



C#

-CHAPTER 11-

SOUL SEEK



목차

1. WinForm



The background is a dark blue gradient with a large, faint, light blue circle in the center. In the four corners, there are white, stylized circuit board traces. These traces consist of straight lines of varying lengths and angles, some ending in small white circles, resembling electronic components or connectors.

WINFORM

1. WINFORM

WinForm

- 폼 디자이너 툴을 제공해서 프로그래머가 그림 그리듯 **UI**를 만들 수 있게 제공 한 것.
- **What You See Is What You Get(WYSIWYG)**방식의 개발 방법.
- 반응할 수 있는 컨트롤러 들을 배치해서 이벤트를 체크하고 **UI**에 특정 동작에 대한 이벤트 함수를 연결해두면 스스로 해당함수를 호출해서 내부 코드를 실행하는 방법
 - ➔ 이런 방식이 익숙해지면 차후에 **Unity**에서 함수를 **GameObject**에 연결해서 사용하는 것에 대한 이해에 도움이 많이 된다.
- 이런 **GUI**를 편성하고 도움을 주는 툴들은 항상 내부 네임스페이스나 클래스 **API**의 도움을 받고 있기 때문에 콘솔 환경에서도 바로 제작이 가능하다.

Win32 API로 윈도우를 만드는 절차 vs WinForm으로 윈도우를 만드는 절차

Win32 API 절차

1. 윈도우 클래스(**OOP**의 클래스와는 다르다.)를 정의한다.
2. 정의된 윈도우 클래스를 등록한다.
3. 윈도우를 생성한다.
4. 생성한 윈도우를 사용자에게 보여준다.
5. 메시지 루프를 돌면서 프로그램을 시작한다.

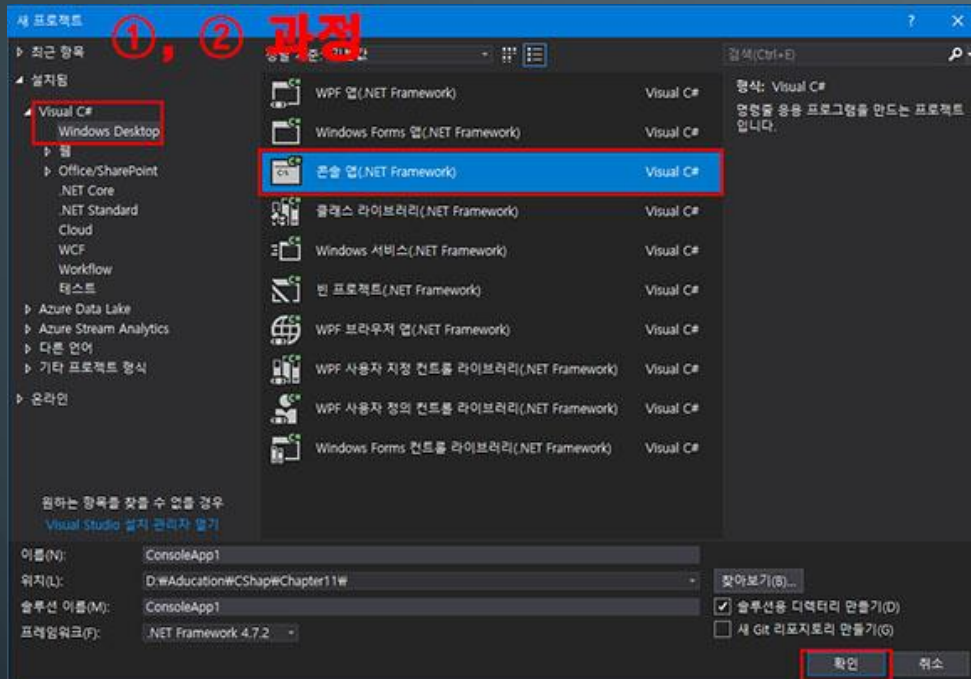
WinForm 절차

1. **System.Windows.Forms.Form** 클래스에서 파생된 윈도우 폼 클래스를 선언한다. ➔ 상속.
2. 1번에서 만든 클래스의 인스턴스를 **System.Windows.Forms.Application.Run()** 메소드에 매개 변수로 넘겨 호출한다.

- 콘솔프로그램에서 직접 호출해서 사용하는 방법, **UI**를 배치해서 코드를 연결하는 방법, 앞서 배운 비동기를 이용한 실습 순서의 학습으로 진행할 것이다.

1. WINFORM

기본 Window



①, ② 과정

```
using System;
using System.Windows.Forms;

namespace SimpleWinForms
{
    // MainApp이 System.Windows.Forms 네임스페이스의
    // Form 클래스로부터 상속받도록 선언.
    class MainApp : Form
    {
        static void Main(string[] args)
        {
            // Application.Run() 메소드에 MainApp의 인스턴스를
            // 매개 변수로 넘겨 호출한다.
            Application.Run(new MainApp());
        }
    }
}
```

④ 과정

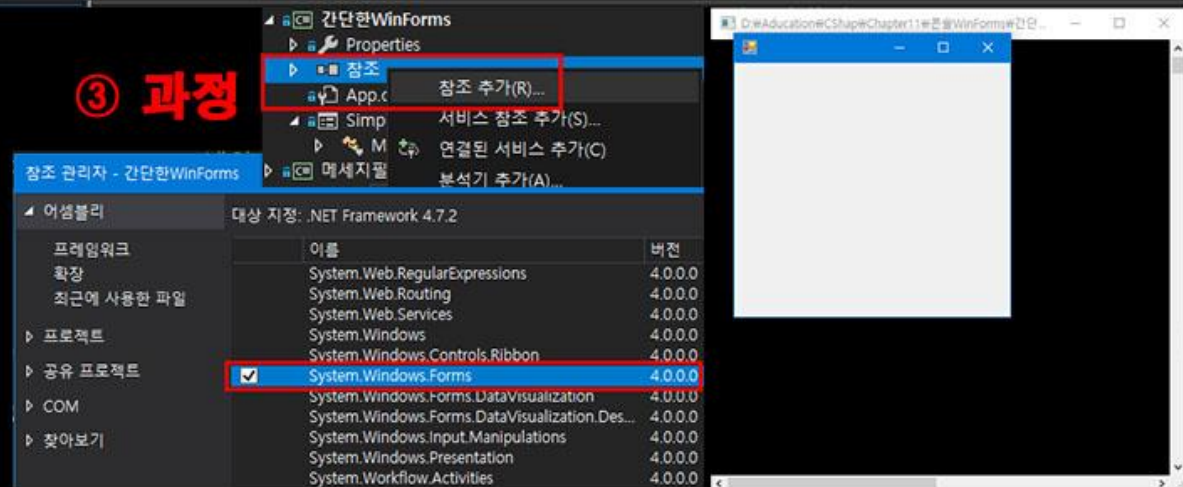
1. 새 프로젝트 만들기에서 [Visual C#]

-> [Window Desktop] 선택

2. 왼쪽의 [콘솔 앱(.NET Framework)] 선택하고
경로와 솔루션 명, 프로젝트 이름을 기입하고
프로젝트를 생성한다.

3. 프로젝트에서 참조추가로 System.Windows.
Forms 어셈블리를 추가한다.

4. 코드를 작성하고 실행한다.



③ 과정

1. WINFORM

Application 클래스 – project Application클래스 참조!

- 윈도우 응용 프로그램을 시작하고 종료 시키는 메소드를 제공한다.
- 윈도우 메시지를 받아서 처리한다.
- **Application.Run()**으로 **Form**으로 정의한 **Window**를 실행한다.
- **Application.Exit()**으로 종료 처리를 한다.
 - **Exit()** 메소드는 호출되는 즉시 종료하는 것이 아니라 모든 윈도우를 닫은 뒤 **Run()** 메소드가 반환할 수 있도록 해주기 때문에 **Run()** 메소드 다음 열로 코드 진행이 되므로 **Run()** 이후에 자원에 대한 해제 코드가 들어가면 종료하면서 메모리 자원을 사용하는 것을 반환 할 수 있게 된다.

class MainApp

```
{  
    static void Main(string[] args)  
    {  
        MyForm form = new MyForm();  
        // Form 클래스는 여러 가지 이벤트를 정의하고 있는데, 그 중 Click 이벤트는  
        // 윈도우를 클릭했을 때 발생하는 이벤트이다.  
        // 따라서 이 코드는 윈도우를 클릭하면 Application.Exit()를 호출하도록 한다.  
        form.Click += new EventHandler((sender, EventArgs) =>  
            {  
                Application.Exit();  
            });  
        Application.Run(form);  
    }  
}
```


1. WINFORM

Message Filtering – project 메시지 필터 참고!

- 윈도우 기반의 응용 프로그램들은 이벤트 기반(**Event Driven**) 방식으로 만들어 진다.
- 사용자가 마우스나 키보드 같은 하드웨어를 제어하면 인터럽트가 발생하고, 이 인터럽트를 윈도우 운영체제가 받아들인다 그리고 운영체제는 다시 이 인터럽트를 바탕으로 윈도우 메시지를 만든 뒤 이벤트를 받아야하는 응용 프로그램은 핸들로 구분해서 전달해 준다.
- **Application.AddMessageFilter()** 메소드로 필터링을 한다.
 - ➔ 매개 변수로 **IMessageFilter** 인터페이스를 구현하는 파생 클래스의 인스턴스를 매개 변수로 받으며, **IMessageFilter**는 **PreFilterMessage()** 메소드를 구현 할 것을 요구한다.

```
public class MessageFilter : IMessageFilter
{
    public bool PreFilterMessage(ref Message m)
    {
        // 마우스 이동부터 마우스의 왼쪽, 오른쪽, 가운데 버튼 동작,
        // 마우스 휠 굴림 메시지를 모두 걸러낸다.
        if(m.Msg >= 0x200 && m.Msg <= 0x20E)
        {
            Console.WriteLine("발생한 메시지: " + m.Msg);
            return true;
        }
        return false;
    }
}
```

1. WINFORM

Form으로 이벤트 처리하기 – project Form이벤트 참조!

- **Win32 API**처럼 우리가 직접적으로 메시지를 알 필요는 없다.
→ **Form**을 비롯한 **WinForm**의 윈도우와 컨트롤 클래스들이 윈도우 메시지를 포장하여 이벤트로 구현 해 놓았다.
- 각 클래스마다 제공하는 이벤트들은 각각 다르기 때문에 사용시 지원 여부를 판별할 필요가 있다.

Form클래스의 **MouseDown** 이벤트에 등록.

```
class MyForm : Form
{
    // 이벤트 처리기 선언
    private void Form_MouseDown(object sender, MouseEventArgs e)
    {
        MessageBox.Show("안녕하세요!");
    }

    public MyForm()
    {
        // 이벤트를 처리기를 이벤트에 연결
        this.MouseDown += new MouseEventHandler(this.Form_MouseDown);
    }
}
```


1. WINFORM

- **MouseEventHandler** 델리게이트로 선언되어 있다.
- **MouseDown**은 **MouseEventHandler**를 이용한 이벤트로 만들어져 있다.
- 그렇기 때문에 앞의 예제에서 보여지는 **MouseDown**의 매개변수와 반환형이 어떤 형태가 되어있는지 알 수 있고 반대로 어떤 델리게이트인지 예측도 가능하다.

```
public event MouseEventHandler MouseDown;  
public delegate void MouseEventHandler(object sender, MouseEventArgs e);
```

MouseEventArgs 형식은 프로퍼티들을 가지고 있다.

프로퍼티	설명
Button	마우스의 어떤 버튼(왼쪽, 오른쪽, 또는 가운데)에서 이벤트가 발생했는지를 나타낸다.
Clicks	마우스의 버튼을 클릭한 횟수를 나타낸다. 사용자가 더블 클릭했을 때만 어떤 기능을 수행하고 싶다면 이 값이 2 일 경우를 확인하면 된다.
Delta	마우스 휠의 회전 방향과 회전한 거리를 나타낸다.
X	마우스 이벤트가 발생한 폼 또는 컨트롤상의 x 좌표를 나타낸다.
Y	마우스 이벤트가 발생한 폼 또는 컨트롤상의 y 좌표를 나타낸다.

1. WINFORM

From Style 설정

- **From** 클래스는 윈도우의 모양을 결정 짓는 크기, 배경색, 전경색, 투명도, 제목, 폰트 등 여러가지 프로퍼티를 갖고 있다.

윈도우 모습을 결정하는 **From** 프로퍼티

종류	프로퍼티	설명
크기	Width	창의 너비를 나타낸다.
	Height	창의 높이를 나타낸다.
색깔	BackColor	창의 배경 색깔을 나타낸다.
	BackgroundImage	창의 배경 이미지를 나타낸다.
	Opacity	창의 투명도를 나타낸다.
스타일	MaximizeBox	최대한 버튼을 설치할 것인지 여부를 나타낸다.
	MinimizeBox	최소화 버튼을 설치할 것인지 여부를 나타낸다.
	Text	창의 제목을 나타낸다.

1. WINFORM

윈도우 크기 조절 – **project Form** 크기조절 참조!

```
MainApp form = new MainApp();  
Form.Width = 300;  
Form.Height = 200;
```

윈도우 배경설정 변경 – **project Form** 백그라운드 설정 참조!

System.Drawing 네임스페이스의 **Color** 형식을 이용해 적용할 수 있다.

```
Form form = Form();
```

```
// Color 구조체에는 Red, Green, Blue 등등을 포함한 기본적으로 바로 쓸 수 있는 색상을 미리  
// 정의한 것을 사용할 수 있다.
```

```
form.BackColor = Color.Red;
```

```
// FromArgb() 메소드를 이용한다 첫 번째 매개 변수는 투명도를 나타내는 Alpha, 두 번째는 Red, 세 번째는 Green,  
// 네 번째는 Blue 값을 나타내며 각 매개 변수는 0부터 255 사이의 값을 가질 수 있다.
```

```
form.BackColor = Color.FromArgb(255, 255, 0, 0); // 불투명한 빨간색
```

```
// 윈도우 창의 투명도 조절
```

```
Form.Opacity = 0.87;
```

```
form.Opacity = 1.00;
```

```
// 윈도우 배경 이미지 적용
```

```
form.BackgroundImage = Image.FromFile("파일경로");
```

1. WINFORM

윈도우 이름, 최소화 최대화의 아이콘 박스 컨트롤 – **project Form**스타일 참조!

// 최소화, 최대화 아이콘 활성화, 비활성화

```
Form form = new Form();  
form.MaximizeBox = true;  
form.MinimizeBox = false;
```

// 윈도우 이름 변경

```
Form form = new Form();  
Form.Text = “Form 프로퍼티 테스트”;
```

Form에 컨트롤 적용하기 – **project Form**컨트롤러 참조!

Controll?

- **Window OS**가 제공하는 사용자 인터페이스 요소를 말한다.
- 응용 프로그램을 제어하는데 사용하는 도구라는 의미에서 붙여진 이름이다.
- 버튼, 텍스트 박스, 콤보 박스 등이 이에 해당한다.
- 유닉스의 모티브나 자바의 스윙에서는 **Widget**이라고 부르고 델파이에서는 **VCL**이라고 부른다.

코드작성 순서

- 컨트롤의 인스턴스를 생성하고, 컨트롤의 프로퍼티 값을 지정한다.
- 컨트롤의 이벤트에 이벤트 처리기를 등록하고 **Form**에 컨트롤을 추가한다.

1. WINFORM

Step1. 컨트롤의 인스턴스 생성

→ 컨트롤은 **System.Windows.Forms.Control**을 상속한다.

```
Button button = new Button();
```

Step2. 컨트롤의 프로퍼티에 값 지정

→ 인스턴스를 만들었으면 각 프로퍼티에 값을 지정해서 컨트롤 모양과 위치를 결정한다.

```
button.Text = "Click Me!";
```

```
button.Left = 100;
```

```
button.Top = 50;
```

Step3. 컨트롤 이벤트에 이벤트 처리기 등록

```
button.Click += (object sender, EventArgs e) =>  
    {  
        MessageBox.Show( "딸깍!" );  
    }
```

Step4. Form에 컨트롤 추가

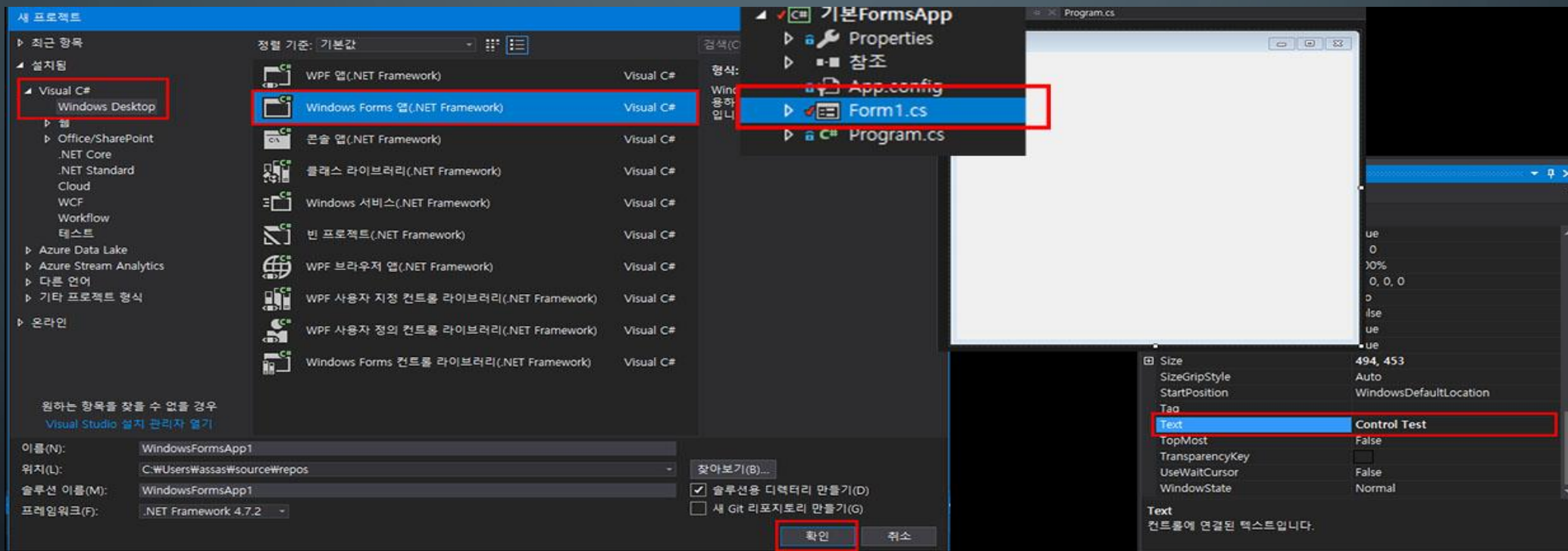
```
MainApp form = new MainApp();
```

```
Form.Controls.Add(button);
```

1. WINFORM

WinForm UI구성

- **UI**를 편리하게 구축하기 위해 만들어진 폼 디자이너
- **VisualStudio IDE**이 일부로, 우리가 코드를 통해 컨트롤을 폼 위에 배치하고 프로퍼티를 변경했던 작업을 통해 만든 것보다 품질이 우수한 **UI**를 제공한다.



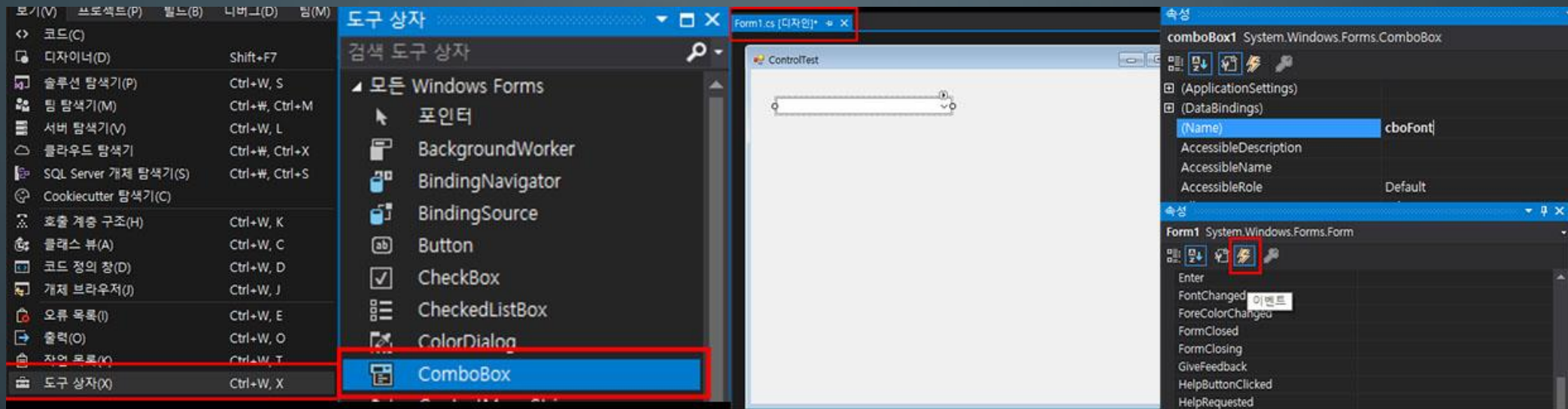
1. [Windows Forms 앱]으로 새로운 프로젝트를 생성한다.

2. 오른쪽 프로젝트창에서 Form1.cs를 클릭하면 리소스 디자인 창을 볼 수 있다.

3. Form1.cs 탭을 눌러서 활성화를 시키면 오른쪽의 속성창에 디자인의 정보를 알려주는데 여기서 Text라는 항목으로 기본 이름 네이밍이 가능하다. Name이라는 항목은 따로 존재하는데 이 UI들의 ID이기 때문에 주의 해야한다.

1. WINFORM

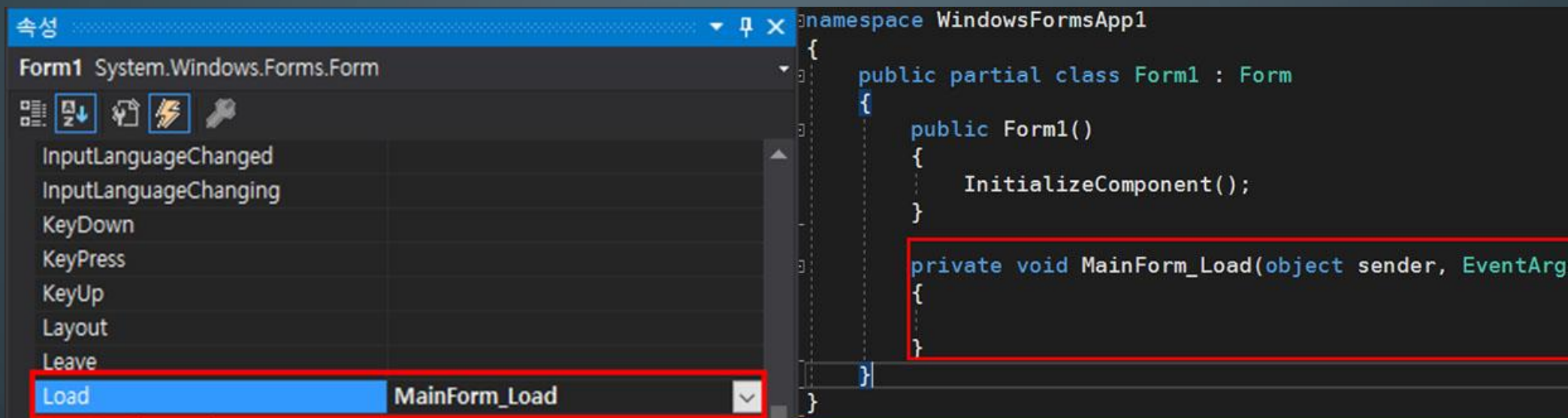
컨트롤러 배치 방법과 로더 이벤트 연결 준비



1. 일반적으로 도구 상자를 건드린 적이 없다면 **VisualStudio 왼쪽 끝에 툴바 형태로 [도구 상자]**라는 것이 보일 것이다. 만약, 보이지 않는다면 **[보기]->[도구 상자]**를 선택해서 **활성화** 시켜주고 원하는 위치에 두고 작업을 진행하자!
2. 도구 상자에서 원하는 컨트롤들을 디자이너 위에 배치 할 수 있다. 도구 상자에서 원하는 항목을 선택하고 디자이너 위에서 **클릭하고 드레그** 하면 **컨트롤의 크기를 설정**할 수 있다.
3. 배치한 후에는 해당 컨트롤이 선택된 상태이기 때문에 속성창에 해당 컨트롤의 속성이 나온다. 그 중 **Name이라는 속성을 사용할 ID명으로 작성**해준다.
4. 이제 이렇게 배치된 컨트롤이나 여러 이벤트를 디자이너가 감지해야 하기 때문에 **메인 Loder가 필요하다**. 해당 디자이너 작성 상단 탭을 눌러서 **디자이너가 활성화** 되었는지 **확인**하자. 그리고 속성창에 가면 해당 디자이너 이름의 속성창인지 확인하고 **번개 모양을 클릭해서 이벤트 연결을 준비**하자.

1. WINFORM

메인로더 및 이벤트 Method 생성

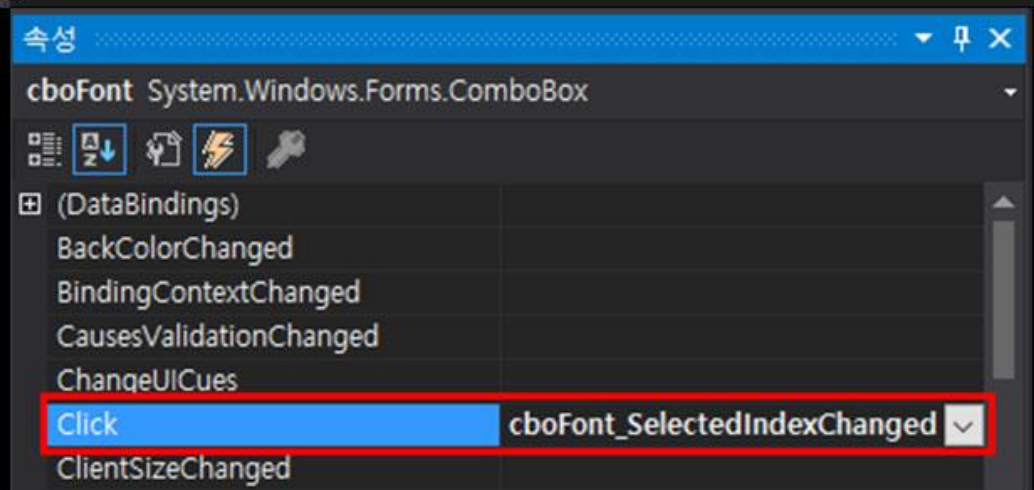


The screenshot shows the Visual Studio IDE. On the left, the '속성' (Properties) window for 'Form1' (System.Windows.Forms.Form) is open. The 'Load' event is selected, and the 'MainForm_Load' method is being created. On the right, the 'Form1.cs' file is open, showing the C# code. The 'MainForm_Load' method is highlighted with a red box.

```
namespace WindowsFormsApp1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void MainForm_Load(object sender, EventArgs e)
        {
        }
    }
}
```

1. **Load 항목을 찾아서 원하는 Method 이름을 적은 후 Enter를 누르면 자동으로 Method를 만들어 준다.**
2. 이와 같이 메인 로더를 만들 수 있고 디자이너 위에 올라가 있는 모든 컨트롤들도 이와 같은 방법으로 Method를 작성할 수 있다.
3. 콤보 박스가 올려져 있으니 콤보박스를 선택한 후 메인 로더 만들때와 마찬가지로 원하는 **이벤트에 Method를 이름을 작성** 하고 Enter를 눌러주면 **자동으로 이벤트 Method가 등록된다.**



The screenshot shows the Visual Studio IDE. On the left, the '속성' (Properties) window for 'cboFont' (System.Windows.Forms.ComboBox) is open. The 'Click' event is selected, and the 'cboFont_SelectedIndexChanged' method is being created. The 'cboFont_SelectedIndexChanged' method is highlighted with a red box.

```
namespace WindowsFormsApp1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void MainForm_Load(object sender, EventArgs e)
        {
        }

        private void cboFont_SelectedIndexChanged(object sender, EventArgs e)
        {
        }
    }
}
```

1. WINFORM

GroupBox, Label, ComboBox, CheckBox, TextBox 배치 및 사용

Control : Label
Name : lblFont
Text : Font

Control : GroupBox
Name : grpFont
Text : ComboBox, CheckBox, TextBox

Control : CheckBox
Name : chkItalic
Text : 이탤릭

Form1

콤보박스, 체크박스, 텍스트박스

Font : cboFont

☐ 굵게 ☐ 이탤릭

Hello, C#

Control : TextBox
Name : txtSampleText
Text : Hello, C#

Control : ComboBox
Name : cboFont

Control : CheckBox
Name : chkBold
Text : 굵게

1. WINFORM

Step1. Control의 배치가 끝났다면 **MainLoader**를 만들어야 하고 만들고 나면 빈 껍데기만 존재하므로 내용을 채워 줘야 한다.

// **Load**를 **MainForm_Load**로 적었을 경우에 그 이름 그대로 **Method**가 작성.

```
private void MainForm_Load()
```

```
{
```

```
    // System.Drawing 네임스페이스를 사용해야 한다.
```

```
    FontFamily[] fontFamilies;
```

```
    // System.Drawing.Text 네임스페이스를 사용해야 한다.
```

```
    InstalledFontCollection installedFontCollection = new InstalledFontCollection();
```

```
    // OS에 설치되어 있는 모든 Font정보를 가져 온다.
```

```
    fontFamilies = installedFontCollection.Families;
```

```
    foreach (var fontFamily in fontFamilies)
```

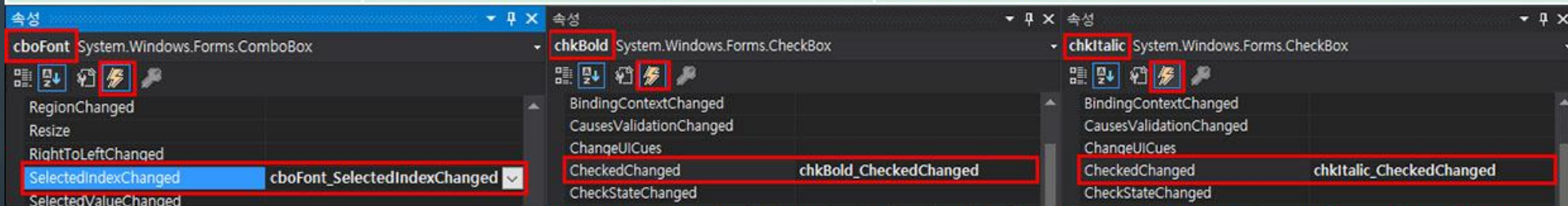
```
        cboFont.Items.Add(fontFamily.Name);
```

```
}
```

1. WINFORM

Step2. 이제 해당 컨트롤의 각 이벤트 부분과 해당 컨트롤 이벤트 **Method**를 작성해 보자.

컨트롤	이벤트	이벤트 Method
cboFont	SelectedIndexChanged	cboFont_SelectedIndexChanged
chkBold	CheckedChanged	chkBold_CheckedChanged
chkItalic	CheckedChanged	chkItalic_CheckedChanged



ComboBox의 Name이 cboFont 굵기를 나타내는 CheckBox의 Name이 chkBold 이탤릭을 나타내는 CheckBox의 Name이 chkItalic으로 각각 지정했다.

각 Control에 이벤트에서 연결할 이벤트 Method를 작성해주면 알아서 껌대기를 만들어 주는데 각 Method에 실행 Method를 작성해 주자!

```
private void cboFont_SelectedIndexChanged(object sender, EventArgs e)
{
    ChangeFont();
}

private void chkBold_CheckedChanged(object sender, EventArgs e)
{
    ChangeFont();
}

private void chkItalic_CheckedChanged(object sender, EventArgs e)
{
    ChangeFont();
}
```

1. WINFORM

Step3. 각 이벤트 **Method**가 실행할 동작 **Method**를 작성해 보자.

```
void ChangeFont()
```

```
{
```

```
    // cboFont에서 선택한 항목이 없으면 메소드 종료
```

```
    if (cboFont.SelectedIndex < 0)
```

```
        return;
```

```
    // FontStyle 객체를 초기화
```

```
    FontStyle style = FontStyle.Regular;
```

```
    // "굵게" 체크 박스가 선택되어 있으면 Bold 논리합 수행
```

```
    if (chkBold.Checked)
```

```
        style |= FontStyle.Bold;
```

```
    // "이텔릭" 체크 박스가 선택되어 있으면 Italic 논리합 수행
```

```
    if (chkItalic.Checked)
```

```
        style |= FontStyle.Italic;
```

```
    // txtSampleText의 Font 프로퍼티를 앞에서 만든 style로 수정
```

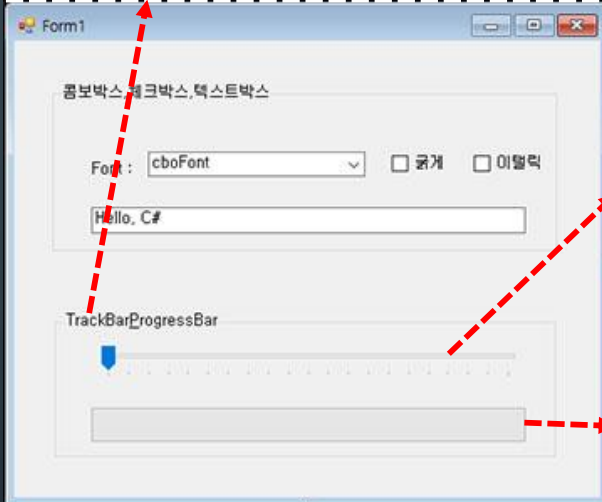
```
    txtSampleText.Font = new Font((string)cboFont.SelectedItem, 10, style);
```

```
}
```


1. WINFORM

TrackBar, ProgressBar 사용하기

Control : GroupBox
Name : grpBar
Text : TrackBar & ProgressBar



Control : TrackBar
Name : tbDummy
Maximum : 20

Control : ProgressBar
Name : pgDummy
Maximum : 20

컨트롤	이벤트	이벤트 처리기
tbDummy	Scroll	tbDummy_Scroll

```
private void tbDummy_Scroll(object sender, EventArgs e)
{
    //슬라이더의 위치에 따라 프로그레스바의 내용도 변경
    pgDummy.Value = tbDummy.Value;
}
```

1. TrackBar 컨트롤과 ProgressBar 컨트롤을 각각 설치한다.
2. 이벤트가 필요한것은 TrackBar이며 Scroll 항목에 tbDummy_Scroll이라는 이벤트 Method를 작성한다.
3. ProgressBar는 TrackBar의 값을 그대로 반영하는 목적으로 현재 구현의도를 가지고 구현했기 때문에 TrackBar의 Value 값을 그대로 ProgressBar의 Value에 대입시켜 준다.

※ ProgressBar 단독으로 사용이 가능하며 Value 값에 대입만 시켜주면 ProgressBar는 적용이 가능하다. 총량 비례하는 값을 가지므로 나누기 연산을 하게 되므로 곱연산으로 치환하는 응용을 생각해봐야 한다.

1. WINFORM

Button, Form, Dialog 사용하기

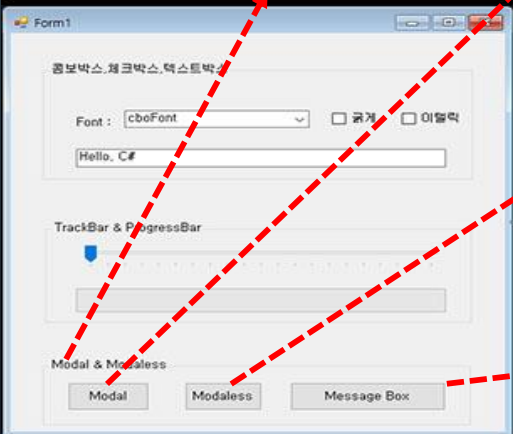
- **Dialog** 형태의 **Child Window**를 보여주는 방법에는 **Modal, Modaleless** 두 가지가 있다.
- **Modal** : 띄우고 나면 창을 닫을 때까지 프로그램의 다른 **UI**를 절대 사용할 수 없다. 주로 프로그램이 심각한 정보를 표시해야 하거나 사용자로부터 중요한 결정 사항을 입력 받아 다음 단계를 진행해야 할 때 주로 사용한다.
- **Modaleless** : 띄우고 난 뒤에도 프로그램의 다른 **UI**에 사용자가 접근할 수 있다. 웹 브라우저의 파일 다운로드 창 같은 것이 이에 가장 좋은 예이다.

Control : GroupBox
Name : grpForm
Text : Model & Modaleless

Control : Button
Name : BtnModal
Text : Modal

Control : Button
Name : btnModaleless
Text : Modaleless

Control : Button
Name : btnMsgBox
Text : MessageBox



컨트롤	이벤트	이벤트 처리
btnModel	Click	btnModal_Click
btnModaleless	Click	btnModaleless_Click
btnMsgBox	Click	btnMsgBox_Click

```
private void btnModal_Click(object sender, EventArgs e)
{
    Form frm = new Form();
    frm.Text = "Modal Form";
    frm.Width = 300;
    frm.Height = 100;
    frm.BackColor = Color.Red;
    frm.ShowDialog(); // Modal 창을 띄운다.
}

private void btnModaleless_Click(object sender, EventArgs e)
{
    Form frm = new Form();
    frm.Text = "Modaleless Form";
    frm.Width = 300;
    frm.Height = 300;
    frm.BackColor = Color.Green;
    frm.Show(); // Modaleless 창을 띄운다.
}

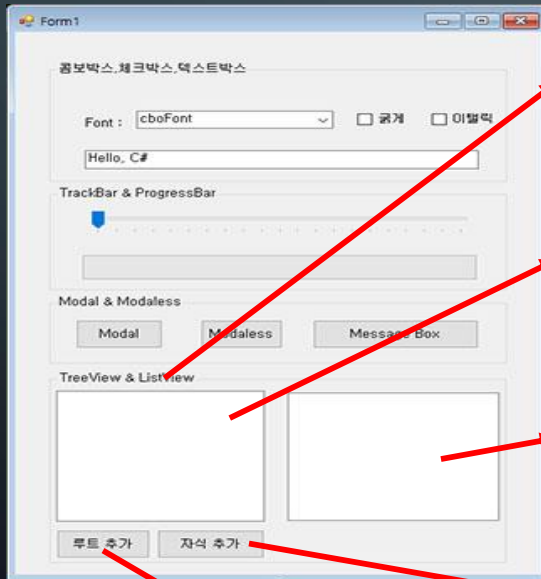
private void btnMsgBox_Click(object sender, EventArgs e)
{
    MessageBox.Show(txtSampleText.Text, "MessageBox Test",
        MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
}
```

1. WINFORM

TreeView, ListView 사용하기

- **TreeView**와 **ListView**는 표시할 **Data**의 상관관계와 **Data**들이 필요하기 때문에 임의의 **Data** 생산을 위해 난수를 생성할 예정이다.

```
public partial class MainForm : Form
{
    Random random = new Random(37);
    // ..
}
```



Control : GroupBox
Name : grpTreeList
Text : TreeView && ListView

Control : TreeView
Name : tvDummy

Control : ListView
Name : lvDummy
View : Details

Control : Button
Name : btnAddRoot
Text : 루트 추가

Control : Button
Name : btnAddChild
Text : 자식 추가

컨트롤	이벤트	이벤트 처리기
btnAddRoot	Click	btnAddRoot_Click
btnAddChild	Click	btnAddChild_Click

```
private void btnAddRoot_Click(object sender, EventArgs e)
{
    tvDummy.Nodes.Add(random.Next().ToString());
    TreeToList();
}

private void btnAddChild_Click(object sender, EventArgs e)
{
    if(tvDummy.SelectedNode == null)
    {
        MessageBox.Show("선택된 노드가 없습니다.", "TreeView Test",
            MessageBoxButtons.OK, MessageBoxIcon.Error);

        return;
    }

    tvDummy.SelectedNode.Nodes.Add(random.Next().ToString());
    tvDummy.SelectedNode.Expand();
    TreeToList();
}
```


1. WINFORM

- **TreeView**와 **ListView**는 각각 **Data**를 보여줄 형식으로 표시 할 수 있기 때문에 이벤트로 받지 않고 사용 할 수 있다.

```
public MainForm()
{
    InitializeComponent();

    lvDummy.Columns.Add("Name");
    lvDummy.Columns.Add("Depth");
}
```

```
void TreeToList()
{
    lvDummy.Items.Clear();

    foreach (TreeNode node in tvDummy.Nodes)
        TreeToList(node);
}
```

```
void TreeToList(TreeNode Node)
{
    //FullPath는 string형이고 string형의 Count는
    //System.Linq 네임스페이스를 사용해야한다.
    lvDummy.Items.Add(new ListViewItem(new string[] { Node.Text,
                                                         Node.FullPath.Count(f => f == '\\').ToString() }));

    foreach (TreeNode node in Node.Nodes)
        TreeToList(node);
}
```

TreeNode 형식의 **FullPath** 프로퍼티는 루트 노드부터 현재 노드까지의 경로를 나타내며, 각 경로는 \로 구분한다.

1. WINFORM

비동기 작업을 적용한 사용자 인터페이스

- 비동기 파일 복사 부분의 코드와 달라지는 건 없지만 복사량을 체크하는 부분을 추가 하기 위해 **ProgressBar**에 표시하는 것을 추가 한다.

```
private async Task<long> CopyAsync(string FromPath, string ToPath)
{
    btnSyncCopy.Enabled = false;
    long totalCopied = 0;

    using (FileStream fromStream = new FileStream(FromPath, FileMode.Open))
    {
        using (FileStream toStream = new FileStream(ToPath, FileMode.Create))
        {
            byte[] buffer = new byte[1024 * 1024];
            int nRead = 0;

            while ((nRead = await fromStream.ReadAsync(buffer, 0, buffer.Length)) != 0)
            {
                await toStream.WriteAsync(buffer, 0, nRead);
                totalCopied += nRead;

                //프로그레스바에 현재 파일 복사 상태 표시
                pbCopy.Value
                    = (int)((((double)totalCopied / (double)fromStream.Length) * pbCopy.Maximum));
            }
        }
    }

    btnSyncCopy.Enabled = true;
    return totalCopied;
}
```

동기 파일 복사의 코드부분이 비슷하다. 이 부분이 실행되면 비동기화 복사 버튼이 비활성화 처리되고 파일 복사를 이루면서 **ProgressBar**의 수치를 조절해서 복사량을 표시해준다.

비동기 파일 복사의 코드부분이 비슷하다. 이 부분이실행되면 동기화 복사 버튼이 비활성화 처리되고, 파일 복사를 이루면서 **ProgressBar**의 수치를 조절해서 복사량을 표시해준다.

```
private Task<long> CopySync(string FromPath, string ToPath)
{
    btnAsyncCopy.Enabled = false;

    using (FileStream fromStream = new FileStream(FromPath, FileMode.Open))
    {
        using (FileStream toStream = new FileStream(ToPath, FileMode.Create))
        {
            byte[] buffer = new byte[1024 * 1024];
            int nRead = 0;

            while ((nRead = fromStream.Read(buffer, 0, buffer.Length)) != 0)
            {
                toStream.Write(buffer, 0, nRead);
                totalCopied += nRead;

                //프로그레스바에 현재 파일 복사 상태 표시
                pbCopy.Value
                    = (int)((((double)totalCopied / (double)fromStream.Length) * pbCopy.Maximum));
            }
        }
    }

    btnAsyncCopy.Enabled = true;
    return totalCopied;
}
```

1. WINFORM

UI 구성하기

컨트롤	이벤트	이벤트 처리기
btnFindSource	Click	btnFindSource_Click
btnFindTarget	Click	btnFindTarget_Click
btnAsyncCopy	Click	btnAsyncCopy_Click
btnSyncCopy	Click	btnSyncCopy_Click
btnCancel	Click	btnCancel_Click

```
private async void btnAsyncCopy_Click(object sender, EventArgs e)
{
    long totalCopied = await CopyAsync(txtSource.Text, txtTarget.Text);
}
```

```
private void btnSyncCopy_Click(object sender, EventArgs e)
{
    long totalCopied = CopySync(txtSource.Text, txtTarget.Text);
}
```

```
private void btnCancel_Click(object sender, EventArgs e)
{
    MessageBox.Show("UI 반응 테스트 성공!!");
}
```

```
private void btnFindSource_Click(object sender, EventArgs e)
{
    OpenFileDialog dlg = new OpenFileDialog();

    if(dlg.ShowDialog() == DialogResult.OK)
    {
        txtSource.Text = dlg.FileName;
    }
}
```

```
private void btnFindTarget_Click(object sender, EventArgs e)
{
    SaveFileDialog dlg = new SaveFileDialog();

    if(dlg.ShowDialog() == DialogResult.OK)
    {
        txtTarget.Text = dlg.FileName;
    }
}
```