

**10.5.1 공통 속성들**  
  
컨트롤 중에서 가장 많이 사용하는 컨트롤에는 어떤 것이 있을까? Visual Studio에서는 이 컨트롤을 공용 컨트롤로 명명하고 있다.  
  
**□ 공용 컨트롤 I**  
◇ Button, Label, CheckBox, ListBox, ComboBox, PictureBox  
◇ TextBox, RadioButton, ScrollBar, TreeView, ListView   
  
**□ 공용 컨트롤 II**  
◇ CheckedListBox, DateTimePicker, LinkLabel, MaskedTextBox, MonthCalendar, NofifyIcon, NumericUpDown, ProgressBar, RichTextBox, ToolTip, WebBrowser  
  
위의 공용 컨트롤 I과 II는 많이 사용하는 기준으로 필자가 임의로 나누었다. 이 책에서는 공용 컨트롤 I 위주로 알아 볼 것이다. 여기서는 먼저 이들 컨트롤에서 많이 사용하는 속성들에 대해서 알아보자.   
  
컨트롤에서 공통으로 많이 사용되는 속성은 크게 모양, 동작, 레이아웃 카테고리로 나눌 수 있다.  
  
**□ 컨트롤에서 많이 사용하는 속성의 카테고리**  
◇ 모양  
◇ 동작  
◇ 레이아웃  
  
모양에 관련된 주요 속성은 Text, Font, ForeColor, BackColor이다. 그 이름만 들어도 알 수 있듯이 텍스트와 폰트, 글자색, 배경색 속성이다.  
  
**▒ 모양에 관련된 속성 중 공통으로 많이 사용하는 속성**

|  |  |
| --- | --- |
| 1  2  3  4 | obj.Text = "Hello World";  obj.Font = new Font("굴림", 12);  obj.ForeColor = Color.Black;  obj.BackColor = SystemColors.Control; |

동작에 관련된 주요 속성은 Visible, Enabled, TabStop, TabIndex 등이 있다.  
  
**▒ 동작에 관련된 속성 중 공통으로 많이 사용하는 속성**

|  |  |
| --- | --- |
| 1  2  3  4 | obj.Visible = true; //보일 것인지를 지정  obj.Enabled = true; //사용 가능한지를 지정  obj.TabStop = true; //탭 키로 포커스를 받을 수 있는지를 지정  obj.TabIndex = 0; //탭의 순서를 지정 |

레이아웃에 관련된 주요 속성은 Location, X, Y, Size, Dock 등이 있다.  
  
**▒ 레이아웃에 관련된 속성 중 공통으로 많이 사용하는 속성**

|  |  |
| --- | --- |
| 1  2  3  4  5 | obj.Location = new Point(100, 100); //위치  obj.X = 100; //X 좌표  obj.Y = 100; //Y 좌표  obj.Size = new Size(300, 300); //크기  obj.Dock = DockStyle.Fill; //Left, Right, Top, Bottom, None, Fill |

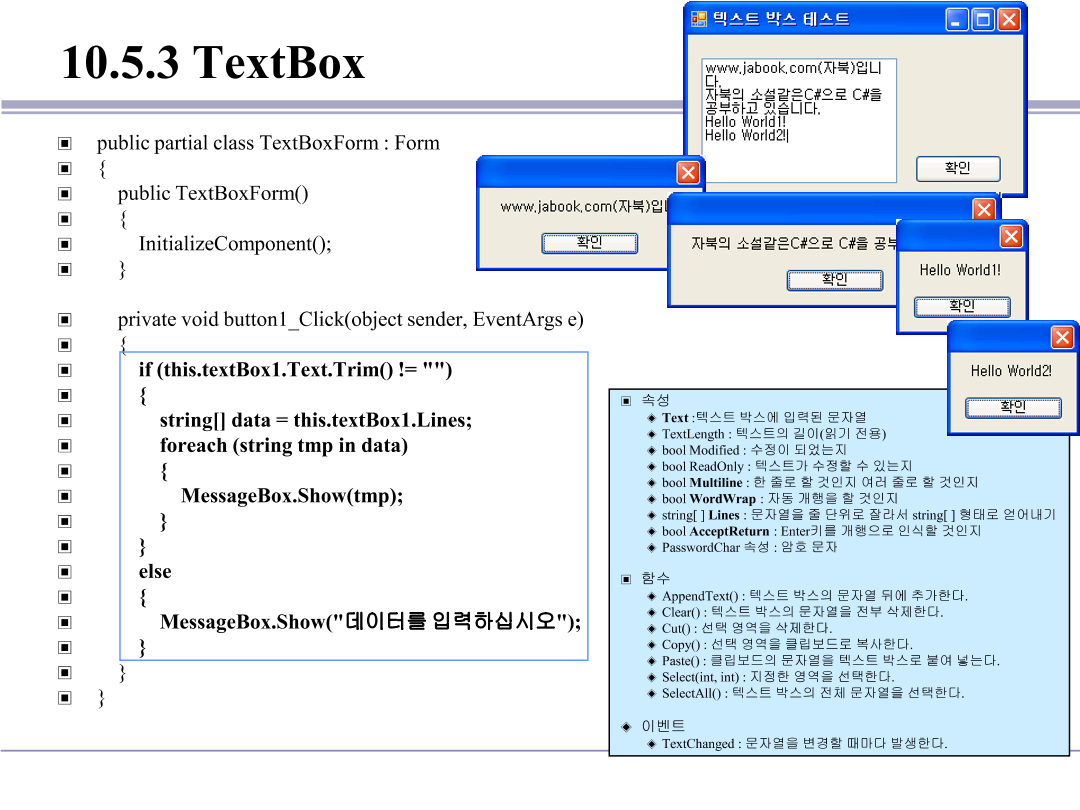
지금까지 언급한 속성들은 너무나도 기본적이며 많이 사용하는 속성들이다. 적어도 이 정도의 속성은 알아두고 넘어가는 것이 좋다.  
  
컨트롤을 사용할 때 반드시 지켜야 하는 규칙이 있다. 자신이 만들고자 하는 프로그램에 어떤 컨트롤을 사용해야 할지를 정확하게 알고 있어야 한다. 즉 컨트롤의 성격에 맞게 컨트롤을 사용해야 하는 것이다.  
  
**□ 컨트롤을 사용할 때의 규칙**  
◇ 컨트롤의 성격에 맞게 컨트롤을 사용하라!  
  
컨트롤을 사용하다 보면 현재의 컨트롤에 추가적으로 필요한 기능이 있다. 현재의 컨트롤에는 없는 새로운 기능이 필요한 경우가 있다. 이럴 경우를 위해서 상속이 존재하는 것이다. 상속으로 새로운 기능을 추가적으로 구현해서 사용하면 된다.  
  
**□ 컨트롤의 성격에 벗어난다면 어떻게 해야 하나?**  
◇ 통째로 상속해서 사용해야 한다.  
◇ 컨트롤을 직접 상속한다.  
  
컨트롤을 직접 상속하는 경우는 하나의 컨트롤에만 해당하는 작업이다. 이런 경우 이외에 여러 개의 컨트롤을 묶어서 하나의 컨트롤로 만들어야 하는 경우도 있다. 이런 경우에는 사용자 정의 컨트롤을 만들어서 사용하면 된다.  
  
**□ 여러 개의 컨트롤을 묶어서 하나의 컨트롤로 만들어야 하는 경우**  
◇ 사용자 정의 컨트롤로 만들어서 새로 정의해서 사용한다.  
◇ UserControl을 상속한 후 새롭게 디자인해서 사용한다.  
  
직접 상속해서 사용하는 방법과 사용자 정의 컨트롤을 만드는 방법은 이 장의 후반부에서 배우게 될 것이다.

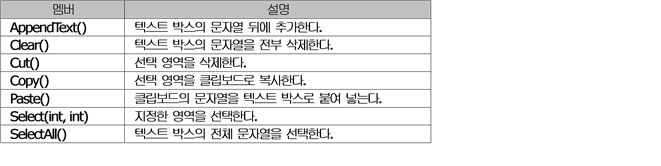
**10.5.2 Button, Label, CheckBox**  
  
**▣ Button 컨트롤**  
  
버튼 컨트롤의 주된 목적은 마우스로 클릭하는 행위를 처리하기 위한 용도로 사용된다. 그래서 Button을 만들고 디자인 영역에서 더블 클릭을 하면 Click 이벤트 핸들러가 자동으로 만들어진다.  
  
**□ Button의 핵심 이벤트**  
◇ Click 이벤트  
  
**☞ 기본 이벤트**

|  |
| --- |
| 컨트롤을 만든 후 더블 클릭하면 기본 이벤트가 추가된다. 각 컴포넌트의 기본 이벤트 정도는 알아두는 것이 좋다. |

Click 이벤트는 버튼이 클릭되었을 때 발생하는 이벤트이다. 사실 Button만큼 간단하면서도 많이 사용되는 컨트롤도 없을 것이다.  
  
**▣ Label 컨트롤**  
  
Label 컨트롤은 화면에 문자열을 표현하는 용도로 많이 사용된다. 가끔 Label에 Click 이벤트를 주고 사용하지만 이것의 주목적은 문자열의 표시에 있다.  
  
**▣ CheckBox 컨트롤**  
  
CheckBox는 문자열 앞에 사각형의 체크 박스가 존재한다. 이 체크 박스의 가장 중요한 문제가 바로 체크되어 있느냐 아니냐의 문제이다. Checked 속성을 이용하면 간단히 CheckBox가 체크되어 있는지 확인할 수 있다. 체크되어 있다면 true, 아니면 false이다.   
  
**□ CheckBox의 핵심 속성**  
◇ Checked 속성 : 체크되어 있다면 true, 아니면 false이다.  
  
Checked 속성이 중요하듯이 CheckBox에서 이벤트를 처리할 때에는 CheckedChagned 이벤트를 주로 사용한다.  
  
**□ CheckBox의 핵심 이벤트**  
◇ CheckedChagned 이벤트  
  
실제 CheckBox에서 조심해야 할 것은 CheckBox의 CheckState이다.   
  
**☞ CheckState 속성**

|  |
| --- |
| CheckBox의 ThreeState라는 속성이 false이면 Checked, Unchecked만 존재하지만 true이면 Indeterminate까지 3가지 상태가 존재한다. Indeterminate는 체크 박스가 트리 방식으로 존재할 때 많이 사용된다. 상위레벨에서 체크를 하면 하위의 모든 체크박스가 체크되는 것을 경험한 적이 있을 것이다. 하위 레벨에서 어떤 것은 체크되고 어떤 것은 체크되어 있지 않다면 어떻게 하겠는가? 상위 레벨의 체크 상태가 모호하지 않는가! 바로 이럴 경우 Indeterminate를 사용한다. CheckState의 상태가 변하면 CheckStateChanged 이벤트가 발생한다. |



**10.5.3 TextBox**  
  
TextBox는 문자열을 입력받을 수 있는 컨트롤이다. 문자열과 관련된 컨트롤일 경우 알아야 할 기본 함수도 많고 기본 속성도 많은 편이다. 여기서는 많이 사용하는 함수나 속성 위주로 알아보자.  
  
입력된 문자열을 얻어내거나 설정하기 위해서는 Text 속성을 이용하면 된다. 그리고 문자열의 길이를 얻기 위해서는 TextLength 속성을 이용하면 된다.  
  
**□ string Text 속성**  
◇ 텍스트 박스에 입력된 문자열     
  
**□ int TextLength 속성**  
◇ 텍스트의 길이(읽기 전용)  
  
Modified 속성은 TextBox 내의 문자열이 수정되었는지, ReadOnly 속성은 텍스트를 수정할 수 있는지 없는지를 나타낸다.  
  
**□ bool Modified 속성**  
◇ 수정이 되었는지     
  
**□ bool ReadOnly 속성**  
◇ 텍스트가 수정할 수 있는지  
  
TextBox를 한 줄 형식으로 사용하지만 여러 줄을 입력받는 형식으로도 사용할 수 있다. 여러 줄을 입력받는 TextBox로 만들어주기 위해서는 Multiline 속성을 true로 설정하면 된다.  
     
**□ bool Multiline 속성**  
◇ 한 줄로 할 것인지 여러 줄로 할 것인지  
  
여러 줄로 하게 되면 문자열이 길어질 경우 자동으로 다음 줄로 넘기는 자동 개행을 지원할 것인지를 나타내는 WordWrap 속성이 있다. 그리고 모든 문자열을 줄 단위로 잘라서 배열형식으로 얻어내는 Lines 속성이 있다.  
  
**□ bool WordWrap 속성**  
◇ 자동 개행을 할 것인지     
  
**□ string[ ] Lines 속성**  
◇ 전체 문자열을 줄 단위로 잘라서 배열(string[ ])형태로 얻어내기  
  
여러 줄을 입력받을 때 기본적으로 Enter 키를 누르게 되면 반응하지 않는다. 이에 반응하게 만들기 위해서 AcceptReturn 속성을 true로 설정하면 된다.  
  
**□ bool AcceptReturn 속성**  
◇ Enter키를 개행으로 인식할 것인지 디폴트 버튼으로 인식할 것인지  
  
PasswordChar 속성은 문자를 입력할 때 암호문자가 출력되도록 하는데 사용한다. 기본적으로 암호문자가 출력되지 않는 0의 값을 가지지만 다른 문자값을 할당하면 문자가 입력될 때마다 설정된 문자가 출력된다.  
  
**□ PasswordChar 속성**  
◇ 암호 문자     
  
TextBox에서 많이 사용하는 함수는 다음과 같다.  
  
  
[표 10\_13] TextBox의 주요 함수  
  
TextBox를 사용할 때 가장 많이 사용하는 이벤트는 문자열이 변경되었을 때 발생하는 TextChanged 이벤트이다. TextBox에 대한 간단한 예를 알아보자.

**§ chap10\TextBoxProject\TextBoxForm.cs**

|  |  |
| --- | --- |
| 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 TextBoxProject{      public partial class TextBoxForm : Form{          public TextBoxForm(){              InitializeComponent();          }            private void button1\_Click(object sender, EventArgs e){              if (this.textBox1.Text.Trim() != ""){                  string[] data = this.textBox1.Lines;                  foreach (string tmp in data){                      MessageBox.Show(tmp);                  }              }else{                  MessageBox.Show("데이터를 입력하십시오");              }          }      }  } |

위의 예는 TextBox에 문자열을 줄단위로 얻어내서 한줄씩 메시지 박스를 띄우는 예이다. TextBox에 설정된 속성은 다음과 같다.  
  
**▒ 텍스트 박스에 설정된 속성**

|  |  |
| --- | --- |
| 1  2  3 | this.textBox1.AcceptsReturn = true;  this.textBox1.AcceptsTab = true;  this.textBox1.Multiline = true; |

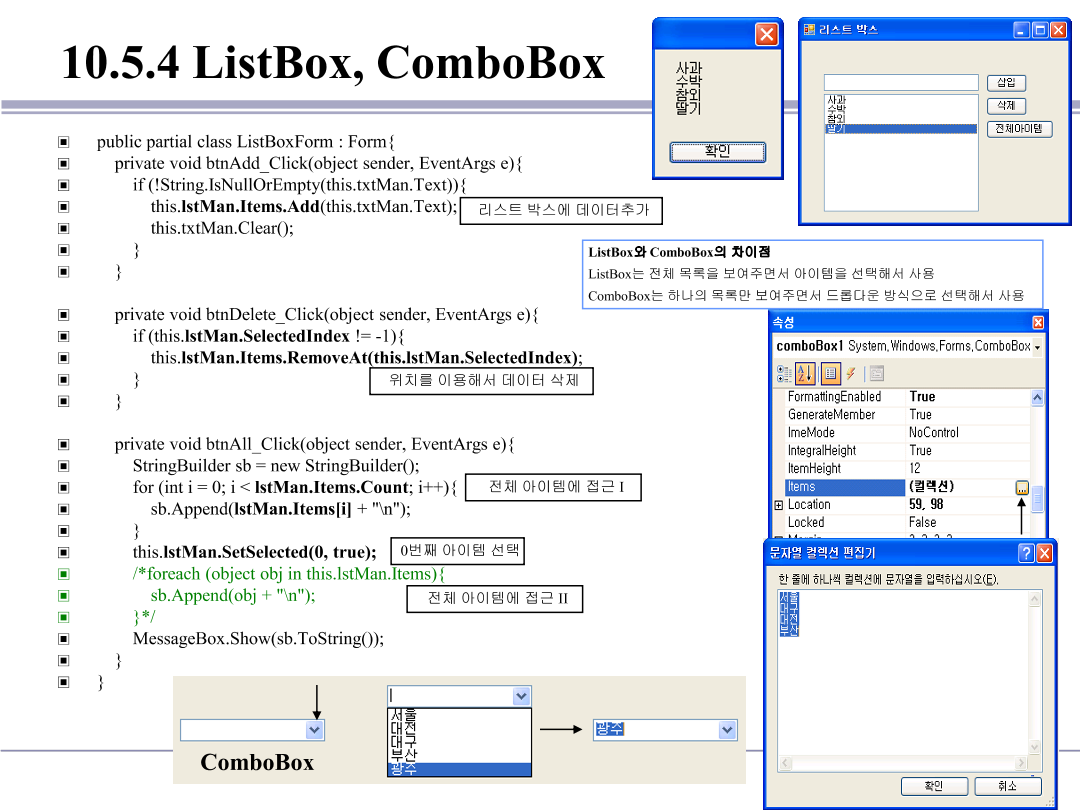
TextBox에 문자열이 포함되어 있는지 확인하기 위해서 TextBox의 Text 속성을 이용하고 있다.  
  
**▒ TextBox에 문자열 포함여부 확인 하기**

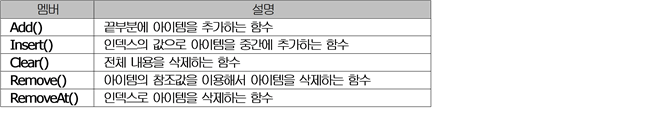
|  |  |
| --- | --- |
| 1  2  3 | if (this.textBox1.Text.Trim() != ""){       //...작업   } |

여기서 Trim() 함수는 문자열의 양쪽 끝 공백을 제거하는 함수이다. Lines 속성을 이용해서 문자열을 줄단위로 얻어내고 있다. 그리고 얻어낸 string[ ]형의 배열을 foreach로 검색하고 있다.  
  
**▒ Lines 속성으로 줄단위로 얻어내기**

|  |  |
| --- | --- |
| 1  2  3  4  5 | string[ ] data = this.textBox1.Lines;   foreach (string tmp in data)   {       MessageBox.Show(tmp);   } |

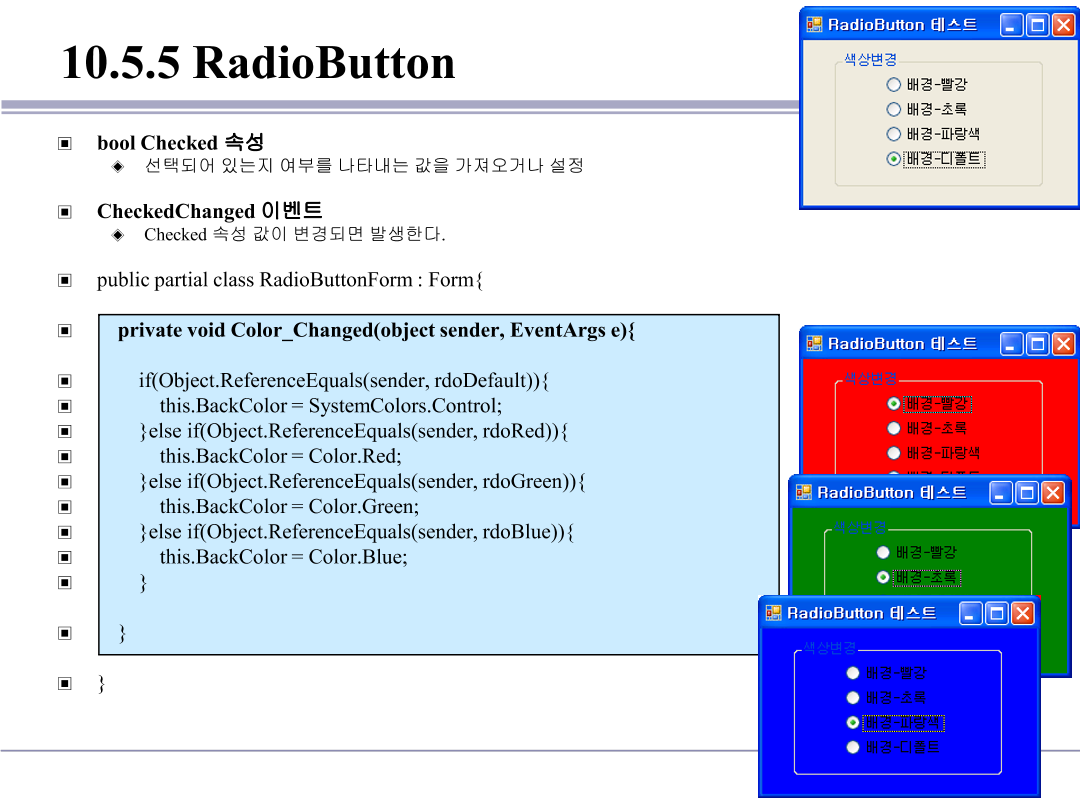
문자열이 입력되면 PPT의 메시지 박스처럼 메시지 박스가 줄 수만큼 표시된다.



**10.5.4 ListBox, ComboBox**  
  
ListBox와 ComboBox는 아이템을 목록 형식으로 보관하고 있으면서 선택할 수 있다는 측면에서는 비슷한 컨트롤이다. ListBox는 전체 목록을 보여주면서 아이템을 선택할 수 있으며, ComboBox는 하나의 목록만 보여주면서 드롭다운 방식으로 선택해서 사용할 수 있다.  
  
**□ ListBox와 ComboBox의 차이점**  
◇ ListBox는 전체 목록을 보여주면서 아이템을 선택해서 사용  
◇ ComboBox는 하나의 목록만 보여주면서 드롭다운 방식으로 선택해서 사용  
  
ListBox와 ComboBox에서 제일 중요한 것은 선택된 아이템이다. 선택된 아이템을 얻어내기 위해서는 SelectedItem 속성을 이용한다.  
  
**□ obj.SelectedItem**  
◇ 선택된 항목  
  
선택된 아이템의 인덱스를 얻어내기 위해서는 SelectedIndex를 이용하면 된다. 이 인덱스를 수정하면 선택 아이템을 변경시킬 수도 있다. 선택되지 않은 상태는 -1이며, 첫번째 아이템의 선택은 0이다.  
  
**□ int i = obj.SelectedIndex;**  
◇ 선택된 항목의 인덱스  
  
ListBox의 경우 MultiColumn이 true인 경우 여러 개의 아이템을 선택할 수 있기 때문에 선택된 아이템 또한 여러 개일 수 있다. 이럴 경우에는 Collection 형식으로 아이템과 아이템 인덱스를 얻어내면 된다.  
  
**□ obj.SelectedItems;**  
◇ 선택된 항목들  
  
**□ obj.SelectedIndice;**  
◇ 선택된 항목의 인덱스들  
  
전체 아이템을 통째로 얻어내야 할 경우가 있다. 이럴 경우엔 Items 속성을 이용한다.  
  
**□ obj.Items**  
◇ 리스트 박스의 모든 항목들   
  
**□ string str = obj.Text;**  
◇ 선택된 항목의 텍스트  
  
ListBox와 ComboBox에서 무엇보다 중요한 것은 아이템의 추가와 삭제이다. 추가 또는 삭제를 할 때에는 ListBox나 ComboBox의 Items 속성을 이용한다.  
  
**□ ListBox와 ComboBox  추가 삭제**  
◇ 추가 삭제를 할 때에는 ListBox나 ComboBox의 Items 속성을 이용한다.  
  
끝부분에 추가하는 함수는 Add()이며, 인덱스의 값으로 중간에 추가하기 위해서는 Insert()를, 전체 내용을 삭제하기 위해서는 Clear()를, 아이템의 참조값을 이용해서 삭제할 때는 Remove()를, 인덱스로 삭제하기 위해서는 RemoveAt() 함수를 이용한다.  
  
  
[표 10\_12] ListBox와 ComboBox의 추가 삭제 함수  
  
ComboBox의 경우에는 텍스트 상자를 포함하고 있으므로 목록에 없는 항목을 입력할 수 있다. 그리고 DropDownStyle 속성이 DropDownList로 설정되어 있는 경우에는 항목의 첫 글자만 입력하면 컨트롤에서 해당 항목이 자동으로 선택된다. DropDownStyle 속성이 Simple로 설정된 경우는 전체 목록이 표시된다.  
  
**□ ComboBox의 DropDownStyle 속성**  
◇ DropDownStyle 속성이 DropDownList로 설정되어 있는 경우에는 항목의 첫 글자만 입력하면 컨트롤에서 해당 항목이 자동으로 선택된다.   
◇ DropDownStyle 속성이 Simple로 설정된 경우는 전체 목록이 표시된다.  
  
**□ ListBox와 ComboBox의 이벤트**  
◇ 선택된 아이템이 변경될 때 작업을 하면 되기 때문에 SelectedIndexChanged 이벤트나 SelectedValueChanged 이벤트가 많이 사용된다.  
  
ListBox와 ComboBox는 선택된 아이템이 변경될 때 작업을 하면 되기 때문에 SelectedIndexChanged 이벤트나 SelectedValueChanged 이벤트가 많이 사용된다.  
  
위에서 설명한 ListBox의 속성과 함수를 이용해서 ListBox의 예를 만들어보자.

**§ chap10\ListBoxProject\ListBoxForm.cs**

|  |  |
| --- | --- |
| 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 | 癤퓎sing System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Text;  using System.Windows.Forms;    namespace ListBoxProject{      public partial class ListBoxForm : Form{          public ListBoxForm(){              InitializeComponent();          }          private void btnAdd\_Click(object sender, EventArgs e){              if (!String.IsNullOrEmpty(this.txtMan.Text)){                  this.lstMan.Items.Add(this.txtMan.Text); //由ъ뒪??諛뺤뒪???곗씠?곗텛媛                  this.txtMan.Clear();              }          }          private void btnDelete\_Click(object sender, EventArgs e){              if (this.lstMan.SelectedIndex != -1){                  this.lstMan.Items.RemoveAt(this.lstMan.SelectedIndex); //?꾩튂瑜??댁슜?댁꽌 ?곗씠????젣              }          }          private void btnAll\_Click(object sender, EventArgs e){              StringBuilder sb = new StringBuilder();              for (int i = 0; i < lstMan.Items.Count; i++){                  sb.Append(lstMan.Items[i] + "\n");              }              this.lstMan.SetSelected(0, true); //0踰덉㎏ ?꾩씠???좏깮              /\*foreach (object obj in this.lstMan.Items){                  sb.Append(obj + "\n");              }\*/              MessageBox.Show(sb.ToString());          }      }  } |



**10.5.5 RadioButton**  
  
하나의 그룹 내에서 여러 개의 RadioButton이 있을 경우 하나를 선택하면 다른 RadioButton이 자동으로 선택이 해제된다. RadioButton 및 CheckBox 컨트롤의 경우 선택하거나 취소하는 면에서는 비슷하지만 RadioButton은 하나의 그룹 내에서 하나만 선택할 수 있으며, CheckBox는 여러 개를 동시에 선택할 수 있다.  
  
**□ RadioButton과 그룹의 관계**  
◇ 하나의 그룹에서 하나의 RadioButton이 선택된다.  
  
하나의 Form은 하나의 그룹으로 지정되며, Form 내에서 여러 개의 그룹을 만들고 싶다면 GroupBox 또는 Panel 컨트롤과 같은 컨테이너에 배치하면 된다.   
  
**□ RadioButton의 그룹**  
◇ 하나의 Form은 하나의 그룹으로 지정  
  
**□ RadioButton의 다중 그룹**  
◇ GroupBox 또는 Panel에 배치할 경우 각각의 그룹으로 된다.  
  
RadioButton의 핵심 속성은 Checked 속성이다. 체크되어 있다면 true를, 체크되어 있지 않다면 false의 값을 가진다.  
  
**□ bool Checked 속성**  
◇ 선택되어 있는지 여부를 나타내는 값을 가져오거나 설정  
  
RadioButton에서 Checked 속성의 값이 변경되면 CheckedChanged 이벤트가 발생한다. RadioButton에서의 작업은 주로 CheckedChanged 이벤트에서 이루어진다.  
  
**□ CheckedChanged**  
◇ Checked 속성 값이 변경되면 발생한다.   
  
RadioButton을 이용하는 예제를 살펴보자. 4개의 RadioButton을 GroupBox에 배치하고 각각의 RadioButton의 이벤트 핸들러를 Color\_Changed로 설정하고 있다. 그리고 각각의 RadioButton이 눌러질 때마다 RadioButton을 확인한 후 폼의 배경색을 변경하는 예이다.

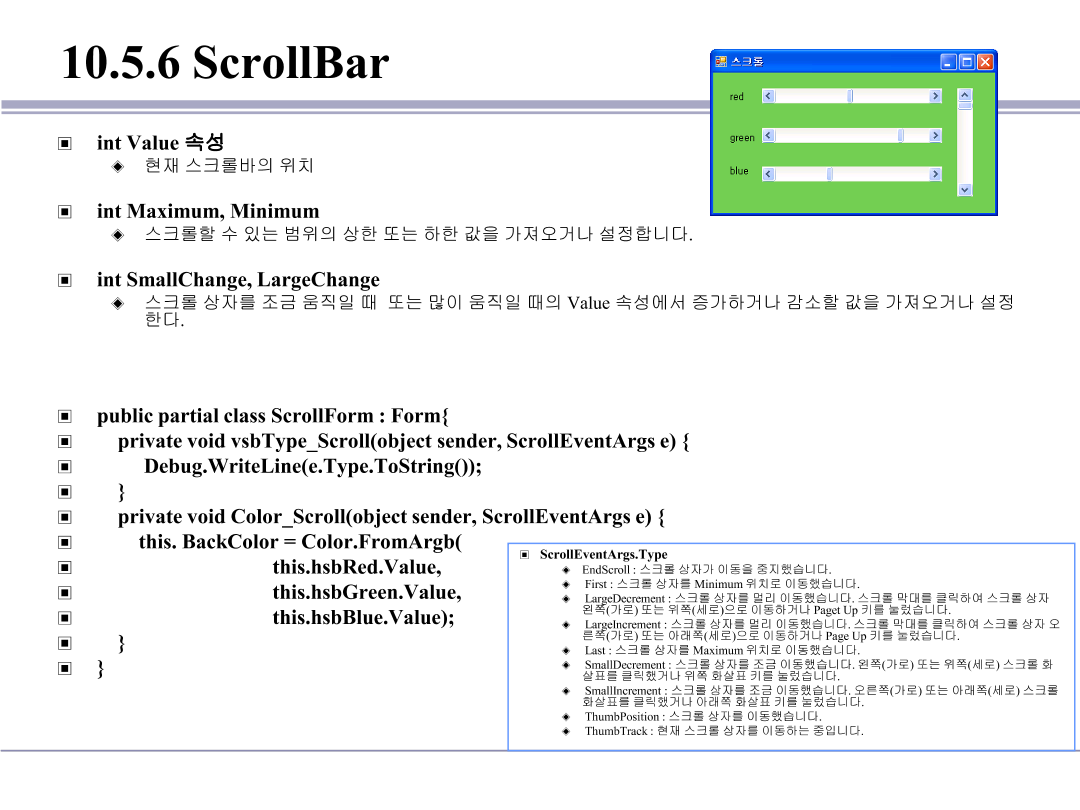
**§ chap10\RadioButtonProject\RadioButtonForm.cs**

|  |  |
| --- | --- |
| 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 | 癤퓎sing System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Text;  using System.Windows.Forms;    namespace RadioButtonProject{      public partial class RadioButtonForm : Form{          public RadioButtonForm(){              InitializeComponent();          }          private void Color\_Changed(object sender, EventArgs e){              if(Object.ReferenceEquals(sender, rdoDefault)){                  this.BackColor = SystemColors.Control;              }else if(Object.ReferenceEquals(sender, rdoRed)){                  this.BackColor = Color.Red;              }else if(Object.ReferenceEquals(sender, rdoGreen)){                  this.BackColor = Color.Green;              }else if(Object.ReferenceEquals(sender, rdoBlue)){                  this.BackColor = Color.Blue;              }          }      }  } |

4개의 RadioButton에 하나의 이벤트 핸들러를 사용했기 때문에 Color\_Changed 이벤트 핸들러에서는 어떤 RadioButton이 눌러졌는지 알아내야 한다. 이 때 눌러진 버튼을 알아내는 방법은 다음과 같다.  
  
**□ 이벤트를 발생시킨 컨트롤 알아내는 방법**  
◇ Object.ReferenceEquals() 함수를 이용해서 sender와 각각의 RadioButton의 참조값을 비교하면 된다.  
  
**▒ 비교하는 방법**

|  |  |
| --- | --- |
| 1  2  3 | if(Object.ReferenceEquals(sender, rdoDefault)){               this.BackColor = SystemColors.Control;   } |

위의 방법으로 4개의 RadioButton을 전부 비교해서 어떤 RadioButton이 눌러졌는지 확인하고 있다.

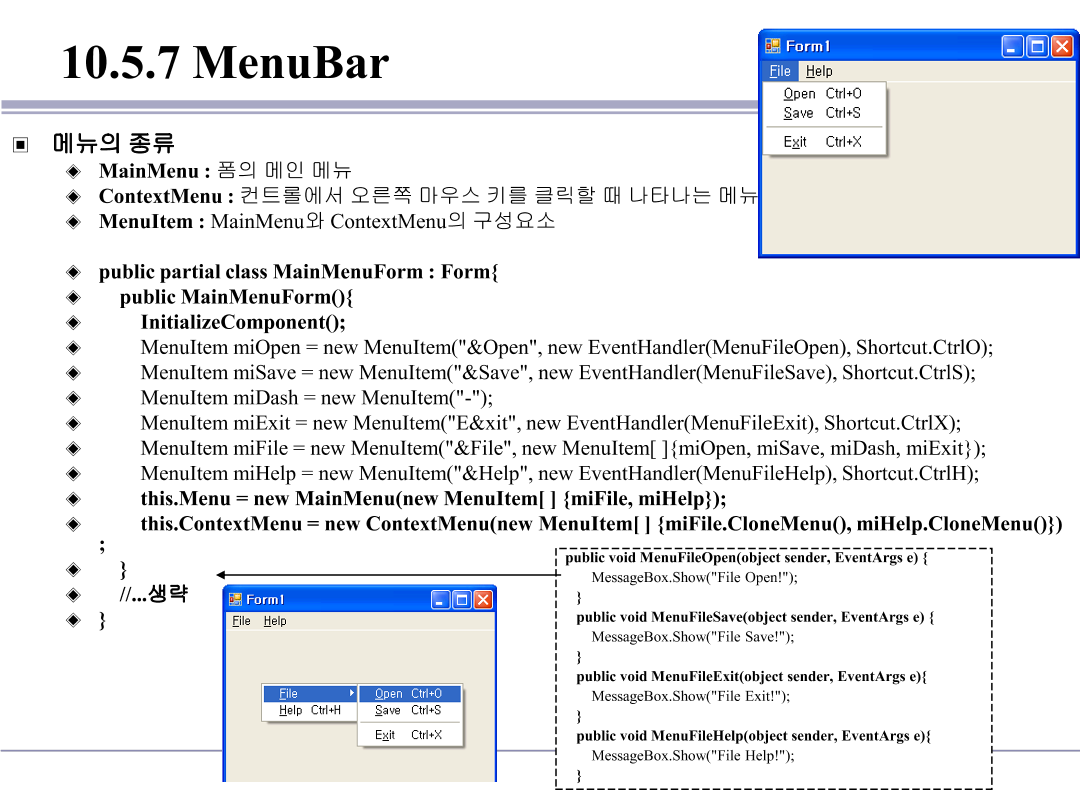


**10.5.6 ScrollBar**  
  
스크롤바는 가로 또는 세로로 스크롤해서 많은 양의 정보를 쉽게 탐색하는데 사용된다. 스크롤바는 수평 스크롤바인 HScrollBar와 수직 스크롤바인 VScrollBar 두 종류가 독립적으로 존재한다.  
  
사용자는 마우스를 클릭해서 스크롤 막대의 값을 변경할 수 있을 뿐 아니라 스크롤 막대에서 원하는 위치로 스크롤 상자를 끌어올 수도 있다.   
  
스크롤바에서의 스크롤된 값은 Value 속성으로 알 수 있으며, 이 값은 Minimum 속성과 Maximum 속성 사이의 값을 가지게 된다.  
  
**□ int Value 속성**  
◇ 현재 스크롤바의 위치  
  
**□ int Maximum, Minimum**  
◇ 스크롤할 수 있는 범위의 상한 또는 하한 값을 가져오거나 설정합니다.    
◇ 스크롤바를 만들면 Maximum과 Minimum 값을 설정하고 사용하는 것이 기본이다.  
  
사용자가 Page Up 키 또는 Page Down 키를 누르거나 스크롤 막대 트랙에서 스크롤 상자의 한 쪽 가장자리를 클릭하면 LargeChange 속성에 설정된 값에 따라 Value 속성이 변경된다. 사용자가 화살표 키 중 하나를 누르면 SmallChange 속성에 설정된 값에 따라 Value 속성이 변경된다.  
  
**□ int SmallChange, LargeChange**  
◇ 스크롤 상자를 조금 움직일 때  또는 많이 움직일 때의 Value 속성에서 증가하거나 감소할 값을 가져오거나 설정한다.  
  
스크롤바에서 스크롤이 되면 Scroll 이벤트가 발생한다. Scroll 이벤트를 사용하면 스크롤 막대를 끌 때 스크롤 막대의 값에 액세스할 수 있다.  
  
스크롤바를 이용하는 예제를 살펴보자. 3개의 HScrollBar를 배치하고 각각의 HScrollBar의 Scroll 이벤트 핸들러를 Color\_Scroll로 설정하고 있다. 그리고  각각의 HScrollBar가 눌러질 때마다 눌러진 ScrollBar의 값을 확인해서 폼의 배경색을 변경하는 예이다.

**§ chap10\ScrollProject\ScrollForm.cs**

|  |  |
| --- | --- |
| 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 | 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.Diagnostics;    namespace ScrollProject{      public partial class ScrollForm : Form{          public ScrollForm(){              InitializeComponent();          }          private void vsbType\_Scroll(object sender, ScrollEventArgs e){              /\*switch (e.Type){                  case ScrollEventType.SmallIncrement:                      this.Text = "SmallIncrement";break;                  case ScrollEventType.SmallDecrement:                      this.Text = "SmallDecrement";break;                  case ScrollEventType.LargeIncrement:                      this.Text = "LargeIncrement"; break;                  case ScrollEventType.LargeDecrement:                      this.Text = "LargeDecrement"; break;                  case ScrollEventType.ThumbTrack:                      this.Text = "ThumbTrack"; break;                  case ScrollEventType.ThumbPosition:                      this.Text = "ThumbPosition"; break;              }\*/              Debug.WriteLine(e.Type.ToString());          }          private void Color\_Scroll(object sender, ScrollEventArgs e){              this.BackColor = Color.FromArgb(this.hsbRed.Value,                                          this.hsbGreen.Value, this.hsbBlue.Value);          }      }  } |

Scroll 이벤트의 Type 속성을 이용하면 스크롤바에서 발생하는 동작의 상태를 얻을 수 있다.



**10.5.7 MenuBar**  
  
메뉴를 만드는 방법은 툴을 이용해서 만드는 MenuStrip 방식과 코드를 이용하는 두가지 방법이 있다. 두가지 방법 모두 사용되기 때문에 알아둘 필요가 있다.  
  
**□ 메뉴를 만드는 방법**  
◇ MenuStrip 방식  
◇ 코드를 이용해서 만드는 방식  
  
메뉴의 종류에는 MainMenu와 ContextMenu가 있다. MainMenu는 폼의 상단부에 장착되는 메뉴를 말하며, ContextMenu는 컨트롤의 오른쪽 마우스 키를 누를 때 나타나는 메뉴를 말한다.  
  
**□ 메뉴의 종류**  
◇ MainMenu  
― 폼의 메인 메뉴  
◇ ContextMenu  
― 컨트롤에서 오른쪽 마우스 키를 클릭할 때 나타나는 메뉴  
― 컨트롤이면 어디에나 사용할 수 있는 메뉴  
  
MainMenu와 ContextMenu는 MenuItem으로 구성되며, MenuItem을 조합해서 새로운 MenuItem을 구성한다. MainMenu와 ContextMenu의 가장 큰 차이점은 바로 어느 컨트롤을 대상으로 하느냐이다. MainMenu는 폼을 대상으로 하지만 ContextMenu의 경우 화면에 나타나는 대부분의 컨트롤에 ContextMenu를 장착할 수 있다.  
  
**□ MainMenu와 ContextMenu의 차이점**  
◇ MainMenu는 폼을 대상으로 하지만 ContextMenu의 경우 화면에 나타나는 대부분의 컨트롤에 ContextMenu를 장착할 수 있다.  
  
먼저 메뉴 아이템을 만드는 방법부터 살펴보자. 보통의 경우 MenuItem에는 3개의 매개변수를 가진 생성자를 많이 사용한다.   
  
**▒ MenuItem의 생성**

|  |  |
| --- | --- |
| 1  2  3  4  5 | MenuItem miOpen = new MenuItem("&Open", new EventHandler(MenuFileOpen), Shortcut.CtrlO);  MenuItem miSave = new MenuItem("&Save", new EventHandler(MenuFileSave), Shortcut.CtrlS);  MenuItem miDash = new MenuItem("-");  MenuItem miExit = new MenuItem("E&xit", new EventHandler(MenuFileExit), Shortcut.CtrlX);  MenuItem miHelp = new MenuItem("&Help", new EventHandler(MenuFileHelp), Shortcut.CtrlH); |

MenuItem의 첫번째 매개변수인 "&Open"은 표시될 텍스트이며, 메뉴가 활성화된 상태에서 &가 붙은 문자를 누를 경우 키보드로 메뉴를 선택할 수 있다. 두번째 매개변수인 new EventHandler(MenuFileOpen)은 MenuItem이 클릭되었을 때 호출될 이벤트 핸들러이다. 그리고 세번째 매개변수 Shortcut.CtrlO는 단축키를 의미한다. 그리고 MenuItem("-")은 메뉴에서 하나의 줄을 표시할 때 사용하는 MenuItem이다.  
  
MenuItem의 경우 여러 개의 MenuItem을 묶어서 새로운 MenuItem으로 만들 수 있다.   
  
**▒ MenuItem을 묶어서 새로운 MenuItem 생성**

|  |  |
| --- | --- |
| 1 | MenuItem miFile = new MenuItem("&File", new MenuItem[ ]{miOpen, miSave, miDash, miExit}); |

이렇게 생성된 MenuItem을 이용해서 폼의 MainMenu에 등록할 수 있다.  
  
**▒ 폼의 MainMenu에 등록하는 방법**

|  |  |
| --- | --- |
| 1 | this.Menu = new MainMenu(new MenuItem[ ] {miFile, miHelp}); |

만약 MainMenu에서 사용한 MenuItem을 ContextMenu에 사용하기 위해서는 다음과 같이 CloneMenu() 함수로 복제한 후 사용해야 한다.  
  
**▒ ContextMenu 생성**

|  |  |
| --- | --- |
| 1 | this.ContextMenu = new ContextMenu(new MenuItem[ ] {miFile.CloneMenu(), miHelp.CloneMenu()}); |

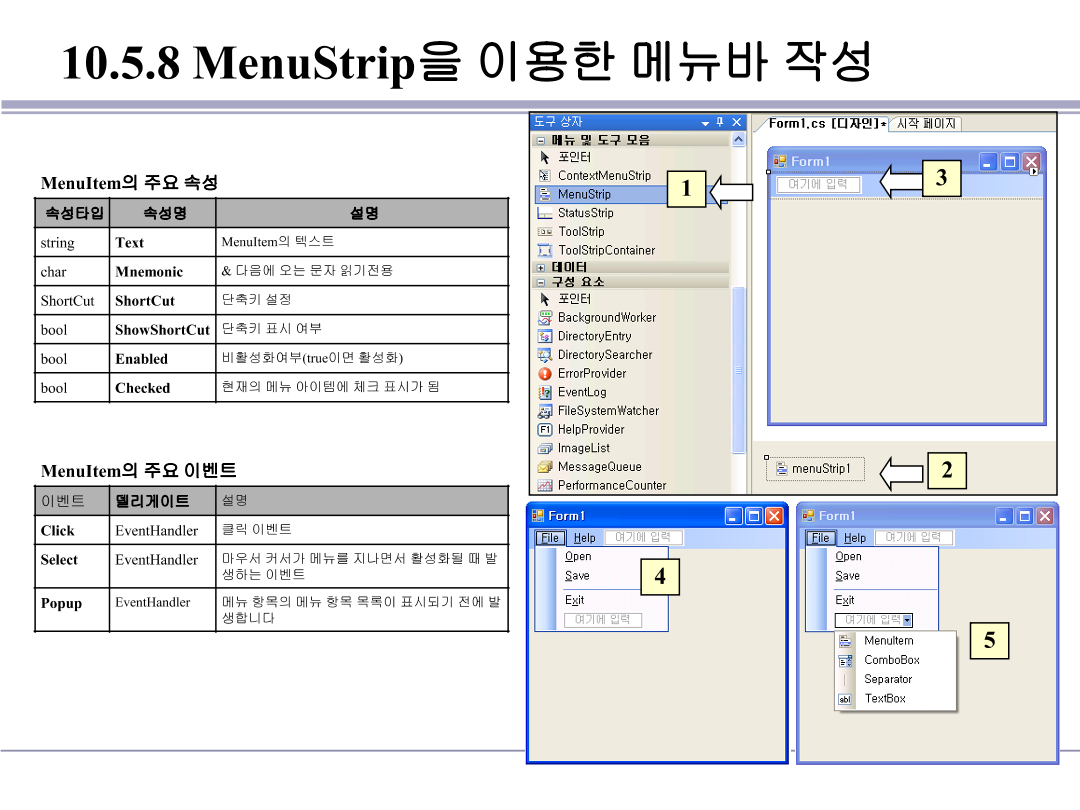
**☞ 참고**

|  |
| --- |
| MainMenu에 표시된 MenuItem 개체를 ContextMenu에서 다시 사용하려면 MenuItem 클래스의 CloneMenu() 함수를 이용해서 해당 메뉴의 복사본을 만들어서 사용해야 한다. |

만약 CloneMenu()를 사용하지 않고 그냥 사용하면 ContextMenu와 MainMenu 둘 중 하나는 나타나지 않는 현상이 발생한다.

**§ chap10\MainMenuProject\MainMenuForm.cs**

|  |  |
| --- | --- |
| 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 | using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Text;  using System.Windows.Forms;    namespace MainMenuProject{      public partial class MainMenuForm : Form{          public MainMenuForm(){              InitializeComponent();                MenuItem miOpen, miSave, miDash, miExit, miFile, miHelp;              miOpen = new MenuItem("&Open", new EventHandler(MenuFileOpen), Shortcut.CtrlO);              miSave = new MenuItem("&Save", new EventHandler(MenuFileSave), Shortcut.CtrlS);              miDash = new MenuItem("-");              miExit = new MenuItem("E&xit", new EventHandler(MenuFileExit), Shortcut.CtrlX);              miFile = new MenuItem("&File", new MenuItem[]{miOpen, miSave, miDash, miExit});              miHelp = new MenuItem("&Help", new EventHandler(MenuFileHelp), Shortcut.CtrlH);                this.ContextMenu = new ContextMenu(new MenuItem[]{miFile.CloneMenu(), miHelp.CloneMenu()});              this.Menu = new MainMenu(new MenuItem[] { miFile, miHelp });          }          public void MenuFileOpen(object sender, EventArgs e){              MessageBox.Show("File Open!");          }          public void MenuFileSave(object sender, EventArgs e){              MessageBox.Show("File Save!");          }          public void MenuFileExit(object sender, EventArgs e){              MessageBox.Show("File Exit!");          }          public void MenuFileHelp(object sender, EventArgs e){              MessageBox.Show("File Help!");          }      }  } |



**10.5.8 MenuStrip을 이용한 메뉴바 작성**  
  
앞에서 우리는 코드를 이용해서 MainMenu와 ContextMenu를 구축하는 방법에 대해서 알아보았다. Visual Studio에서는 메뉴 구성을 편하게 하기 위해서 MainMenu 대신 MenuStrip을 제공하며, ContextMenu 대신 ContextMenuStrip을 제공한다.   
  
**□ Visual Studio에서의 메뉴 구성**  
◇ MainMenu 대신 MenuStrip을 제공한다.  
◇ ContextMenu 대신 ContextMenuStrip을 제공한다.  
  
MenuStrip과 ContextMenuStrip은 닷넷 2.0부터 지원하고 있으며, 화면을 보면서 메뉴를 구축할 수 있는 특징이 있다. 먼저 도구상자에 있는 MenuStrip을 폼에 추가해야 한다. 생성된 MenuStrip의 [여기에 입력]이라고 표시되어 있는 부분에 텍스트를 입력하면 새로운 MenuItem이 생성되며, 데이터를 입력하면 같은 레벨 또는 하위에 입력할 수 있는 [여기에 입력]이 나타난다.  
  
**□ [여기에 입력]**  
◇ MenuStrip과 ContextMenuStrip의 [여기에 입력]에 텍스틀 입력하면 MenuItem이 자동으로 생성된다.  
  
[여기에 입력]에 마우스를 올리면 드롭다운 박스가 나타나며, 이 드롭다운 박스를 클릭하면 삽입할 수 있는 컨트롤들을 보여준다. 단순히 [여기에 입력] 부분에 텍스트를 입력하면 자동으로 MenuItem이 추가된다.  
  
**□ [여기에 입력]의 드롭다운 박스**  
◇ 삽입할 수 있는 컨트롤들을 보여준다.  
◇ 텍스트를 입력하면 MenuItem이 추가된다.  
  
MenuStrip과 ContextMenuStrip에서 이벤트를 추가하기 위해서는 원하는 MenuItem을 선택한 후 속성창의 이벤트 부분에서 추가하면 된다.  
  
MenuStrip과 ContextMenuStrip은 화면을 보면서 누구나 메뉴를 구축할 수 있는 장점이 있다. MenuStrip을 이용하는 방법이 훨씬 쉽다. 하지만 실제 코딩에서 동적으로 메뉴를 구축할 상황이 발생할 수 있다. 이러한 경우를 대비해 두가지 방법을 모두 알고 있어야 한다.