1\_Click(object sender, EventArgs e){              MessageBox.Show("Hello World!");              MessageBox.Show("Hello World?", "Hello Tilte");              MessageBox.Show("Hello World?", "Hello Tilte", MessageBoxButtons.YesNo);                DialogResult result = MessageBox.Show("Hello World?", "Hello Tilte",                                      MessageBoxButtons.YesNo, MessageBoxIcon.Question,                                      MessageBoxDefaultButton.Button1);              if(result == DialogResult.Yes) {                      MessageBox.Show("Yes Click");              }          }      }  } |

**10.1.7 Assertion**  
  
프로그래밍에서 특정 조건을 만족하지 않으면 프로그램 자체를 멈추고 관련된 정보를 보여주는 기능이 있다. 이 기능을 Assertion이라고 한다. Assertion은 크게 Debug.Assert()와 Trace.Assert()로 나눌 수 있다.   
  
**□ Assertion이란**  
◇ 특정 조건을 만족하지 않으면 프로그램 자체를 멈추고 관련된 정보를 보여주는 기능  
  
**□ Assertion의 종류**  
◇ Debug.Assert()  
― 디버그 모드에 적용된다.  
◇ Trace.Assert()  
― 디버그 모드와 릴리즈 모드에 적용된다.  
  
Debug.Assert()는 디버그 모드에 적용되며, Trace.Assert()는 디버그 모드와 릴리즈 모드에 적용된다.  
  
Assertion은 다음과 같이 함수의 매개변수로 주어진 데이터가 true이면 다음 작업으로 넘어가지만 false이면 현재 실행상태의 스택 정보나 매개변수로 주어진 메시지를 출력하게 된다.   
  
**□ Debug의 Assertion**  
◇ Debug.Assert(false);  
― false이면 프로그램을 종료시키면서 현재 실행 상태의 스택 정보를 출력한다.  
◇ Debug.Assert(false, "에러에 대한 메시지 ");  
― false이면 프로그램을 종료시키면서 주어진 메시지를 출력한다.  
  
Debug.Assert(false)는 프로그램을 종료시키면서 다음과 같이 현재 실행 상태의 스택 정보를 보여준다.  
  
  
[그림 10\_1] Debug.Assert(false)의 결과화면  
Debug.Assert()는 디버그 모드에서 적용되며 릴리즈(Release) 모드에서는 무의미한 존재가 되고 만다. Trace.Assert()의 경우에는 디버그와 릴리즈 모두 적용 가능하다. 오류의 내용을 사용자에게 보여줄 필요가 없기 때문에 Debug.Assert() 함수 사용을 권장한다.  
  
디버그 모드에서 디버그 결과창에 정보를 출력하는 용도로 다음의 함수를 사용할 수 있다.  
  
**▒ 디버그창에 문자열 출력**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2 | Debug.WriteLine("Hello Debug!!");  Trace.WriteLine("Hello Trace!!"); |

위의 구문은 둘 다 디버그 모드에서만 적용되는 구문이다.

**§ chap10\AssertProject\AssertForm.cs**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Text;  using System.Windows.Forms;  using System.Diagnostics;    namespace KeyEventFrame{      public partial class AssertForm : Form{          public AssertForm(){              InitializeComponent();          }          private void btnDebug\_Click(object sender, EventArgs e){              Debug.Assert(false);          }          private void btnTrace\_Click(object sender, EventArgs e){              Trace.Assert(false);          }          private void btnInfoWrite\_Click(object sender, EventArgs e){              Debug.WriteLine("Hello Debug!!");              Trace.WriteLine("Hello Trace!!");          }      }  } |

**10.1.8 Application 정보**  
  
우리가 만든 하나의 애플리케이션에서 애플리케이션 그 자체를 대표하는 Application 클래스가 있다. 사용빈도가 높기 때문에 알아두면 아주 유용한 클래스이다. 이 클래스의 기능은 다음과 같다.  
  
**▒ 응용 프로그램 시작 및 중지**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2 | 1. Application.Exit()  2. Application.Run() |

**▒ 응용 프로그램에 대한 정보에 관련된 속성**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4 | 1. Application.StartupPath  2. Application.ProductName  3. Application.ExecutablePath  4. Application.ProductVersion |

**▒ Windows 메시지를 처리하기 위한 함수**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2 | 1. DoEvents()  2. AddMessageFilter() |

우리는 윈도우 애플리케이션을 실행하기 위해서 다음과 같은 구문을 사용하였다. 이 구문은 윈폼 애플리케이션에서 자동으로 생성되는 구문이다.  
  
**▒ 윈도우 프로그램의 Main() 함수**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4  5 | static void Main() {     Application.EnableVisualStyles();     Application.SetCompatibleTextRenderingDefault(false);     Application.Run(new AppInfoForm());  } |

이 구문에서 Application.Run() 함수는 AppInfoForm을 이용해서 윈도우 애플리케이션을 실행하는 역할을 하고 있다. 반대로 Application.Exit()는 애플리케이션 어느 곳에서나 사용만 하면 바로 애플리케이션이 종료된다.  
  
Application 클래스를 이용해서 현재 애플리케이션의 다양한 정보를 얻을 수 있다. 그 대표적인 예는 다음과 같다.  
  
**□ 응용 프로그램에 대한 정보를 가져오기**  
◇ Application.StartupPath : 응용 프로그램을 시작한 실행 파일의 디렉터리   
◇ Application.ProductName : 제품이름  
◇ Application.ExecutablePath : 실행파일의 경로  
◇ Application.ProductVersion : 버전정보  
  
애플리케이션이 종료될 때 뭔가 처리할 일이 있다면 Application의 ApplicationExit 이벤트를 이용하면 된다. ApplicationExit 이벤트는 애플리케이션이 종료하는 순간에 발생하는 이벤트이다.   
  
**▒ ApplicationExit 이벤트 등록과 구현**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4 | private void App\_OnExit(object sender, EventArgs evtArgs){     MessageBox.Show("프로그램을 종료합니다!","App Exit!!");  }  Application.ApplicationExit += new EventHandler(App\_OnExit); |

여러분이 종료버튼을 클릭하면 ApplicationExit 이벤트가 발생할 것이다.

**§ chap10\ApplicationProject\AppInfoForm.cs**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Text;  using System.Windows.Forms;    namespace ApplicationProject{      public partial class AppInfoForm : Form{          public AppInfoForm(){              InitializeComponent();              Application.ApplicationExit += new EventHandler(App\_OnExit);          }          private void App\_OnExit(object sender, EventArgs evtArgs){              MessageBox.Show("?꾨줈洹몃옩??醫낅즺?⑸땲??","App Exit!!");          }          private void btnAppInfo\_Click(object sender, EventArgs e){              StringBuilder sb = new StringBuilder();              sb.Append(Application.StartupPath + "\r\n");              sb.Append(Application.ProductName + "\r\n");              sb.Append(Application.ExecutablePath + "\r\n");              sb.Append(Application.ProductVersion + "\r\n");              tbAppInfo.Text = sb.ToString();          }      }  } |

**10.1.9 Application 레벨의 메시지 처리**  
  
모든 메시지는 애플리케이션 레벨에서 발생한다. 폼에서 메시지를 발생시켰다 하더라도 애플리케이션 레벨에서 먼저 처리된 후 하위 폼으로 전달되는 방식이다. 그렇기 때문에 애플리케이션 레벨에서 메시지를 처리하면 하위의 폼들에게까지 영향을 미칠 수 있다.  
  
**□ 메시지 처리 방식**  
◇ 폼에서 메시지를 발생시켰다 하더라도 애플리케이션 레벨에서 먼저 처리한 후 하위 폼으로 전달되는 방식이다.  
  
애플리케이션 레벨의 메시지 처리를 위해서 IMessageFilter라는 것을 Application에 등록해두면 된다. IMessageFilter가 메시지 필터 역할을 하는 것이다. IMessageFilter는 인터페이스이며 PreFilterMessage()라는 하나의 함수를 구현해야만 사용할 수 있다. 다음은 IMessageFilter를 구현하는 방법을 보여주고 있다.  
  
**▒ 메시지 필터링을 위한 IMessageFilter의 구현**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4  5 | public class MyMessageFilter : IMessageFilter {        public bool PreFilterMessage(ref Message m) {           //구현부        }  } |

IMessageFilter를 구현해서 Application에 등록하는 방법은 다음과 같다.  
  
**▒ Application에 IMessageFilter 구현 객체 등록**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1 | Application.AddMessageFilter(new MyMessageFilter()); |

위와 같이 하면 애플리케이션 레벨의 메시지가 처리되기도 전에 우리의 MyMessageFilter가 먼저 처리될 것이다.  
  
PreFilterMessage() 내부를 보면 메시지가 WM\_LBUTTONDOWN일 때 즉 0x201의 값을 가지면 메시지 박스를 띄우는 프로그램이다. 여기서 우리가 조심해야 되는 것은 바로 return값이다.  
  
**▒ PreFilterMessage 함수의 구현**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

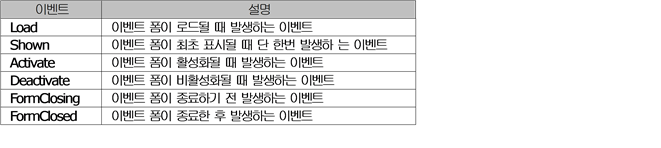
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | public bool PreFilterMessage(ref Message m) {        if (m.Msg == WM\_LBUTTONDOWN) {           MessageBox.Show("애플리케이션에서왼쪽마우스버튼");           return true;//처리했음        }        return false;   //처리하지 않았음  } |

PreFilterMessage의 반환값으로 true 또는 false를 반환해야 하는데 이것이 아주 중요하다. 반환값의 의미는 다음과 같다.  
  
**□ PreFilterMessage() 반환값**  
◇ return true;  
― 현재 발생한 메시지 처리를 완료했으니 이후 이 메시지는 처리하지 않는다.  
◇ return false;  
― 현재 발생한 메시지 처리를 했지만 이후에 이 메시지를 다시 처리한다.  
  
PreFilterMessage()에서 발생한 메시지를 처리하고 true를 반환한다면, 현재 처리를 다했으니 이후에 이 메시지는 무시해도 좋다는 의미가 된다. false라면 처리했더라도 이후에 이 메시지를 다시 처리해야 한다는 의미가 담겨 있다.  
  
예를 들어보자. PreFilterMessage()에서도 왼쪽 마우스 버튼 클릭을 처리하고, 폼에서도 왼쪽 마우스 클릭을 처리했다고 가정하자. 이렇게 되면 애플리케이션과 폼 두 곳에서 왼쪽 마우스 클릭을 처리한 것이 된다. 하지만 PreFilterMessage()의 반환값에 따라서 폼의 메시지가 처리되기도 하고 안 되기도 한다.  
  
**□ 두 곳에서 메시지 처리 작업**  
◇ 애플리케이션의 PreFilterMessage()에서 왼쪽 마우스 클릭 처리  
◇ 폼의 왼쪽 마우스 클릭 처리  
◇ PreFilterMessage()의 반환값에 따라서 폼의 메시지 처리가 달라진다.  
  
PreFilterMessage()에서 왼쪽 마우스 클릭을 처리한 후 true를 반환하면 폼의 왼쪽 마우스 클릭은 처리 되지 않는다. false를 반환하면 폼의 왼쪽 마우스 클릭이 처리된다. true와 false를 변경해가면서 테스트해보기 바란다.

**§ chap10\ApplicationProject2\Form1.cs**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

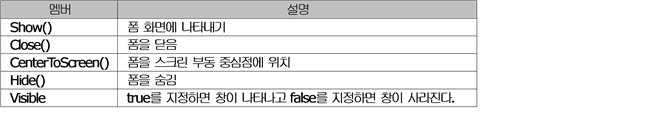
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Text;  using System.Windows.Forms;    namespace ApplicationProject2{      public partial class Form1 : Form{          public Form1(){              InitializeComponent();              Application.AddMessageFilter(new MyMessageFilter());          }          private void Form1\_MouseDown(object sender, MouseEventArgs e){              if (e.Button == MouseButtons.Left)                  MessageBox.Show("이벤트에서 마우스 왼쪽 버튼");          }      }        public class MyMessageFilter : IMessageFilter{          private const int WM\_LBUTTONDOWN = 0x201;          public bool PreFilterMessage(ref Message m){              if (m.Msg == WM\_LBUTTONDOWN){                  MessageBox.Show("애플리케이션에서 왼쪽 마우스 버튼");                  return true;//처리했음              }              return false;   //처리하지 않았음          }      }  } |

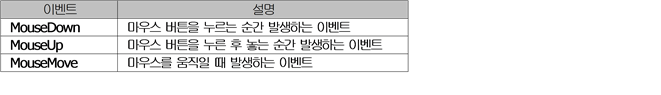
**10.2.1 Form Event**  
  
폼은 윈도우 프로그래밍의 기본이기 때문에 폼에서 발생하는 이벤트를 이해하는 것은 필수적이다. 특히 폼의 이벤트가 어느 시점에 발생하는가는 반드시 알아두어야 한다. 폼 이벤트는 크게 6가지 정도로 나누어 생각해 볼 수 있다.   
  
  
[표 10\_4] 폼에서 발생하는 이벤트  
  
위의 폼에서 발생하는 이벤트를 코드로 만들어보자. 6개의 이벤트를 등록한 후 각각의 이벤트를 테스트하는 예는 다음과 같다.

**§ chap10\FormEventProject\EventForm.cs**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Text;  using System.Windows.Forms;    namespace FormEventProject{      public partial class EventForm : Form{          public EventForm(){              InitializeComponent();          }          private void EventForm\_Load(object sender, EventArgs e){              MessageBox.Show("Load....!!", "濡쒕뵫");          }          private void EventForm\_FormClosed(object sender, FormClosedEventArgs e){              MessageBox.Show("Closed....!!", "醫낅즺");          }          private void EventForm\_FormClosing(object sender, FormClosingEventArgs e){              if (MessageBox.Show("醫낅즺?섏떆寃좎뒿?덇퉴?", "醫낅즺", MessageBoxButtons.YesNo)                  == DialogResult.No)                  e.Cancel = true;//醫낅즺 痍⑥냼          }          private void EventForm\_Activated(object sender, EventArgs e){              this.BackColor = Color.Blue;          }          private void EventForm\_Deactivate(object sender, EventArgs e){              this.BackColor = SystemColors.ControlLight;          }          private void EventForm\_Shown(object sender, EventArgs e){              MessageBox.Show("Shown ?대깽?몃뒗 ?쇱씠 泥섏쓬 ?쒖떆???????쒕쾲 諛쒖깮?⑸땲??");          }      }  } |

폼의 이벤트는 언제 어떤 이벤트가 발생하는지만 기억하면 된다. 그리고 폼에 관련된 주요 함수는 다음과 같다.  
  
  
[표 10\_5] 폼에 관련된 함수 및 속성  
  
폼에 관련된 많은 함수와 이벤트들이 존재하지만 폼과 관련된 기본적인 사항들만 알아보았다.

**10.2.2 Mouse Event**  
  
마우스 이벤트는 컨트롤에서 마우스 Down, Up, Move할 때 발생하는 이벤트를 말한다.  
  
  
[표 10\_6] 마우스 이벤트의 종류  
  
마우스 이벤트가 필요한 곳에 이벤트 핸들러를 구현해서 사용하면 된다. 마우스 이벤트 핸들러의 원형은 다음과 같다.  
  
**▒ 마우스 이벤트 핸들러**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1 | public delegate void MouseEventHandler(object sender, MouseEventArgs e) |

마우스 이벤트의 핸들러에는 2개의 매개변수를 가진다. object sender는 마우스 이벤트를 발생시킨 컨트롤의 참조값이며, MouseEventArgs e는 마우스 이벤트에 관련된 정보가 들어있다. 이 매개변수를 이용하면 마우스의 좌표와 눌러진 버튼의 종류를 알 수 있다.  
  
**□ MouseEventHandler의 매개변수**  
◇ object sender : 마우스 이벤트를 발생시킨 컨트롤의 참조값  
◇ MouseEventArgs e : 마우스 이벤트에 관련된 정보(마우스의 좌표, 눌러진 버튼의 종류)  
  
MouseEventHandler를 구현할 때 가장 중요한 것은 어떤 마우스 버튼이 눌러졌는지와 어떤 키가 눌러졌는지와 마우스의 좌표이다. 이벤트가 발생했을 때 MouseEventArgs로부터 좌표를 구하는 방법은 다음과 같다.   
  
**▒ MouseEventArgs로부터 마우스 좌표를 구해 타이틀에 출력하기**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1 | this.Text = string.Format("Move {0}, {1}", e.X, e.Y); |

눌러진 마우스의 버튼은 MouseEventArgs의 Button 속성으로 알 수 있으나 함께 눌러진 Control, Shift, Alt 키는 확인할 수 없다. 이 키를 확인하기 위해서는 Control.ModifierKeys를 이용해야 한다. 다음은 마우스 키와 눌러진 키를 알아내는 방법이다.  
  
**▒ 마우스 키와 눌러진 키보드 알아내기**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | if (e.Button == MouseButtons.Left && Control.ModifierKeys == Keys.Control){         //...Control + 마우스 왼쪽   }else if (e.Button == MouseButtons.Middle && Control.ModifierKeys == Keys.Alt){         //...Alt + 마우스 중간   }else if (e.Button == MouseButtons.Right && Control.ModifierKeys == (Keys.Control | Keys.Alt)){         //...Control + Alt + 마우스 오른쪽  } |

마우스가 움직일 때 좌표를 폼의 타이틀에 출력하고, 마우스 버튼과 눌러진 키를 확인하는 전체 예는 다음과 같다.

**§ chap10\MouseProject\MouseForm.cs**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | 癤퓎sing System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Text;  using System.Windows.Forms;    namespace MouseProject{      public partial class MouseForm : Form{          public MouseForm(){              InitializeComponent();          }          private void MouseForm\_MouseDown(object sender, MouseEventArgs e){              if (e.Button == MouseButtons.Left && Control.ModifierKeys == Keys.Control)                  MessageBox.Show("Ctrl + ?쇱そ留덉슦??踰꾪듉", "Mouse ?뚯뒪??);              else if (e.Button == MouseButtons.Middle && Control.ModifierKeys == Keys.Alt)                  MessageBox.Show("Alt + 以묎컙留덉슦??踰꾪듉", "Mouse ?뚯뒪??);              else if (e.Button == MouseButtons.Right &&                                          Control.ModifierKeys == (Keys.Control | Keys.Alt))                  MessageBox.Show("Ctrl + Alt + ?ㅻⅨ履쎈쭏?곗뒪 踰꾪듉", "Mouse ?뚯뒪??);          }          private void MouseForm\_MouseUp(object sender, MouseEventArgs e){              //...          }          private void MouseForm\_MouseMove(object sender, MouseEventArgs e){              this.Text = string.Format("Move {0}, {1}", e.X, e.Y);          }      }  } |

**☞ 이벤트 핸들러의 첫번째 매개변수**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 거의 모든 이벤트 핸들러의 첫번째 매개변수는 이벤트를 발생시킨 놈의 참조값이다. 예를 들어 버튼에서 마우스 클릭을 했을 때 MouseDown이라는 이벤트가 제일 먼저 발생한다. 이 때 MouseDown 이벤트 핸들러의 첫번째 매개변수가 바로 버튼이 되는 것이다. 정확히 말해 버튼의 참조값이 되는 것이다. 즉 이벤트가 발생한 장소가 버튼이기 때문이다. 그래서 매개변수의 이름도 sender이다. 즉 이 이벤트를 보낸 놈이라는 뜻이다.  **10.2.3 Key Event**  키 이벤트는 키가 눌러졌을 때 발생하는 이벤트이다. 키 이벤트는 발생하는 순서에 따라 다음과 같이 4가지로 나눌 수 있다.  http://www.jabook.com/jabook2/images/upload/book/book_73/bookmark1-12.png [표 10\_7] 키 이벤트의 종류  PreviewKeyDown은 키를 누르기 직전 발생하는 이벤트이며, KeyDown은 키를 누른 후 발생하는 이벤트이다. 그리고 KeyPress는 키를 놓은 후 발생하는 이벤트이며 KeyUp은 키를 업하는 순간 발생하는 이벤트이다. 4가지 이벤트는 키를 한번 누르고 놓을 때마다 발생하며, 키 이벤트 핸들러에서 이벤트에 대한 정보를 얻을 수 있다.  키 이벤트의 정보는 이벤트 핸들러의 매개변수를 통해 얻을 수 있으며, 대표적으로 어떠한 키가 눌러졌는지를 알아내는 것이 가장 주된 임무이다. KeyDown에서 눌러진 키를 확인하는 예를 알아보자. 다음은 텍스트 박스에서 KeyDown이 발생했을 때 호출되는 이벤트 핸들러이다.  **▒  KeyDown 이벤트 핸들러**  [?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)   |  |  | | --- | --- | | 1  2  3 | private void tbKeyBox\_KeyDown(object sender, KeyEventArgs e){     //.......  } |   위의 이벤트 핸들러에서 KeyEventArgs를 이용해서 눌러진 키를 확인할 수 있다. 확인하는 방법은 KeyEventArgs의 KeyCode를 이용하는 방법과 KeyData를 이용하는 방법이 있다.   **□ 눌러진 키를 확인하는 방법** ◇ KeyEventArgs의 KeyCode를 이용하는 방법 ◇ KeyEventArgs의 KeyData를 이용하는 방법  KeyCode의 경우에는 키보드에 눌러진 값 중 하나의 키 값을 갖는다. 이 값은 Keys 열거형 중 하나의 값을 가진다. KeyCode의 경우에는 Modifier 즉 Control, Alt, Shift와 같은 키를 확인할 수는 없다. KeyCode와 눌러진 Control, Alt, Shift를 OR 연산시킨 것이 바로 KeyData 값이다.  **□ KeyCode** ◇ 키보드에 눌러진 값 중 하나의 키보드 값을 갖는다.(Keys 열거형)  **□ KeyData** ◇ KeyCode와 Modifier(Control, Alt, Shift) 키의 OR 연산된 값  KeyCode를 이용해서 Modifier와 눌러진 키를 확인하는 방법은 다음과 같다.   **▒ KeyCode를 이용한 키 확인**  [?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)   |  |  | | --- | --- | | 1  2  3 | if (e.KeyCode == Keys.F && (e.Alt || e.Control || e.Shift)){     //....Ctrl + Shift + Alt + F  } |   KeyCode 내에는 Modifier의 값이 없기 때문에 KeyEventArgs의 Alt와 Control 그리고 Shift 값을 확인하고 있다. Alt, Control, Shift가 눌러졌다면 이 값들은 true가 될 것이다.   KeyData는 눌러진 키와 Modifier 값의 OR 연산된 값이기 때문에 다음과 같은 방법으로 눌러진 키를 확인할 수 있다.  **▒ KeyData를 이용한 키 확인**  [?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)   |  |  | | --- | --- | | 1  2  3 | if (e.KeyData == (Keys.G | Keys.Control | Keys.Alt | Keys.Shift)){        tbKeyBox.Text = "Ctrl + Shift + Alt + G";  } |   눌러진 키와 Modifier를 OR 연산시킨 뒤 KeyData와 비교해서 눌러진 키들을 확인하면 된다. 키 이벤트를 테스트하는 전체 예는 다음과 같다.  **§ chap10\KeyProject\KeyForm.cs**  [?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | 癤퓎sing System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Text;  using System.Windows.Forms;    namespace KeyProject{      public partial class KeyForm : Form{          public KeyForm(){              InitializeComponent();          }          private void tbKeyBox\_KeyDown(object sender, KeyEventArgs e){              //2. ?ㅻ? ?ㅼ슫?섎뒗 ?쒓컙              if (e.KeyCode == Keys.F && (e.Alt || e.Control || e.Shift)){                  tbKeyBox.Text = "Ctrl + Shift + Alt + F";              }else if (e.KeyData == (Keys.G | Keys.Control | Keys.Alt | Keys.Shift)){                  tbKeyBox.Text = "Ctrl + Shift + Alt + G";              }          }          private void tbKeyBox\_KeyPress(object sender, KeyPressEventArgs e){              //3. ?ㅻ? ?ㅼ슫?????낇븯???쒓컙          }          private void tbKeyBox\_KeyUp(object sender, KeyEventArgs e){              //4. ?ㅻ? ?낇븯???쒓컙          }          private void tbKeyBox\_PreviewKeyDown(object sender, PreviewKeyDownEventArgs e){              //1. ?ㅻ? ?ㅼ슫?섍린 ??          }      }  } |   **10.2.4 Novel! : override Event 함수**  모든 이벤트에 On만 붙이면 override Event 함수가 된다. 이벤트의 경우 이벤트 핸들러를 등록해서 사용하지만 On override Event 함수의 경우에는 override해서 사용한다.   예를 들어보자. Form에 MouseDown이라는 이벤트를 구현할 때 다음과 같은 이벤트 핸들러를 사용한다.  **▒ 폼의 MouseDown 이벤트를 위한 이벤트 핸들러**  [?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)   |  |  | | --- | --- | | 1  2  3 | private void MouseForm\_MouseDown(object sender, MouseEventArgs e) {     //....  } |   이벤트 핸들러 대신 다음과 같이 override 함수를 사용할 수도 있다.  **▒ override 함수를 이용한 메시지 처리**  [?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)   |  |  | | --- | --- | | 1  2  3 | protected override void OnMouseDown(MouseEventArgs e) {     //....  } |   위의 둘 다 MouseDown 이벤트가 발생할 때 반응한다. 하지만 이 둘은 미세한 차이가 있다. 이 차이점에 대해서 알아보자.  MouseDown 이벤트는 어딘가에서 발생해야 한다. 그 이벤트가 발생하는 곳이 OnMouseDown() 오버라이드 함수이다. MSDN에 보면 이 함수의 설명을 MouseDown 이벤트를 발생시키는 함수라고 되어 있다. 즉 OnMouseDown() 함수에서 MouseDown에 해당하는 이벤트 핸들러가 등록되어 있으면 MouseDown 이벤트를 발생시키는 역할을 하는 것이다.  **□ OnMouseDown() 오버라이드 함수** ◇ MouseDown 이벤트를 발생시키는 함수 ◇ MouseDown 이벤트 핸들러가 등록되어 있으면 MouseDown 이벤트 핸들러를 호출한다.  OnMouseDown()에서 이벤트를 발생시켜 주기 때문에 MouseDown 이벤트가 발생한다는 의미가 된다. 결국 OnMouseDown()이 먼저 호출되고 MouseDown 이벤트 핸들러가 호출되는 것이다.  만약 다음과 같이 OnMouseDown() 함수를 오버라이드하고, MouseDown 이벤트도 등록했다고 가정하자.  **▒ OnMouseDown() 함수와 MouseDown 이벤트의 구현**  [?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9 | public partial class OverrideEventForm : Form {     protected override void OnMouseDown(MouseEventArgs e) {        MessageBox.Show("OnMouseDown() 함수");     }     private void OverrideEventForm\_MouseDown(object sender, MouseEventArgs e) {        MessageBox.Show("handler: MouseDown 이벤트처리");     }  }  //Visual Studio에서 작성했기 때문에 MouseDown 핸들러 추가하는 부분은 디자인 페이지에 있음 |   위의 상황에서 폼에 마우스를 다운하면 이론상으로는 OnMouseDown()이 호출되고, OnMouseDown() 내에서 MouseDown의 이벤트 핸들러인 OverrideEventForm\_MouseDown() 함수를 호출하게 된다.   하지만 문제가 발생하였다. 아버지의 OnMouseDown()의 내부에서 MouseDown의 이벤트 핸들러를 호출하는 것이지 새로 오버라이드한 아들의 OnMouseDown()에서 MouseDown의 이벤트를 호출하는 것이 아니기 때문이다. 그렇기 때문에 위의 상황에서는 새로 만든 아들의 OnMouseDown() 함수만 호출되며, MouseDown 이벤트 핸들러는 호출되지 않는다.  이것은 문제가 있다. OnMouseDown()을 오버라이드하면서 이벤트를 발생시키는 메커니즘이 사라져 버렸기 때문이다. 새로 만든 OnMouseDown()에서 이벤트를 발생시키기 위해서는 다음과 같이 아들의 OnMouseDown() 함수 내에서 아버지의 OnMouseDown()을 호출해주면 된다.  **▒ OnMouseDown() 함수와 MouseDown 이벤트의 구현**  [?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9 | public partial class OverrideEventForm : Form {     protected override void OnMouseDown(MouseEventArgs e) {        MessageBox.Show("OnMouseDown() 함수");        base.OnMouseDown(e);     }     private void OverrideEventForm\_MouseDown(object sender, MouseEventArgs e) {        MessageBox.Show("handler: MouseDown 이벤트처리");     }  } |   위와 같이 아들의 OnMouseDown() 내에서 아버지의 OnMouseDown()을 호출하기 위해서 base.OnMouseDown(e)를 호출한다. 이렇게 하면 마우스를 다운하는 순간 아들의 OnMouseDown()이 호출되면서 base.OnMouseDown(e)이 호출되어 MouseDown 이벤트 핸들러가 호출되는 것이다.  **☞ 참고**   |  | | --- | | 여기서 base를 붙이지 않으면 아들의 OnMouseDown()이 되기 때문에 아버지를 의미하는 base 키워드를 붙여 둔 것이다. |   이 메커니즘은 너무나도 일반적인 방식이다. 거의 대부분의 이벤트는 이벤트를 발생시키는 On 오버라이드 이벤트 함수를 가지고 있다. 그리고 위의 방식대로 동작한다.   그렇다면 마우스가 다운된다면 어디에서 코딩을 해주는 것이 맞는가를 한번 생각해보아야 한다.   **□ 질문** ◇ 마우스 다운에 해당하는 작업을 OnMouseDown()에서 해주어야 할까? MouseDown 이벤트 핸들러에서 해주어야 할까?  분명한 것은 MouseDown에 해당하는 작업을 OnMouseDown()에서 해주어도 되고, MouseDown 이벤트 핸들러에서 해주어도 된다. 일반적인 경우 대부분 MouseDown 이벤트 핸들러를 처리하는 방식으로 사용한다.   **□ 답변** ◇ On 오버라이드 함수와 이벤트 핸들러 어느 곳이든 상관 없다. ◇ 일반적인 경우 이벤트 핸들러 방식을 더 많이 사용한다.  사실 OnMouseDown()이라는 함수가 존재하는 것도 모르는 사람이 많다. 여러분은 오버라이드 방식과 이벤트 핸들러 방식으로 이벤트 작업을 처리할 수 있다는 것을 꼭 알아두기 바란다.   오버라이드 방식이 꼭 필요할 경우가 있다. MouseDown 이벤트가 발생하기 직접 작업을 해주어야 한다고 가정하자. 하지만 MouseDown 직전에 발생하는 이벤트는 없다고 가정하자. 이럴 경우 OnMouseDown() 함수 내에서 다음과 같은 방식으로 사용하면 된다.  **§ chap10\OverrideEventProject\OverrideEventForm.cs**  [?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Text;  using System.Windows.Forms;    namespace OverrideEventProject{      public partial class OverrideEventForm : Form{          public OverrideEventForm(){              InitializeComponent();          }          protected override void OnMouseDown(MouseEventArgs e){              MessageBox.Show("override: base.OnMouseDown(e) 호출이전");              base.OnMouseDown(e);              MessageBox.Show("override: base.OnMouseDown(e) 호출이후");          }          private void OverrideEventForm\_MouseDown(object sender, MouseEventArgs e){              MessageBox.Show("handler: MouseDown 이벤트 처리");          }      }  } |   위의 OnMouseDown() 내에서 base.OnMouseDown(e) 이전과 이후에 호출되는 메시지 박스는 이벤트 핸들러가 처리되기 직전과 직후에 호출되는 효과가 있다.  **☞ 참고**   |  | | --- | | 일반적인 경우 이벤트와 이벤트를 발생시키는 On 오버라이드 이벤트 함수는 쌍으로 존재한다. | |

**10.2.5 Timer**  
  
Timer 컨트롤은 타이머에 등록된 핸들러를 일정한 주기마다 반복적으로 호출하는 컴포넌트이다. 사용방법은 Timer 객체를 생성한 후 Interval 속성을 설정하고, Tick 이벤트 핸들러를 추가하면 Timer의 기본적인 설정은 완료된다. 최종적으로 Timer를 구동하기 위해서 Start() 함수만 호출하면 된다.  
  
**▒ Timer 컨트롤 사용하기**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4 | Timer t = new Timer();  t.Interval = 1000; //1초  t.Tick += new EventHandler(Timer\_Tick);  t.Start(); |

**▒ Tick 이벤트 핸들러**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

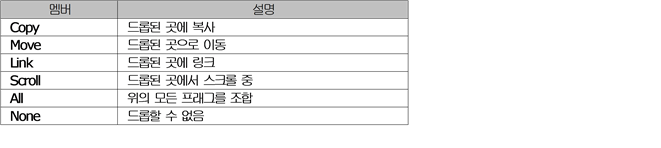
|  |  |
| --- | --- |
| 1  2  3 | private void Timer\_Tick(Object myObject, EventArgs myEventArgs) {     //...  } |

Interval은 Timer의 Tick 이벤트가 발생할 주기를 의미하며, Start()를 호출할 때 Tick 이벤트가 주기적으로 발생하여 Tick 이벤트 핸들러를 호출하게 된다.  
  
**□ 도구 상자를 이용한 타이머 생성**  
◇ 도구상자로부터 Timer 객체 생성  
◇ 속성창에서 Interval 속성과 Tick 이벤트를 추가할 수 있다.  
  
위의 예는 코드 상으로 Timer 객체를 생성했지만, 도구상자를 이용해서 Timer 객체를 생성할 수도 있다. 도구상자를 이용해서 객체를 생성하면 속성창을 이용해서 Interval 속성과 Tick 이벤트 핸들러를 등록할 수 있는 장점이 있다.

**§ chap10\TimerProject\TimerForm.cs**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Text;  using System.Windows.Forms;    namespace TimerProject{      public partial class TimerForm : Form{          private int count = 0;          public TimerForm(){              InitializeComponent();          }          private void TimerForm\_Load(object sender, EventArgs e){              Timer t = new Timer();              t.Interval = 1000;              t.Tick += new EventHandler(Timer\_Tick);              t.Start();          }          private void Timer\_Tick(Object myObject, EventArgs myEventArgs){              this.Text = ++this.count + "초입니다.";          }      }  } |

**10.2.6 Drag & Drop Event**  
  
기본적으로 Control로부터 상속된 모든 컨트롤은 Drag & Drop 기능을 지원한다. 우리는 특정 컨트롤을 드래그하기 시작한다. 그리고 원하는 컨트롤에 드롭을 시킨다. 그래서 Drag & Drop을 구현하기 위해서는 두개의 컨트롤이 필요하다. 편의상 드래그 되는 컨트롤을 A 컨트롤이라 하고, 드롭받는 컨트롤을 B 컨트롤이라고 하자. 여기서 A 컨트롤은 Label이며 B 컨트롤은 Panel이다.  
  
**□ 규칙**  
◇ 편의상 드래그 되는 컨트롤을 A 컨트롤이라 하고, 드롭받는 컨트롤을 B 컨트롤이라고 하자.  
  
우선 드롭을 받는 B 컨트롤 측에서 드롭을 허용하는 AllowDrop 속성이 true로 설정되어 있어야만 드롭을 받을 수가 있다.  
  
**□ AllowDrop 속성**  
◇ 드롭을 받기 위해서 드롭을 받는 B 컨트롤의 AllowDrop 속성을 true로 설정해야 한다.  
◇ 디폴트는 false로 설정되어 있다.  
  
**▣ 드래그의 시작 함수 DoDragDrop()**   
  
Drag & Drop을 구현할 때에는 항상 두 개의 컨트롤의 입장에서 생각해야 한다. 먼저 드래그를 시작하기 위해서는 A(드래그할) 컨트롤의 DoDragDrop()을 호출해야 한다. 이 함수가 호출되면 드래그가 끝날 때까지 리턴되지 않고 계속적으로 Drag & Drop에 관련된 이벤트를 발생시킨다.   
  
**□ 드래그의 시작**  
◇ 드래그를 시작하기 위해서는 A(드래그할) 컨트롤에 DoDragDrop()을 호출해야 한다.  
  
**□ DoDragDrop()의 호출**  
◇ DoDragDrop() 함수가 호출되면 드래그가 끝날 때까지 리턴되지 않고 계속적으로 Drag & Drop에 관련된 이벤트를 발생시킨다.   
  
**□ DoDragDrop()의 결과**  
◇ Drag & Drop이 완료되면 DoDragDrop() 함수의 리턴값으로 DragDropEffects 열거형으로 리턴한다.  
  
Drag & Drop이 완료되면 DoDragDrop() 함수의 리턴값으로 DragDropEffects 열거형으로 리턴한다. DragDropEffects의 열거형은 멤버는 다음과 같다.  
  
  
[표 10\_8] DragDropEffects 열거형 멤버  
  
DragDropEffects에 의해서 어떠한 행위가 일어나는 것은 아니다. 단순히 Drag & Drop을 할 때 Copy나 Move 또는 Link의 동작이 발생했음을 의미한다.   
  
보통의 경우 드래그는 마우스의 조작에 의해서 시작하기 때문에 MouseDown에서 DoDragDrop()을 사용하는 경우가 많다. 다음의 예는 A(Label) 컨트롤에서 MouseDown 이벤트가 발생할 때 DoDragDrop()을 호출하는 예이다.  
  
**▒ Label의 MouseDown에서의 드래그의 시작(DoDragDrop())**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | public partial class DragEventForm : Form{     private void Label\_MouseDown(object sender, MouseEventArgs e){        DragDropEffects effect = ((Control)sender).DoDragDrop(sender, DragDropEffects.Move);        if (effect == DragDropEffects.Move){           //작업        }     }     //.....생략  } |

**▣ B(드롭되는) 컨트롤의 DragDrop 이벤트**  
  
B 컨트롤에 드롭이 되면 DragDrop 이벤트가 발생한다. 그렇기 때문에 B 컨트롤의 DragDrop 이벤트 핸들러를 구현해서 Drag & Drop을 구현하면 된다. 다음은 B 컨트롤(Panel)에 A 컨트롤(Label)이 드롭될 때 DragDrop 이벤트를 구현한 예이다.  
  
**▒ Panel의 DragDrop 이벤트 핸들러**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | public partial class DragEventForm : Form{     private void Label\_MouseDown(object sender, MouseEventArgs e){        //....... DoDragDrop()의 호출     }     private void pnContainer\_DragDrop(object sender, DragEventArgs e){        Control mycontrol = (Control)e.Data.GetData(typeof(Label));        Panel panel = (Panel)sender;        panel.Controls.Add(mycontrol);        mycontrol.Location= pnContainer.PointToClient(new Point(e.X, e.Y));     }  } |

B 컨트롤에 드롭되면 DragDrop 이벤트 핸들러인 pnContainer\_DragDrop()이 호출된다. 핸들러 내부에서는 드롭된 컨트롤의 참조값을 얻어내기 위해서 DragEventArgs를 이용하고 있다.  
  
**▒ 드롭되는 객체의 참조값 얻어내기**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1 | Control mycontrol = (Control)e.Data.GetData(typeof(Label)); |

DragEventArgs의 Data 속성에는 드롭에 관련된 모든 정보가 들어있다. Data의 GetData()를 이용하면 드롭된 A 컨트롤의 참조값을 얻을 수 있다. 이 때 매개변수로 얻어낼 컨트롤의 Type 객체를 넣어주면 된다.  
  
B 컨트롤에서 드롭된 A 컨트롤을 이동시키기 위해서 B 컨트롤(Panel)의 참조값과 드롭된 곳의 마우스의 좌표를 이용하고 있다. Panel의 참조값을 이용해서 Label을 Panel에 삽입하고, Label의 좌표를 지정해주면 이동이 완료된다.  
  
DragEventArgs에서 넘어오는 좌표는 화면좌표이기 때문에 이것을 Panel의 지역좌표로 바꾸어 주어야 한다. 이 때 이용되는 것이 PointToClient() 함수이다. 이 함수를 이용하면 화면좌표를 지역좌표로 변환할 수 있다.   
  
**▒ 화면좌표를 Panel의 지역좌표로 변환하기**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1 | pnContainer.PointToClient(new Point(e.X, e.Y)); |

반대로 지역좌표를 전역좌표로 바꾸기 위해서는 PointToScreen() 함수를 이용하면 된다.  
  
**▣ A 컨트롤에서 발생하는 이벤트**  
  
DoDragDrop() 함수에 의해 드래그가 시작되면 현재 커서가 위치한 곳의 컨트롤을 확인한다. 그런 다음 컨트롤이 Drag & Drop 가능한 대상인지 여부를 확인한다. 유효한 대상일 경우 이 상태에서 GiveFeedback 이벤트가 발생한다. GiveFeedback 이벤트를 사용하면 끌어서 놓기 작업 중에 사용자에게 시각적 효과를 주기 위해 마우스 포인터의 모양을 변경시킬 수 있는 최적의 장소이다.  
  
**☞ 참고**

|  |
| --- |
| DoDragDrop() 함수를 호출하면 사용자가 마우스를 놓기 전까지는 이후의 구문을 실행하지 않는다. 이렇게 되면 다른 작업 특히 또 다른 컴포넌트 위에 위치했는지와 같은 정보는 알 수 없다. 이러한 문제점을 해결하기 위해서 DoDragDrop() 함수가 호출된 상태에서 드래그하는 도중 유효한 컴포넌트일 경우 GiveFeedback 이벤트를 발생시킨다. |

**▣ B 컨트롤에서 발생하는 이벤트**  
  
기본적으로 드롭받는 B 컨트롤에서는 DragDrop 이벤트가 발생한다. 그리고 드래그 상태에서 마우스가 해당 컨트롤에 진입할 때 DragEnter 이벤트가, 컨트롤을 떠날 때 DragLeave 이벤트가, 컨트롤 내에서 이동하면 DragOver 이벤트가 발생한다.   
  
B 컨트롤에 A 컨트롤이 진입하는 DragEnter 이벤트에서 DragDropEffects를 지정하고 있다.   
  
**▒ Panel의 DragEnter 이벤트 핸들러**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | public partial class DragEventForm : Form{     private void pnContainer\_DragEnter(object sender, DragEventArgs e){        if (e.Data.GetDataPresent(typeof(Label))){           e.Effect = DragDropEffects.Move;        }else{           e.Effect = DragDropEffects.None;        }     }     //.....생략  } |

DragEnter 이벤트 핸들러 내부에서는 DragEventArgs의 Data 속성을 이용해서 드롭된 A 컨트롤의 데이터 타입을 확인하고 있다. Data의 GetDataPresent()를 이용해서 드롭된 컨트롤이 Label형이 맞는지 확인하고 있다. 맞으면 true를 반환한다.

**§ chap10\DragEventProject\DragEventForm.cs**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40 | 癤퓎sing System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Text;  using System.Windows.Forms;    namespace DragDropEventProject{      public partial class DragEventForm : Form{          public DragEventForm(){              InitializeComponent();          }          private void Label\_MouseDown(object sender, MouseEventArgs e){              DragDropEffects effect;              effect = ((Control)sender).DoDragDrop(sender, DragDropEffects.Move);              if (effect == DragDropEffects.Move){                  Label myLabel = sender as Label;                  MessageBox.Show("[" + myLabel.Text + "]???대룞?섏??듬땲??");              }          }          private void pnContainer\_DragEnter(object sender, DragEventArgs e){              if (e.Data.GetDataPresent(typeof(Label))){                  e.Effect = DragDropEffects.Move;              }else{                  e.Effect = DragDropEffects.None;              }          }          private void pnContainer\_DragDrop(object sender, DragEventArgs e){              Control mycontrol = (Control)e.Data.GetData(typeof(Label));              Panel panel = (Panel)sender;              panel.Controls.Add(mycontrol);              mycontrol.Location= pnContainer.PointToClient(new Point(e.X, e.Y));          }          int i=0;          private void Label\_GiveFeedback(object sender, GiveFeedbackEventArgs e){              this.Text = e.Effect.ToString() + i++;          }      }  } |

**10.2.7 드래그 & 드롭**  
  
이번에는 순수하게 MouseDown과 MouseMove 그리고 MouseUp 이벤트를 이용해서 드래그 & 드롭을 구현해보자.  
  
먼저 DragDropHandler라는 것을 만들고 다음과 같은 세개의 멤버변수와 MouseDown, MouseMove, MouseUp에 해당하는 3개의 함수를 두고 있다.  
  
**▒ 드래그 & 드롭 구현을 위한 DragDropHandler 클래스**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | class DragDropHandler{     private bool isDragging = false;     private Control myparent = null;     private Point offset;     public DragDropHandler(Control myparent)    {        this.myparent = myparent;     }     public void DragDrop\_MouseDown(object sender, MouseEventArgs e)    {        //작업     }     public void DragDrop\_MouseMove(object sender, MouseEventArgs e)    {        //작업     }     public void DragDrop\_MouseUp(object sender, MouseEventArgs e)    {        //작업     }  } |

DragDropHandler의 생성자에서는 매개변수로 들어온 Control형의 참조값을 멤버 변수 myparent에 보관하고 있다. 그리고 isDragging 멤버변수는 마우스가 눌러졌는지 아닌지를 나타내며, offset은 마우스를 다운하는 순간의 마우스 좌표를 보관하는 Point 구조체이다.  
  
먼저 마우스가 다운되었을 때 즉 MouseDown 이벤트에서는 isDragging을 true로 설정하고, 마우스가 눌러진 순간의 마우스 좌표를 offset 멤버변수에 저장한다.  
  
**▒ MouseDown 이벤트 핸들러**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | public void DragDrop\_MouseDown(object sender, MouseEventArgs e)    {     Control mycontrol = (Control)sender;     this.isDragging = true;     this.offset = new Point(e.X, e.Y);     mycontrol.BringToFront();  } |

그리고 마우스를 놓을 때 즉 MouseUp 이벤트에서는 isDragging만 false로 만들어준다.     
  
**▒ MouseUp 이벤트 핸들러**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3 | public void DragDrop\_MouseUp(object sender, MouseEventArgs e)    {     this.isDragging = false;  } |

실제 작업은 MouseMove에서 이루어진다. MouseMove 이벤트가 발생할 때마다 현재의 컨트롤의 좌표를 이동시키기만 하면 된다. isDragging이 true인 상태에서 MouseMove 이벤트를 발생시킨 컨트롤의 좌표를 다시 계산해서 설정하고 있다.  
  
**▒ MouseMove 이벤트 핸들러**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | public void DragDrop\_MouseMove(object sender, MouseEventArgs e)    {      Control mycontrol = (Control)sender;     if (this.isDragging == true)        {         mycontrol.Left = mycontrol.Left  + (e.X - offset.X) ;         mycontrol.Top = mycontrol.Top + (e.Y - offset.Y);      }   } |

위의 컨트롤의 이동 위치 계산법은 MouseDown일 때의 마우스 좌표와 MouseMove일 때의 마우스 좌표의 차이를 컨트롤의 Left와 Top에 반영시키는 방식이다. MouseDown일 때의 좌표는 offset으로 저장해두었으며, MouseMove일 때의 좌표는 MouseEventArgs로부터 얻을 수 있다.  
  
DragDropHandler 클래스는 MouseDown, MouseMove, MouseUp에 해당하는 이벤트 핸들러를 구현했을 뿐 어떠한 컨트롤에도 사용하지는 않은 상태이다. 이를 위해 DragDrop\_AddControl()이라는 함수를 두고 있다. 이 함수를 이용하면 쉽게 이벤트 핸들러들을 장착시킬 수 있다.  
  
**▒ 컨트롤에 이벤트 핸들러들을 추가하는 함수**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4  5 | public void DragDrop\_AddControl(Control sender)    {     sender.MouseDown += this.DragDrop\_MouseDown;     sender.MouseUp += this.DragDrop\_MouseUp;     sender.MouseMove += this.DragDrop\_MouseMove;  } |

이제 모든 작업이 완료되었다. DragDropHandler를 객체를 생성한 후 각각의 컨트롤에 DragDrop\_AddControl()이라는 함수만 호출하면 드래그 & 드롭이 동작할 것이다. DragDropHandler을 컨트롤에 연결하는 전체 예는 다음과 같다.

**§ chap10\DragDropProject\DragDropHandler.cs**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46 | using System;  using System.Collections.Generic;  using System.Text;  using System.Windows.Forms;  using System.Drawing;  using System.Diagnostics;    namespace DragDropProject{      class DragDropHandler{          private bool isDragging = false;          private Control myparent = null;          private Point offset;          public DragDropHandler(Control myparent){              this.myparent = myparent;          }          #region DragDropHandler 멤버          public void DragDrop\_MouseDown(object sender, MouseEventArgs e){              Control mycontrol = (Control)sender;              this.isDragging = true;              this.offset = new Point(e.X, e.Y);              mycontrol.BringToFront();          }          public void DragDrop\_MouseMove(object sender, MouseEventArgs e){              Control mycontrol = (Control)sender;              if (this.isDragging == true){                  String str = String.Format("e.x={0},e.y={1} offset.x={2},offset.y={3} my.x={4},my.y={5}", e.X, e.Y, offset.X, offset.Y, mycontrol.Left, mycontrol.Top);                  Debug.WriteLine(str);                    this.myparent.Text = str;                  mycontrol.Left = mycontrol.Left + (e.X - offset.X);                  mycontrol.Top = mycontrol.Top + (e.Y - offset.Y);              }else{                  Console.Write("A");              }          }          public void DragDrop\_MouseUp(object sender, MouseEventArgs e){              this.isDragging = false;          }          public void DragDrop\_AddControl(Control sender){              sender.MouseDown += this.DragDrop\_MouseDown;              sender.MouseUp += this.DragDrop\_MouseUp;              sender.MouseMove += this.DragDrop\_MouseMove;          }          #endregion      }  } |

**§ chap10\DragDropProject\DragDropForm.cs**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | 癤퓎sing System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Text;  using System.Windows.Forms;    namespace DragDropProject{      public partial class DragDropForm : Form{          DragDropHandler ddrop = null;          public DragDropForm(){              InitializeComponent();              ddrop = new DragDropHandler(this);              ddrop.DragDrop\_AddControl(this.button1);              ddrop.DragDrop\_AddControl(this.label1);              ddrop.DragDrop\_AddControl(this.checkBox1);          }      }  } |

위와 같이 하면 DragDropHandler 하나로 여러 개의 컨트롤의 드래그 & 드롭을 구현할 수 있다.

**10.2.8 Application.DoEvents()**  
  
Application.DoEvents() 함수는 이벤트를 핸들할 때 만날 수 있는 함수이다. 이벤트가 발생하면 이벤트들은 메시지 큐에 순서대로 적재된다. 이들을 하나씩 끄집어 내서 메시지를 처리하는 것이 바로 이벤트 처리이다.   
  
**□ 메시지 처리 방식**  
◇ 메시지 큐에서 메시지를 하나씩 끄집어 내어서 처리하는 방식  
  
메시지 큐에서 하나의 메시지를 끄집어 내서 처리할 때 처리하는 작업이 너무 오래 걸린다고 가정하자. 현재 상황은 하나의 메시지를 처리하고 있기 때문에 메시지 큐에서 다른 메시지를 끄집어 내서 처리할 수 없는 상황이 발생한다. 이 상황에 사용자의 조작과 내부의 로직에 의해 메시지가 계속 발생하게 된다면 흔히 알고 있는 [응답없음] 메시지가 창에 나타난다.  
  
**□ 응답없음**  
◇ 하나의 메시지를 처리하는 작업이 너무 오래 걸릴 경우 다른 메시지를 처리할 수 없기 때문에 나타나는 현상  
  
이러한 문제점은 메시지 처리에서 너무나도 자주 발생하는 상황이다. 예를 들어보자. 다음과 같이 버튼의 이벤트 핸들러에서 double형 연산을 아주 많이 한다고 가정하자.  
  
**▒ 작업량이 많은 버튼의 이벤트 핸들러**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | public partial class DoEventForm : Form{     private void button1\_Click(object sender, EventArgs e)    {        for (long i = 0; i < 100000000; i++) {           double d = i \* 0.453432F \* 0.23423D;        }        this.Text = "DoEvents() 사용 안 함 완료";     }  } |

버튼을 누르는 순간 다른 작업을 할 수 없는 상황이 발생한다. 버튼을 누르는 순간 메시지 큐에서 버튼 클릭에 관련된 메시지를 끄집어 내서 button1\_Click() 이벤트 핸들러를 호출할 것이다. 하지만 button1\_Click()에서는 100000000번의 for문이 기다리고 있다. 그렇기 때문에 for문을 빠져나가지 못하면 다른 메시지를 처리할 수 없는 것이다. 이 상태에서 창을 움직이거나 다른 버튼을 클릭한다면 반응이 없을 것이다.  
  
이것의 해결책은 바로 Application.DoEvents()에 있다. 이 함수를 이용하면 for문 중간에 메시지를 끄집어 내서 처리한 후 다시 for문을 재개할 수 있다.  
  
**▒ 작업량이 많은 버튼의 이벤트 핸들러**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | public partial class DoEventForm : Form{     private void button1\_Click(object sender, EventArgs e)    {        for (long i = 0; i < 100000000; i++) {           double d = i \* 0.453432F \* 0.23423D;           if(i%1000000==0)              Application.DoEvents();        }        this.Text = "DoEvents() 사용함 완료";     }  } |

위와 같이 하면 버튼의 이벤트 핸들러를 처리하는 중간 중간 Application.DoEvents()를 호출하기 때문에 현재의 모든 메시지를 처리하면서 자신의 작업을 처리할 수 있게 된다.

**§ chap10\DoEventProject\DoEventForm.cs**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Text;  using System.Windows.Forms;    namespace DoEventProject{      public partial class DoEventForm : Form{          public DoEventForm(){              InitializeComponent();          }          private void button1\_Click(object sender, EventArgs e){              for (long i = 0; i < 100000000; i++)              {                  double d = i \* 0.453432F \* 0.23423D;              }              this.Text = "DoEvents() 사용 안함 완료";          }          private void button2\_Click(object sender, EventArgs e){              for (long i = 0; i < 100000000; i++){                  double d = i \* 0.453432F \* 0.23423D;                  if(i%1000000==0)                      Application.DoEvents();              }              this.Text = "DoEvents() 사용함 완료";          }      }  } |

**10.2.9 Novel! : 메시지 후킹**  
  
메시지 후킹(Hooking)에는 레벨이 있다. 앞에서 우리는 Application 레벨의 메시지 후킹 기법을 배운 적이 있다. Application 레벨의 메시지가 처리되고 나면, 애플리케이션 내에 있는 윈도우의 WndProc()에서 이 메시지들이 처리된다. 그리고 나면 최종적으로 On 이벤트 함수가 호출된다.  
  
**▒ 메시지 처리 절차**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3 | 1. Application 레벨의 메시지 처리  2. WndProc()에서의 메시지 처리  3. On 계열의 이벤트 함수의 메시지 처리 |

보통의 경우엔 On 계열의 이벤트 함수를 오버라이드하거나 이벤트 핸들러를 이용해서 메시지를 처리한다. 하지만 그 상위 레벨에서 메시지를 처리할 필요가 있을 때가 있다. 이럴 경우에 WndProc()을 오버라이드해서 사용하면 된다. WndProc()을 오버라이드하는 일반적인 형태는 다음과 같다.  
  
**▒ WndProc()의 일반적인 형태**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | protected override void WndProc(ref Message m) {     switch (m.Msg) {        case WM\_MOUSEMOVE:           //....작업;           break;        case WM\_KEYDOWN:           //....작업;           break;     }     base.WndProc(ref m);  } |

위의 상황에서 WndProc() 내에서 base.WndProc(ref m)을 호출해주지 않으면 그 이후의 메시지는 하위의 이벤트로 전달되지 않는다. 하지만 윈도우의 기본적인 메시지 처리는 해주어야 한다. 이를 위해 base.DefWndProc(ref m)을 호출해주면 된다.  
  
**□ base.WndProc(ref m)**  
◇ base.WndProc(ref m)을 호출해주지 않으면 그 이후의 메시지는 하위의 이벤트를 발생시키지 않는다.   
  
**□ base.DefWndProc(ref m)**  
◇ 윈도우의 기본적인 메시지 처리는 해주는 함수를 호출한다.  
◇ base.WndProc(ref m)을 호출하지 않을 경우에는 base.DefWndProc(ref m)을 호출해서 기본 메시지가 호출되게 해야 한다.  
  
위의 WM\_MOUSEMOVE와 WM\_KEYDOWN과 같은 상수는 다음과 같이 직접 정의해서 사용해야 한다.  
  
**▒ 윈도우 메시지 상수**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2 | private const int WM\_MOUSEMOVE = 0x200;  private const int WM\_KEYDOWN = 0x100; |

이 상수들의 값은 Win32 API에 정의되어 있으며, C#에서 별도로 제공하지 않기 때문에 사용자가 직접 선언한 후 사용해야 한다.   
  
원도우의 가장 기본적인 메시지 처리 방식은 우리가 지금까지 앞 절들에서 배워왔던 이벤트 처리 기법이다. 하지만 이 기법으로 핸들할 수 없는 미세한 작업들이 있을 수 있다. 이럴 경우에는 이벤트 처리 전에 호출되는 WndProc()에서 작업하면 된다.  
  
다음은 WndProc()을 오버라이드한 후 WM\_MOUSEMOVE를 처리하는 예를 보여주고 있다.

**§ chap10\MessageHookProject\MessageHookForm.cs**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Text;  using System.Windows.Forms;  using System.Diagnostics;    namespace MessageHookProject{      public partial class MessageHookForm : Form{          public MessageHookForm(){              InitializeComponent();          }          private const int WM\_MOUSEMOVE = 0x200;          protected override void WndProc(ref Message m){              switch (m.Msg){                  case WM\_MOUSEMOVE:                      Point p = new Point(m.LParam.ToInt32());                      this.Text = p.X + "," + p.Y;                      base.WndProc(ref m);                      //base.DefProc(ref m);                      return;              }              base.WndProc(ref m);          }          private void MessageHookForm\_MouseMove(object sender, MouseEventArgs e){              Trace.WriteLine( e.X +"," + e.Y);          }      }  } |

이 예제는 WM\_MOUSEMOVE 메시지가 발생했을 때 전달된 Message 객체의 LParam에서 마우스의 좌표를 추출해서 폼의 타이틀에 출력하는 예이다.

**10.3 그래픽 GDI+**  
  
  
다음은 여러분들이 이 절에서 학습할 각 소절들에 대한 제목들이다.  
  
**□  10.3 그래픽 GDI+**  
◇ 10.3.1 Novel! : Graphics 클래스  
◇ 10.3.2 그리기 함수  
◇ 10.3.3 Color 클래스  
◇ 10.3.4 Image 그리기  
◇ 10.3.5 Novel! : 더블 버퍼링  
◇ 10.3.6 이미지 크기 변경  
◇ 10.3.7 좌표계  
  
위의 주제들을 하나씩 학습해 나가면서 각각의 주제에 대한 내용을 익히도록 하자.

**10.3.1 Novel! : Graphics 클래스**  
  
윈도우나 이미지를 상대로 그림을 그릴 때 사용하는 도구가 바로 Graphics 클래스이다. 이 클래스를 이용해서 도형이나 텍스트 또는 이미지를 그릴 수 있다. 그렇다면 먼저 Graphics를 얻어내는 방법부터 알아보자. Graphics를 얻는 방법은 크게 두 가지로 나눌 수 있다.  
  
**□ Graphics를 얻는 방법**  
◇ Paint 이벤트의 매개변수를 통해서 Graphics 얻어내기  
◇ 사용자가 직접 원하는 윈도우나 원하는 이미지로부터 Graphics 얻어내기  
  
**▣ Paint 이벤트 핸들러의 Graphics**  
  
먼저 Paint 이벤트의 매개변수부터 살펴보자. Paint 이벤트를 구현하는 목적은 그림이 다시 그려져야 할 필요가 있을 때마다 Paint 이벤트가 발생하기 때문이다. Paint 이벤트 핸들러의 매개변수로 넘어오는 PaintEventArgs의 Graphics 속성을 이용해서 그림을 그리면 된다.  
  
**□ Paint 이벤트를 구현하는 목적**  
◇ 그림이 다시 그려져야 할 필요가 있을 때마다 Paint 이벤트가 발생하기 때문이다.  
  
**□ Paint 이벤트 핸들러의 매개변수**  
◇ 매개변수로 넘어오는 PaintEventArgs의 Graphics 속성을 이용해서 그림을 그리면 된다.  
  
다음은 Paint 이벤트 핸들러를 구현한 후 Graphics 클래스를 사용하는 예를 보여주고 있다.  
  
**▒ Paint 이벤트 핸들러에서의 Graphics 사용**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3 | private void PaintForm\_Paint(object sender, PaintEventArgs e) {        e.Graphics.DrawString("Hello Paint!!", new Font("굴림", 20), Brushes.Black, 50, 50);  } |

PaintEventArgs의 Graphics의 경우에는 사용자가 생성 또는 소멸할 필요 없이 Paint 이벤트 핸들러 내에서 마음대로 사용해도 되는 Graphics이다. 실제 Graphics가 가장 많이 사용되는 곳이기도 하다. 그림을 계속해서 그려지게 하기 위해서는 Paint 이벤트 핸들러가 최적의 장소이다.  
  
**▣ 사용자가 직접 Graphics 객체 얻어내기**  
  
다음으로 사용자가 직접 원하는 윈도우나 이미지로부터 Graphics를 얻어내는 기법이다. 모든 윈도우로부터 Graphics를 얻어낼 수 있으며, 이 때 사용하는 함수가 바로 CreateGraphics()이다. 이렇게 얻은 Graphics는 반드시 Dispose()를 호출해서 자원을 해제해야 한다.  
  
**□ 윈도우로부터 직접 얻어낸 Graphics**  
◇ CreateGraphics()를 사용해서 모든 윈도우로부터 Graphics를 얻어낼 수 있다.  
◇ 이렇게 얻은 Graphics는 반드시 자원 해제를 위해 Dispose()를 호출해야 한다.  
  
다음은 버튼이 눌러지는 순간 폼의 Graphics를 얻어내서 문자열을 폼에 그리는 예이다.  
  
**▒ 윈도우로부터 직접 Graphics 얻어내기**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | public partial class PaintForm : Form{     private void btnPaint\_Click(object sender, EventArgs e){        Graphics g = this.CreateGraphics();        //g를 이용한 그림 그리기        g.Dispose();     }  } |

Graphics는 기본적으로 누구의 Graphics냐에 따라서 그리는 대상이 달라진다. 현재 폼으로부터 Graphics를 얻었다면, 이 Graphics를 이용해서 그림을 그리면 폼에 그림이 그려진다.  
  
**▣ Paint 이벤트의 Graphics와 CreateGraphics의 차이점**  
  
Paint 이벤트의 Graphics와 CreateGraphics의 차이점은 Paint 이벤트의 특징과 관련이 있다. 화면이 다시 그려질 필요가 있을 때 즉 Paint 이벤트가 발생할 때마다 매번 그림이 그려진다. 반면 CreateGraphics()의 경우 사용자가 얻어내서 그리는 순간에만 그려지는 특징이 있다. 사실 이것은 Graphics의 특징이 아니라 Paint 이벤트의 특징이다.  
  
**□ Paint 이벤트의 Graphics**  
◇ 화면이 다시 그려질 필요가 있을 때 즉 Paint 이벤트가 발생할 때마다 매번 그려진다.  
  
**□ CreateGraphics()**  
◇ 그리는 작업을 할 때에만 그려진다.  
◇ 화면이 다시 그려지면 CreateGraphics()를 이용해서 그린 내용은 사라진다.  
  
또 다른 차이점으로는 Paint 이벤트의 Graphics는 그냥 사용하기만 하면 된다. 하지만 CreateGraphics()로 얻은 Graphics는 반드시 사용 후에는 Dispose()를 호출해서 자원을 해제해주어야 한다.  
  
Paint 이벤트의 Graphics와 CreateGraphics()를 테스트하는 예는 다음과 같다.

**§ chap10\PaintProject\PaintForm.cs**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | 癤퓎sing System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Text;  using System.Windows.Forms;    namespace PaintProject{      public partial class PaintForm : Form{          public PaintForm(){              InitializeComponent();          }          private void PaintForm\_Paint(object sender, PaintEventArgs e){              e.Graphics.DrawString("Paint Event!!", new Font("援대┝", 20),                                              Brushes.Black, 50, 50);          }          private void btnPaint\_Click(object sender, EventArgs e){              Graphics g = this.CreateGraphics();              g.DrawString("Create Graphics!!", new Font("援대┝", 20),                                              Brushes.Black, 50, 80);              g.Dispose();          }      }  } |

**10.3.3 Color 클래스**  
  
Color 구조체는 RGB에 해당하는 색상을 표현하기 위한 구조체이다. Color 구조체는 실전에 자주 등장하기 때문에 한번쯤 정리하고 넘어갈 필요가 있다.  
  
기본적인 색상은 다음과 같이 Color 구조체에서 스태틱 형태로 제공된다.  
  
**□ static으로 정의된 색상**  
◇ Color.Red  
◇ Color.Greean  
◇ Color.Blue  
  
Color.FromArgb() 함수를 이용해서 사용자가 직접 RGB 색상을 조합해서 사용할 수도 있다.  
  
**▒ RGB 색상 조합하기**

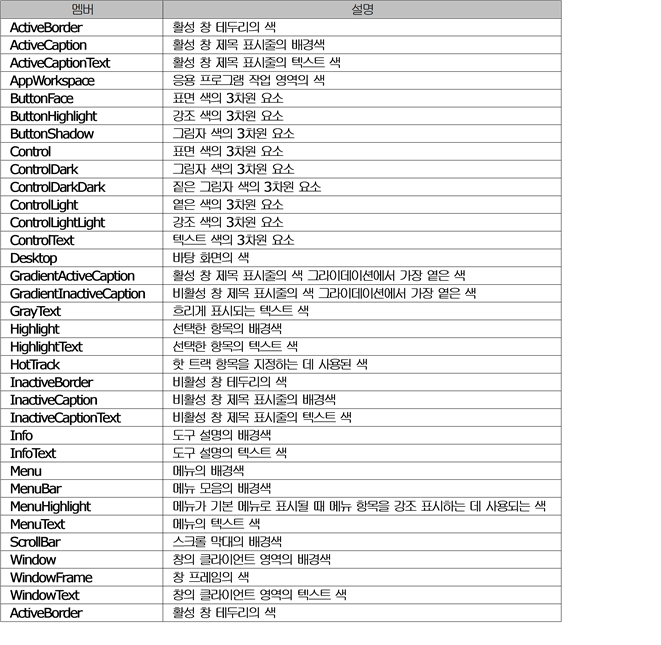
[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3 | Color red = Color.FromArgb(255, 0, 0);  Color green = Color.FromArgb(0, 255, 0);  Color blue = Color.FromArgb(0, 0, 255); |

Color.FromArgb()의 매개변수는 순서대로 R, G, B에 해당하며 각각 0부터 255의 값을 가질 수 있다.   
  
또 다른 방법으로 RGB 이외에 투명도를 지정할 수도 있으며 이 때 첫번째 매개변수가 투명도에 해당하며 나머지는 RGB에 해당한다.  
  
**▒ 투명도 지정**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1 | Color alpa = Color.FromArgb(128, 0, 128, 128); |

Color 구조체에서 A, R, G, B의 속성을 이용해서 각각의 색상을 추출할 수 있으며, 추출된 데이터는 byte형이다.  
  
**□ Color 구조체의 속성**  
◇ ARGB의 각각의 색상을 바이트형으로 A, R, G, B 속성으로 얻을 수 있다.  
  
Color 구조체에 정의된 색상과 사용자가 직접 만들어서 사용하는 색상 이외에 시스템에서 사용하는 색상이 있다. 시스템에서 사용하는 색상은 SystemColor 클래스의 속성으로 지정되어 있으며 다음과 같은 형식으로 사용할 수 있다.  
  
  
[표 10\_9] SystemColor의 속성  
  
Color의 스태틱 색상과 Color.FormArgb()를 이용한 색상 그리고 SystemColor를 이용한 시스템 색상을 이용할 줄 알면 색상에서는 문제되는 것이 없을 것이다.

**10.3.4 Image 그리기**  
  
C++ 계열에서 골치 아픈 것이 바로 이미지를 로딩하고 저장하는 것이다. 하지만 C#에서는 근본적으로 이미지를 로딩하는 것이 쉬워졌다. 일단 다음의 한 줄이면 이미지를 로딩할 수 있다.   
  
**▒ 이미지 로딩**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1 | Image img1 = Image.FromFile(Application.StartupPath + @"\img1.gif"); |

이미지를 로딩하기 위해서는 Image.FromFile()을 이용하면 된다. 이 때 리턴되는 객체는 Image형 객체이다. 이 Image형 객체가 바로 이미지를 핸들하기 위한 기본적인 클래스이다.  
  
Image 클래스는 상속되어서 Bitmap과 Metafile로 구분된다. Bitmap은 말 그대로 비트맵 이미지를 관리하기 위한 클래스이며, Metafile은 벡터 이미지를 관리하기 위한 클래스이다.   
  
**□ Bitmap**  
◇ 비트맵 이미지를 관리하기 위한 클래스  
  
**□ Metafile**  
◇ 벡터 이미지를 관리하기 위한 클래스  
  
Bitmap과 Metafile은 둘 다 Image를 상속받으며 이미지를 로딩하는 방법은 다음과 같다.  
  
**▒ Bitmap과 Metafile의 이미지 로딩**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2 | Bitmap img2 = new Bitmap(Application.StartupPath + @"\img2.gif");  Metafile img3 = new Metafile(Application.StartupPath + @"\img3.emf"); |

이미지를 로딩하는 방법이 쉬운 것은 개발자에게 너무나 다행스러운 일이다. 이미지를 로딩했다면 그 다음은 그리는 작업이다. 이미지를 로딩해서 그리는 가장 일반적인 예는 다음과 같다.

**§ chap10\ImageProject\ImageForm.cs**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | 癤퓎sing System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Text;  using System.Windows.Forms;  using System.Drawing.Imaging;  using System.IO;    namespace ImageProject{      public partial class ImageForm : Form{          private Image img1 = null;          private Bitmap img2 = null;          private Metafile img3 = null;            public ImageForm(){              InitializeComponent();              this.img1 = Image.FromFile(Application.StartupPath + @"\img1.gif");              this.img2 = new Bitmap(Application.StartupPath + @"\img2.gif");              this.img3 = new Metafile(Application.StartupPath + @"\img3.emf");          }          private void ImageForm\_Paint(object sender, PaintEventArgs e){              if (this.img1 != null)                  e.Graphics.DrawImage(img1, 10, 10);              if (this.img2 != null)                  e.Graphics.DrawImage(img2, 10, 100);              if (this.img3 != null)                  e.Graphics.DrawImage(img3, 10, 200);          }      }  } |

Paint 이벤트 핸들러에서 Image 객체가 로딩되어 있는지 확인한 후 그리는 방식은 가장 많이 사용하는 이미지 그리기 기법이다.

**10.3.5 Novel! : 더블 버퍼링**  
  
뭔가를 그리기 위해서 Graphics를 이용한다. Graphics를 이용해서 그림을 그릴 경우 디바이스를 향해서 직접 그린다는 생각으로 그림을 그리면 된다. 하지만 그림의 크기가 클 경우나 자주 그려져야 할 경우 깜박임이 발생하게 된다. 이러한 깜박임은 더블 버퍼링으로 해결할 수 있다.  
  
더블 버퍼링이란 Graphics로 직접 그리는 것이 아니라, 메모리상의 버퍼에 그림을 그린 후 메모리 버퍼를 통째로 전달해서 그림을 그리는 방식이다. 이 방법은 Buffered 스트림과 비슷한 동작 방식이라고 생각하면 된다.  
  
**□ 더블 버퍼링이란**  
◇ Graphics로 직접 그리는 것이 아니라, 메모리상의 버퍼에 그림을 그린 후 메모리 버퍼를 통째로 전달해서 그림을 그리는 방식  
  
닷넷 프레임웍 2.0에서는 기본적으로 폼 차원에서 더블 버퍼링을 지원한다. Graphics로 그리면 폼의 메모리 버퍼에 그림을 그리고, 버퍼를 한꺼번에 그리는 폼 레벨의 더블 버퍼링을 지원한다. 폼의 DoubleBuffered 기능을 true로 만들어주면 된다.  
  
**□ 폼의 더블 버퍼링 지원**  
◇ 폼의 DoubleBuffered 속성을 true로 설정하면 폼에서 자동으로 더블 버퍼링을 지원한다.  
  
기본으로 제공하는 더블 버퍼링의 단점은 폼차원에서 제어된다는 데 있다. 어느 시점에 더블 버퍼링이 지원되어야 하며, 언제 그려져야 하는지에 대한 사항은 전부 폼의 제어에 따른다.   
  
이러한 제어를 사용자가 직접 하고자 한다면 더블 버퍼링을 구현해야 한다. 그리고 폼에서 더블 버퍼링을 지원한다 할지라도 더블 버퍼링이 사용되는 곳이 많기 때문에 알아둘 필요가 있다.  
  
**▣ 더블 버퍼링을 위한 메모리 버퍼 이미지 만들기**  
  
더블 버퍼링을 구현하기 위해서는 먼저 메모리 버퍼가 필요하다. 다음은 메모리 버퍼로 사용하기 위한 이미지를 생성하는 예이다.  
  
**▒ 메모리 버퍼로 사용할 비트맵 객체 생성**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1 | Bitmap bmp = new Bitmap(100, 100); |

이 메모리 버퍼에 그림을 그리기 위해서는 비트맵 이미지 객체로부터 Graphics 객체를 얻어내야 한다. Bitmap 객체로부터 Graphics를 얻어내는 방법은 다음과 같다.  
  
**▒ Bitmap으로부터 Graphics 얻어내기**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4  5 | Bitmap bmp = new Bitmap(100, 100);  Graphics g = Graphics.FromImage(bmp);  g.Clear(Color.Yellow);  //.....작업  g.Dispose(); |

보통의 경우 Graphics를 윈도우로부터 얻어내지만 Image 객체로부터 Graphics를 얻어낼 수도 있다. 이 때 이용되는 함수가 바로 Graphcis.FromImage()이다. Graphics를 얻었다면 그리기만 하면 된다. 이렇게 하면 메모리 이미지가 하나 완성되는 것이다. 완성된 메모리 이미지를 다시 한번 폼에 그려주기만 하면 된다.  
  
**▣ 저장하기**  
  
메모리 형태의 Bitmap 객체를 만들고 그림까지 그렸다면, 이것을 저장할 필요가 있을 것이다. 저장을 위해서 파일 저장 대화상자를 띄우고, 파일명을 지정한 후 저장하면 된다. 먼저 파일 저장 대화상자를 띄우는 예는 다음과 같다.  
  
**▒ 파일 저장 대화 상자 띄우기**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | SaveFileDialog sfd = new SaveFileDialog();  sfd.InitialDirectory = "c:\\";  sfd.Filter = "JPEG 파일(\*.jpg)|\*.jpg | PNG 파일(\*.png)|\*.png | All files (\*.\*)|\*.\*";  sfd.FilterIndex = 3;  sfd.RestoreDirectory = true;  if (sfd.ShowDialog() == DialogResult.OK){     //이미지 저장 루틴 삽입  } |

파일 저장 대화상자의 역할을 하는 SaveFileDailog 객체를 생성한 후 ShowDialog()를 호출하면 파일 저장 대화상자가 화면에 나타난다. 사용자가 파일명을 입력한 후 버튼 누르는 순간 ShowDialog()는 DialogResult 값을 리턴한다. 이 때 리턴된 값이 DialogResult.OK이면 이미지 저장 루틴을 삽입하면 된다. 이미지를 저장하는 방법은 다음과 같다.  
  
**▒ Bitmap 이미지 저장하기**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2 | bmp.Save(sfd.FileName);//디폴트는 PNG 인코딩 사용  //bmp.Save(sfd.FileName, ImageFormat.Jpeg); |

Bitmap의 Save() 함수에서 파일명만 주고 저장하면 PNG 인코딩으로 저장된다. 만약 JPG 형식으로 저장하고 싶다면 ImageFormat.Jpeg를 두번째 매개변수로 사용하면 된다.

**§ chap10\GraphicsProject\GraphicsForm.cs**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43 | 癤퓎sing System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Text;  using System.Windows.Forms;  using System.Drawing.Imaging;    namespace GraphicsProject{      public partial class GraphicsForm : Form{          private int count = 0;          private Bitmap bmp = new Bitmap(100, 100);          public GraphicsForm(){              InitializeComponent();          }          private void btnDraw\_Click(object sender, EventArgs e){              this.count++;              Graphics g = Graphics.FromImage(bmp);              g.Clear(Color.Yellow);              g.DrawString(Convert.ToString(count), new Font("援대┝", 30),                                                              Brushes.Blue, 30, 30);              g.DrawEllipse(new Pen(Color.Red, 5), 25, 25, 50, 50);              g.Dispose();              this.Invalidate();          }          private void GraphicsForm\_Paint(object sender, PaintEventArgs e){              if (bmp != null)                  e.Graphics.DrawImage(bmp, 100, 100);          }          private void btnSave\_Click(object sender, EventArgs e){              SaveFileDialog sfd = new SaveFileDialog();              sfd.InitialDirectory = "c:\\";              sfd.Filter = "JPEG ?뚯씪(\*.jpg)|\*.jpg | PNG ?뚯씪(\*.png)|\*.png | All files (\*.\*)|\*.\*";              sfd.FilterIndex = 3;              sfd.RestoreDirectory = true;              if (sfd.ShowDialog() == DialogResult.OK){                  bmp.Save(sfd.FileName);//?뷀뤃?몃뒗 PNG ?몄퐫???ъ슜                  bmp.Save(sfd.FileName, ImageFormat.Jpeg);              }          }      }  } |

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

using System.Drawing.Imaging;

using System.Drawing.Drawing2D;

namespace ImageSizeProject{

    public partial class ImageForm : Form{

        private Image sImage = Image.FromFile(Application.StartupPath + @"\ndm.jpg");

        Bitmap tImage = null;

        public ImageForm(){

            InitializeComponent();

        }

        private void btnResize\_Click(object sender, EventArgs e){

            int width = sImage.Width / 2; //異뺤냼 Width

            int height = sImage.Height / 2; //異뺤냼 Height

            tImage = new Bitmap(width, height, sImage.PixelFormat);

            tImage.SetResolution(sImage.HorizontalResolution,

                                                        sImage.VerticalResolution);

            Graphics g = Graphics.FromImage(tImage);

            g.InterpolationMode = InterpolationMode.Bilinear;

            g.DrawImage(sImage, //洹몃┫ ?대?吏

                    new Rectangle(0, 0, width, height),//異뺤냼 ?ш린

                    new Rectangle(0, 0, sImage.Width, sImage.Height),//?먮낯 ?ш린

                    GraphicsUnit.Pixel);

            g.Dispose();

            this.Invalidate(); //?ㅼ떆 洹몃━湲?

        }

        private void ImageForm\_Paint(object sender, PaintEventArgs e){

            if (sImage != null)

                e.Graphics.DrawImage(sImage, 10, 10);

            if (tImage != null)

                e.Graphics.DrawImage(tImage, 10 + sImage.Width + 10, 10);

        }

    }

}

**10.3.7 좌표계**  
  
Graphics를 이용해서 뭔가를 그릴 때 사용되는 기본 좌표계의 원점은 폼의 왼쪽 상단이다. 다음과 같은 코드를 출력해보라. 그러면 정확하게 왼쪽 상단의 0,0 지점을 확인할 수 있을 것이다.  
  
**▒ 기본 좌표계**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3 | Graphics g = this.CreateGraphics();  g.DrawString("기본 좌표계 0,0 지점", new Font("굴림", 12), Brushes.Brown, 0, 0);  g.Dispose(); |

만약 이 원점을 바꾸고자 한다면 Graphics로 뭔가를 그리기 전에 TranslateTransform() 함수를 이용해서 원점을 변경해주면 된다.  
  
**▒ 원점 변환**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4 | Graphics g = this.CreateGraphics();  g.TranslateTransform(100, 100);  g.DrawString("원점변화(100,100) 좌표계 0,0 지점", new Font("굴림", 12), Brushes.Brown, 0, 0);  g.Dispose(); |

기본적으로 적용되는 좌표의 단위는 픽셀 단위이다. 만약 사용자가 이것을 다른 단위로 바꾸고자 한다면 Graphics의 PageUnit 속성을 이용하면 된다.  
  
**▒ 측정 단위 변환**

[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4 | Graphics g = this.CreateGraphics();  g.PageUnit = GraphicsUnit.Inch;  g.DrawString("기본 좌표계(인치) 1인치, 2인치 지점", new Font("굴림", 12), Brushes.Brown, 1, 2);  g.Dispose(); |

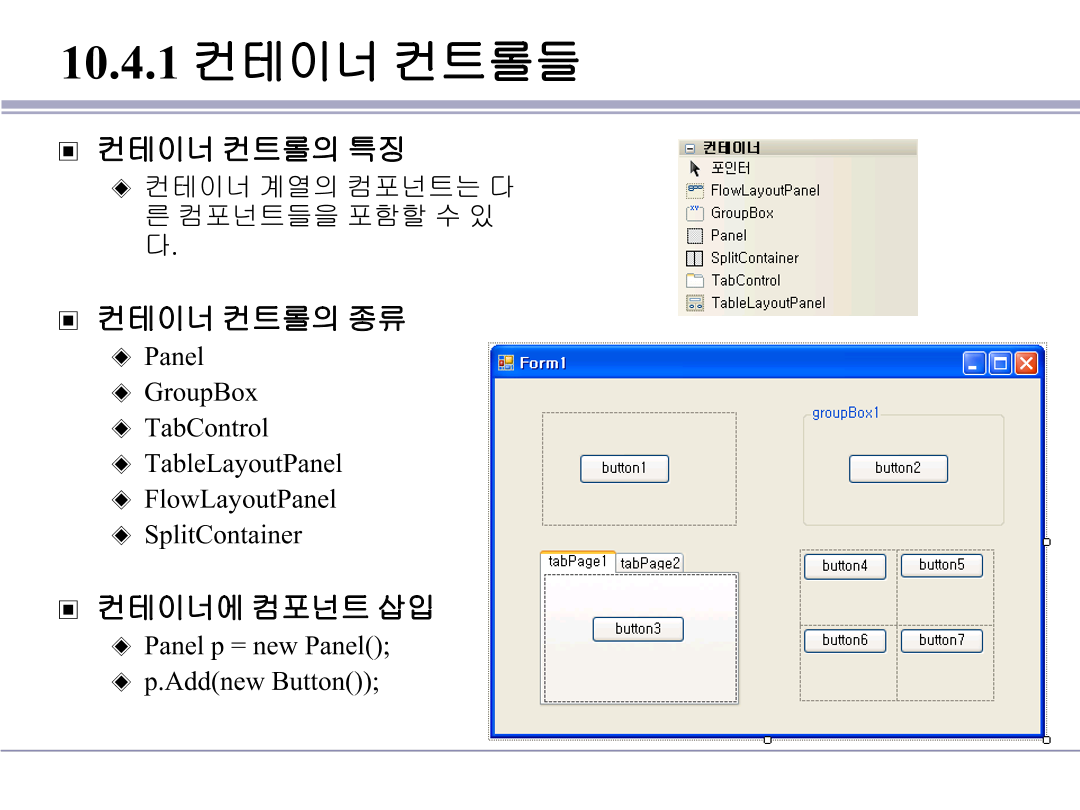
Graphics의 PageUnit 속성에 GraphicsUnit.Inch를 설정하고 있다. 이렇게 되면 좌표(1,2)의 값은 좌표(1인치, 2인치)가 된다. 참고로 1인치는 25.4mm이다.

**§ chap10\CoordinateProject\CoordinateForm.cs**

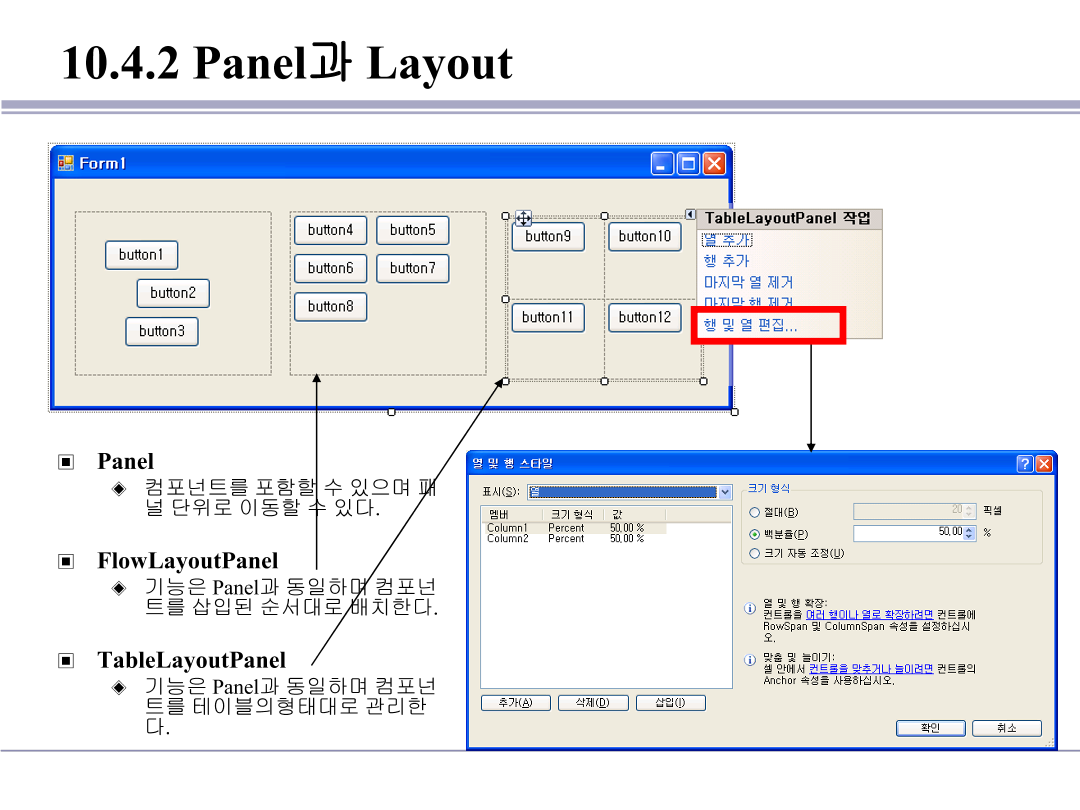
[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Text;  using System.Windows.Forms;    namespace CoordinateProject{      public partial class CoordinateForm : Form{          public CoordinateForm(){              InitializeComponent();          }          private void btnBasic\_Click(object sender, EventArgs e){              Graphics g = this.CreateGraphics();              g.DrawString("湲곕낯 醫뚰몴怨?0,0 吏??, new Font("援대┝", 12), Brushes.Brown, 0, 0);              g.Dispose();          }          private void btnTransform\_Click(object sender, EventArgs e){              Graphics g = this.CreateGraphics();              g.TranslateTransform(100, 100);              g.DrawString("?먯젏蹂??100,100) 醫뚰몴怨?0,0 吏??,                                                      new Font("援대┝", 12), Brushes.Brown, 0, 0);              g.Dispose();          }          private void btnUnit\_Click(object sender, EventArgs e){              Graphics g = this.CreateGraphics();              g.PageUnit = GraphicsUnit.Inch;              g.DrawString("湲곕낯 醫뚰몴怨??몄튂) 1?몄튂, 2?몄튂 吏??,                                                      new Font("援대┝", 12), Brushes.Brown, 1, 2);              g.Dispose();          }      }  } |

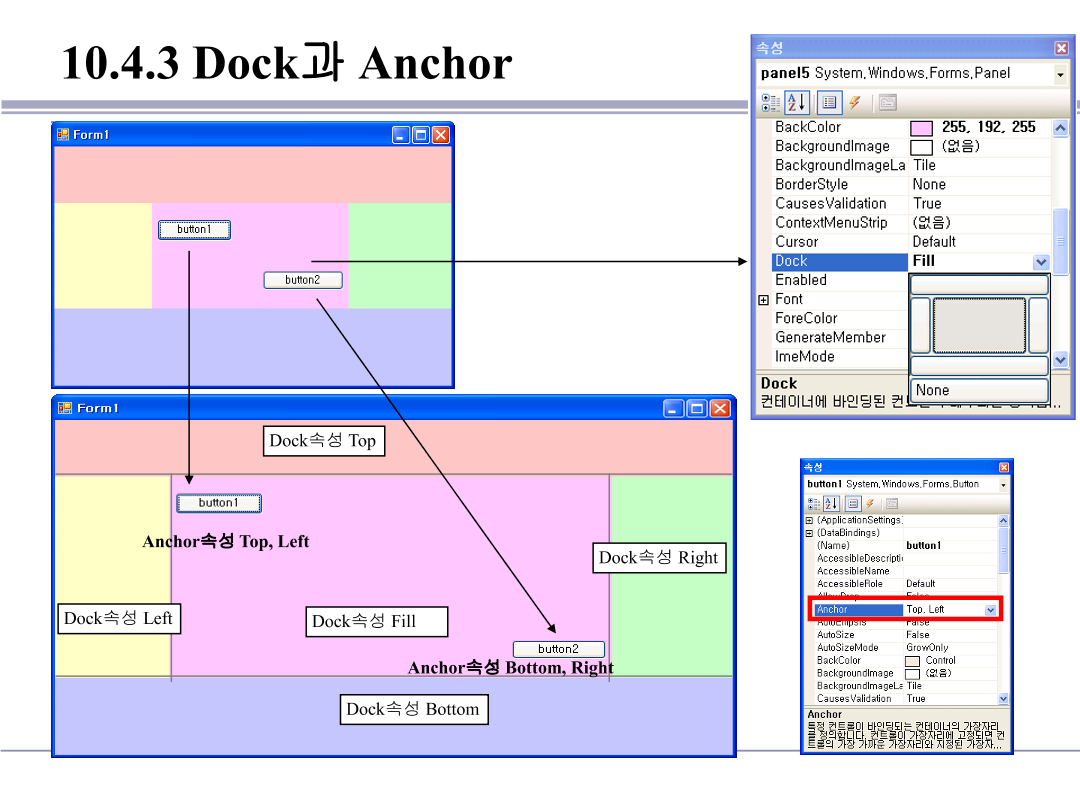
**10.4 컨테이너 컨트롤**  
  
  
다음은 여러분들이 이 절에서 학습할 각 소절들에 대한 제목들이다.  
  
**□  10.4 컨테이너 컨트롤**  
◇ 10.4.1 컨테이너 컨트롤들  
◇ 10.4.2 Panel과 Layout  
◇ 10.4.3 Dock과 Anchor  
◇ 10.4.4 SplitContainer  
  
위의 주제들을 하나씩 학습해 나가면서 각각의 주제에 대한 내용을 익히도록 하자

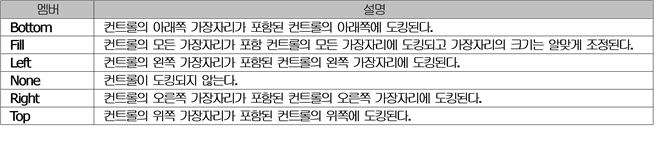


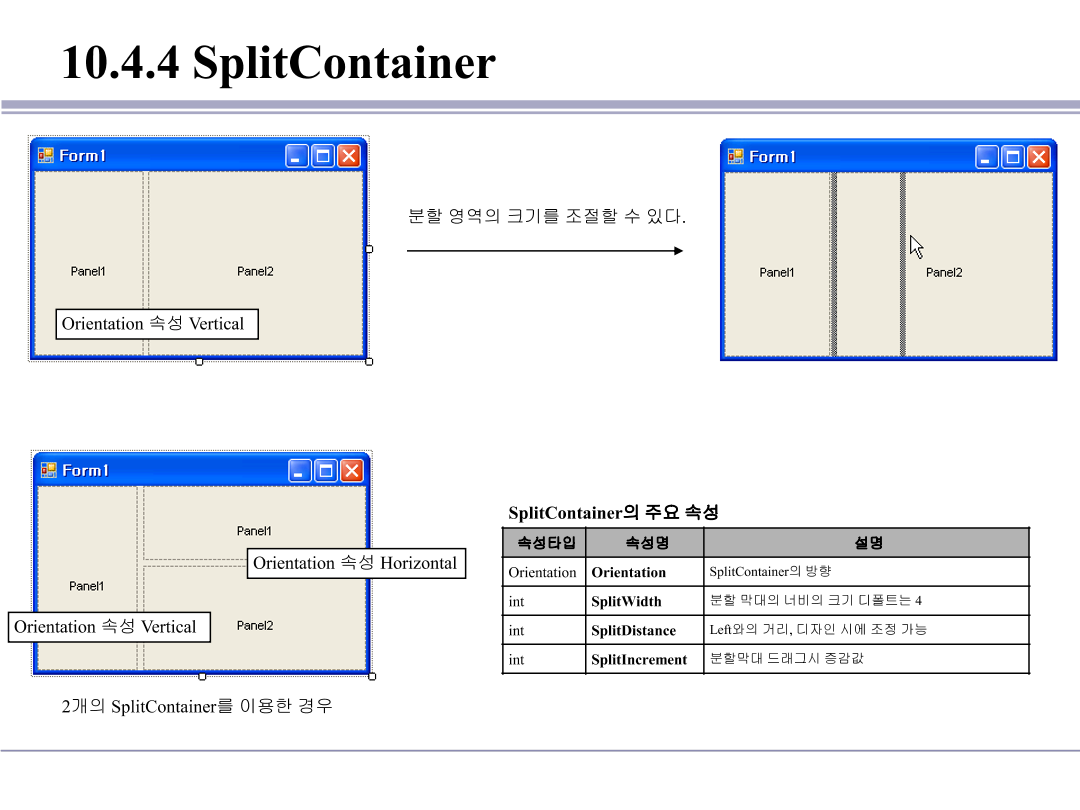
**10.4.1 컨테이너 컨트롤들**  
  
컨테이너 계열의 컴포넌트는 말 그대로 다른 컴포넌트들을 포함할 수 있다. Button, Label, Choice 등의 기본 컴포넌트들 뿐만 아니라, 같은 계열의 컨테이너 컴포넌트들도 포함할 수 있다.  
  
**□ 컨테이너 계열 컴포넌트의 특징**   
◇ 컨테이너 계열의 컴포넌트는 다른 컴포넌트들을 포함할 수 있다.  
  
대표적인 컨테이너 컨트롤들은 다음과 같다.  
  
**□ 컨테이너 컨트롤의 종류**  
◇ Panel  
◇ GroupBox  
◇ TabControl  
◇ TableLayoutPanel  
◇ FlowLayoutPanel  
◇ SplitContainer  
  
컨테이너에서 중요한 것은 묶어서 관리하는 것과 컨트롤의 배치에 있다. 컨테이너를 사용하는 것은 컨트롤들을 묶어서 관리할 수 있다는 장점을 제공한다. 그리고 컨트롤의 디자인적인 배치를 위해서 컨테이너를 사용하기도 한다. 컨테이너와 컨트롤을 조합해서 사용자가 원하는 디자인으로 배치하는 것은 쉬워 보이지만 그렇게 만만한 작업이 아니다.   
  
**□ 컨테이너 컨트롤의 핵심**  
◇ 컨트롤들을 묶어서 관리할 수 있다.  
◇ 컨트롤을 원하는 형태대로 배치한다.  
  
컨테이너에 컨트롤을 집어넣는 것은 쉽다. 단순히 컨테이너에 끌어다 놓기만 하면 된다. 그 이후부터는 컨테이너 단위로 관리할 수 있다.   
  
가장 단순한 형태의 컨테이너가 바로 Panel이다. 이에 반해 GroupBox는 묶어서 관리하면서 테두리와 타이틀을 표시할 수 있다. TabControl은 여러 개의 컨테이너를 Tab 형식으로 관리하기 때문에 하나의 컨테이너에 많은 양의 정보를 담는 목적으로 사용된다.   
  
**□ Panel**  
◇ 가장 단순한 형태의 컨테이너가 바로 Panel이다.   
  
**□ GroupBox**  
◇ 묶어서 관리하면서 테두리와 타이틀을 표시할 수 있다.   
  
**□ TabControl**  
◇ 하나의 컨테이너에 많은 양의 정보를 담는 목적으로 많이 사용한다.   
  
다음으로 배치에 관련된 컨테이너 컨트롤인 FlowLayoutPanel과 TableLayoutPanel에 대해서 알아보자.



**10.4.2 Panel과 Layout**  
  
Panel 중에서도 배치하는 방법이 미리 정해져 있는 패널이 있다. Panel은 기본적으로 자유 배치이지만, FlowLayoutPanel의 경우 컨트롤을 삽입하면 삽입하는 순서대로 가로 또는 세로로 배치되는 컨테이너이며, TableLayoutPanel은 테이블의 형태대로 컴포넌트를 배치할 수 있는 컨테이너이다.  
  
**□ FlowLayoutPanel**   
◇ 컨트롤을 삽입하면 삽입하는 순서대로 가로 또는 세로로 배치되는 컨테이너  
  
**□ TableLayoutPanel**  
◇ 테이블의 형태대로 컴포넌트를 배치할 수 있는 컨테이너  
  
TableLayoutPanel의 행과 열을 편집하기 위해서는 먼저 TableLayoutPanel을 선택한 후 컨트롤의 오른쪽 상단에 나타나는 화살표를 클릭하면 TableLayoutPanel 작업창을 만날 수 있다. 여기서 열과 행 추가 및 삭제를 할 수 있으며, [행 및 열 편집] 메뉴를 클릭하면 행과 열의 크기를 조정할 수 있다.  
  
**□ TableLayoutPanel 작업창**  
◇ [행 및 열 편집] 메뉴를 클릭하면 행과 열의 크기를 조정할 수 있다.  
  
앞에서 배운 Panel, GroupBox, TabControl은 단순 컨테이너적인 성격이 강하다. 여기서의 FlowLayoutPanel과 TableLayoutPanel의 경우에는 컴포넌트의 배치의 성격이 강하다. 이들 컨테이너 컨트롤을 잘 이용하면 대부분의 디자인을 만들 수 있다. 필자 또한 특별한 디자인 기법을 사용하는 것이 아니라 이 컨테이너를 이용해서 디자인하는 것이 전부이다.



**10.4.3 Dock과 Anchor**  
  
컨테이너를 디자인하는 방법 중 Dock과 Anchor라는 것이 있다. Dock이라는 것은 컨테이너 컨트롤의 영역을 Top, Bottom, Left, Right, Fill로 나누고 컴포넌트를 각각의 영역에 꽉 차게 배치하는 것을 말한다. 여기서 Fill은 가운데를 의미한다.   
  
**□ Dock**  
◇ 컨테이너 컨트롤의 영역을 Top, Bottom, Left, Right, Fill 나누고 컴포넌트를 각각의 영역에 꽉 차게 배치하는 것을 말한다.  
  
디폴트 Dock은 None이며 이 Dock의 값은 DockStyle 열거형의 값으로 설정할 수 있다. 그리고 이 Dock 속성은 삽입할 컴포넌트에 지정하면 된다.  
  
  
[표 10\_11] DockStyle 열거형 멤버  
  
Dock으로 설정되었을 경우 컨테이너의 크기가 변경되면 함께 크기가 변경되는 장점이 있다. 사실 Dock은 일반적인 컨트롤을 배치하기 보다는 주로 컨테이너를 배치하는 목적으로 사용된다. 컨테이너 내에서 다른 컨테이너를 배치할 때 주로 이용된다.  
  
**□ Dock 속성**  
◇ 컨테이너 내에서 다른 컨테이너를 배치할 때 주로 사용된다.  
  
이에 반해 Anchor 속성은 컨테이너의 크기를 변경했을 때 어느 위치를 기준으로 이동할 것인지가 결정된다. 위의 PPT 그림을 보면 Anchor 속성의 특징을 정확하게 알 수 있다. button1의 경우 Anchor 속성이 Top, Left로 설정되어 있으며, button2의 경우에는 Anchor 속성이 Bottom, Right로 설정되어 있다. 이 상태에서 폼의 크기를 변경시키면 button1과 button2가 Anchor에 따라서 위치가 달라지는 것을 확인할 수 있을 것이다.  
  
**□ Anchor 속성**  
◇ 컨트롤을 배치하는 기준 위치를 Anchor라고 한다.  
  
button1은 Anchor 속성이 Top, Left이기 때문에 크기를 변경해도 위치가 변경되지 않는다. 반면 button2의 경우 크기를 변경하면 Bottom, Right를 기준으로 button2가 이동된다. 보통 Anchor를 생각하지 않고 작업하는 경우가 많다. 이 경우 일반적인 기준은 Top, Left이다. 그리고 디폴트 Anchor 또한 Top, Left이다.

****

**10.4.4 SplitContainer**  
  
마지막으로 소개할 컨테이너 컨트롤은 SplitContainer이다. 이 컨트롤을 이용하면 화면을 분할해서 관리할 수 있다. SplitContainer를 만들면 자동으로 Panel1과 Panel2로 분할된다. 분할 방향을 SplitContainer의 Orientation 속성으로 지정할 수 있으며, 수직 또는 수평의 Orientation 열거형의 값을 가진다.  
  
**□ Orientation 열거형의 멤버**  
◇ Horizontal : 컨트롤의 요소를 가로 레이아웃으로 배치한다.   
◇ Vertical : 컨트롤의 요소를 세로 레이아웃으로 배치한다.  
  
위의 PPT에는 2개의 SplitContainer를 이용해서 폼을 전체 3개의 영역으로 분할하고 있다. 먼저 SplitContainer를 하나 생성한다. 디폴트 Orientation이 Vertical이기 때문에 세로로 만들어진다. 그리고 SplitContainer의 Panel2에 새로운 SplitContainer를 하나 더 만들고 있다. 이 때의 Orientation은 Horizontal로 지정하고 있다. 이렇게 하면 위의 PPT의 그림과 같이 3개의 영역으로 분할된 폼을 얻을 수 있다.  
  
SplitContainer의 가장 큰 장점은 화면을 분할할 수 있다는 것이며, 프로그램이 실행된 후에도 분할 영역의 크기를 변경할 수 있다는 것이다.