

**10.6.1 Novel! : 대화상자**  
  
대화상자란 프로그램에서 사용자와 대화를 하기 위해서 사용하는 폼을 말한다. 일반적인 폼과 별반 다를 것이 없지만 굳이 대화상자라고 하는 것은 바로 대화상자가 가지고 있는 독특한 기능 때문이다. 예를 들어서 프린터 대화상자를 열었다고 가정하자. 이 대화상자는 프린트할 페이지와 매수를 지정하고 [확인]이나 [취소] 버튼을 누르면 된다. 또는 [Enter] 키나 [ESC] 키를 누르면 된다. 대화상자가 닫히고 나면 사용자가 어떤 버튼을 클릭했는지, 어떤 키를 눌렀는지를 알아야 그 다음 작업을 할 수 있다. 바로 이것이 대화상자의 가장 큰 특징이다.  
  
**□ 대화상자**  
◇ Form을 상속받아 대화상자를 만들기 때문에 Form과 다른 것이 없다.  
◇ 대화 상자는 사용자와의 정보교환을 위해서 화면에 나타나며 정보와 결과를 반환하면서 닫힌다.  
  
별것 아닌 것처럼 생각할 수도 있겠지만 대화상자의 주목적은 사용자가 입력한 정보와 결과를 반환하는 목적으로 사용된다. 이것을 한마디로 하면 사용자와 대화를 하기 위해서 사용한다고 볼 수 있다.  
  
대화상자를 만드는 방법은 일반적인 폼을 만드는 방법과 동일하다. 일단 Form을 상속받아 새로운 MyDialog 폼을 하나 만들고 있다.  
  
**▒ 대화상자 만들기**

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

|  |  |
| --- | --- |
| 1  2  3 | class MyDialog : Form{       //...   } |

폼을 띄우는 방식은 모달과 모달리스라는 두 가지 방식이 있다. 우리가 일반적으로 알고 있는 Show() 함수를 이용해서 폼을 띄우는 방식은 모달리스 방식이다.   
  
**▒ 대화상자  모달리스로 띄우기**

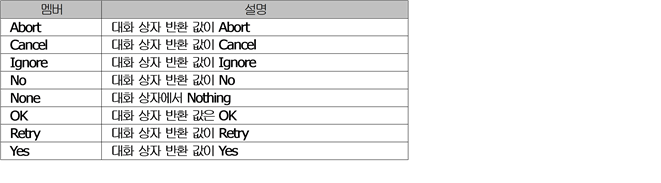
[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3 | MyDialog dlg = new MyDialog();  dlg.Owner = this; //부모 폼  dlg.Show (); |

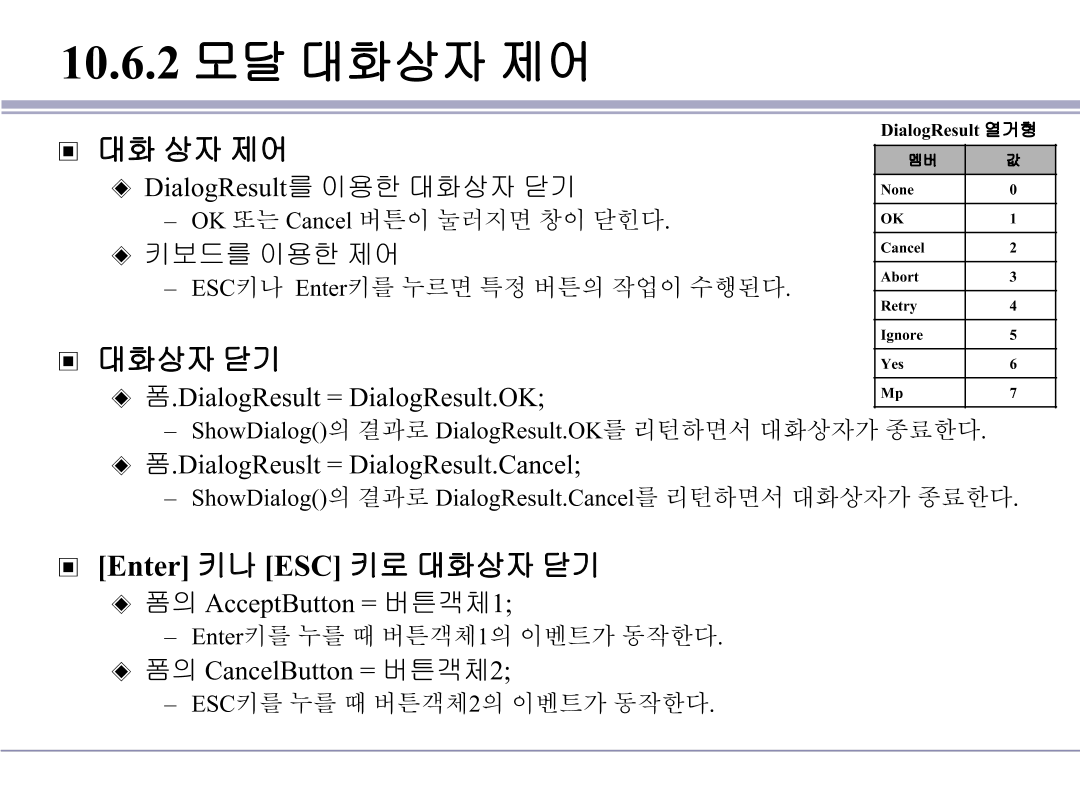
현재 활성화된 폼 이외에 다른 폼을 활성화할 수 있다면 이 방식은 모달리스 방식이다. 반면에 현재 활성화된 창을 닫기 전에는 다른 작업을 할 수 없다면 이 방식은 모달 방식이다.  
  
**□ 모달리스(Modalless)방식**  
◇ 현재 활성화된 폼 이외에 다른 폼을 활성화할 수 있다면 이 방식은 모달리스 방식  
  
**□ 모달(Modal)  방식**  
◇ 현재 활성화된 창을 닫기 전에는 다른 작업을 할 수 없다면 이 방식은 모달 방식이다.  
  
위의 Owner 속성의 경우 자신의 소유폼을 지정하는 용도로 사용된다. 폼이 다른 폼에 소유된 경우 소유자 폼으로 해당 폼을 최소화하거나 닫을 수 있다. 예를 들어 MyChild 폼이 MyOwner 폼에 소유된 상태에서 MyOwner가 닫히거나 최소화되면 MyChild도 함께 닫히거나 최소화된다.   
  
**□ Owner 속성**  
◇ Owner 속성으로 지정된 폼이 닫히거나 최소화되면 현재의 폼도 함께 닫히거나 최소화된다.  
  
Show() 함수는 모달리스 방식으로 폼을 띄우는 반면 ShowDialog() 함수를 이용하면 모달 방식으로 폼을 띄울 수 있다. 모달 방식이 대화상자 방식이라고 생각하는 것이 편하다. 다음은 모달 방식으로 대화상자를 띄우는 예이다.  
  
**▒ 대화상자 모달로 띄우기**

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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | MyDialog dlg = new MyDialog();  DialogResult dr = dlg.ShowDialog();  if(dr == DialogResult.OK){     //...OK 버튼이 눌러졌을 때의 작업  }else{     //...  } |

Show()와 ShowDialog()의 차이점 즉 모달리스와 모달의 차이점은 ShowDialog()의 반환값에서 찾을 수 있다. 위의 ShowDialog()의 경우 DialogResult 열거형을 리턴한다. 이 값이 바로 대화상자에서 어떠한 버튼을 눌렀느냐에 해당한다.  
  
잘 생각해보라 DialogResult 열거형을 배운 적이 있다. 메시지 박스를 띄운 후 메시지 박스가 반환하는 값이 바로 DialogResult 열거형의 값이다.   
  
  
[표 10\_14] DialogResult 열거형  
  
DialogResult의 값에 따라서 사용자의 프로그램이 달라질 것이다. 대화상자의 의미가 넓기 때문에 Show()를 호출해서 사용하는 것도 대화상자라고 할 수 있다. 진정한 대화상자는 ShowDialog()를 이용하는 모달 대화상자이다.

=======================================================================



**10.6.2 모달 대화상자 제어**  
  
대화상자는 OK 또는 Cancel 버튼이 눌러지면 창이 닫힌다. 이것은 버튼으로 제어하는 방식이다. 이 방식 이외에 키보드의 [Enter]와 [ESC] 키를 이용해서 창을 닫을 수도 있다.   
  
**□ 대화 상자 제어**  
◇ DialogResult를 이용한 대화상자 닫기  
― OK 또는 Cancel 버튼이 눌러지면 창이 닫힌다.  
◇ 키보드를 이용한 제어  
― ESC키나  Enter키를 누르면 특정 버튼의 작업이 수행된다.  
  
실제 대화상자를 닫는 효과는 Form의 DialogResult 속성을 이용한다. DialogResult 속성에 DialogResult.OK 값이나 DialogResult.Cancel 값이 할당되면 대화상자가 닫히면서 할당된 DialogResult의 값이 리턴된다.  
  
**□ this.DialogResult = DialogResult.OK;**   
◇ ShowDialog()의 결과로 DialogResult.OK를 리턴하면서 대화상자가 종료한다.  
  
**□ this.DialogResult = DialogResult.Cancel;**  
◇ ShowDialog()의 결과로 DialogResult.Cancel를 리턴하면서 대화상자가 종료한다.  
  
키보드의 [Enter]나 [ESC] 키를 이용할 때에는 버튼과 연결하는 방식을 이용한다. 폼의 AcceptButton 속성에 버튼 객체를 지정해두면, [Enter] 키가 눌러 지는 순간 버튼 객체의 이벤트가 동작하는 방식을 이용한다. [ESC]의 경우에는 CancelButton 속성을 이용하면 된다.  
  
**□ 키보드 [Enter]나 [ESC] 키로 대화상자 닫기**  
◇ this.AcceptButton = 버튼객체1;  
― Enter키를 누를 때 버튼객체1의 이벤트가 동작한다.  
◇ this.CancelButton = 버튼객체2;  
― ESC키를 누를 때 버튼객체2의 이벤트가 동작한다.  
  
다음은 대화상자로 사용할 폼의 소스코드이다. 버튼이 눌러지는 순간 DialogResult.OK와 DialogResult.Cancel을 설정하고 있다. 그리고 AcceptButton과 CancelButton 또한 설정하고 있다.

**§ chap10\DialogProject\MyDialog.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;    namespace DialogProject{      public partial class MyDialog : Form{          public MyDialog(){              InitializeComponent();              this.AcceptButton = btnOK;              this.CancelButton = btnCancel;          }          private void btnExit\_Click(object sender, EventArgs e){              this.Close();          }          private void btnOK\_Click(object sender, EventArgs e){              this.DialogResult = DialogResult.OK;          }          private void btnCancel\_Click(object sender, EventArgs e){              this.DialogResult = DialogResult.Cancel;          }      }  } |

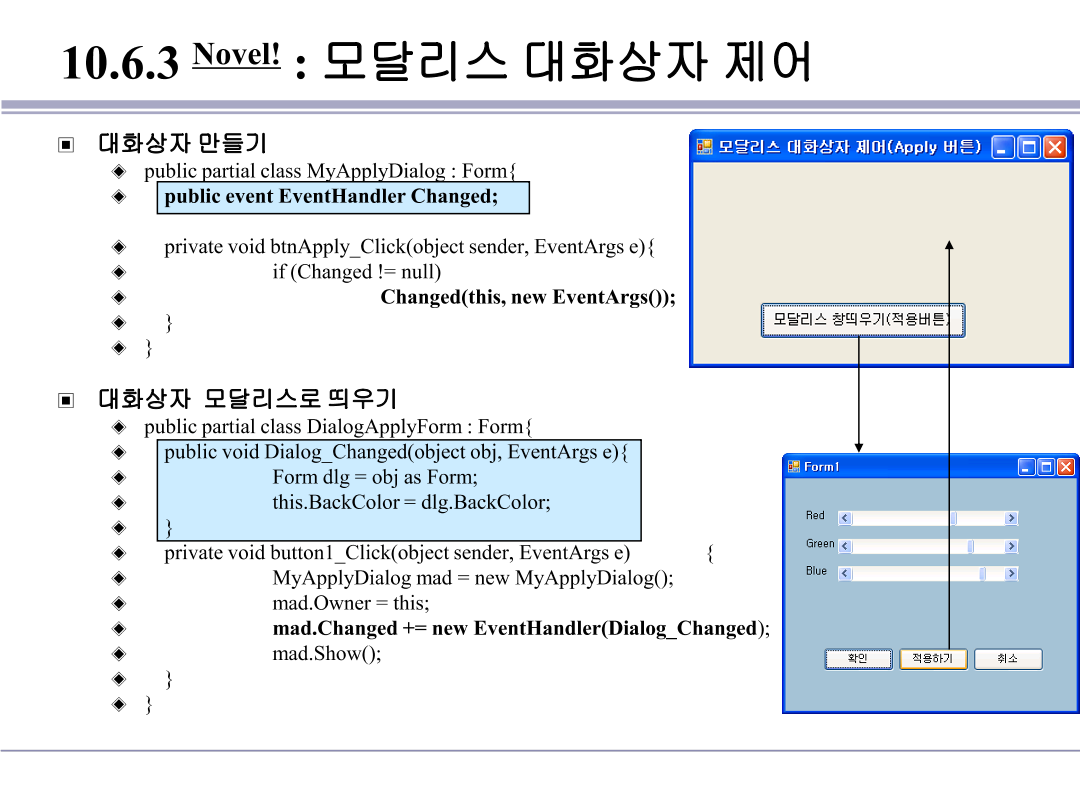
이 대화상자를 이용해서 모달과 모달리스로 창을 띄우는 예제는 다음과 같다.

**§ chap10\DialogProject\DialogForm.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 | using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Text;  using System.Windows.Forms;    namespace DialogProject{      public partial class DialogForm : Form{          public DialogForm(){              InitializeComponent();          }          private void button1\_Click(object sender, EventArgs e){              MyDialog md = new MyDialog();              md.Owner = this;              md.Show();          }          private void button2\_Click(object sender, EventArgs e){              MyDialog md = new MyDialog();              DialogResult dResult = md.ShowDialog();              if (dResult == DialogResult.OK){                  MessageBox.Show("OK 버튼입니다.");              }else{                  MessageBox.Show("Cancel 버튼입니다.");              }          }      }  } |

모달리스 형식으로 대화상자를 띄울 때에는 DialogResult와 AcceptButton 그리고 CancelButton이 동작하지 않는 것을 볼 수 있다. 그리고 모달 형식으로 띄울 때 DialogResult 속성으로 닫지 않고 종료 버튼이나 Close() 함수로 닫아도 DialogResult.Cancel을 리턴하는 것도 기억해두기 바란다.



**10.6.3 Novel! : 모달리스 대화상자 제어**  
  
모달리스 대화상자나 모달 대화상자를 사용할 때 창을 닫지 않고 현재 설정된 값들을 바로 적용하는 경우가 있다. 이 적용 버튼을 구현하는 프로그램 기법에 대해서 알아보자.  
  
적용 버튼을 구현하기 위해서 이벤트를 사용한다. 여기에서 소개되는 이벤트의 사용법만 알아도 이벤트를 다 아는 것이나 마찬가지이다. 구현 방법은 너무나 쉽다. Form을 상속받아서 이벤트를 하나 선언한다. 그리고 적용버튼에 해당 이벤트를 호출해주기만 하면 된다.  
  
**▒ 대화상자 만들기**

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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | public partial class MyApplyDialog : Form{     public event EventHandler Changed;       private void btnApply\_Click(object sender, EventArgs e){        if (Changed != null)           Changed(this, new EventArgs());     }  } |

이벤트를 public으로 선언한 것은 외부에서 이벤트를 등록할 수 있도록 하기 위해서이다.   
  
**☞ 참고**

|  |
| --- |
| 여기서는 구현을 간단히 하기 위해서 EventArgs 객체를 생성하고 있지만, EventArgs를 상속받아 여러분이 원하는 형태의 클래스로 만들어서 이벤트의 정보인 EventArgs를 첨부해도 된다. |

다음으로 외부에서 이벤트를 등록하고 모달리스나 모달로 창을 띄우기만 하면 된다. 이벤트를 등록하고 창을 띄우는 예는 다음과 같다.  
  
**▒ 대화상자 모달리스로 띄우기**

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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | public partial class DialogApplyForm : Form{     public void Dialog\_Changed(object obj, EventArgs e){        Form dlg = obj as Form;        this.BackColor = dlg.BackColor;     }     private void button1\_Click(object sender, EventArgs e)   {        MyApplyDialog mad = new MyApplyDialog();        mad.Owner = this;        mad.Changed += new EventHandler(Dialog\_Changed);        mad.Show();     }  } |

Dialog\_Changed() 이벤트 함수는 대화상자 내에서 적용버튼이 클릭되는 순간 호출되는 이벤트 핸들러이다. 물론 Dialog\_Changed() 이벤트 핸들러는 MyApplyDialog에 등록해야만 한다. 위의 대화상자를 띄우는 부분에 Changed 이벤트를 등록하는 부분을 확인할 수 있다.  
  
MyApplyDialog 대화상자 내의 적용 버튼에서 Changed 이벤트가 등록되어 있다면 해당 이벤트를 호출하게 된다. 이 단순한 기법이 일반적인 윈도우 프로그래밍의 이벤트 호출 방식이다. 모달리스 대화상자 제어를 구현하는 전체 예는 다음과 같다.

**§ chap10\DialogApplyProject\MyApplyDialog.cs**

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

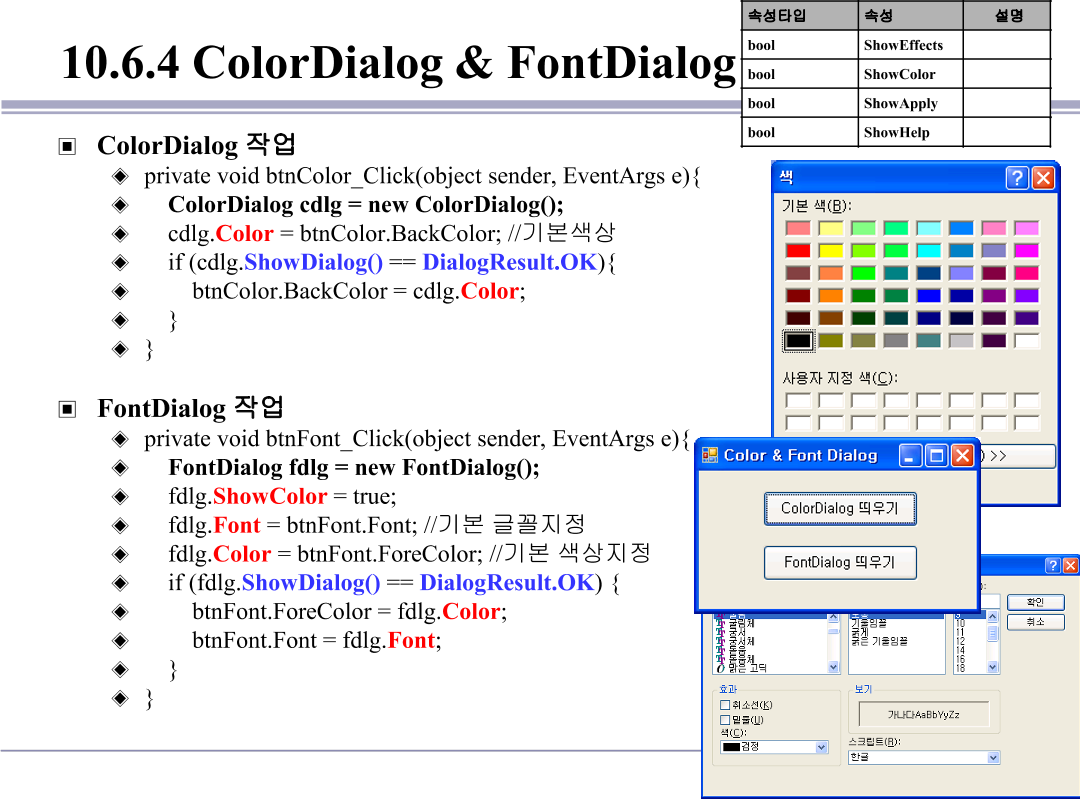
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Text;  using System.Windows.Forms;    namespace DialogApplyProject{      public partial class MyApplyDialog : Form{          public event EventHandler Changed;            public MyApplyDialog(){              InitializeComponent();              this.btnApply.Enabled = false;          }          private void Color\_Scroll(object sender, ScrollEventArgs e){              this.btnApply.Enabled = true;              this.BackColor = Color.FromArgb(this.hsbRed.Value,                                  this.hsbGreen.Value, this.hsbBlue.Value);          }          private void btnCancel\_Click(object sender, EventArgs e){              this.Close();          }          private void btnOK\_Click(object sender, EventArgs e){              if (Changed != null)                  if(this.btnApply.Enabled == true)                      Changed(this, new EventArgs());              this.Close();          }          private void btnApply\_Click(object sender, EventArgs e){              if (Changed != null)                  Changed(this, new EventArgs());          }      }  } |

**§ chap10\DialogApplyProject\DialogApplyForm.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 DialogApplyProject{      public partial class DialogApplyForm : Form{          public DialogApplyForm(){              InitializeComponent();          }          public void Dialog\_Changed(object obj, EventArgs e){              Form dlg = obj as Form;              this.BackColor = dlg.BackColor;          }          private void button1\_Click(object sender, EventArgs e){              MyApplyDialog mad = new MyApplyDialog();              mad.Owner = this;              mad.Changed += new EventHandler(Dialog\_Changed);              mad.Show();          }      }  } |

단순히 적용 버튼을 구현하는 예라고 생각하지 말고 이벤트를 직접 구현하는 가장 일반적인 예라고 생각하라.



**10.6.4 ColorDialog & FontDialog**  
  
윈도우 환경에서 사용되는 일반적인 대화상자가 있는데 이를 공통 대화상자라고 한다. 대표적인 공통 대화상자는 바로 색상 선택, 폰트 선택, 파일 열기, 저장하기, 프린터 대화상자이다.   
  
**□ 공통 대화상자**  
◇ 색상 대화상자  
◇ 폰트 대화상자  
◇ 파일 열기 대화상자  
◇ 파일 저장 대화상자  
◇ 프린터 대화상자  
  
**☞ 참고**

|  |
| --- |
| 공통 대화상자들은 모두 도구상자에 포함되어 있기 때문에 도구상자에서 끌어다 사용해도 된다. 이 책에서는 객체를 선언하는 방식으로 대화상자를 생성할 것이다. 도구상자를 이용해서 객체를 생성하면 속성창에서 속성을 수정할 수 있는 장점이 있지만 수작업으로 객체를 생성하면 속성창은 이용할 수 없다. |

이들 대화상자의 사용법은 동일하다. 대화상자를 띄우고, 눌러진 버튼을 확인하고, 정보를 얻어내는 절차에 의해서 대화상자를 사용한다.  
  
**▒ 공통 대화상자의 사용법**

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

|  |  |
| --- | --- |
| 1  2  3 | 1. 대화상자를 띄운다.  2. 대화상자가 닫힐 때 눌러진 버튼을 확인한다.  3. 대화상자로부터 정보를 얻어낸다. |

**▣ ColorDialog**  
  
ColorDialog를 예를 들어보자. ColorDialog를 먼저 띄워야 한다. 사용자가 색상을 선택한 후 버튼을 클릭하면 대화상자가 리턴하는 DialogResult 값을 확인해야 한다. OK가 눌러지면 ColorDialog 대화상자로부터 선택된 색상만 얻어내면 된다. 이 때는 ColorDialog의 Color 속성을 이용해서 색상 값을 얻어내면 된다.   
  
**▒ ColorDialog 작업**

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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | private void btnColor\_Click(object sender, EventArgs e){       ColorDialog cdlg = new ColorDialog();       cdlg.Color = btnColor.ForeColor; //기본색상       if (cdlg.ShowDialog() == DialogResult.OK){           btnColor.ForeColor = cdlg.Color;       }   } |

ColorDialog를 띄우기 전에 ColorDialog의 Color 속성을 버튼의 글자색으로 설정한 후 띄우고 있다. 그리고 DialogResult.OK를 리턴하면 선택된 색상을 버튼의 글자색으로 설정하는 예이다.  
  
**▣ FontDialog**  
  
FontDialog 또한 ColorDialog와 다르지 않다. 단지 다른 것은 FontDialog의 경우 Color와 Font 두가지 값을 얻어낼 수 있다는 것이 다르다. 다음은 FontDialog를 띄운 후 Color와 Font를 얻어내는 예이다.  
  
**▒ FontDialog 작업**

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

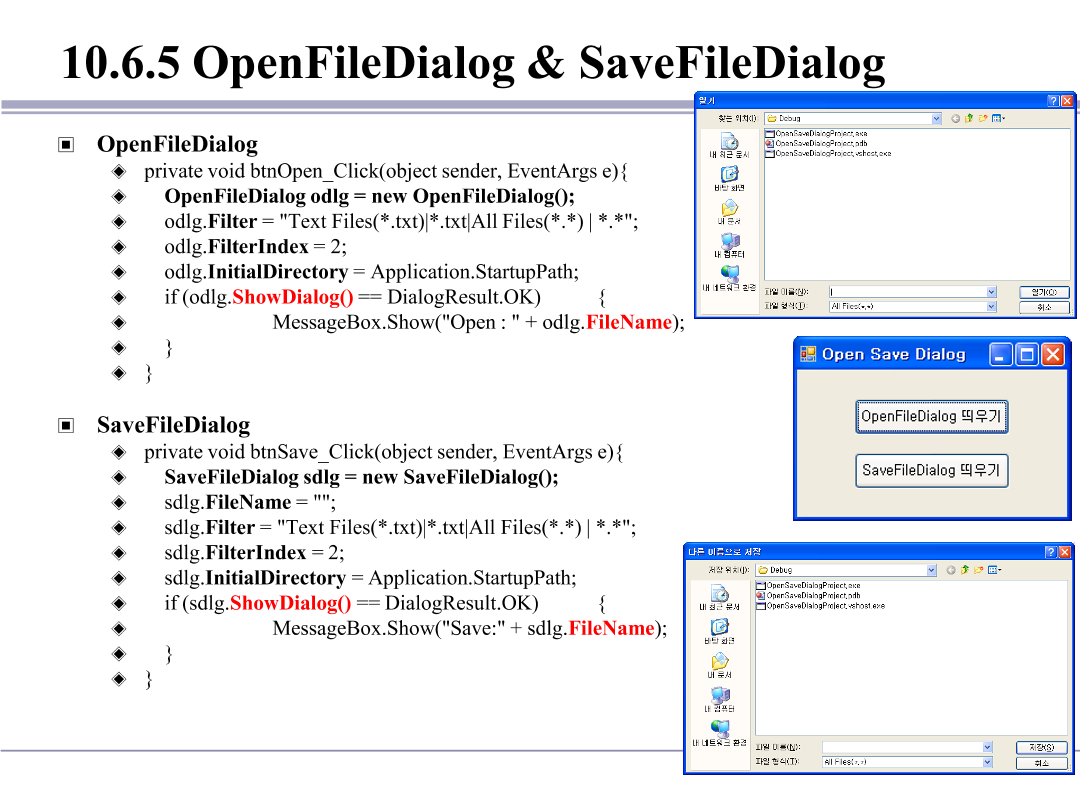
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | private void btnFont\_Click(object sender, EventArgs e){       FontDialog fdlg = new FontDialog();       fdlg.ShowColor = true;       fdlg.Font = btnFont.Font; //기본 글꼴지정       fdlg.Color = btnFont.ForeColor; //기본 색상지정       if (fdlg.ShowDialog() == DialogResult.OK) {           btnFont.ForeColor = fdlg.Color;           btnFont.Font = fdlg.Font;       }   } |

FontDialog 객체를 생성한 후 ShowColor를 true로 설정하고 있다. 이 속성이 true로 설정되면 FontDialog에서 색상을 선택할 수 있다.  
  
**□ FontDialog의 ShowColor**  
◇ 이 속성이 true로 설정되면 FontDialog에서 색상을 선택할 수 있다.  
  
그리고 FontDialog의 디폴트 Color와 Font를 버튼의 ForeColor와 Font로 지정한 후 ShowDialog()로 띄우고 있다. DialogResult.OK를 리턴하면 Color와 Font를 얻어내서 버튼의 ForeColor와 Font로 지정하는 예이다. 이렇게 하면 사용자가 설정한 색상과 폰트로 버튼의 텍스트가 변경된다.

**§ chap10\ColorFontDialogProject\ColorFontForm.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 | using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Text;  using System.Windows.Forms;    namespace ColorFontDialogProject{      public partial class ColorFontForm : Form{          public ColorFontForm(){              InitializeComponent();          }          private void btnColor\_Click(object sender, EventArgs e){              ColorDialog cdlg = new ColorDialog();              cdlg.Color = this.btnColor.ForeColor; //기본색상              if (cdlg.ShowDialog() == DialogResult.OK){                  this.btnColor.ForeColor = cdlg.Color;              }          }          private void btnFont\_Click(object sender, EventArgs e){              FontDialog fdlg = new FontDialog();              fdlg.ShowColor = true;              fdlg.Font = btnFont.Font; //기본 글꼴지정              fdlg.Color = btnFont.ForeColor; //기본 색상지정                if (fdlg.ShowDialog() == DialogResult.OK){                  btnFont.ForeColor = fdlg.Color;                  btnFont.Font = fdlg.Font;              }          }      }  } |



**10.6.5 OpenFileDialog & SaveFileDialog**  
  
대화상자에서 제일 중요한 것은 대화상자에서 리턴되는 DialogResult 값이며, 대화상자에서 설정된 정보이다. 이것을 위해서 우리는 대화상자를 사용하는 것이다. OpenFileDialog와 SaveFileDailog도 DialogResult 값을 리턴하며 대화상자의 정보는 사용자가 선택한 파일명이 된다.   
  
**□ OpenFileDialog와 SaveFileDailog**  
◇ DialogResult 값을 리턴하며 대화상자의 정보는 사용자가 선택한 파일명이다.  
  
**▣ OpenFileDialog**  
  
OpenFileDialog는 파일을 열기 위한 대화상자를 말한다. OpenFileDialog을 생성한 후 파일 형식을 위한 Filter와 FilterIndex 그리고 초기 디렉터리 지정을 위해서 InitialDirectory를 이용하고 있다.  
  
**▒ OpenFileDialog의 속성 지정**

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

|  |  |
| --- | --- |
| 1  2  3  4 | OpenFileDialog odlg = new OpenFileDialog();  odlg.Filter = "Text Files(\*.txt)|\*.txt|All Files(\*.\*) | \*.\*";  odlg.FilterIndex = 2;  odlg.InitialDirectory = Application.StartupPath; |

Filter 속성은 두개를 쌍으로 Text Files(\*.txt)|\*.txt에서 Text Files(\*.txt)는 보여주는 텍스트이며, \*.txt는 실제 적용되는 확장자이다. 마찬가지로 All Files(\*.\*) | \*.\*에서 All Files(\*.\*)는 파일형식에 보여지는 텍스트이며, \*.\*은 실제 적용되는 확장자이다. 그리고 FilterIndex를 2로 설정했기 때문에 OpenFileDialog를 띄웠을 때 All Files(\*.\*)가 기본 파일 형식으로 설정된다.  
  
파일을 띄운 후에는 파일을 선택하고 OK 버튼을 누르게 되면 OpenFileDialog의 FileName 속성을 이용해서 Open할 파일명을 얻을 수 있다.  
  
**▒ OpenFileDialog에서 선택된 파일명 얻어내기**

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

|  |  |
| --- | --- |
| 1  2  3  4  5 | OpenFileDialog odlg = new OpenFileDialog();  //....  if (odlg.ShowDialog() == DialogResult.OK)   {     MessageBox.Show("Open : " + odlg.FileName);  } |

**▣ SaveFileDialog**  
  
SaveFileDialog는 파일을 저장할 때 사용하는 파일 저장 대화상자이다. 이 대화상자를 이용해서 저장할 새로운 파일명이나 기존의 파일명을 얻을 수 있다.   
  
**▒ SaveFileDialog로 저장할 파일명 또는 기존의 파일명 얻어 내기**

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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | SaveFileDialog sdlg = new SaveFileDialog();  sdlg.FileName = "";  sdlg.InitialDirectory = Application.StartupPath;  sdlg.Filter = "Text Files(\*.txt)|\*.txt|All Files(\*.\*) | \*.\*";  sdlg.FilterIndex = 2;  if (sdlg.ShowDialog() == DialogResult.OK)   {     MessageBox.Show("Save:" + sdlg.FileName);  } |

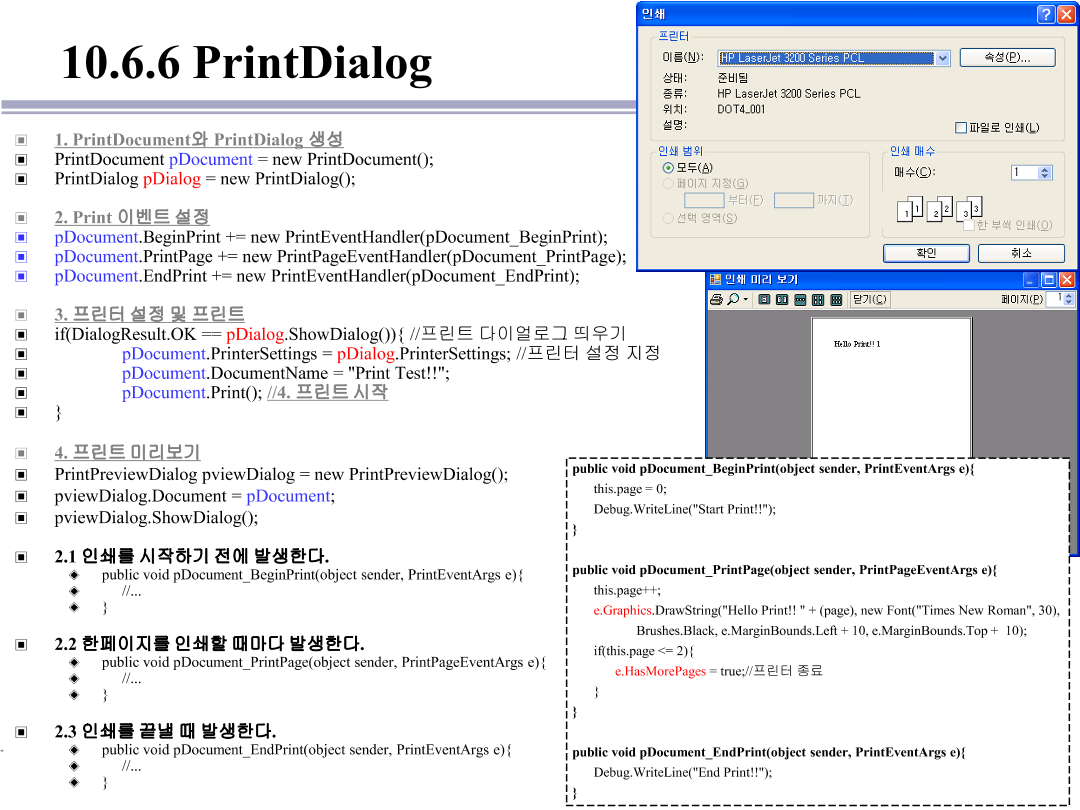
OpenFileDialog와 SaveFileDialog로부터 얻은 파일명을 이용해서 Open이나 Save을 위한 코드를 작성하면 된다.

**§ chap10\OpenSaveDialogProject\OpenSaveForm.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 OpenSaveDialogProject{      public partial class OpenSaveForm : Form{          public OpenSaveForm(){              InitializeComponent();          }          private void btnOpen\_Click(object sender, EventArgs e){              OpenFileDialog odlg = new OpenFileDialog();              odlg.Filter = "Text Files(\*.txt)|\*.txt|All Files(\*.\*) | \*.\*";              odlg.FilterIndex = 2;              odlg.InitialDirectory = Application.StartupPath;              if (odlg.ShowDialog() == DialogResult.OK){                  MessageBox.Show("Open : " + odlg.FileName);              }          }          private void btnSave\_Click(object sender, EventArgs e){              SaveFileDialog sdlg = new SaveFileDialog();              sdlg.FileName = "";              sdlg.InitialDirectory = Application.StartupPath;              sdlg.Filter = "Text Files(\*.txt)|\*.txt|All Files(\*.\*) | \*.\*";              sdlg.FilterIndex = 0;              if (sdlg.ShowDialog() == DialogResult.OK){                  MessageBox.Show("Save:" + sdlg.FileName);              }          }      }  } |

대부분의 대화상자의 패턴은 ColorDialog, FontDialog, OpenFileDialog, SaveFileDialog의 패턴을 벗어나지 않는다. 대화상자를 만났을 때에는 리턴되는 값과 대화상자가 넘겨주는 값을 확인해서 사용하면 된다.



**10.6.6 PrintDialog**  
  
프린트를 하기 위해서는 PrintDocument와 PrintDialog 클래스가 필요하다. PrintDocument는 프린트할 내용을 보관하고 있으면서 한 페이지씩 프린트하는 역할을 담당한다. 그리고 PrintDialog는 프린트하기 위한 다양한 정보를 얻기 위한 대화상자이다. 이 대화상자를 이용해서 프린터의 종류, 페이지 레이아웃, 출력할 페이지 수 등 프린트에 관련된 다양한 정보를 얻을 수 있다.  
  
**□ PrintDocument**  
◇ 프린트할 내용을 보관하고 있으면서 한 페이지씩 프린트하는 역할을 담당한다.  
  
**□ PrintDialog**  
◇ 프린트하기 위한 다양한 정보를 얻기 위한 대화상자이다.  
  
전체 프린트를 하기 위한 절차는 다음과 같다.  
  
**▒ 프린터 절차**

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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | 1. PrintDocument와 PrintDialog 객체 생성  2. PrintDocument의 Print 이벤트 설정  3. PrintDialog을 띄운 후 PrintDocument의 PrinterSettings 속성 설정  4. PrintDocument의 Print() 호출  5. PrintDocument의 BeginPrint 이벤트 발생  6. PrintDocument의 PrintPage 이벤트 발생  7. PrintPage에서 리턴하는 결과에 따라 PrintPage 이벤트 다시 발생(여러 페이지 프린터)  8. PrintDocument의 EndPrint 이벤트 발생 |

**▣ 1단계 : PrintDocument와 PrintDialog 객체 생성**   
  
제일 먼저 PrintDocument와 PrintDialog 객체를 생성해야 한다. 디폴트 생성자를 이용해서 다음과 같이 객체를 생성하면 된다.  
  
**▒ PrintDocument와 PrintDialog 생성**

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

|  |  |
| --- | --- |
| 1  2 | PrintDocument pDocument = new PrintDocument();  PrintDialog pDialog = new PrintDialog(); |

**▣ 2단계 PrintDocument의 Print 이벤트 설정**  
  
PrintDocument는 프린트할 내용을 담고 있으면서 직접적으로 프린트할 내용을 제어하는 클래스이다. 프린트를 제어하기 위해서는 BeginPrint, PrintPage, EndPrint 이벤트를 설정하면 된다.  
  
**▒ Print 이벤트 설정**

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

|  |  |
| --- | --- |
| 1  2  3 | pDocument.BeginPrint += new PrintEventHandler(pDocument\_BeginPrint);  pDocument.PrintPage += new PrintPageEventHandler(pDocument\_PrintPage);  pDocument.EndPrint += new PrintEventHandler(pDocument\_EndPrint); |

BeginPrint 이벤트에서는 프린트를 시작하기 전에 필요한 작업을 넣어두면 된다. 여기서는 프린트할 페이지를 의미하는 page 변수를 0으로 초기화하고 있다.  
  
**▒ BeginPrint 이벤트 핸들러**

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

|  |  |
| --- | --- |
| 1  2  3  4 | public void pDocument\_BeginPrint(object sender, PrintEventArgs e){     this.page = 0;     Debug.WriteLine("Start Print!!");  } |

EndPrint 이벤트에는 프린트를 끝낼 때 필요한 작업을 넣어두면 된다. 여기서는 단순히 "End Print!!"라는 문자열만 출력하고 있다.  
  
**▒ EndPrint 이벤트 핸들러**

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

|  |  |
| --- | --- |
| 1  2  3 | public void pDocument\_EndPrint(object sender, PrintEventArgs e){     Debug.WriteLine("End Print!!");  } |

실제 중요한 것은 PrintPage 이벤트이다. 하나의 페이지가 프린트될 때마다 발생하는 이벤트이기 때문에 여기서 프린트할 내용을 기입하면 된다.   
  
**▒ PrintPage 이벤트 핸들러**

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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | public void pDocument\_PrintPage(object sender, PrintPageEventArgs e){     this.page++;     e.Graphics.DrawString("Hello Print!! " + (page), new Font("Times New Roman", 30),           Brushes.Black, e.MarginBounds.Left + 10, e.MarginBounds.Top +  10);     if(this.page <= 2){        e.HasMorePages = true;//프린트 종료     }  } |

PrintPage 이벤트가 발생할 때 page 변수에 1씩 증가시켜서 현재의 페이지를 계산하고 있다. 그리고 PrintPageEventArgs의 Graphics를 이용해서 프린트할 내용을 그리고 있다. Graphics로 그리는 작업은 화면에 그림을 그리는 작업과 동일한 방식으로 작업하면 된다.  
  
**□ PrintPageEventArgs의 Graphics**  
◇ Graphics로 그리는 작업은 화면에 그림을 그리는 작업과 동일한 방식으로 작업하면 된다.  
  
페이지 레이아웃에서 마진(Margin) 부분은 PrintPageEventArgs의 MarginBounds를 이용해서 Left와 Top으로 계산하고 있다.   
  
**□ 페이지 레이아웃에서 마진(Margin)**  
◇ PrintPageEventArgs의 MarginBounds를 이용해서 Left와 Top으로 계산  
  
PrintPage에서 가장 중요한 것은 PrintPageEventArgs의 HasMorePages의 설정이다. 이 값을 true로 설정하면 더 이상 프린트할 페이지가 없다는 것을 의미한다. 즉 이 값에 따라서 다음 페이지의 인쇄여부가 결정되는 것이다.  
  
**□ PrintPageEventArgs의 HasMorePages**  
◇ HasMorePages의 값에 따라서 다음 페이지의 인쇄여부가 결정된다.  
  
**▣ 3단계 프린터 설정 및 프린트**  
  
PrintDocument가 준비되었다면, PrintDialog를 띄워서 PrinterSettings 값을 얻어내기만 하면 된다. PrintDialog에 세팅된 모든 정보가 PrinterSettings에 들어 있기 때문에 PrintDialog가 DialogResult.OK 값을 리턴하면 PrintDialog의 PrinterSettings 값을 PrintDocument의 PrinterSettings 속성에 설정만 해주면 된다.  
  
**▒ 프린터 설정 및 프린트**

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

|  |  |
| --- | --- |
| 1  2  3  4  5 | if(DialogResult.OK == pDialog.ShowDialog()){ //프린트 다이얼로그 띄우기     pDocument.PrinterSettings = pDialog.PrinterSettings; //프린터 설정 지정     pDocument.DocumentName = "Print Test!!";     pDocument.Print(); //4. 프린트 시작  } |

프린터 설정이 완료되었다면 PrintDocument의 Print() 함수만 호출하면 프린트할 내용이 프린터로 보내져 프린트가 될 것이다.  
  
**▣ 4단계 PrintPreviewDialog을 이용한 미리보기**  
  
PrintPreviewDialog 객체를 생성한 후 PrintDocument 객체만 지정하면 프린트할 내용을 미리 볼 수 있다.   
  
**▒ 프린트 미리보기**

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

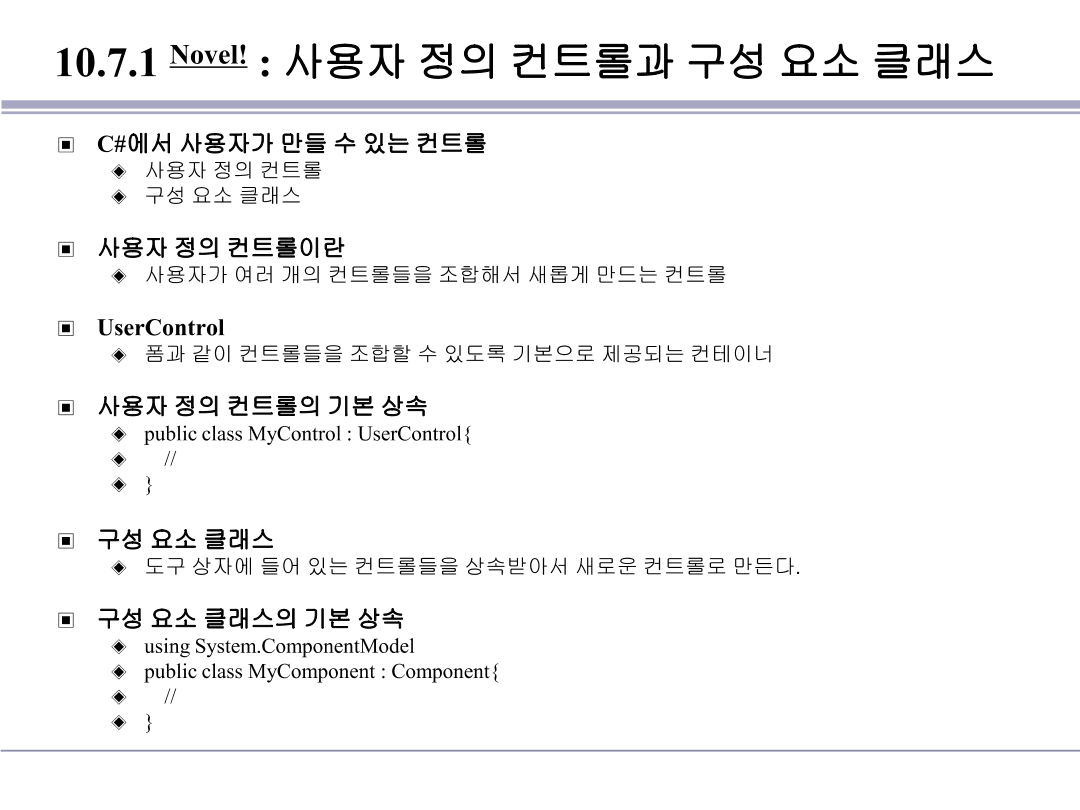
|  |  |
| --- | --- |
| 1  2  3 | PrintPreviewDialog pviewDialog = new PrintPreviewDialog();  pviewDialog.Document = this.pDocument;  pviewDialog.ShowDialog(); |

PrintPreviewDialog를 이용하면 대화상자만으로 프린트 미리보기를 쉽게 구현할 수 있다.

**§ chap10\PrintProject\PrintForm.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  47  48  49  50  51  52  53  54  55 | 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.Printing;  using System.Diagnostics;    namespace PrintProject{      public partial class PrintForm : Form{          private int page = 0;          //1. PrintDocument와 PrintDialog 생성          private PrintDocument pDocument = new PrintDocument();          private PrintDialog pDialog = new PrintDialog();            public PrintForm(){              InitializeComponent();              //2. Print 이벤트 설정              pDocument.BeginPrint += new PrintEventHandler(pDocument\_BeginPrint);              pDocument.PrintPage += new PrintPageEventHandler(pDocument\_PrintPage);              pDocument.EndPrint += new PrintEventHandler(pDocument\_EndPrint);          }          private void btnPrint\_Click(object sender, EventArgs e){              //3. 프린터 설정 및 프린트              if(DialogResult.OK == pDialog.ShowDialog()){ //프린트 다이얼로그 띄우기                  pDocument.PrinterSettings = pDialog.PrinterSettings; //프린터 설정 지정                  pDocument.DocumentName = "Print Test!!";                  pDocument.Print(); //4. 프린트 시작              }          }          private void btnPrintPreview\_Click(object sender, EventArgs e){              PrintPreviewDialog pviewDialog = new PrintPreviewDialog();              pviewDialog.Document = this.pDocument;              pviewDialog.ShowDialog();          }          public void pDocument\_BeginPrint(object sender, PrintEventArgs e){              this.page = 0;              Debug.WriteLine("Start Print!!");          }          public void pDocument\_PrintPage(object sender, PrintPageEventArgs e){              this.page++;              e.Graphics.DrawString("Hello Print!! " + (page),                      new Font("Times New Roman", 30),                      Brushes.Black, e.MarginBounds.Left + 10, e.MarginBounds.Top + 10);              if (this.page < 3){//3페이지까지 출력                  e.HasMorePages = true;//프린터 종료              }          }          public void pDocument\_EndPrint(object sender, PrintEventArgs e){              Debug.WriteLine("End Print!!");          }      }  } |

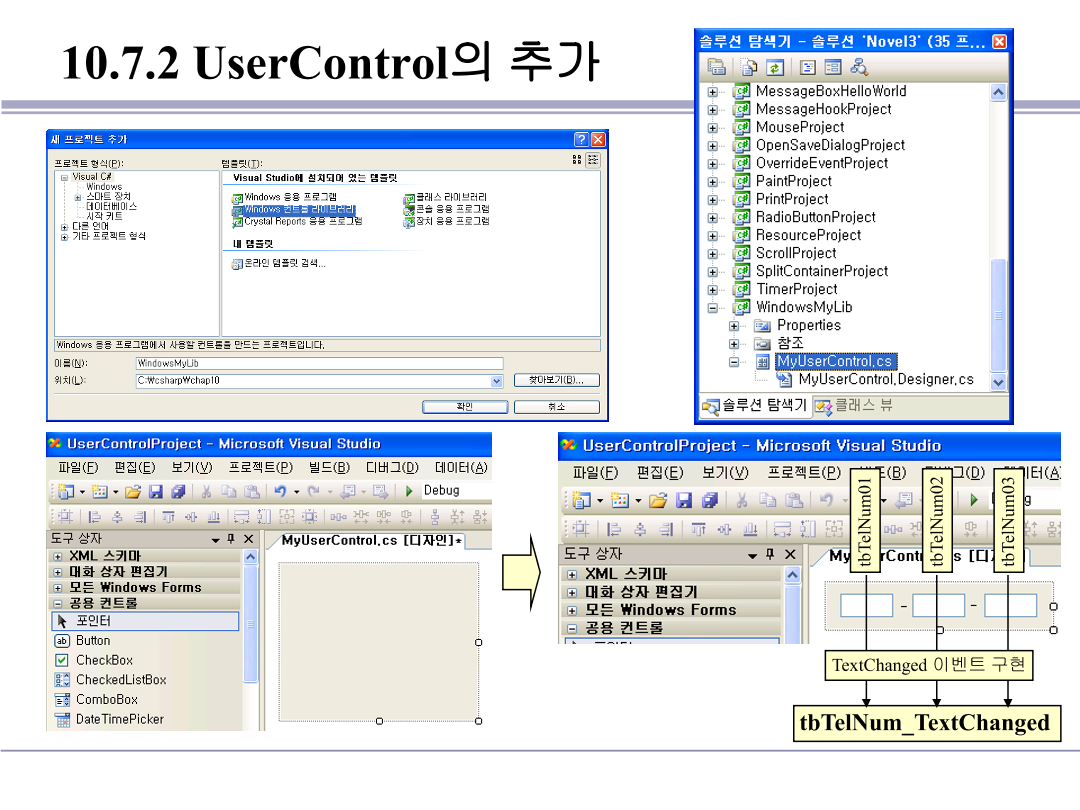


**10.7.1 Novel! : 사용자 정의 컨트롤과 구성 요소 클래스**  
  
C#에서는 사용자가 만들어 사용할 수 있는 컨트롤은 크게 사용자 정의 컨트롤과 구성 요소 클래스 두가지가 있다.  
  
**□ C#에서 사용자가 만들 수 있는 컨트롤**  
◇ 사용자 정의 컨트롤  
◇ 구성 요소 클래스  
  
**▣ 사용자 정의 컨트롤**  
  
사용자 정의 컨트롤이란 사용자가 여러 개의 컨트롤들을 조합해서 새로운 컨트롤을 만드는 것을 말한다. 여러 개의 컨트롤을 조합하기 위해서는 폼과 같은 기본 컨테이너가 있어야 한다. 이를 위해 UserControl이라는 기본 컨테이너를 제공한다.  
  
**□ 사용자 정의 컨트롤이란**  
◇ 사용자가 여러 개의 컨트롤들을 조합해서 새롭게 만드는 컨트롤  
  
**□ UserControl**  
◇ 폼과 같이 컨트롤들을 조합할 수 있도록 기본으로 제공되는 컨테이너  
  
새로운 사용자 정의 컨트롤을 만들면 기본으로 UserControl을 상속받으며, 이것은 다른 컨트롤들을 배치할 수 있는 기본적인 컨테이너가 된다.  
  
**▣ 구성 요소 컨트롤**  
  
구성 요소 클래스는 기본적으로 Component를 상속받거나 Component에서 파생되는 클래스를 상속받아야 한다. 구성 요소 클래스란 도구 상자에 들어 있는 컨트롤들을 상속받아서 새로운 컨트롤로 만든다고 생각하면 된다.  
  
**□ 구성 요소 클래스**  
◇ 도구 상자에 들어 있는 컨트롤들을 상속받아서 새로운 컨트롤로 만든다.  
  
Button이나 TextBox, ListBox, Panel과 같은 컨트롤들을 상속받아 여러분들만의 새로운 컨트롤로 만들고자 할 때에 구성 요소 클래스 형식으로 만들면 된다. 구성 요소 컨트롤을 생성하면 다음과 같은 형식의 코드가 만들어진다.  
  
**▒ 구성 요소 클래스의 형식**

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

|  |  |
| --- | --- |
| 1  2  3  4 | using System.ComponentModel  public class MyComponent : Component{     //  } |

기본적으로 Component를 상속받지만 Component 대신 사용자가 원하는 컨트롤을 상속받아 사용해도 된다.



**10.7.2 UserControl의 추가**  
  
사용자 정의 컨트롤을 직접 만들어보자. 현재의 솔루션에서 오른쪽 마우스 버튼을 누르면 새로운 프로젝트를 추가할 수 있다. [새 프로젝트 추가] 창에서 [Windows 컨트롤 라이브러리]를 선택한 후 WindowsMyLib라는 이름으로 프로젝트를 생성한다.  
  
**□ 새 프로젝트 생성**  
◇ [새 프로젝트 추가] 창에서 [Windows 컨트롤 라이브러리]를 선택  
◇ 프로젝트의 이름에 WindowsMyLib라고 입력한 후 [확인] 버튼 클릭  
  
위와 같이 하면 자동으로 WindowsMyLib라는 프로젝트로 사용자 정의 컨트롤이 하나 만들어진다. 사용자 정의 컨트롤의 이름을 MyUserControl.cs로 변경하기 바란다.  
  
**□ 사용자 정의 컨트롤 이름 변경**  
◇ 사용자 정의 컨트롤의 이름을 MyUserControl.cs로 변경  
  
이름을 변경한 후 MyUserControl.cs 파일을 보면 다음과 같은 코드를 만날 수 있다.  
  
**▒ MyUserControl.cs의 코드**

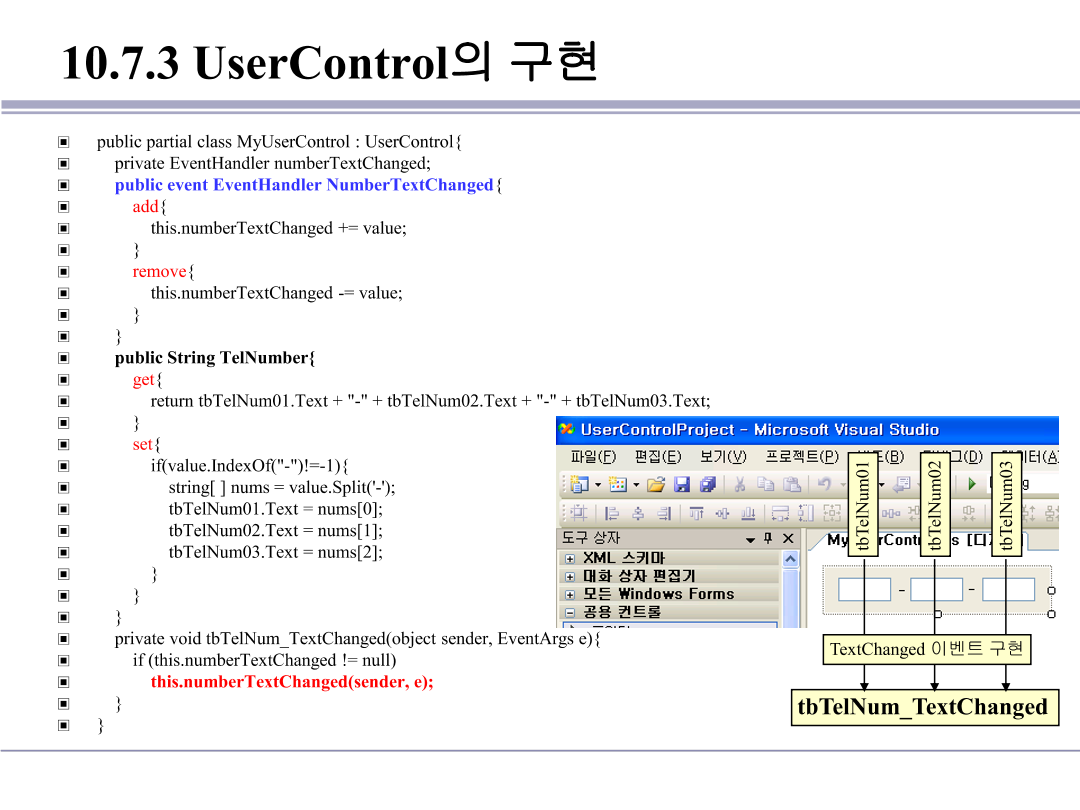
[?](http://www.jabook.com/jabook2/bs/bsTreeLoad.do?ba_no=73)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | namespace WindowsMyLib {       public partial class MyUserControl : UserControl {           public MyUserControl() {               InitializeComponent();           }       }   } |

UserControl을 상속받는 단순한 컨트롤이다. 솔루션 탐색기의 MyUserControl.cs를 더블 클릭하면 다른 컨트롤을 배치할 수 있는 컨테이너를 만날 수 있다. 여기에 3개의 TextBox를 추가한 후 각각에 TextChanged 이벤트를 장착하기 바란다.  
  
**□ 3개의 TextBox 생성**  
◇ TextBox 이름 : tbTelNumber01  
◇ TextBox 이름 : tbTelNumber02  
◇ TextBox 이름 : tbTelNumber03  
  
**□ 3개의 TextBox에 TextChanged 이벤트 장착**  
◇ tbTelNumber01에 tbTelNumber\_TextChanged 이벤트 핸들러 구현  
◇ tbTelNumber02에 tbTelNumber\_TextChanged 이벤트 핸들러 구현  
◇ tbTelNumber03에 tbTelNumber\_TextChanged 이벤트 핸들러 구현  
  
**☞ 참고**

|  |
| --- |
| 3개의 TextBox에 하나의 이벤트 핸들러를 장착하기 위해서는 먼저 하나의 이벤트 핸들러를 만든 후, 다른 TextBox의 이벤트에서는 만들어진 이벤트 핸들러를 이벤트 선택창에서 선택하면 된다. |

여기서 만들려고 하는 것은 전화번호 입력 컨트롤이다. 3개의 텍스트 박스를 이용해서 전화번호를 입력받을 것이다.



**10.7.3 UserControl의 구현**  
  
사용자 정의 컨트롤을 만들 때 명심해야 하는 것은 이것이 라이브러리라는 점이다. 그렇기 때문에 나를 위해서가 아니라 다른 사람을 위해서 이 컨트롤을 만드는 것이다.  
  
**▣ TelNumber 속성 구현하기**  
  
먼저 MyUserControl에 두가지 기능을 넣으려고 한다. 첫번째 기능은 3개의 TextBox에 들어 있는 전화번호를 외부에서 얻어내거나 설정할 수 있는 속성을 만들려고 한다.  
  
**□ MyUserControl의 새로운 속성**  
◇ 기능은 3개의 TextBox에 들어 있는 전화번호를 외부에서 얻어내거나 설정할 수 있는 속성  
  
public string TelNumber라는 속성을 두고 있으며 기본 구현은 다음과 같다.  
  
**▒ TelNumber 속성**

[?](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 | public String TelNumber{     get{     //3개의 텍스트 박스를 조합해서 전화번호로 만들어서 리턴하기        return tbTelNum01.Text + "-" + tbTelNum02.Text + "-" + tbTelNum03.Text;     }     set{        //대시(-)로 연결된 전화번호를 잘라서 각각의 텍스트 박스에 잘라서 넣기        if(value.IndexOf("-")!=-1){           string[ ] nums = value.Split('-');           tbTelNum01.Text = nums[0];           tbTelNum02.Text = nums[1];           tbTelNum03.Text = nums[2];        }     }  } |

**▣ 이벤트 구현하기**  
  
두번째로 MyUserControl을 만들었을 때 외부에서 사용할 이벤트를 만들어보자. 제일 먼저 이벤트를 선언해야 한다. 이벤트를 선언하는 일반적인 예이다.  
  
**▒ NumberTextChanged 이벤트의 구현**

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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | private EventHandler numberTextChanged;   public event EventHandler NumberTextChanged{           add{               this.numberTextChanged += value;           }           remove{               this.numberTextChanged -= value;           }   } |

이벤트를 선언하는 방법을 보면 거의 속성과 비슷하다. 단지 다른 것은 속성에서는 set과 get을 사용하는데 이벤트에서는 add와 remove를 사용하는 것이 다르다. 그리고 내부에서 value라는 매개변수를 사용할 수 있는 것도 동일하다.  
  
위와 같이 하면 외부에서 사용할 수 있는 NumberTextChanged 이벤트가 하나 만들어진 것이다. 하지만 이 이벤트는 만들어져 있을 뿐 어디에도 사용되지는 않았다. 즉 외부에서 이벤트를 등록해도 아무 일도 일어나지 않는다.  
  
이것이 동작하기 위해서는 MyUserControl 내부에서 등록된 이벤트를 호출해주어야 한다. 내부에서 3개의 TextBox의 TextChanged 이벤트가 발생할 때 NumberTextChanged 이벤트가 발생하도록 하자.   
  
**□ NumberTextChanged와 내부 이벤트 연결**  
◇ 3개의 TextBox의 TextChanged 이벤트가 발생할 때 NumberTextChanged 이벤트가 발생하도록 연결  
  
**▒ 3개의 TextBox의 TextChanged 이벤트**

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

|  |  |
| --- | --- |
| 1  2  3  4 | private void tbTelNum\_TextChanged(object sender, EventArgs e){           if (this.numberTextChanged != null)               this.numberTextChanged(sender, e);  } |

이 이벤트가 사용되는 절차는 제일 먼저 외부에서 NumberTextChanged 이벤트 핸들러를 등록할 것이다. 3개의 TextBox에서 TextChanged 이벤트가 발생할 때마다 this.numberTextChanged(sender, e)가 호출되면서 등록된 NumberTextChanged 이벤트 핸들러가 호출되는 것이다.  
  
MyUserControl을 구현하는 전체 예는 다음과 같다.

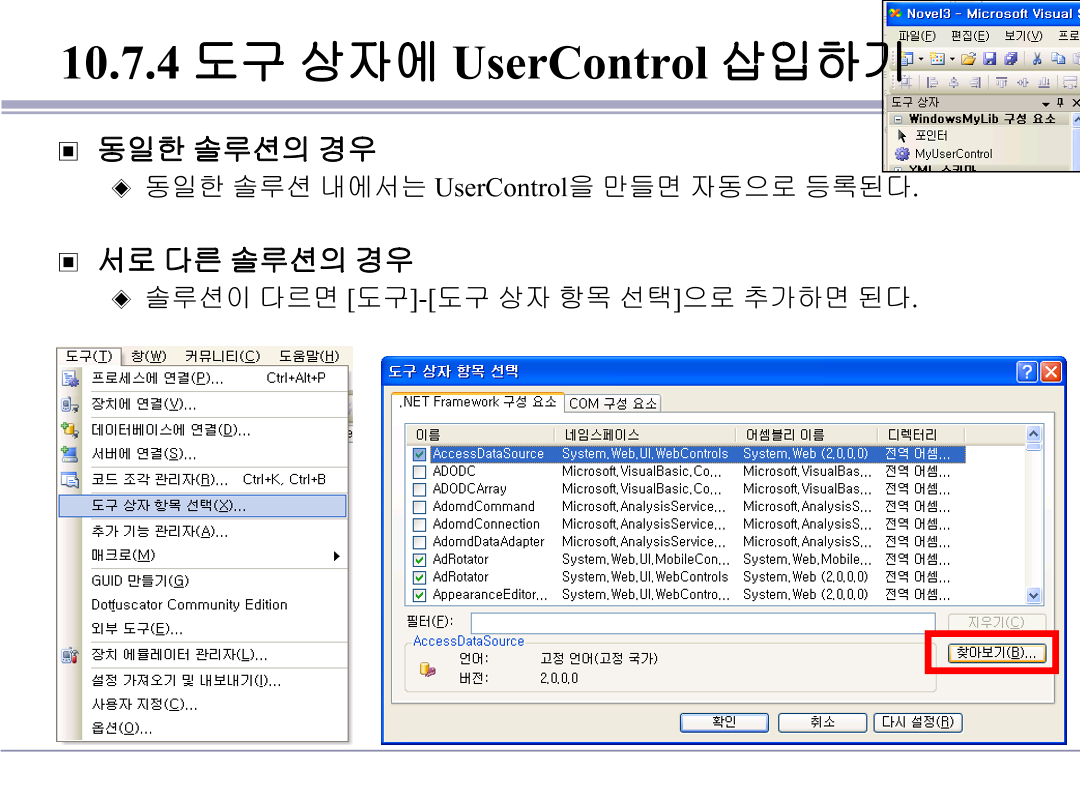
**§ chap10\WindowsMyLib\MyUserControl.cs**

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

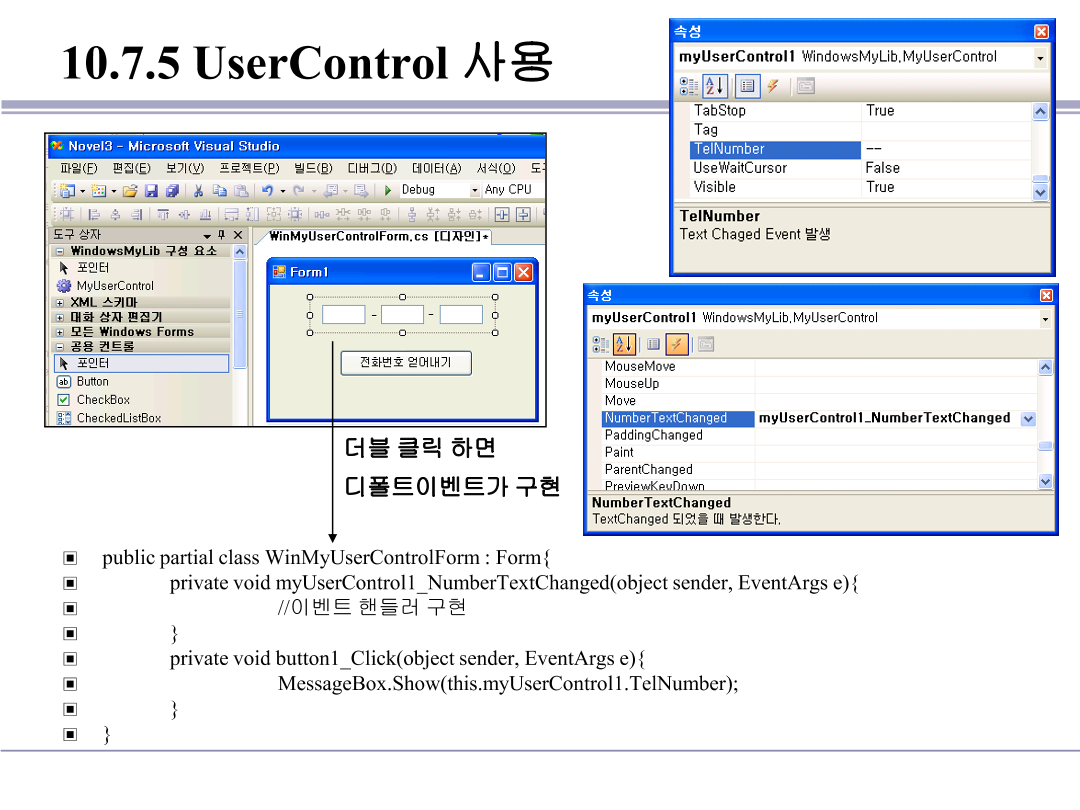
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Drawing;  using System.Data;  using System.Text;  using System.Windows.Forms;    namespace WindowsMyLib{      [DefaultEvent("NumberTextChanged")]      [DefaultProperty("TelNumber")]      public partial class MyUserControl : UserControl{          public MyUserControl(){              InitializeComponent();          }          private EventHandler numberTextChanged;            [Category("속성 변경"),          Description("TextChanged 되었을 때 발생한다.")]          public event EventHandler NumberTextChanged{              add{                  this.numberTextChanged += value;              }              remove{                  this.numberTextChanged -= value;              }          }          [Category("모양"),          Description("전화번호 값을 얻어낸다.")]          public String TelNumber{              get{                  return tbTelNum01.Text + "-" + tbTelNum02.Text + "-" + tbTelNum03.Text;              }              set{                  if(value.IndexOf("-")!=-1){                      string[] nums = value.Split('-');                      tbTelNum01.Text = nums[0];                      tbTelNum02.Text = nums[1];                      tbTelNum03.Text = nums[2];                  }              }          }          private void tbTelNum\_TextChanged(object sender, EventArgs e){              if (this.numberTextChanged != null)                  this.numberTextChanged(sender, e);          }      }  } |

**☞ 참고**

|  |
| --- |
| 여기서 사용하는 기법은 사용자 정의 컨트롤에서 이벤트와 속성을 구현하는 정형화된 방법이기 때문에 꼭 숙지해두기 바란다. |



**10.7.4 도구 상자에 UserControl 삽입하기**  
  
하나의 솔루션 내에서 윈도우 응용 애플리케이션도 만들고 윈도우 사용자 정의 컨트롤도 만들면 자동으로 새로 만든 사용자 정의 컨트롤이 도구 상자에 등록된다.  
  
**□ 동일한 솔루션의 경우**  
◇ 동일한 솔루션 내에서는 UserControl을 만들면 자동으로 등록된다.  
  
하지만 서로 다른 솔루션에서 만들 경우에는 새로 만든 사용자 정의 컨트롤은 도구상자에 등록되지 않는다. 이럴 경우에는 사용자자 직접 도구상자에 등록시켜 주면 된다.   
  
**□ 서로 다른 솔루션의 경우**  
◇ 솔루션이 다르면 [도구]-[도구 상자 항목 선택]으로 추가하면 된다.  
  
Visual Studio 메뉴의 [도구]-[도구 상자 항목 선택] 메뉴를 클릭하면 [도구 상자 항목 선택] 대화 상자가 나타난다. 이 대화 상자에서 [찾아보기]를 이용해서 새로 만든 [사용자 정의 컨트롤]을 등록시켜 주면 된다.



**10.7.5 UserControl 사용**  
  
앞에서 만든 MyUserControl을 사용해보자. 사용하는 방법은 너무 싶다. 새로운 윈도우 응용 애플리케이션을 만든 후 도구상자에서 MyUserControl을 끌어다 놓기만 하면 된다.  
  
**□ MyUserControl 사용하기**  
◇ 새로운 윈도우 응용 애플리케이션 만들기  
◇ 도구상자에 MyUserControl 등록하기(동일한 솔루션에서는 자동 등록)  
◇ 도구상자에서 폼으로 MyUserControl 끌어다 놓기  
  
중요한 것은 MyUserControl에서 만든 TelNumber 속성과 NumberTextChanged 이벤트이다. 이 이벤트는 속성창의 속성 부분과 이벤트 부분에서 찾을 수 있다. 우리가 만든 속성과 이벤트가 속성창에 자동으로 등록된 것이다.  
  
**▣ NumberTextChanged 이벤트의 사용**  
  
실제 NumberTextChanged 이벤트를 구현해보자. 현재 3개의 TextBox에서 TextChanged 이벤트가 발생할 때마다 NumberTextChanged 이벤트가 발생한다. 이러한 장점을 이용해서 입력된 내용이 숫자인지 아닌지를 체크해서 숫자만 입력되도록 해보자. 구현 방법은 다음과 같다.  
  
**▒ MyUserControl의 NumberTextChanged 이벤트 구현**

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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | private void myUserControl1\_NumberTextChanged(object sender, EventArgs e){       TextBox tb = sender as TextBox;       StringBuilder sb = new StringBuilder(tb.Text);       for (int i = 0; i < sb.Length; i++) {           if (!Char.IsDigit(sb[i])) {               sb.Remove(i, 1);               tb.Text = sb.ToString();               MessageBox.Show("숫자 형식이 아닙니다.");           }       }   } |

구현된 내부를 보면 이벤트가 발생한 TextBox의 참조값을 얻어내고 있다. 그리고 TextBox의 문자열을 이용해서 StringBuilder를 생성하고 있다. StringBuilder는 TextBox의 문자열 내에 숫자가 아닌 다른 문자가 포함되어 있다면 문자열을 편집하기 위해서 사용한다. 일반적인 string으로는 편집을 할 수 없기 때문에 StringBuilder로 변경한 것이다.  
  
**□ StringBuilder**  
◇ TextBox의 문자열 내에 숫자가 아닌 다른 문자가 포함되어 있다면 문자열을 편집하기 위해서 StringBuilder를 사용한다.  
◇ 일반적인 string은 편집할 수 없다. 단지 조합해서 새로운 문자열로 만드는 것이다.  
  
StringBuilder sb로 인덱서(Indexer)의 기법을 이용하면 배열 형식으로 각각의 문자에 접근할 수 있다. for문을 돌려서 숫자가 아니면 제거해버리면 되는 것이다. toString() 함수를 이용해서 숫자만으로 된 StringBuilder의 문자열을 다시 얻어내서 TextBox에 다시 할당하면 모든 작업이 끝난다.  
  
**▣ TelNumber 속성의 사용**  
  
TelNumber 속성을 이용하면 조합된 전화번호를 얻을 수 있다. 속성을 사용하는 방식으로 사용하면 된다. 그리고 대시(-)가 포함된 전화번호를 입력하면 자동으로 잘라서 각각의 TextBox에 넣어준다.   
  
**▒ TelNumber속성을 사용하는 예**

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

|  |  |
| --- | --- |
| 1  2  3 | private void button1\_Click(object sender, EventArgs e){       MessageBox.Show(this.myUserControl1.TelNumber);   } |

만약 전화번호를 설정하고자 한다면 다음과 같은 방식으로 사용하면 된다.  
  
**▒ TelNumber 값 설정**

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

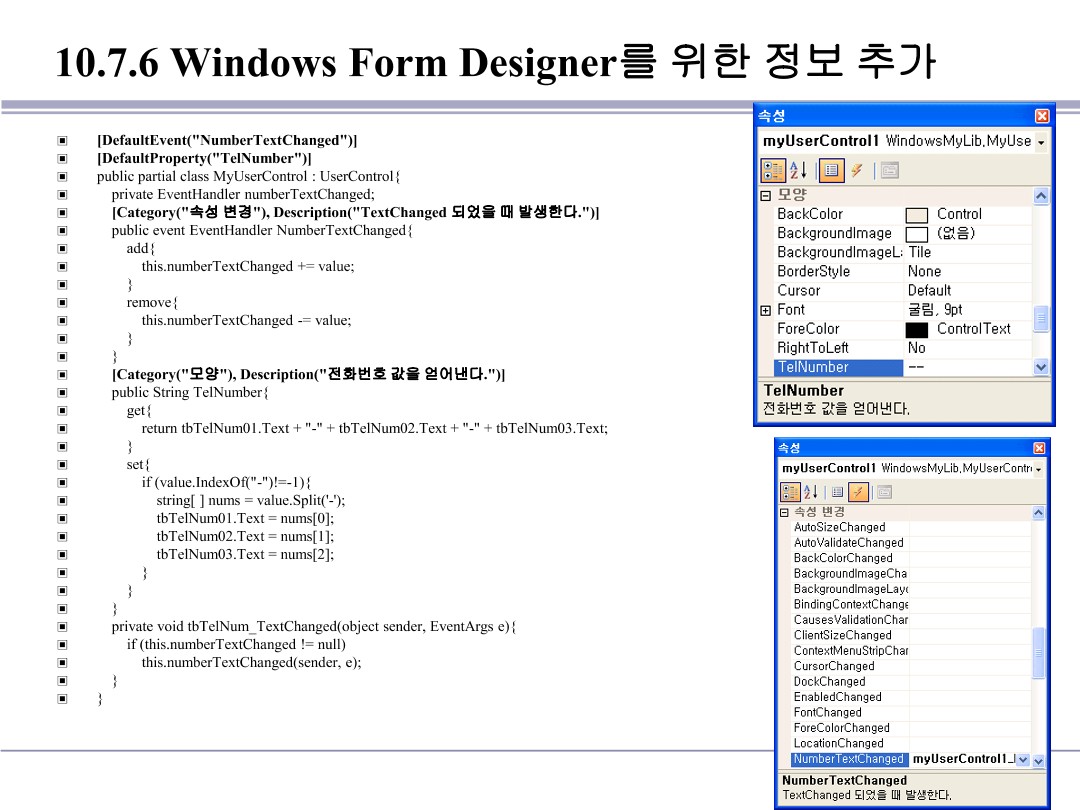
|  |  |
| --- | --- |
| 1 | myUserControl1.TelNumber = "02-777-7777"; |

위와 같이하면 자동으로 잘라서 각각의 TextBox에 전화번호가 입력된다.  
  
MyUserControl을 사용하는 전체 예는 다음과 같다.

**§ chap10\WinMyUserControlProject\WinMyUserControlForm.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 | using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Text;  using System.Windows.Forms;    namespace WinMyUserControlProject{      public partial class WinMyUserControlForm : Form{          public WinMyUserControlForm(){              InitializeComponent();          }          private void myUserControl1\_NumberTextChanged(object sender, EventArgs e){              TextBox tb = sender as TextBox;              StringBuilder sb = new StringBuilder(tb.Text);              for (int i = 0; i < sb.Length; i++){                  if (!Char.IsDigit(sb[i])){                      sb.Remove(i, 1);                      tb.Text = sb.ToString();                      MessageBox.Show("숫자 형식이 아닙니다.");                  }              }          }          private void button1\_Click(object sender, EventArgs e){              MessageBox.Show(this.myUserControl1.TelNumber);          }      }  } |



**10.7.6 Windows Form Designer를 위한 정보 추가**  
  
폼에 Button 컨트롤을 배치한 후 더블 클릭하면 자동으로 Click 이벤트가 추가되면서 편집할 수 있는 화면이 나타난다. 이것은 버튼에서 Click 이벤트를 디폴트 이벤트로 등록해두었기 때문이다.  
  
**□ Button 컨트롤의 디폴트 이벤트**  
◇ Button의 Click 이벤트가 디폴트 이벤트이기 때문에 폼에 Button 컨트롤을 배치한 후 더블 클릭하면 자동으로 Click 이벤트가 추가되면서 편집할 수 있는 화면이 나타난다  
  
이러한 디폴트 이벤트를 추가하기 위해서는 클래스 선언 앞부분에 다음과 같은 속성을 추가하면 된다.  
  
**▒ 디폴트 이벤트와 디폴트 속성 선언**

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

|  |  |
| --- | --- |
| 1  2  3  4  5 | [DefaultEvent("NumberTextChanged")]  [DefaultProperty("TelNumber")]  public partial class MyUserControl : UserControl{        //...  } |

MyUserControl에 디폴트 이벤트를 NumberTextChanged로 선언하고 있으며, 디폴트 속성을 TelNumber로 선언하고 있다. 디폴트 이벤트나 속성으로 선언해두면 속성 창을 열었을 때 최초로 선택되는 이벤트나 속성이 된다. 그리고 Button 컨트롤의 Click 이벤트처럼 폼에 MyUserControl 컨트롤을 배치하고 더블 클릭하면, 자동으로 NumberTextChanged 이벤트가 추가되면서 편집할 수 있는 장점이 있다.  
  
**□ 디폴트 이벤트와 디폴트 속성**  
◇ 디폴트 이벤트나 속성으로 선언해두면 속성 창을 열었을 때 최초로 선택되는 이벤트나 속성이 된다.  
  
속성창에서 속성들을 카테고리 형식으로 볼 수도 있으며, 알파벳 형식으로 볼 수도 있다. 만약 TelNumber 속성을 존재하는 카테고리에 집어 넣고자 한다면 다음과 같이 하면 된다.  
  
**▒ 이벤트의 카테고리 지정**

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

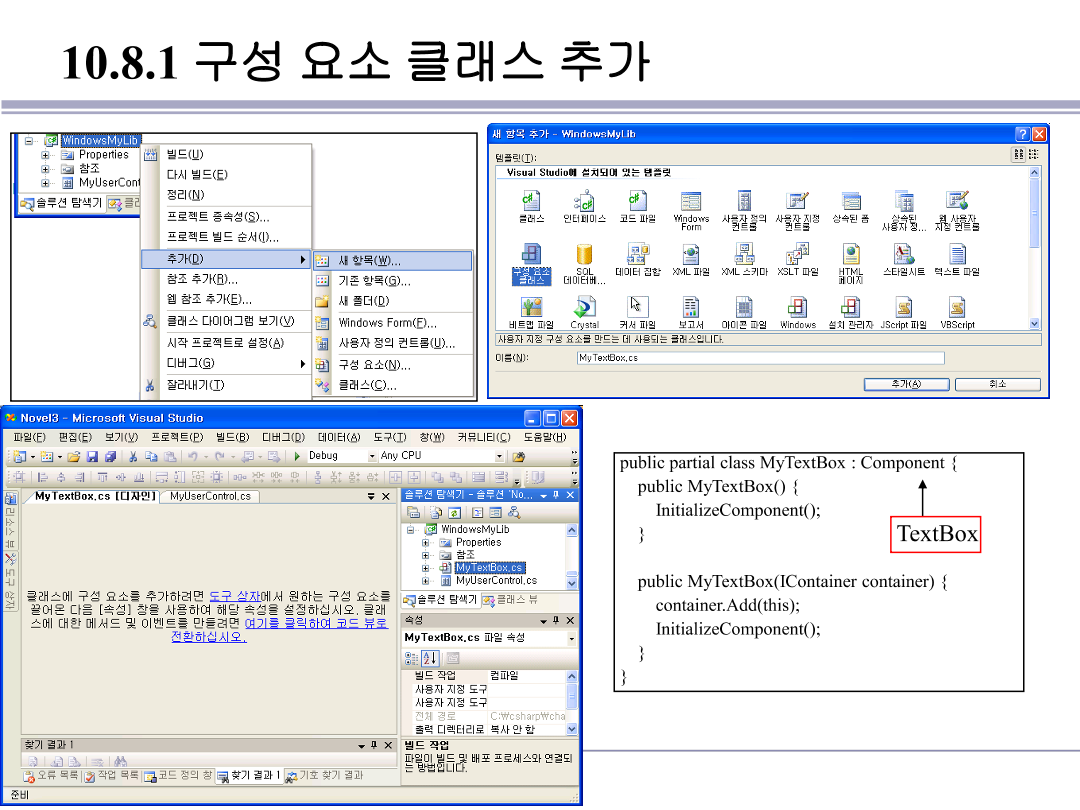
|  |  |
| --- | --- |
| 1  2  3  4 | [Category("속성 변경"), Description("TextChanged 되었을 때 발생한다.")]  public event EventHandler NumberTextChanged{     //....  } |

**▒ 속성의 카테고리 지정**

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

|  |  |
| --- | --- |
| 1  2  3  4 | [Category("모양"), Description("전화번호 값을 얻어낸다.")]  public String TelNumber{     //....  } |

위와 같이 하면 속성창에서 속성과 이벤트가 각각의 카테고리에 들어가 있는 모습을 확인할 수 있을 것이다



**10.8.1 구성 요소 클래스 추가**  
  
구성 요소 클래스는 도구 상자에 들어 있는 컨트롤들을 상속받아서 새로운 컨트롤로 만드는 컨트롤이라고 생각하면 쉽게 이해할 수 있다. 사용자 정의 클래스와 다른 점은 컨트롤을 조합해서 만드는 것이 아니라 컨트롤을 직접 상속하는 것이 다르다.  
  
앞에서 만든 WindowsMyLib 라이브러리 프로젝트에 추가적으로 구성 요소 클래스를 만들어보자. WindowsMyLib 프로젝트에 오른쪽 마우스 키를 누른 후 [추가]-[새 항목]을 선택하면 [새 항목 추가] 창을 만날 수 있다. 여기서 [구성 요소 클래스]를 선택한 후 파일 이름에 MyTextBox.cs라고 입력하자. 그리고 [추가] 버튼을 누르면 프로젝트에 구성 요소 클래스가 추가된다.  
  
**▒ 구성 요소 클래스 추가하는 절차**

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

|  |  |
| --- | --- |
| 1  2  3  4 | 1. WindowsMyLib 프로젝트에 오른쪽 마우스 키  2. [추가]-[새 항목]을 선택  3. [새 항목 추가] 창에서 [구성 요소 클래스]를 선택한 후 파일 이름에 MyTextBox.cs라고 입력  4. [추가] 버튼을 누르면 프로젝트에 구성 요소 클래스가 추가된다. |

구성 요소 클래스를 추가하면 폼과 같은 컨테이너가 나오는 것이 아니라 다음과 같은 메시지가 나온다.  
  
**☞ 구성 요소 클래스의 초기화면 메시지**

|  |
| --- |
| 클래스에 구성 요소를 추가하려면 도구 상자에서 원하는 구성 요소를 끌어온 다음 [속성] 창을 사용해서 해당 속성을 설정하십시오. 클래스에 대한 함수 및 이벤트를 만들려면 여기를 클릭하여 코드 뷰로 전환하십시오. |

구성 요소에는 컨테이너가 없다. 단지 도구상자에 있는 컨트롤을 현재의 구성 요소 클래스에 삽입할 수 있으며, 속성창에서 속성이나 이벤트를 편집할 수 있는 장점만 존재한다. 이것이 싫으면 객체를 생성하는 구문을 수작업으로 넣고 속성이나 이벤트를 직접 구현해서 사용하면 된다.   
  
Visual Studio에서는 속성을 다루는 기능이 발전되어 있기 때문에 툴을 이용해서 속성을 변경하고자 한다면 도구상자의 컨트롤을 구성요소 초기화면에 끌어다 놓고 사용하기만 하면 된다.  
  
MyTextBox 구성요소 클래스를 추가하면 다음과 같은 코드가 자동으로 생성된다.  
  
**▒ MyTextBox 구성 요소 클래스 생성**

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

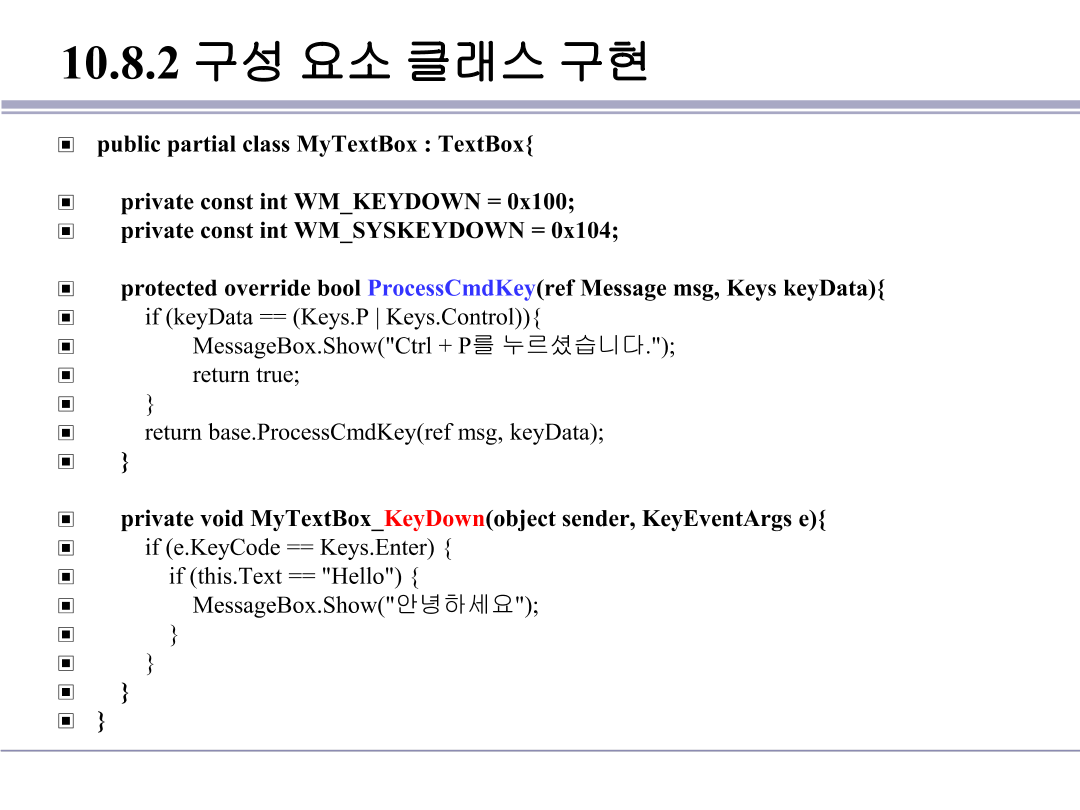
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | public partial class MyTextBox : Component {       public MyTextBox() {           InitializeComponent();       }       public MyTextBox(IContainer container) {           container.Add(this);           InitializeComponent();       }   } |

여기서는 TextBox를 상속받아 새로운 나만의 TextBox를 만들려고 한다. 현재 기본으로 Component를 상속하고 있는 부분을 TextBox로 변경하기 바란다.  
  
**▒ TextBox를 상속받는 MyTextBox 구성 요소 클래스**

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

|  |  |
| --- | --- |
| 1  2  3 | public partial class MyTextBox : TextBox{     //...  } |

이 클래스는 기본적으로 TextBox의 기능을 한다. 그리고 추가적인 구현을 사용자가 직접 밀어 넣을 수 있는 장점이 있다.



**10.8.2 구성 요소 클래스 구현**  
  
앞에서 생성한 MyTextBox의 세부적인 구현을 해보자. TextBox에 단축키 기능과 특정 단어를 입력한 후 엔터를 누르면 반응하는 나만의 TextBox를 구현하려 한다.  
  
**□ MyTextBox의 새로운 구현**  
◇ MyTextBox에서 Ctrl + P를 누르면 "Ctrl + P를 누르셨습니다."라는 메시지가 출력된다.  
◇ Hello라는 단어를 입력하고 엔터를 누르면 "안녕하세요"라는 메시지가 출력된다.  
  
**▣ 단축키의 구현**   
  
먼저 Ctrl + P의 단축키를 구현하기 위해서는 TextBox의 ProcessCmdKey() 함수를 오버라이딩하는 방법이 가장 좋은 방법이다. ProcessCmdKey() 함수 자체가 명령키(Command Key)를 처리하기 위한 함수이다. 명령키는 단축키 정도로 생각하면 된다. 명령키의 경우 항상 일반 입력 키보다 우선 순위가 높다.  
  
**□ ProcessCmdKey()**   
◇ 명령키(Command Key-단축키)를 처리하기 위한 함수이다.  
  
ProcessCmdKey() 함수를 오버라이딩하고 이 함수 내에서 true를 반환하면 명령 키를 처리했음을 의미하고, false를 반환하면 해당 키가 명령 키가 아님을 의미한다. 일반적인 경우 폼 레벨에서 단축키를 처리하기 때문에 폼 레벨에서 ProcessCmdKey()를 많이 사용한다.  
  
**▒ ProcessCmdKey()의 오버라이딩**

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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | protected override bool ProcessCmdKey(ref Message msg, Keys keyData){         if (keyData == (Keys.P | Keys.Control)){                   MessageBox.Show("Ctrl + P를 누르셨습니다.");                   return true;         }         return base.ProcessCmdKey(ref msg, keyData);   } |

입력된 키는 ProcessCmdKey()의 keyData 매개변수로 넘어온다. keyData의 값이 Ctrl + P일 경우 메시지 박스를 띄워주고 있다. true를 반환하면 Ctrl + P 단축키는 처리되었음을 의미한다. 만약 다른 단축키일 경우를 위해서 base.ProcessCmdKey()를 호출하고 그 결과를 리턴한다.  
  
**▣ "Hello" 문자열을 입력한 후 엔터에 반응하기**   
  
Hello라는 단어를 입력하고 엔터를 누르면 "안녕하세요"라는 메시지가 출력되도록 만들기 위한 최적의 장소는 KeyDown 이벤트이다. KeyDown 이벤트 핸들러에서 엔터키를 확인한 후 메시지를 띄우면 된다.  
  
**▒ KeyDown 이벤트 핸들러 구현**

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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | private void MyTextBox\_KeyDown(object sender, KeyEventArgs e){           if (e.KeyCode == Keys.Enter) {               if (this.Text == "Hello") {                   MessageBox.Show("안녕하세요");               }           }   } |

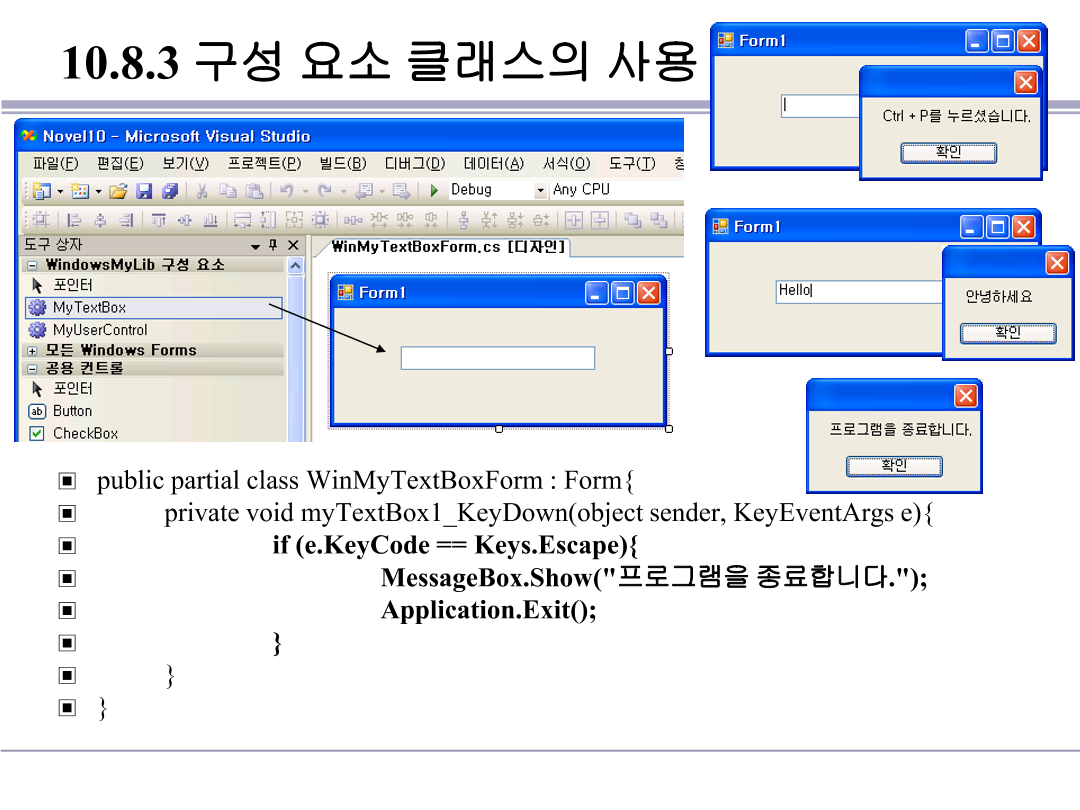
KeyEventArgs의 KeyCode 속성을 이용해서 현재 입력된 키를 확인하고 있다. Keys.Enter 키가 입력되면 "안녕하세요" 메시지를 띄우고 있다.  
  
단축키와 "Hello"라는 문자열에 반응하는 새로운 MyTextBox를 구현이 완료되었다. 이것을 사용하는 예를 보자.  
  
**☞ 참고**

|  |
| --- |
| KeyEventArgs의 KeyCode의 경우에는 현재 눌러진 키 값만 포함하지만, KeyData의 경우에는 현재 눌러진 키 값과 보조키(Control, Alt, Shift) 키 값을 함께 포함한다. |

**§ chap10\WindowsMyLib\MyTextBox.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 | using System;  using System.ComponentModel;  using System.Collections.Generic;  using System.Diagnostics;  using System.Text;  using System.Windows.Forms;  namespace WindowsMyLib{      public partial class MyTextBox : TextBox    {         public MyTextBox(){              InitializeComponent();          }          public MyTextBox(IContainer container){              container.Add(this);              InitializeComponent();          }          //private const int WM\_KEYDOWN = 0x100;          //private const int WM\_SYSKEYDOWN = 0x104;          protected override bool ProcessCmdKey(ref Message msg, Keys keyData){              //if (msg.Msg == WM\_KEYDOWN || msg.Msg == WM\_SYSKEYDOWN){                  if (keyData == (Keys.P | Keys.Control)){                      MessageBox.Show("Ctrl + P를 누르셨습니다.");                      return true;                  }              //}              return base.ProcessCmdKey(ref msg, keyData);          }          private void MyTextBox\_KeyDown(object sender, KeyEventArgs e){              Debug.WriteLine("MyTextBox 내부의 KeyDown");              if (e.KeyCode == Keys.Enter){                  if (this.Text == "Hello"){                      MessageBox.Show("안녕하세요");                  }              }          }      }  } |



**10.8.3 구성 요소 클래스의 사용**  
  
새로운 윈도우 응용 애플리케이션을 만든 후 도구상자에서 MyTextBox을 끌어다 놓기만 하면 모든 작업이 완료된다.  
  
**□ MyTextBox 사용하기**  
◇ 새로운 윈도우 응용 애플리케이션 만들기  
◇ 도구상자에 MyTextBox 등록하기(동일한 솔루션에서는 자동 등록)  
◇ 도구상자에서 폼으로 MyTextBox 끌어다 놓기  
  
이 상태에서 프로그램을 실행시키면 단축키와 "Hello" 엔터에 반응한다. 이것은 MyTextBox가 새로운 기능을 장착하고 있기 때문이다.  
  
MyTextBox에 새로운 이벤트를 하나 더 구현해보자. MyTextBox에서 [ESC] 키를 입력하면 프로그램이 종료하도록 만들어보자. 우선 키의 입력을 받기 위해서 MyTextBox의 KeyDown 이벤트를 구현하자.  
  
**▒ MyTextBox의 KeyDown 이벤트 추가**

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

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | private void myTextBox1\_KeyDown(object sender, KeyEventArgs e){     if (e.KeyCode == Keys.Escape){        MessageBox.Show("프로그램을 종료합니다.");        Application.Exit();     }  } |

MyTextBox의 KeyDown 이벤트를 다시 구현하고 있다. 실제 MyTextBox 내부에서는 [Enter] 키에 반응하는 예를 구현하였으며, 외부에서는 [ESC] 키에 반응하는 예를 만들었다.  
  
**□ 질문**  
◇ MyTextBox 내부에서 KeyDown 이벤트를 구현하고 있고 외부에서 KeyDown 이벤트를 다시 구현하고 있으면 둘 다 동작합니까?  
  
**□ 해답**  
◇ 이벤트의 경우 다중으로 등록해서 사용할 수 있다.  
◇ 내부의 KeyDown 이벤트가 먼저 동작하고 외부의 KeyDown 이벤트가 동작  
◇ 내부의 KeyDown 이벤트는 이미 등록되어 있는 상태이며, MyTextBox를 사용할 때 KeyDown 이벤트가 등록된다.  
  
이벤트의 경우 다중으로 등록해서 사용할 수 있다. 내부의 KeyDown 이벤트는 이미 등록되어 있는 상태이며, MyTextBox를 사용할 때 다시 KeyDown 이벤트를 등록해서 사용하기 때문에 이벤트는 내부의 KeyDown 이벤트가 먼저 동작하고 외부의 KeyDown 이벤트가 동작한다.

**§ chap10\WinMyTextBoxProject\WinMyTextBoxForm.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;  using System.Diagnostics;    namespace WinMyTextBoxProject{      public partial class WinMyTextBoxForm : Form{          public WinMyTextBoxForm(){              InitializeComponent();          }          private void myTextBox1\_KeyDown(object sender, KeyEventArgs e){              Debug.WriteLine("MyTextBox 외부의 KeyDown");              if (e.KeyCode == Keys.Escape){                  MessageBox.Show("프로그램을 종료합니다.");                  Application.Exit();              }          }      }  } |