

# Contents

## Windows Forms

Windows Forms 시작

Windows Forms 개요

새 Windows Form 만들기

방법: 명령줄에서 Windows Forms 애플리케이션 만들기

Windows Forms 작표

Windows Forms에서 이벤트 처리기 만들기

이벤트 개요

이벤트 처리기 개요

방법: 런타임 시 Windows Forms의 이벤트 처리기 만들기

방법: Windows Forms에서 단일 이벤트 처리기에 여러 이벤트 연결

Windows Forms에서의 이벤트 순서

Windows Forms의 크기 및 배율 조정

방법: Windows Forms 크기 조정

Windows Forms의 자동 크기 조정

방법: Windows Forms 애플리케이션에서 글꼴 구성표 변경 내용에 대응

Windows Forms의 높은 DPI 지원

Windows Forms의 모양 변경

방법: Windows Forms의 테두리 변경

Windows Forms 컨트롤

Windows Forms에 사용자 입력

Windows Forms 애플리케이션의 사용자 입력

Windows Forms 애플리케이션의 키보드 입력

키보드 입력 작동 방식

키보드 이벤트 사용

방법: 표준 컨트롤로 키보드 입력 수정

방법: 누른 보조 키 확인

방법: 양식 수준에서 키보드 입력 처리

Windows Forms 애플리케이션의 마우스 입력

Windows Forms에서 마우스 입력이 작동하는 방식

Windows Forms의 마우스 이벤트

방법: 클릭과 두 번 클릭 간 구별

Windows Forms의 마우스 포인터

Windows Forms의 마우스 캡처

Windows Forms에서의 끌어서 놓기 기능

방법: 코드에서 마우스 및 키보드 이벤트 시뮬레이션

방법: Windows Forms 컨트롤에서 사용자 입력 이벤트 처리

Windows Forms에서 사용자 입력 유효성 검사

Windows Forms 대화 상자

방법: Windows Forms 대화 상자 표시

Windows Forms 데이터 바인딩

데이터 바인딩 및 Windows Forms

Windows Forms에서 지원하는 데이터 소스

데이터 바인딩과 관련된 인터페이스

Windows Forms 데이터 바인딩의 변경 알림

방법: PropertyChanged 패턴 적용

방법: 바인딩된 컨트롤 만들기 및 표시된 데이터 서식 지정

방법: Windows Form에 단순 바인딩된 컨트롤 만들기

방법: 동일한 데이터 소스에 바인딩된 여러 컨트롤의 동기화 상태가 유지되도록 설정

방법: 자식 테이블에서 선택된 행이 올바른 위치에 유지되도록 설정

방법: IListSource 인터페이스 구현

방법: INotifyPropertyChanged 인터페이스 구현

방법: IList 인터페이스 구현

방법: Windows Forms에서 데이터 탐색

Windows Forms 보안

Windows Forms의 보안 개요

Windows Forms의 파일 및 데이터 액세스 추가 보안

Windows Forms의 인쇄 추가 보안

Windows Forms의 추가 보안 고려 사항

Windows Forms에 대한 ClickOnce 배포

.NET Core 3.0을 통한 접근성 개선 사항

방법: Windows Forms에서 키 컬렉션에 액세스  
Windows Forms 애플리케이션 강화

# Windows Forms

2020-02-03 • 4 minutes to read • [Edit Online](#)

폼은 애플리케이션의 기본 단위이므로 폼의 기능과 디자인을 고려해야 합니다. 폼은 기본적으로 개발자가 컨트롤을 통해 향상시켜 사용자 인터페이스를 만들고 코드를 통해 향상시켜 데이터를 조작하는 빈 슬레이트입니다. 이를 위해 Visual Studio는 코드 작성을 돕는 IDE (통합 개발 환경)와 .NET Framework로 작성된 다양한 컨트롤 집합을 제공합니다. 이러한 컨트롤의 기능을 코드로 보완하면 필요한 솔루션을 쉽고 빠르게 개발할 수 있습니다.

## 섹션 내용

### [Windows Forms 시작](#)

Windows Forms의 기능을 활용하여 데이터를 표시하고, 사용자 입력을 처리하며, 쉽고 보다 강력한 보안으로 애플리케이션을 배포하는 방법에 대한 항목의 링크를 제공합니다.

### [Windows Forms 애플리케이션 강화](#)

다양한 기능으로 Windows Forms를 향상시키는 방법에 대한 항목의 링크를 제공합니다.

## 관련 섹션

### [Windows Forms 컨트롤](#)

Windows Forms 컨트롤을 설명하고 구현 방법을 설명하는 항목의 링크를 포함합니다.

### [Windows Forms 데이터 바인딩](#)

Windows Forms 데이터 바인딩 아키텍처를 설명하는 항목의 링크를 포함합니다.

### [그래픽 개요](#)

Windows 그래픽 디자인 인터페이스의 고급 구현을 사용하여 그래픽을 만들고, 텍스트를 그리고, 그래픽 이미지를 개체로 조작하는 방법을 설명합니다.

### [ClickOnce 보안 및 배포](#)

ClickOnce 배포의 원칙을 설명합니다.

### [Windows Forms/MFC 프로그래밍의 차이점](#)

MFC 애플리케이션과 Windows Forms 간의 차이점을 설명합니다.

### [Visual Studio에서 데이터 액세스](#)

애플리케이션에 데이터 액세스 기능을 통합하는 방법을 설명합니다.

### [Windows Forms 애플리케이션](#)

Windows 애플리케이션 프로젝트 템플릿을 사용하여 만든 애플리케이션을 디버그하는 프로세스와 디버그 및 릴리스 구성을 변경하는 방법을 설명합니다.

### [먼저 Visual Studio에서 배포 살펴보기](#)

다른 컴퓨터에 설치할 완성된 애플리케이션 또는 구성 요소를 배포하는 프로세스를 설명합니다.

### [콘솔 애플리케이션 만들기](#)

[Console](#) 클래스를 사용하여 콘솔 애플리케이션 만드는 방법의 기본 사항을 설명합니다.

# Windows Forms 시작

2020-02-03 • 4 minutes to read • [Edit Online](#)

Windows Forms를 사용 하면 강력한 Windows 기반 응용 프로그램을 만들 수 있습니다. 다음 항목에서는 Windows Forms 기능을 활용 하여 데이터를 표시 하고, 사용자 입력을 처리 하고, 응용 프로그램을 쉽게 배포 하고 보안을 강화 하는 방법에 대해 자세히 설명 합니다.

## 섹션 내용

### [Windows Forms 개요](#)

Windows Forms 및 스마트 클라이언트 응용 프로그램에 대 한 개요를 포함 합니다.

### [새 Windows Form 만들기](#)

Windows Forms 응용 프로그램을 만들기 위한 기본 개념을 설명 하는 항목의 링크를 포함 합니다.

### [Windows Forms에서 이벤트 처리기 만들기](#)

Windows Forms 이벤트 처리기를 만드는 방법을 설명 하는 항목에 대 한 링크를 포함 합니다.

### [Windows Forms의 크기 및 배율 조정](#)

Windows Forms의 크기와 크기를 조정 하는 방법을 보여 주는 항목의 링크를 포함 합니다.

### [Windows Forms의 모양 변경](#)

Windows Forms 애플리케이션의 모양을 변경하는 방법을 보여 주는 항목의 링크를 포함합니다.

### [Windows Forms 컨트롤](#)

Windows Forms 컨트롤 및 구성 요소를 사용 하는 방법을 설명 하고 보여 주는 항목의 링크를 포함 합니다.

### [Windows Forms에 사용자 입력](#)

Windows Forms 응용 프로그램에서 사용자 입력을 처리 하는 방법을 설명 하고 보여 주는 항목의 링크를 포함 합니다.

### [Windows Forms 대화 상자](#)

Windows Forms에서 사용할 수 있는 여러 대화 상자를 설명 하는 항목의 링크를 포함 합니다.

### [Windows Forms 데이터 바인딩](#)

Windows Forms 데이터 바인딩 아키텍처 및 Windows Forms 응용 프로그램에서 사용 하는 방법에 대해 설명 하는 항목의 링크를 포함 합니다.

### [Windows Forms 보안](#)

보안이 강화 된 Windows Forms 응용 프로그램을 빌드하는 방법을 설명 하는 항목의 링크를 포함 합니다.

### [Windows Forms에 대한 ClickOnce 배포](#)

Windows Forms 응용 프로그램을 쉽게 배포 하는 방법을 설명 하는 항목에 대 한 링크가 포함 되어 있습니다.

### [방법: Windows Forms에서 키 컬렉션 액세스](#)

인덱스가 아닌 키를 사용 하여 컬렉션에 액세스 하는 방법을 보여 줍니다.

## 관련 섹션

### [Windows Forms 애플리케이션 강화](#)

Windows Forms 응용 프로그램을 만들기 위한 고급 개념을 설명 하는 항목의 링크를 포함 합니다.

# Windows Forms 개요

2020-02-03 • 22 minutes to read • [Edit Online](#)

다음 개요에서는 스마트 클라이언트 애플리케이션의 이점, Windows Forms 프로그래밍의 주요 기능 및 Windows Forms를 사용하여 오늘날의 기업과 최종 사용자의 요구를 충족하는 스마트 클라이언트를 빌드하는 방법을 설명합니다.

## Windows Forms 및 스마트 클라이언트 앱

Windows Forms를 사용하여 스마트 클라이언트를 개발합니다. *스마트 클라이언트*는 쉽게 배포 및 업데이트되며, 인터넷에 연결되었거나 연결이 끊어졌을 때 작동할 수 있고, 기존의 Windows 기반 애플리케이션보다 더 안전하게 로컬 컴퓨터의 리소스에 액세스할 수 있는 시각적으로 풍부한 애플리케이션입니다.

풍부한 대화형 사용자 인터페이스 빌드

Windows Forms는 파일 시스템 읽기 및 쓰기와 같은 일반적인 응용 프로그램 작업을 간소화 하는 관리 되는 라이브러리 집합인 .NET Framework에 대한 스마트 클라이언트 기술입니다. Visual Studio와 같은 개발 환경을 사용하는 경우 정보를 표시 하고, 사용자의 입력을 요청 하고, 네트워크를 통해 원격 컴퓨터와 통신 하는 Windows Forms 스마트 클라이언트 응용 프로그램을 만들 수 있습니다.

Windows Forms에서 *폼*은 정보를 사용자에게 표시하는 비주얼 화면입니다. 일반적으로 폼에 컨트롤을 추가하고 마우스 클릭이나 키 누름과 같은 사용자 동작에 대한 응답을 개발하여 Windows Forms 애플리케이션을 빌드합니다. *컨트롤*은 데이터를 표시하거나 데이터 입력을 수락하는 고유한 UI(사용자 인터페이스) 요소입니다.

사용자가 폼이나 컨트롤 중 하나에 작업을 수행하면 이벤트가 생성됩니다. 애플리케이션은 코드를 사용하여 이러한 이벤트에 대응하고, 발생 시 이벤트를 처리합니다. 자세한 내용은 [Windows Forms에서 이벤트 처리기 만들기](#)를 참조하세요.

Windows Forms에는 텍스트 상자, 단추, 드롭다운 상자, 라디오 단추 및 웹 페이지를 표시하는 컨트롤 등 폼에 추가할 수 있는 다양한 컨트롤이 포함되어 있습니다. 폼에서 사용할 수 있는 모든 컨트롤의 목록은 [Windows Forms에서 사용할 수 있는 컨트롤](#)을 참조하세요. 기존 컨트롤이 요구를 충족하지 않는 경우 Windows Forms에서 [UserControl](#) 클래스를 사용하여 고유한 사용자 지정 컨트롤을 만들 수도 있습니다.

Windows Forms에는 Microsoft Office와 같은 고급 애플리케이션의 기능을 에뮬레이트하는 풍부한 UI 컨트롤이 있습니다. [ToolStrip](#) 및 [MenuStrip](#) 컨트롤을 사용하는 경우 텍스트와 이미지를 포함하고, 하위 메뉴를 표시하며, 텍스트 상자 및 콤보 상자와 같은 기타 컨트롤을 호스트하는 도구 모음과 메뉴를 만들 수 있습니다.

Visual Studio에서 끌어서 놓기 **Windows Forms 디자이너** 를 사용 하면 Windows Forms 응용 프로그램을 쉽게 만들 수 있습니다. 커서를 사용하여 컨트롤을 선택하고 폼에서 원하는 위치에 추가하면 됩니다. 디자이너는 컨트롤을 쉽게 배치하기 위한 모눈선 및 맞춤선과 같은 도구를 제공합니다. Visual Studio를 사용 하거나 명령줄에서 컴파일하는 경우에는 [FlowLayoutPanel](#), [TableLayoutPanel](#) 및 [SplitContainer](#) 컨트롤을 사용 하여 더 짧은 시간 안에 고급 폼 레이아웃을 만들 수 있습니다.

끝으로, 고유한 사용자 지정 UI 요소를 만들어야 하는 경우 [System.Drawing](#) 네임스페이스에는 선, 원 및 기타 도형은 폼에 직접 렌더링하는 다양한 클래스가 포함되어 있습니다.

### NOTE

Windows Forms 컨트롤은 애플리케이션 도메인 간에 마샬링되도록 설계되어 있지 않습니다. 이런 이유로 Microsoft는 [AppDomain](#)의 [Control](#) 기본 형식이 가능한 것처럼 표시해도 [MarshalByRefObject](#) 경계를 넘어서 Windows Forms 컨트롤을 전달하는 기능을 지원하지 않습니다. 애플리케이션 도메인 경계를 넘어서 Windows Forms 컨트롤이 전달되지 않는 한 여러 개의 애플리케이션 도메인을 가진 Windows Forms 애플리케이션이 지원됩니다.

## 폼 및 컨트롤 만들기

이러한 기능을 사용하는 방법에 대한 단계별 정보는 다음 도움말 항목을 참조하세요.

DESCRIPTION	도움말 항목
폼에서 컨트롤 사용	방법: <a href="#">Windows Forms에 컨트롤 추가</a>
<a href="#">ToolStrip</a> 컨트롤 사용	방법: <a href="#">디자이너를 사용하여 표준 항목이 있는 기본 ToolStrip 만들기</a>
<a href="#">System.Drawing</a> 을 사용하여 그래픽 만들기	<a href="#">그래픽 프로그래밍 시작</a>
사용자 지정 컨트롤 만들기	방법: <a href="#">UserControl 클래스에서 상속</a>

## 데이터 표시 및 조작

많은 애플리케이션은 데이터베이스, XML 파일, XML Web services 또는 기타 데이터 소스의 데이터를 표시해야 합니다. Windows Forms는 각 데이터 조각이 해당 셀을 사용하도록 이러한 표 형식 데이터를 기존의 행과 열 형식으로 표시하기 위해 [DataGridView](#) 컨트롤이라는 유연한 컨트롤을 제공합니다. [DataGridView](#)를 사용하는 경우 다른 기능 중에서도 개별 셀의 모양을 사용자 지정하고, 임의의 행과 열을 제자리에 잠그고, 셀 안에 복잡한 컨트롤을 표시할 수 있습니다.

네트워크를 통해 데이터 소스에 연결하는 것은 Windows Forms 스마트 클라이언트에서 간단한 작업입니다. [BindingSource](#) 구성 요소는 데이터 소스에 대한 연결을 나타내며 데이터를 컨트롤에 바인딩하고, 이전 및 다음 레코드로 이동하고, 레코드를 편집하고, 변경 내용을 다시 원래 소스에 저장하기 위한 메서드를 노출합니다. [BindingNavigator](#) 컨트롤은 [BindingSource](#) 구성 요소를 통해 사용자가 레코드를 탐색하기 위한 간단한 인터페이스를 제공합니다.

데이터 소스 창을 통해 데이터 바인딩된 컨트롤을 쉽게 만들 수 있습니다. 창에는 데이터베이스, 웹 서비스 및 개체와 같은 프로젝트의 데이터 소스가 표시됩니다. 이 창에서 프로젝트의 폼으로 항목을 끌어 데이터 바인딩된 컨트롤을 만들 수 있습니다. 데이터 소스 창에서 기존 컨트롤로 개체를 끌어 기존 컨트롤을 데이터에 바인딩할 수도 있습니다.

Windows Forms에서 관리할 수 있는 다른 형식의 데이터 바인딩은 [설정](#)입니다. 대부분의 스마트 클라이언트 애플리케이션은 마지막으로 알려진 폼 크기와 같은 런타임 상태에 대한 일부 정보를 유지하고 저장된 파일의 기본 위치와 같은 사용자 기본 설정 데이터를 유지해야 합니다. 애플리케이션 설정 기능은 클라이언트 컴퓨터에 두 유형의 설정을 모두 쉽게 저장할 수 있는 방법을 제공하여 이러한 요구 사항을 충족합니다. Visual Studio 또는 코드 편집기를 사용하여 이러한 설정을 정의한 후에는 설정이 XML로 유지되며 런타임에 자동으로 메모리로 다시 읽혀집니다.

## 데이터 표시 및 조작

이러한 기능을 사용하는 방법에 대한 단계별 정보는 다음 도움말 항목을 참조하세요.

DESCRIPTION	도움말 항목
<a href="#">BindingSource</a> 구성 요소 사용	방법: <a href="#">디자이너를 사용하여 Windows Forms 컨트롤에 BindingSource 구성 요소 바인딩</a>
ADO.NET 데이터 원본 사용	방법: <a href="#">Windows Forms BindingSource 구성 요소를 사용하여 ADO.NET 데이터 정렬 및 필터링</a>
데이터 소스 창 사용	<a href="#">Visual Studio에서 데이터에 Windows Forms 컨트롤 바인딩</a>
애플리케이션 설정 사용	방법: <a href="#">애플리케이션 설정 만들기</a>

## 클라이언트 컴퓨터에 앱 배포

애플리케이션을 작성한 후 해당 클라이언트 컴퓨터에 설치하고 실행할 수 있도록 사용자에게 애플리케이션을 보내야 합니다. ClickOnce 기술을 사용 하는 경우 몇 번의 클릭 만으로 Visual Studio 내에서 응용 프로그램을 배포 하고 사용자에게 웹의 응용 프로그램을 가리키는 URL을 제공할 수 있습니다. ClickOnce는 응용 프로그램의 모든 요소와 종속성을 관리 하고 클라이언트 컴퓨터에 응용 프로그램이 올바르게 설치 되었는지 확인 합니다.

ClickOnce 응용 프로그램은 사용자가 네트워크에 연결 된 경우에만 실행 되도록 구성 하거나 온라인 및 오프 라인에서 실행할 수 있습니다. 응용 프로그램이 오프 라인 작업을 지원 하도록 지정 하면 ClickOnce는 사용자의 시작 메뉴에 응용 프로그램에 대한 링크를 추가 합니다. 그러면 사용자가 URL을 사용하지 않고도 애플리케이션을 열 수 있습니다.

애플리케이션을 업데이트하는 경우 새 배포 매니페스트와 애플리케이션의 새 복사본을 웹 서버에 게시합니다. ClickOnce에서 사용 가능한 업데이트가 있음을 감지 하고 사용자의 설치를 업그레이드 합니다. 이전 어셈블리를 업데이트 하는 데 사용자 지정 프로그래밍이 필요 하지 않습니다.

#### ClickOnce 앱 배포

ClickOnce에 대한 전체 개요는 [Clickonce 보안 및 배포](#)를 참조 하세요. 이러한 기능을 사용하는 방법에 대한 단계별 정보는 다음 도움말 항목을 참조하세요.

DESCRIPTION	도움말 항목
ClickOnce를 사용 하여 응용 프로그램 배포	방법: <a href="#">게시 마법사를 사용하여 ClickOnce 애플리케이션 게시</a>  연습: <a href="#">ClickOnce 애플리케이션 수동 배포</a>
ClickOnce 배포 업데이트	방법: <a href="#">ClickOnce 애플리케이션에 대한 업데이트 관리</a>
ClickOnce를 사용 하여 보안 관리	방법: <a href="#">ClickOnce 보안 설정 사용</a>

#### 기타 컨트롤 및 기능

Windows Forms에는 대화 상자 만들기, 인쇄, 도움말 및 설명서 추가 및 여러 언어로 애플리케이션 지역화 지원과 같이 일반적인 작업을 쉽고 빠르게 구현할 수 있게 해주는 다른 여러 기능이 있습니다. 또한 Windows Forms은 .NET Framework의 강력한 보안 시스템에 의존 합니다. 이 시스템을 통해 보다 안전한 애플리케이션을 고객에게 릴리스할 수 있습니다.

#### 기타 컨트롤 및 기능 구현

이러한 기능을 사용하는 방법에 대한 단계별 정보는 다음 도움말 항목을 참조하세요.

DESCRIPTION	도움말 항목
폼의 콘텐츠 인쇄	방법: <a href="#">Windows Forms의 그래픽 인쇄</a>  방법: <a href="#">Windows Forms에서 다중 페이지 텍스트 파일 인쇄</a>
Windows Forms 보안에 대한 자세한 정보	<a href="#">Windows Forms의 보안 개요</a>

## 참고 항목

- [Windows Forms 시작](#)
- [새 Windows Form 만들기](#)
- [ToolStrip 컨트롤 개요](#)
- [DataGridView 컨트롤 개요](#)
- [BindingSource 구성 요소 개요](#)
- [애플리케이션 설정 개요](#)
- [ClickOnce 보안 및 배포](#)





# 새 Windows Form 만들기

2019-10-23 • 3 minutes to read • [Edit Online](#)

이 항목에는 첫 번째 Windows Forms 애플리케이션을 만드는 방법을 설명하는 항목의 링크가 포함되어 있습니다. 또한 이 섹션의 항목에서는 Windows Forms 애플리케이션을 만들 때 알고 있어야 하는 일부 기본 용어와 지침을 소개합니다. Windows Forms 응용 프로그램에 대한 자세한 내용은 이러한, 이벤트 및 이벤트 처리, 그리고 사용자로부터 입력을 처리 하는 방법에 사용할 수 있습니다 컨트롤에는 관련된 항목 목록을 참조 하십시오.

## 섹션 내용

[Windows Forms 좌표](#)합니다.

클라이언트 좌표와 화면 좌표를 설명합니다.

[방법: 명령줄에서 Windows Forms 응용 프로그램 만들기](#)

명령줄에서 기본 Windows Form을 만들고 컴파일하는 방법을 설명합니다.

## 참조

[Form](#)

이 클래스를 설명하고 모든 해당 멤버의 링크를 포함합니다.

[Control](#)

이 클래스를 설명하고 모든 해당 멤버의 링크를 포함합니다.

## 관련 단위

[사용자 입력 처리](#)

Windows Forms 애플리케이션의 사용자 입력 및 처리 방법을 설명하는 항목의 링크를 포함합니다.

[Windows Forms에서 이벤트 처리기 만들기](#)

Windows Forms 애플리케이션의 이벤트 처리 방법을 설명하는 항목의 링크를 포함합니다.

[Windows Forms의 모양 변경](#)

Windows Forms 애플리케이션의 모양을 변경하는 방법을 보여 주는 항목의 링크를 포함합니다.

[기능별 Windows Forms 컨트롤](#)

Windows Forms 애플리케이션에서 사용할 수 있는 컨트롤을 설명하는 항목의 링크를 포함합니다.

# 방법: 명령줄에서 Windows Forms 응용 프로그램 만들기

2020-03-21 • 8 minutes to read • [Edit Online](#)

다음 절차에서는 명령줄에서 Windows Forms 애플리케이션을 만들고 실행하기 위해 완료해야 하는 기본 단계를 설명합니다. Visual Studio에서는 이러한 절차가 광범위하게 지원됩니다. 또한 [연습: WPF에서 Windows 양식 컨트롤 호스팅](#) 참조.

## 절차

폼을 만들려면

1. 빈 코드 파일에서 다음 `Imports` 또는 `using` 문을 입력합니다.

```
using System;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;
```

```
Imports System.ComponentModel
Imports System.Drawing
Imports System.Windows.Forms
```

2. Form 클래스에서 `Form1` 상속되는 클래스를 선언합니다.

```
public class Form1 : Form
```

```
Public Class Form1
    Inherits Form
```

3. 예 대한 `Form1` 매개 변수 없는 생성자 만들기

이후 절차에서 생성자에 더 많은 코드를 추가합니다.

```
public Form1() {}
```

```
Public Sub New()

End Sub
```

4. 클래스에 `Main` 메서드를 추가합니다.

- a. C# `STAThreadAttribute` `Main` 메서드에 적용하여 Windows Forms 응용 프로그램이 단일 스레드 애플리케이션을 지정합니다. (Visual Basic으로 개발된 Windows forms 응용 프로그램은 기본적으로 단일 스레드 아파트 모델을 사용하므로 이 특성은 Visual Basic에서 필요하지 않습니다.)

- b. 응용 `EnableVisualStyles` 프로그램에 운영 체제 스타일을 적용하려면 호출합니다.

- c. 폼 인스턴스를 만들고 실행합니다.

```
[STAThread]
public static void Main()
{
    Application.EnableVisualStyles();
    Application.Run(new Form1());
}
```

```
Public Shared Sub Main()
    Application.EnableVisualStyles()
    Application.Run(New Form1())

End Sub
End Class
```

애플리케이션을 컴파일하고 실행하려면

1. .NET Framework 명령 프롬프트에서 `Form1` 클래스를 만든 디렉터리로 이동합니다.

2. 폼을 컴파일합니다.

- C#을 사용하는 경우 다음을 입력합니다. `csc form1.cs`

-or-

- 시각적 기본을 사용하는 경우 다음을 입력합니다. `vbc form1.vb`

3. 명령 프롬프트에 다음을 입력합니다. `Form1.exe`

## 컨트롤 추가 및 이벤트 처리

이전 절차의 단계에서는 컴파일 및 실행되는 기본 Windows Form을 만드는 방법만 보여 주었습니다. 다음 절차에서는 컨트롤을 만들고 폼에 추가한 다음 컨트롤에 대한 이벤트를 처리하는 방법을 보여 줍니다. Windows 양식에 추가할 수 있는 컨트롤에 대한 자세한 내용은 [Windows 양식 컨트롤](#)을 참조하십시오.

Windows Forms 애플리케이션을 만드는 방법을 이해하는 것은 물론 이벤트 기반 프로그래밍 및 사용자 입력을 처리하는 방법도 이해해야 합니다. 자세한 내용은 [Windows 양식에서 이벤트 처리기 만들기](#) 및 사용자 입력 [처리를](#) 참조하십시오.

단추 컨트롤을 선언하고 해당 **click** 이벤트를 처리하려면

1. `button1`이라는 단추 컨트롤을 선언합니다.
2. 생성자에서 단추를 만들고 해당 [Size](#), [Location](#) 및 [Text](#) 속성을 설정합니다.
3. 폼에 단추를 추가합니다.

다음 코드 예제에서는 단추 컨트롤을 선언 하는 방법을 보여 줍니다.

```
public Button button1;
public Form1()
{
    button1 = new Button();
    button1.Size = new Size(40, 40);
    button1.Location = new Point(30, 30);
    button1.Text = "Click me";
    this.Controls.Add(button1);
    button1.Click += new EventHandler(button1_Click);
}
```

```
Public WithEvents button1 As Button

Public Sub New()
    button1 = New Button()
    button1.Size = New Size(40, 40)
    button1.Location = New Point(30, 30)
    button1.Text = "Click me"
    Me.Controls.Add(button1)

End Sub
```

4. 단추에 대한 [Click](#) 이벤트를 처리할 메서드를 만듭니다.
  5. click 이벤트 처리기에서 "Hello World" 메시지와 함께 [MessageBox](#)를 표시합니다.
- 다음 코드 예제에서는 단추 컨트롤의 클릭 이벤트를 처리하는 방법을 보여 줍니다.

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Hello World");
}
```

```
Private Sub button1_Click(ByVal sender As Object, _
    ByVal e As EventArgs) Handles button1.Click
    MessageBox.Show("Hello World")
End Sub
```

6. 만든 메서드에 [Click](#) 이벤트를 연결합니다.
- 다음 코드 예제에서는 이벤트를 메서드에 연결하는 방법을 보여 줍니다.

```
button1.Click += new EventHandler(button1_Click);
```

```
Private Sub button1_Click(ByVal sender As Object, _
    ByVal e As EventArgs) Handles button1.Click
```

7. 이전 절차에서 설명한 대로 애플리케이션을 컴파일 및 실행합니다.

## 예제

다음 코드 예제는 이전 프로시저의 전체 예제입니다.

```

using System;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;

namespace FormWithButton
{
    public class Form1 : Form
    {
        public Button button1;
        public Form1()
        {
            button1 = new Button();
            button1.Size = new Size(40, 40);
            button1.Location = new Point(30, 30);
            button1.Text = "Click me";
            this.Controls.Add(button1);
            button1.Click += new EventHandler(button1_Click);
        }
        private void button1_Click(object sender, EventArgs e)
        {
            MessageBox.Show("Hello World");
        }
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.Run(new Form1());
        }
    }
}

```

```

Imports System.ComponentModel
Imports System.Drawing
Imports System.Windows.Forms

Public Class Form1
    Inherits Form
    Public WithEvents button1 As Button

    Public Sub New()
        button1 = New Button()
        button1.Size = New Size(40, 40)
        button1.Location = New Point(30, 30)
        button1.Text = "Click me"
        Me.Controls.Add(button1)
    End Sub

    Private Sub button1_Click(ByVal sender As Object, _
        ByVal e As EventArgs) Handles button1.Click
        MessageBox.Show("Hello World")
    End Sub

    <STAThread()> _
    Shared Sub Main()
        Application.EnableVisualStyles()
        Application.Run(New Form1())
    End Sub
End Class

```

- [Form](#)
- [Control](#)
- [Windows Forms의 모양 변경](#)
- [Windows Forms 애플리케이션 강화](#)
- [Windows Forms 시작](#)

# Windows Forms 좌표

2020-02-03 • 5 minutes to read • [Edit Online](#)

Windows Form의 좌표계는 장치 좌표를 기반으로 하며 Windows Forms 그릴 때의 기본 측정 단위는 장치 단위 (일반적으로 픽셀)입니다. 화면의 점은 x 좌표와 y 좌표 쌍으로 표현되며 x 좌표는 오른쪽으로, y 좌표는 위쪽에서 아래쪽으로 늘어납니다. 화면을 기준으로 하는 원본의 위치는 화면 또는 클라이언트 좌표를 지정 하는지 여부에 따라 달라 집니다.

## 화면 좌표

Windows Forms 응용 프로그램 화면 좌표에서 화면에 있는 창의 위치를 지정 합니다. 화면 좌표의 경우 원점은 화면의 왼쪽 위 모퉁이입니다. 창의 전체 위치는 창의 왼쪽 위 모퉁이와 오른쪽 아래 모퉁이를 정의 하는 두 점의 화면 좌표를 포함 하는 [Rectangle](#) 구조로 설명 되는 경우가 많습니다.

## 클라이언트 좌표

Windows Forms 응용 프로그램은 폼 이나 컨트롤에서 클라이언트 좌표를 사용 하여 지점의 위치를 지정 합니다. 클라이언트 좌표에 대 한 원본은 컨트롤이 나 폼의 클라이언트 영역에 대 한 왼쪽 위 모퉁이입니다. 클라이언트 좌표는 화면에 폼 이나 컨트롤의 위치에 관계 없이 응용 프로그램에서 폼 이나 컨트롤을 그릴 때 일관 된 좌표 값을 사용할 수 있도록 합니다.

클라이언트 영역의 차원은 영역에 대 한 클라이언트 좌표를 포함 하는 [Rectangle](#) 구조에도 설명 되어 있습니다. 모든 경우에서 사각형의 왼쪽 위 좌표는 클라이언트 영역에 포함 되 고, 오른쪽 아래 좌표는 제외 됩니다. 그래픽 작업에는 클라이언트 영역의 오른쪽 및 아래쪽 가장자리가 포함 되지 않습니다. 예를 들어 [FillRectangle](#) 메서드는 지정 된 사각형의 오른쪽 및 아래쪽 가장자리에만 채워지고 이러한 가장자리를 포함 하지는 않습니다.

## 한 좌표 유형에 서 다른 유형으로 매핑

경우에 따라 화면 좌표에서 클라이언트 좌표로 매핑해야 할 수도 있습니다. [Control](#) 클래스에서 사용할 수 있는 [PointToClient](#) 및 [PointToScreen](#) 메서드를 사용 하여이를 쉽게 수행할 수 있습니다. 예를 들어 [Control](#)의 [MousePosition](#) 속성은 화면 좌표로 보고 되지만 이러한 속성을 클라이언트 좌표로 변환 하는 것이 좋습니다.

## 참고 항목

- [PointToClient](#)
- [PointToScreen](#)



# Windows Forms에서 이벤트 처리기 만들기

2020-02-03 • 3 minutes to read • [Edit Online](#)

이벤트 처리기는 사용자가 단추를 클릭하거나 메시지 큐에서 메시지를 수신하는 등의 이벤트 발생 시 수행되는 작업을 결정하는 코드 내의 프로시저입니다. 이벤트가 발생하면 해당 이벤트를 수신하는 하나 이상의 이벤트 처리기가 실행됩니다. 이벤트는 여러 처리기에 할당될 수 있으며 특정 이벤트를 처리하는 메서드가 동적으로 변경될 수 있습니다. Visual Studio에서 Windows Forms 디자이너를 사용하여 이벤트 처리기를 만들 수도 있습니다.

## 섹션 내용

### [이벤트 개요](#)

이벤트 모델과 대리자 역할에 대해 설명합니다.

### [이벤트 처리기 개요](#)

이벤트를 처리하는 방법을 설명합니다.

[방법: 런타임에 Windows Forms\에 대한 이벤트 처리기 만들기](#)

[방법: Windows Forms\에서 단일 이벤트 처리기에 여러 이벤트 연결](#)

[Windows Forms\의 이벤트 순서](#)

[방법: 디자이너를 사용하여 이벤트 처리기 만들기](#) Windows Forms 디자이너를 사용하여 이벤트 처리기를 만드는 방법을 설명합니다.

## 관련 섹션

### [이벤트](#)

.NET Framework를 사용하여 이벤트 처리 및 발생에 대한 항목의 링크를 제공 합니다.

[Visual Basic에서 상속된 이벤트 처리기 관련 문제 해결](#)

상속된 구성 요소의 이벤트 처리기에서 발생하는 일반적인 문제에 대해 설명합니다.

# 이벤트 개요(Windows Forms)

2020-02-03 • 9 minutes to read • [Edit Online](#)

이벤트는 코드에서 응답, 즉 "처리"할 수 있는 작업입니다. 마우스를 클릭하거나 키를 누르는 등의 사용자 작업, 프로그램 코드 또는 시스템에 의해 이벤트가 생성될 수 있습니다.

이벤트 구동 애플리케이션은 이벤트에 대한 응답으로 코드를 실행합니다. 각 폼과 컨트롤은 미리 정의된 이벤트 집합을 표시하며 이러한 이벤트에 대해 프로그래밍을 수행할 수 있습니다. 이러한 이벤트 중 하나가 발생할 때 연결된 이벤트 처리기에 코드가 있으면 해당 코드가 호출됩니다.

개체에 의해 발생하는 이벤트 형식은 다양하지만 대부분의 형식은 대다수 컨트롤에 공통적으로 적용됩니다. 예를 들어 대부분의 개체는 [Click](#) 이벤트를 처리합니다. 사용자가 폼을 클릭하면 폼의 [Click](#) 이벤트 처리기에 포함된 코드가 실행됩니다.

## NOTE

대부분의 이벤트는 다른 이벤트와 함께 발생합니다. 예를 들어 [DoubleClick](#) 이벤트가 발생하는 과정에서는 [MouseDown](#), [MouseUp](#) 및 [Click](#) 이벤트도 발생합니다.

이벤트를 발생 시키고 사용 하는 방법에 대 한 자세한 내용은 [이벤트](#)를 참조 하세요.

## 대리자 및 해당 역할

대리자는 이벤트 처리 메커니즘을 빌드하기 위해 .NET Framework 내에서 일반적으로 사용 되는 클래스입니다. 대리자는 일반적으로 시각적 C++ 개체 및 기타 개체 지향 언어에서 사용 되는 함수 포인터와 거의 같습니다. 그러나 함수 포인터와는 달리 대리자는 개체 지향적이고 형식이 안전하며 보안이 유지됩니다. 또한 함수 포인터는 특정 함수에 대한 참조만을 포함하지만 대리자는 개체 참조와 해당 개체 내에 있는 하나 이상의 메서드에 대한 참조로 구성됩니다.

이 이벤트 모델은 *대리자*를 사용 하여 이벤트를 처리 하는 데 사용 되는 메서드에 이벤트를 바인딩합니다. 대리자는 처리기 메서드를 지정하여 다른 클래스가 이벤트 알림을 등록할 수 있도록 설정합니다. 이벤트가 발생하면 대리자가 bound 메서드를 호출합니다. 대리자를 정의 하는 방법에 대 한 자세한 내용은 [이벤트](#)를 참조 하세요.

대리자는 단일 메서드에 바인딩될 수도 있고 여러 메서드에 바인딩(멀티캐스트)될 수도 있습니다. 이벤트에 대한 대리자를 만들 때 (또는 Windows)는 일반적으로 멀티 캐스트 이벤트를 만듭니다. 이때 논리적으로 이벤트당 여러 번 반복되지 않는 특정 절차(예: 대화 상자 표시)를 수행하는 이벤트가 드물지만 예외적으로 발생할 수 있습니다. 멀티 캐스트 대리자를 만드는 방법에 대 한 자세한 내용은 [대리자를 결합 하는 방법 \(멀티 캐스트 대리자\)](#)을 참조 하세요.

멀티캐스트 대리자는 바인딩 대상 메서드의 호출 목록을 유지 관리하며, 호출 목록에 메서드를 추가하기 위한 [Combine](#) 메서드와 해당 메서드를 제거하기 위한 [Remove](#) 메서드를 지원합니다.

애플리케이션이 이벤트를 기록하면 컨트롤은 해당 이벤트의 대리자를 호출하여 이벤트를 발생시킵니다. 그러면 대리자는 bound 메서드를 호출합니다. 가장 일반적인 경우(멀티캐스트 대리자)에서 대리자는 호출 목록의 각 bound 메서드를 차례로 호출하므로 일대다 알림이 제공됩니다. 이러한 전략이 사용되므로 컨트롤이 이벤트 알림을 위해 대상 개체 목록을 유지하지 않아도 됩니다. 대리자가 모든 등록과 알림을 처리하기 때문입니다.

또한 대리자는 여러 이벤트를 같은 메서드에 바인딩할 수 있도록 함으로써 다대일 알림도 허용합니다. 예를 들어 단추 클릭 이벤트와 메뉴 명령 클릭 이벤트 모두가 같은 대리자를 호출할 수 있으며, 이 대리자는 단일 메서드를 호출하여 두 개별 이벤트를 같은 방식으로 처리합니다.

대리자에는 동적 바인딩 메커니즘이 사용되므로 대리자는 런타임에 서명이 이벤트 처리기의 서명과 일치하는 모

든 메서드에 바인딩될 수 있습니다. 이 기능을 사용하면 조건에 따라 bound 메서드를 설정하거나 변경하고 이벤트 처리기를 컨트롤에 동적으로 연결할 수 있습니다.

## 참고 항목

- [Windows Forms에서 이벤트 처리기 만들기](#)
- [이벤트 처리기 개요](#)

# 이벤트 처리기 개요(Windows Forms)

2020-03-21 • 4 minutes to read • [Edit Online](#)

이벤트 처리기는 이벤트에 바인딩된 메서드입니다. 이벤트가 발생하면 이벤트 처리기 내의 코드가 실행됩니다. 각 이벤트 처리기는 이벤트를 올바르게 처리할 수 있는 두 개의 매개 변수를 제공합니다. 다음 예제에서는 [Button](#) 컨트롤의 [Click](#) 이벤트에 대한 이벤트 처리기를 보여 줍니다.

```
Private Sub button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles button1.Click

End Sub
```

```
private void button1_Click(object sender, System.EventArgs e)
{

}
```

```
private:
void button1_Click(System::Object ^ sender,
    System::EventArgs ^ e)
{

}
```

첫 번째 `sender` 매개 변수는 이벤트를 발생시킨 개체에 대한 참조를 제공합니다. 위의 예제에서 `e` 두 번째 매개 변수는 처리 중인 이벤트에 특정한 개체를 전달합니다. 개체의 속성(및 해당 메서드)을 참조하여 마우스 이벤트에 대한 마우스 위치 또는 끌어서 놓기 이벤트에서 전송되는 데이터와 같은 정보를 얻을 수 있습니다.

일반적으로 각 이벤트는 두 번째 매개 변수에 대해 다른 이벤트 개체 형식을 가진 이벤트 처리기를 생성합니다. [MouseDown](#) 및 이벤트와 [MouseUp](#) 같은 일부 이벤트 처리기는 두 번째 매개 변수에 대해 동일한 개체 형식을 갖습니다. 이러한 유형의 이벤트의 경우 동일한 이벤트 처리기를 사용하여 두 이벤트를 모두 처리할 수 있습니다.

동일한 이벤트 처리기를 사용하여 다른 컨트롤에 대해 동일한 이벤트를 처리할 수도 있습니다. 예를 들어 폼에 [RadioButton](#) 컨트롤 그룹이 있는 경우 [Click](#) 이벤트에 대한 단일 이벤트 처리기를 만들고 각 [Click](#) 컨트롤의 이벤트가 단일 이벤트 처리기에 바인딩되어 있을 수 있습니다. 자세한 내용은 [Windows Forms의 단일 이벤트 처리기에 여러 이벤트를 연결하는 방법](#)을 참조하십시오.

## 참고 항목

- [Windows Forms에서 이벤트 처리기 만들기](#)
- [이벤트 개요](#)

# 방법: 런타임에 Windows Forms의 이벤트 처리기 만들기

2020-02-03 • 3 minutes to read • [Edit Online](#)

Visual Studio에서 Windows Forms 디자이너 사용 하여 이벤트를 만드는 것 외에도 런타임에 이벤트 처리기를 만들 수 있습니다. 이렇게 하면 프로그램이 처음 시작될 때 이벤트 처리기를 연결하는 대신 런타임에 코드를 사용하여 조건에 따라 이벤트 처리기를 연결할 수 있습니다.

## 런타임에 이벤트 처리기 만들기

1. 이벤트 처리기를 추가 하려는 폼을 엽니다.
2. 처리할 이벤트의 메서드 시그니처가 있는 양식에 메서드를 추가합니다.

예를 들어 **Button** 컨트롤의 **Click** 이벤트를 처리 하는 경우 다음과 같은 메서드를 만듭니다.

```
Private Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs)
    ' Add event handler code here.
End Sub
```

```
private void button1_Click(object sender, System.EventArgs e)
{
    // Add event handler code here.
}
```

```
private:
    void button1_Click(System::Object ^ sender,
        System::EventArgs ^ e)
    {
        // Add event handler code here.
    }
```

3. 애플리케이션에 적합한 코드를 이벤트 처리기에 추가합니다.
4. 이벤트 처리기를 만들 양식 또는 컨트롤을 결정합니다.
5. 양식의 클래스에 있는 메서드에 이벤트를 처리할 이벤트 처리기를 지정하는 코드를 추가합니다. 예를 들어 다음 코드는 **Button** 컨트롤의 **Click** 이벤트 `button1_Click` 처리할 이벤트 처리기를 지정 합니다.

```
AddHandler Button1.Click, AddressOf Button1_Click
```

```
button1.Click += new EventHandler(button1_Click);
```

```
button1->Click += gcnew System::EventHandler(this, &Form1::button1_Click);
```

위의 Visual Basic 코드에 설명 된 **AddHandler** 메서드는 단추에 대 한 클릭 이벤트 처리기를 설정 합니다.

## 참고 항목

- [Windows Forms에서 이벤트 처리기 만들기](#)
- [이벤트 처리기 개요](#)
- [Visual Basic에서 상속된 이벤트 처리기 관련 문제 해결](#)

# 방법: Windows Forms에서 단일 이벤트 처리기에 여러 이벤트 연결

2020-02-03 • 4 minutes to read • [Edit Online](#)

응용 프로그램 디자인에서 여러 이벤트에 대해 단일 이벤트 처리기를 사용해야 하거나 여러 이벤트에서 동일한 절차를 수행해야 하는 경우가 있을 수 있습니다. 예를 들어, 동일한 기능을 노출하는 경우 메뉴 명령에서 폼의 단추와 동일한 이벤트를 발생시키도록 하는 것이 효과적입니다. 이 작업을 수행하려면 C# 속성 창의 이벤트 뷰를 사용하거나 Visual Basic 코드 편집기의 클래스 이름 및 메서드 이름 드롭다운 상자를 **Handles** 사용합니다.

**Visual Basic**에서 단일 이벤트 처리기에 여러 이벤트를 연결하려면

1. 폼을 마우스 오른쪽 단추로 클릭하고 **코드 보기**를 선택합니다.
2. 클래스 이름 드롭다운 상자에서 이벤트 처리기를 처리하려는 컨트롤 중 하나를 선택합니다.
3. 메서드 이름 드롭다운 상자에서 이벤트 처리기가 처리할 이벤트 중 하나를 선택합니다.
4. 코드 편집기는 적절한 이벤트 처리기를 삽입하고 메서드 내에 삽입 지점을 배치합니다. 아래 예제에서는 **Button** 컨트롤에 대한 **Click** 이벤트입니다.


```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles  
    Button1.Click  
    ' Add event-handler code here.  
End Sub
```

5. 처리하려는 다른 이벤트를 **Handles** 절에 추가합니다.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles  
    Button1.Click, Button2.Click  
    ' Add event-handler code here.  
End Sub
```

6. 이벤트 처리기에 적절한 코드를 추가합니다.

**C#**에서 단일 이벤트 처리기에 여러 이벤트를 연결하려면

1. 이벤트 처리기를 연결하려는 컨트롤을 선택합니다.
2. 속성 창에서 **이벤트 단추** ()를 클릭합니다.
3. 처리하려는 이벤트의 이름을 클릭합니다.
4. 이벤트 이름 옆의 값 섹션에서 드롭다운 단추를 클릭하여 처리하려는 이벤트의 메서드 시그니처와 일치하는 기존 이벤트 처리기의 목록을 표시합니다.
5. 목록에서 적절한 이벤트 처리기를 선택합니다.

기존 이벤트 처리기에 이벤트를 바인딩하기 위해 코드를 폼에 추가합니다.

## 참고 항목

- [Windows Forms에서 이벤트 처리기 만들기](#)
- [이벤트 처리기 개요](#)





# Windows Forms에서의 이벤트 순서

2020-02-03 • 4 minutes to read • [Edit Online](#)

Windows Forms 애플리케이션에서 이벤트가 발생하는 순서는 반대로 이러한 각 이벤트의 처리와 관련된 개발자에게 특히 관심 사항입니다. 폼 부분을 다시 그려야 하는 경우와 같이 이벤트를 세심하게 처리해야 하는 상황에서 런타임에 이벤트가 발생한 정확한 순서를 알아야 합니다. 이 항목에서는 애플리케이션 및 컨트롤 수명에서 여러 중요한 단계 중에 발생하는 이벤트 순서에 대한 세부 정보를 제공합니다. 마우스 입력 이벤트의 순서에 대한 자세한 내용은 [Windows Forms의 마우스 이벤트를 참조](#) 하세요. Windows Forms 이벤트에 대한 개요는 [이벤트 개요](#)를 참조 하세요. 이벤트 처리기의 구성에 대한 자세한 내용은 [이벤트 처리기 개요](#)를 참조 하세요.

## 애플리케이션 시작 및 종료 이벤트

`Form` 및 `Control` 클래스는 애플리케이션 시작 및 종료와 관련된 이벤트 집합을 노출합니다. Windows Forms 애플리케이션이 시작되면 주 폼의 시작 이벤트가 다음 순서대로 발생합니다.

- `Control.HandleCreated`
- `Control.BindingContextChanged`
- `Form.Load`
- `Control.VisibleChanged`
- `Form.Activated`
- `Form.Shown`

애플리케이션이 닫히면 주 폼의 종료 이벤트가 다음 순서대로 발생합니다.

- `Form.Closing`
- `Form.FormClosing`
- `Form.Closed`
- `Form.FormClosed`
- `Form.Deactivate`

`ApplicationExit` 클래스의 `Application` 이벤트는 주 폼의 종료 이벤트 뒤에 발생합니다.

### NOTE

Visual Basic 2005에는 `WindowsFormsApplicationBase.Startup` 및 `WindowsFormsApplicationBase.Shutdown`과 같은 추가 애플리케이션 이벤트가 포함됩니다.

## 포커스 및 유효성 검사 이벤트

키보드(TAB, SHIFT+TAB 등)를 사용하거나, `Select` 또는 `SelectNextControl` 메서드를 호출하거나, `ActiveControl` 속성을 현재 폼으로 설정하여 포커스를 변경하면 `Control` 클래스의 포커스 이벤트가 다음 순서대로 발생합니다.

- `Enter`
- `GotFocus`

- [Leave](#)
- [Validating](#)
- [Validated](#)
- [LostFocus](#)

마우스를 사용하거나 [Focus](#) 메서드를 호출하여 포커스를 변경하면 [Control](#) 클래스의 포커스 이벤트가 다음 순서대로 발생합니다.

- [Enter](#)
- [GotFocus](#)
- [LostFocus](#)
- [Leave](#)
- [Validating](#)
- [Validated](#)

## 참고 항목

- [Windows Forms에서 이벤트 처리기 만들기](#)

# Windows 양식의 크기와 규모 조정

2020-03-21 • 2 minutes to read • [Edit Online](#)

이 항목에서는 Windows Forms의 크기를 조정하는 방법에 대한 정보 링크를 제공합니다.

## 섹션 내용

[방법: Windows Forms 크기 조정](#)

Windows Forms의 크기를 지정하는 방법에 대한 지침을 제공합니다.

[Windows 양식의 자동 크기 조정](#)

자동 크기 조정을 통해 컴퓨터 간에 폼과 컨트롤을 적절하게 표시할 수 있는 방법을 설명합니다.

[Windows 양식에서 높은 DPI 지원](#) 높은 DPI 및 동적 크기 조정에 대한 Windows Forms의 지원에 대해 설명합니다.

## 참조

[Size](#)

이 클래스를 설명하고 모든 해당 멤버의 링크를 포함합니다.

[TableLayoutPanel](#)

이 클래스를 설명하고 모든 해당 멤버의 링크를 포함합니다.

[FlowLayoutPanel](#)

이 클래스를 설명하고 모든 해당 멤버의 링크를 포함합니다.

## 관련 단위

[윈도우 양식의 모양 변경](#)

Windows Forms의 모양을 변경하는 기타 방법을 설명하는 항목의 링크를 제공합니다.

# 방법: Windows Forms 크기 조정

2020-02-03 • 4 minutes to read • [Edit Online](#)

여러가지 방법으로 Windows Form의 크기를 지정할 수 있습니다. [Size](#) 속성에 대해 새 값을 설정하거나 [Height](#) 또는 [Width](#) 속성을 개별적으로 조정하여 프로그래밍 방식으로 폼의 높이와 너비를 모두 변경할 수 있습니다. Visual Studio를 사용 하는 경우 Windows Forms 디자이너를 사용 하여 크기를 변경할 수 있습니다. 또한 [방법: 디자이너를 사용하여 Windows Forms 크기 조정](#)을 참조 하세요.

## 프로그래밍 방식으로 폼 크기 조정

폼의 [Size](#) 속성을 설정하여 런타임에 폼의 크기를 정의합니다.

다음 코드 예제에서는 100 × 100 픽셀로 설정된 폼 크기를 보여 줍니다.

```
Form1.Size = New System.Drawing.Size(100, 100)
```

```
Form1.Size = new System.Drawing.Size(100, 100);
```

```
Form1->Size = System::Drawing::Size(100, 100);
```

## 프로그래밍 방식으로 폼 너비 및 높이 변경

[Size](#)가 정의된 후 [Width](#) 또는 [Height](#) 속성을 사용하여 폼 높이나 너비를 변경합니다.

다음 코드 예제에서는 높이가 일정하게 유지되고 폼의 왼쪽 가장자리에서 300픽셀로 설정된 폼의 너비를 보여 줍니다.

```
Form1.Width = 300
```

```
Form1.Width = 300;
```

```
Form1->Width = 300;
```

또는

[Width](#) 속성을 설정하여 [Height](#) 또는 [Size](#)를 변경합니다.

그러나 다음 코드 예제와 같이 이 접근 방식은 단순히 [Width](#) 또는 [Height](#) 속성을 설정하는 것보다 성가십니다.

```
Form1.Size = New Size(300, Form1.Size.Height)
```

```
Form1.Size = new Size(300, Form1.Size.Height);
```

```
Form1->Size = System::Drawing::Size(300, Form1->Size.Height);
```

## 프로그래밍 방식으로 증가 하 여 폼 크기 변경

폼의 크기를 증가시키려면 **Width** 및 **Height** 속성을 설정합니다.

다음 코드 예제에서는 현재 설정보다 200픽셀 넓게 설정된 폼의 너비를 보여 줍니다.

```
Form1.Width += 200
```

```
Form1.Width += 200;
```

```
Form1->Width += 200;
```

### Caution

**Height** 속성을 새로운 **Width** 구조체로 설정하여 동시에 높이 및 너비 크기를 설정하지 않는 한 항상 **Size** 또는 **Size** 속성을 사용하여 폼의 크기를 변경합니다. **Size** 속성은 값 형식인 **Size** 구조체를 반환합니다. 값 형식의 속성에 새 값을 할당할 수 없습니다. 따라서 다음 코드 예제는 컴파일되지 않습니다.

```
' NOTE: CODE WILL NOT COMPILE
Dim f As New Form()
f.Size.Width += 100
```

```
// NOTE: CODE WILL NOT COMPILE
Form f = new Form();
f.Size.Width += 100;
```

```
// NOTE: CODE WILL NOT COMPILE
Form^ f = gcnew Form();
f->Size->X += 100;
```

## 참고 항목

- [Windows Forms 시작](#)
- [Windows Forms 애플리케이션 강화](#)

# Windows Forms 자동 크기 조정

2020-02-03 • 20 minutes to read • [Edit Online](#)

자동 크기 조정은 한 컴퓨터에서 특정 디스플레이 해상도 또는 시스템 글꼴로 디자인된 폼과 해당 컨트롤이 다른 디스플레이 해상도 또는 시스템 글꼴을 사용하는 다른 컴퓨터에서 제대로 표시될 수 있게 합니다. 이 기능은 사용자 및 다른 개발자 컴퓨터 둘 다의 네이티브 Windows 및 기타 애플리케이션과 일치하도록 폼과 해당 컨트롤의 크기가 지능적으로 조정되도록 합니다. 자동 크기 조정 및 시각적 스타일에 대한 .NET Framework 지원을 통해 .NET Framework 응용 프로그램은 각 사용자 컴퓨터의 네이티브 Windows 응용 프로그램과 비교하여 일관된 모양과 느낌을 유지할 수 있습니다.

대부분의 경우 자동 크기 조정은 .NET Framework 버전 2.0 이상에서 예상대로 작동합니다. 그러나 글꼴 구성표 변경은 문제가 될 수 있습니다. 이 문제를 해결하는 방법에 대한 예제는 [방법: Windows Forms 응용 프로그램에서 글꼴 구성표 변경에 응답](#)을 참조하세요.

## 자동 크기 조정 필요

자동 크기 조정이 없으면 특정 디스플레이 해상도 또는 글꼴로 디자인된 애플리케이션이 해당 해상도나 글꼴이 변경될 경우 너무 작거나 너무 크게 나타납니다. 예를 들어 애플리케이션이 Tahoma 9 포인트를 기준으로 디자인된 경우 조정하지 않으면 시스템 글꼴이 Tahoma 12 포인트인 컴퓨터에서 실행할 경우 너무 작게 나타납니다. 제목, 메뉴, 텍스트 상자 내용 등의 텍스트 요소가 다른 애플리케이션보다 작게 렌더링됩니다. 또한 제목 표시줄, 메뉴, 많은 컨트롤 등 텍스트가 포함된 UI(사용자 인터페이스) 요소의 크기는 사용하는 글꼴에 따라 달라집니다. 이 예제에서는 이러한 요소도 상대적으로 더 작게 나타납니다.

애플리케이션이 특정 디스플레이 해상도로 디자인된 경우 비슷한 상황이 발생합니다. 가장 일반적인 디스플레이 해상도는 96 dpi (인치당 도트 수)입니다. 이 해상도는 100% 표시 크기 조정에 해당하지만, 더 높은 해상도는 125%, 150%, 200% (각각 120, 144 및 192 DPI와 같음) 이상이 점점 더 보편화되고 있습니다. 조정하지 않으면 특정 해상도로 디자인된 애플리케이션, 특히 그래픽 기반 애플리케이션은 다른 해상도로 실행할 경우 너무 크거나 너무 작게 나타납니다.

자동 크기 조정은 상대 글꼴 크기나 디스플레이 해상도에 따라 폼과 해당 자식 컨트롤의 크기를 자동으로 조정하여 이러한 문제를 개선하려고 합니다. Windows 운영 체제는 대화 상자 단위라는 상대 측정 단위를 사용하여 대화 상자의 자동 크기 조정을 지원합니다. 대화 상자 단위는 시스템 글꼴을 기반으로 하며, Win32 SDK 함수

`GetDialogBaseUnits`를 통해 픽셀과의 해당 관계를 확인할 수 있습니다. 사용자가 Windows에서 사용되는 테마를 변경하면 모든 대화 상자가 자동으로 적절하게 조정됩니다. 또한 .NET Framework는 기본 시스템 글꼴 또는 디스플레이 해상도에 따라 자동 크기 조정을 지원합니다. 필요에 따라 애플리케이션에서 자동 크기 조정을 사용하지 않도록 설정할 수 있습니다.

## 자동 크기 조정에 대한 원래 지원

.NET Framework 버전 1.0 및 1.1은 `DEFAULT_GUI_FONT` Win32 SDK 값으로 표시되는 UI에 사용되는 Windows 기본 글꼴에 따라 달라지는 간단한 방식으로 자동 크기 조정을 지원합니다. 이 글꼴은 일반적으로 디스플레이 해상도가 변경되는 경우에만 변경됩니다. 자동 크기 조정을 구현하기 위해 다음 메커니즘이 사용되었습니다.

1. 디자인 타임에 `AutoScaleBaseSize` 속성(이제 사용되지 않음)이 개발자 컴퓨터의 기본 시스템 글꼴 높이와 너비로 설정되었습니다.
2. 런타임에 사용자 컴퓨터의 기본 시스템 글꼴이 `Font` 클래스의 `Form` 속성을 초기화하는 데 사용되었습니다.
3. 폼을 표시하기 전에 `ApplyAutoScaling` 메서드가 폼의 크기 조정을 위해 호출되었습니다. 이 메서드는 `AutoScaleBaseSize` 및 `Font`에서 상대 크기를 계산한 다음 `Scale` 메서드를 호출하여 실제로 폼과 자식의 크기를 조정했습니다.

4. 후속 [AutoScaleBaseSize](#) 호출에서 점진적으로 폼의 크기를 조정하지 않도록 [ApplyAutoScaling](#) 값이 업데이트되었습니다.

이 메커니즘은 대부분의 용도에 충분했지만 다음과 같은 제한 사항이 있었습니다.

- [AutoScaleBaseSize](#) 속성은 기준선 글꼴 크기를 정수 값으로 나타내기 때문에 여러 해상도를 통해 폼이 순환될 때 발생 하는 반올림 오류가 발생 합니다.
- 자동 크기 조정이 [Form](#) 클래스에서만 구현되고 [ContainerControl](#) 클래스에서는 구현되지 않았습니다. 따라서 사용자 정의 컨트롤이 폼과 동일한 해상도로 디자인되고 디자인 타임에 폼에 배치된 경우에만 제대로 크기가 조정되었습니다.
- 컴퓨터 해상도가 동일한 경우에만 여러 개발자가 동시에 폼과 해당 자식 컨트롤을 디자인할 수 있었습니다. 마찬가지로, 폼의 상속이 부모 폼과 연결된 해상도에 따라 달라지도록 했습니다.
- [FlowLayoutPanel](#) 및 [TableLayoutPanel](#)와 같은 .NET Framework 버전 2.0에 도입 된 최신 레이아웃 관리자와 호환 되지 않습니다.
- .NET Compact Framework와의 호환성을 위해 필요한 디스플레이 해상도를 직접 기반으로 하는 크기 조정을 지원 하지 않았습니다.

이 메커니즘은 이전 버전과의 호환성을 유지 하기 위해 .NET Framework 버전 2.0에서 유지 되었지만 다음에 설명된 보다 강력한 크기 조정 메커니즘으로 대체 되었습니다. 따라서 [AutoScale](#), [ApplyAutoScaling](#), [AutoScaleBaseSize](#) 및 특정 [Scale](#) 오버로드는 사용되지 않는 것으로 표시됩니다.

#### NOTE

레거시 코드를 .NET Framework 버전 2.0으로 업그레이드 하는 경우 이러한 멤버에 대 한 참조를 안전 하 게 삭제할 수 있습니다.

## 자동 크기 조정에 대 한 현재 지원

.NET Framework 버전 2.0은 Windows Forms의 자동 크기 조정을 다음과 같이 변경 하여 이전 제한 사항들에서는.

- 폼, 네이티브 복합 컨트롤 및 사용자 정의 컨트롤이 모두 균일한 크기 조정 지원을 받을 수 있도록 크기 조정에 대한 기본 지원이 [ContainerControl](#) 클래스로 이동되었습니다. 새 멤버 [AutoScaleFactor](#), [AutoScaleDimensions](#), [AutoScaleMode](#) 및 [PerformAutoScale](#)이 추가되었습니다.
- [Control](#) 클래스에는 크기 조정에 참여하고 동일한 폼에서 혼합된 크기 조정을 지원할 수 있게 해주는 여러 개의 새 멤버도 있습니다. 구체적으로 [Scale](#), [ScaleChildren](#) 및 [GetScaledBounds](#) 멤버가 크기 조정을 지원합니다.
- [AutoScaleMode](#) 열거형에서 정의된 시스템 글꼴 지원을 보완하기 위해 화면 해상도를 기반으로 하는 크기 조정 지원이 추가되었습니다. 이 모드는 .NET Compact Framework에서 지원하는 자동 크기 조정과 호환되므로 응용 프로그램을 쉽게 마이그레이션할 수 있습니다.
- [FlowLayoutPanel](#) 및 [TableLayoutPanel](#)과 같은 레이아웃 관리자와의 호환성이 자동 크기 조정 구현에 추가되었습니다.
- 이제 배율 인수가 일반적으로 [SizeF](#) 구조체를 사용하여 부동 소수점 값으로 표시되므로 반올림 오류가 거의 제거되었습니다.

#### Caution

DPI 및 글꼴 크기 조정 모드의 임의 혼합은 지원되지 않습니다. 특정 모드(예: DPI)를 사용하여 사용자 정의 컨트롤의 크기를 조정하고 다른 모드(글꼴)를 사용하여 폼에 배치하는 것은 아무 문제가 없지만 특정 모드의 기본 폼과 다른 모드의 파생 폼을 혼합하면 예기치 않은 결과가 발생할 수 있습니다.

자동 크기 조정 작동

이제 Windows Forms에서 다음 논리를 사용하여 폼과 해당 내용의 크기를 자동으로 조정합니다.

1. 디자인 타임에 각 `ContainerControl`이 크기 조정 모드와 현재 해상도를 `AutoScaleMode` 및 `AutoScaleDimensions`에 각각 기록합니다.
2. 런타임에 실제 해상도가 `CurrentAutoScaleDimensions` 속성에 저장됩니다. `AutoScaleFactor` 속성이 런타임 및 디자인 타임 크기 조정 해상도 간의 비율을 동적으로 계산합니다.
3. 폼이 로드될 때 `CurrentAutoScaleDimensions` 및 `AutoScaleDimensions`의 값이 서로 다르면 `PerformAutoScale` 메서드가 호출되어 컨트롤 및 해당 자식의 크기를 조정합니다. 이 메서드는 레이아웃을 일시 중단하고 `Scale` 메서드를 호출하여 실제 크기 조정을 수행합니다. 그 후에 `AutoScaleDimensions` 값이 업데이트되어 점진적 크기 조정을 방지합니다.
4. 다음 상황에서는 `PerformAutoScale`도 자동으로 호출됩니다.
  - 크기 조정 모드가 `OnFontChanged`인 경우 `Font` 이벤트에 대한 응답으로
  - 컨테이너 컨트롤의 레이아웃이 다시 시작되고 `AutoScaleDimensions` 또는 `AutoScaleMode` 속성에서 변경 내용이 검색되는 경우
  - 위에서 암시한 대로 부모 `ContainerControl`의 크기가 조정되는 경우 각 컨테이너 컨트롤은 부모 컨테이너의 배율 인수가 아니라 고유한 배율 인수를 사용하여 자식의 크기를 조정해야 합니다.
5. 자식 컨트롤은 다음과 같은 여러 수단을 통해 해당 크기 조정 동작을 수정할 수 있습니다.
  - `ScaleChildren` 속성을 재정의하여 해당 자식 컨트롤의 크기를 조정할지 여부를 결정할 수 있습니다.
  - `GetScaledBounds` 메서드를 재정의하여 컨트롤의 크기가 조정되는 범위를 조정할 수 있지만 크기 조정 논리는 조정할 수 없습니다.
  - `ScaleControl` 메서드를 재정의하여 현재 컨트롤에 대한 크기 조정 논리를 변경할 수 있습니다.

## 참고 항목

- `AutoScaleMode`
- `Scale`
- `PerformAutoScale`
- `AutoScaleDimensions`
- 비주얼 스타일을 사용하여 컨트롤 렌더링
- 방법: 자동 크기 조정 없이 성능 향상



# 방법: Windows Forms 애플리케이션에서 글꼴 구성표 변경에 응답

2020-02-03 • 7 minutes to read • [Edit Online](#)

Windows 운영 체제에서 사용자는 시스템 수준 글꼴 설정을 변경 하여 기본 글꼴이 더 크거나 작게 표시 되도록 할 수 있습니다. 이러한 글꼴 설정을 변경 하는 것은 시각적으로 손상 된 사용자에게 중요 하며, 화면에서 텍스트를 읽는 데 더 큰 형식이 필요 합니다. 글꼴 구성표가 변경 될 때마다 폼 및 포함 된 모든 텍스트의 크기를 늘리거나 줄여서 이러한 변경 내용에 대응 하도록 Windows Forms 응용 프로그램을 조정할 수 있습니다. 폼에 변경 내용을 동적으로 적용 하려면 폼에 코드를 추가할 수 있습니다.

일반적으로 Windows Forms에서 사용 하는 기본 글꼴은 `GetObject(DEFAULT_GUI_FONT)` 에 대한 [Microsoft.Win32](#) 네임 스페이스 호출에서 반환 되는 글꼴입니다. 이 호출에서 반환 되는 글꼴은 화면 해상도가 변경 되는 경우에만 변경 됩니다. 다음 절차에 표시 된 것 처럼 코드는 글꼴 크기의 변경 내용에 응답 하려면 [IconTitleFont](#) 기본 글꼴을 변경 해야 합니다.

바탕 화면 글꼴을 사용 하고 글꼴 구성표 변경에 응답 하려면

1. 양식을 만들고 원하는 컨트롤을 추가 합니다. 자세한 내용은 [방법: 명령줄에서 Windows Forms 응용 프로그램 만들기](#) 및 [Windows Forms에서 사용할 컨트롤](#)을 참조 하세요.
2. [Microsoft.Win32](#) 네임 스페이스에 대 한 참조를 코드에 추가 합니다.

```
using Microsoft.Win32;
```

```
Imports Microsoft.Win32
```

3. 폼의 생성자에 다음 코드를 추가 하여 필요한 이벤트 처리기를 연결 하고 폼에 사용 되는 기본 글꼴을 변경 합니다.

```
this.Font = SystemFonts.IconTitleFont;  
SystemEvents.UserPreferenceChanged += new  
UserPreferenceChangedEventHandler(SystemEvents_UserPreferenceChanged);  
this.FormClosing += new FormClosingEventHandler(Form1_FormClosing);
```

```
Public Sub New()  
    ' This call is required by the Windows Form Designer.  
    InitializeComponent()  
  
    ' Add any initialization after the InitializeComponent() call.  
    AddHandler SystemEvents.UserPreferenceChanged, New UserPreferenceChangedEventHandler(AddressOf  
SystemEvents_UserPreferenceChangesEventHandler)  
End Sub
```

4. [Window](#) 범주가 변경 될 때 폼이 자동으로 확장 되도록 하는 [UserPreferenceChanged](#) 이벤트에 대 한 처리기를 구현 합니다.

```
void SystemEvents_UserPreferenceChanged(object sender, UserPreferenceChangedEventArgs e)
{
    if (e.Category == UserPreferenceCategory.Window)
    {
        this.Font = SystemFonts.IconTitleFont;
    }
}
```

```
Private Sub SystemEvents_UserPreferenceChangesEventHandler(ByVal sender As Object, ByVal e As
UserPreferenceChangedEventArgs)
    If (e.Category = UserPreferenceCategory.Window) Then
        Me.Font = SystemFonts.IconTitleFont
    End If
End Sub
```

5. 마지막으로 **UserPreferenceChanged** 이벤트 처리기를 분리 하는 **FormClosing** 이벤트에 대 한 처리기를 구현 합니다.

#### IMPORTANT

이 코드를 포함 하지 않으면 응용 프로그램에서 메모리 누수가 발생 합니다.

```
void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    SystemEvents.UserPreferenceChanged -= new
UserPreferenceChangedEventArgs(SystemEvents_UserPreferenceChanged);
}
```

```
Private Sub Form1_FormClosing(ByVal sender As Object, ByVal e As
System.Windows.Forms.FormClosingEventArgs) Handles Me.FormClosing
    RemoveHandler SystemEvents.UserPreferenceChanged, New UserPreferenceChangedEventArgs(AddressOf
SystemEvents_UserPreferenceChangesEventHandler)
End Sub
```

6. 코드를 컴파일하고 실행합니다.

#### Windows XP에서 글꼴 구성표를 수동으로 변경 하려면

1. Windows Forms 응용 프로그램이 실행 되는 동안 Windows 바탕 화면을 마우스 오른쪽 단추로 클릭 하 고 바로 가기 메뉴에서 속성 을 선택 합니다.
2. 표시 속성 대화 상자에서 모양 탭을 클릭 합니다.
3. 글꼴 크기 드롭다운 목록 상자에서 새 글꼴 크기를 선택 합니다.

이제 양식이 바탕 화면 글꼴 구성표의 런타임 변경 내용에 반응 하는 것을 알 수 있습니다. 사용자가 보통, 대형 글꼴 및 초대형 글꼴 사이에서 변경 하는 경우 폼이 글꼴을 변경 하 고 크기를 올바르게 조정 합니다.

## 예제

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using Microsoft.Win32;

namespace WinFormsAutoScaling
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

            this.Font = SystemFonts.IconTitleFont;
            SystemEvents.UserPreferenceChanged += new
UserPreferenceChangedEventHandler(SystemEvents_UserPreferenceChanged);
            this.FormClosing += new FormClosingEventHandler(Form1_FormClosing);
        }

        void SystemEvents_UserPreferenceChanged(object sender, UserPreferenceChangedEventArgs e)
        {
            if (e.Category == UserPreferenceCategory.Window)
            {
                this.Font = SystemFonts.IconTitleFont;
            }
        }

        void Form1_FormClosing(object sender, FormClosingEventArgs e)
        {
            SystemEvents.UserPreferenceChanged -= new
UserPreferenceChangedEventHandler(SystemEvents_UserPreferenceChanged);
        }
    }
}

```

```

Imports Microsoft.Win32

Public Class Form1
    Public Sub New()
        ' This call is required by the Windows Form Designer.
        InitializeComponent()

        ' Add any initialization after the InitializeComponent() call.
        AddHandler SystemEvents.UserPreferenceChanged, New UserPreferenceChangedEventHandler(AddressOf
SystemEvents_UserPreferenceChangesEventHandler)
    End Sub

    Private Sub SystemEvents_UserPreferenceChangesEventHandler(ByVal sender As Object, ByVal e As
UserPreferenceChangedEventArgs)
        If (e.Category = UserPreferenceCategory.Window) Then
            Me.Font = SystemFonts.IconTitleFont
        End If
    End Sub

    Private Sub Form1_FormClosing(ByVal sender As Object, ByVal e As
System.Windows.Forms.FormClosingEventArgs) Handles Me.FormClosing
        RemoveHandler SystemEvents.UserPreferenceChanged, New UserPreferenceChangedEventHandler(AddressOf
SystemEvents_UserPreferenceChangesEventHandler)
    End Sub
End Class

```

이 코드 예제의 생성자에는 Visual Studio에서 새 Windows Forms 프로젝트를 만들 때 정의 되는

`InitializeComponent`에 대한 호출이 포함되어 있습니다. 명령줄에서 응용 프로그램을 빌드하는 경우이 코드 줄을 제거 합니다.

## 참고 항목

- [PerformAutoScale](#)
- [Windows Forms의 자동 크기 조정](#)

# Windows Forms의 높은 DPI 지원

2020-02-03 • 10 minutes to read • [Edit Online](#)

.NET Framework 4.7부터 Windows Forms 일반적인 높은 DPI 및 동적 DPI 시나리오에 대한 향상된 기능이 포함되어 있습니다. 여기에는 다음이 포함됩니다.

- [MonthCalendar](#) 컨트롤 및 [CheckedListBox](#) 컨트롤과 같은 여러 Windows Forms 컨트롤의 크기 조정 및 레이아웃이 개선되었습니다.
- 단일 패스 크기 조정. .NET Framework 4.6 이전 버전에서는 여러 패스를 통해 크기 조정을 수행했습니다. 이로 인해 일부 컨트롤의 크기가 필요 이상으로 조정되었습니다.
- 사용자가 Windows Forms 응용 프로그램이 시작된 후 DPI 또는 배율 인수를 변경하는 동적 DPI 시나리오에 대한 지원.

.NET Framework 4.7부터 시작하는 .NET Framework 버전에서는 향상된 높은 DPI 지원이 옵트인 기능입니다. 응용 프로그램을 사용하도록 구성해야 합니다.

## 높은 DPI 지원을 위해 Windows Forms 앱 구성

높은 DPI 인식을 지원하는 새로운 Windows Forms 기능은 .NET Framework 4.7를 대상으로 하고 Windows 10 크리에이터 업데이트부터 Windows 운영 체제에서 실행되는 응용 프로그램에서만 사용할 수 있습니다.

또한 Windows Forms 응용 프로그램에서 높은 DPI 지원을 구성하려면 다음을 수행해야 합니다.

- Windows 10과의 호환성을 선언합니다.

이렇게 하려면 매니페스트 파일에 다음을 추가합니다.

```
<compatibility xmlns="urn:schemas-microsoft-com:compatibility.v1">
  <application>
    <!-- Windows 10 compatibility -->
    <supportedOS Id="{8e0f7a12-bfb3-4fe8-b9a5-48fd50a15a9a}" />
  </application>
</compatibility>
```

- *App.config* 파일에서 모니터 단위 dpi 인식을 사용하도록 설정합니다.

Windows Forms에서는 새 기능을 지원하고 .NET Framework 4.7부터 추가된 사용자 지정을 지원하기 위해 새로운 `<System.Windows.Forms.ApplicationConfigurationSection>` 요소를 소개합니다. 높은 DPI를 지원하는 새 기능을 활용하려면 응용 프로그램 구성 파일에 다음을 추가합니다.

```
<System.Windows.Forms.ApplicationConfigurationSection>
  <add key="DpiAwareness" value="PerMonitorV2" />
</System.Windows.Forms.ApplicationConfigurationSection>
```

### IMPORTANT

이전 버전의 .NET Framework에서는 매니페스트를 사용하여 높은 DPI 지원을 추가했습니다. 이 방법은 *app.config* 파일에 정의된 설정을 재정의하므로 더 이상 권장되지 않습니다.

- 정적 [EnableVisualStyles](#) 메서드를 호출합니다.

응용 프로그램 진입점의 첫 번째 메서드 호출 이어야 합니다. 예를 들어 다음과 같습니다.

```
static void Main()
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Application.Run(new Form2());
}
```

## 개별 높은 DPI 기능 옵트아웃

`DpiAwareness` 값을 `PerMonitorV2`로 설정 하면 .NET Framework 버전에서 지원하는 모든 높은 DPI 인식 기능을 .NET Framework 4.7부터 사용할 수 있습니다. 일반적으로 대부분의 Windows Forms 응용 프로그램에 적합 합니다. 그러나 하나 이상의 개별 기능을 옵트아웃 (opt out) 할 수 있습니다. 이 작업을 수행 하는 가장 중요 한 이유는 기존 응용 프로그램 코드에서 해당 기능을 이미 처리 하는 것입니다. 예를 들어 응용 프로그램에서 자동 크기 조정을 처리 하는 경우 다음과 같이 자동 크기 조정 기능을 사용 하지 않도록 설정할 수 있습니다.

```
<System.Windows.Forms.ApplicationConfigurationSection>
  <add key="DpiAwareness" value="PerMonitorV2" />
  <add key="EnableWindowsFormsHighDpiAutoResizing" value="false" />
</System.Windows.Forms.ApplicationConfigurationSection>
```

개별 키와 해당 값의 목록은 [Windows Forms Add Configuration 요소](#)를 참조 하세요.

## 새 DPI 변경 이벤트

.NET Framework 4.7부터 3 개의 새 이벤트를 사용 하여 동적 DPI 변경을 프로그래밍 방식으로 처리할 수 있습니다.

- [DpiChangedAfterParent](#)는 부모 컨트롤 또는 폼에 대한 DPI 변경 이벤트가 발생 한 후 컨트롤의 DPI 설정이 프로그래밍 방식으로 변경 될 때 발생 합니다.
- [DpiChangedBeforeParent](#)는 부모 컨트롤 또는 양식에 대한 DPI 변경 이벤트가 발생 하기 전에 컨트롤의 DPI 설정이 프로그래밍 방식으로 변경 될 때 발생 합니다.
- [DpiChanged](#)-폼이 현재 표시 되는 디스플레이 장치에서 DPI 설정이 변경 될 때 발생 합니다.

## 새 도우미 메서드 및 속성

또한 .NET Framework 4.7은 DPI 크기 조정에 대한 정보를 제공 하고 DPI 크기 조정을 수행할 수 있도록 하는 여러 가지 새로운 도우미 메서드 및 속성을 추가 합니다. 여기에는 다음이 포함됩니다.

- [LogicalToDeviceUnits](#)-값을 논리적에서 장치 픽셀로 변환 합니다.
- 비트맵 이미지를 장치에 대한 논리 DPI로 크기를 조정 하는 [ScaleBitmapLogicalToDevice](#).
- [DeviceDpi](#)-현재 장치에 대한 DPI를 반환 합니다.

## 버전 관리 고려 사항

.NET Framework 4.7 및 Windows 10 크리에이터 업데이트를 실행 하는 것 외에도 응용 프로그램은 높은 DPI의 향상 된 기능과 호환 되지 않는 환경에서 실행 될 수 있습니다. 이 경우에는 응용 프로그램에 대한 대체 (fallback)를 개발 해야 합니다. 이 작업을 수행 하여 [사용자 지정 그리기](#) 를 수행 하여 크기 조정을 처리할 수 있습니다.

이렇게 하려면 앱이 실행 되는 운영 체제도 확인 해야 합니다. 다음과 같은 코드를 사용 하여이 작업을 수행할 수 있습니다.

```
// Create a reference to the OS version of Windows 10 Creators Update.
Version OsMinVersion = new Version(10, 0, 15063, 0);

// Access the platform/version of the current OS.
Console.WriteLine(Environment.OSVersion.Platform.ToString());
Console.WriteLine(Environment.OSVersion.VersionString);

// Compare the current version to the minimum required version.
Console.WriteLine(Environment.OSVersion.Version.CompareTo(OsMinVersion));
```

응용 프로그램은 응용 프로그램 매니페스트에서 지원 되는 운영 체제로 나열 되지 않은 경우 Windows 10을 성공적으로 검색 하지 못합니다.

응용 프로그램을 빌드한 .NET Framework 버전을 확인할 수도 있습니다.

```
Console.WriteLine(AppDomain.CurrentDomain.SetupInformation.TargetFrameworkName);
```

## 참고 항목

- [구성 요소 Windows Forms 추가](#)
- [Windows Forms의 크기 및 배율 조정](#)

# Windows Forms의 모양 변경

2020-02-03 • 2 minutes to read • [Edit Online](#)

테두리, 불투명도, 도형, 스타일 변경이나 Windows Forms 애플리케이션에 대한 배경 이미지 설정과 같은 다양한 방법으로 Windows Forms 애플리케이션의 모양을 사용자 지정할 수 있습니다.

## 섹션 내용

방법: [Windows Forms의 테두리 변경](#)

폼의 테두리 스타일을 변경하는 방법을 보여 줍니다.

## 참조

[Form](#)

이 클래스를 설명하고 모든 해당 멤버의 링크를 포함합니다.

[FormBorderStyle](#)

이 열거형을 설명하고 모든 해당 멤버에 대한 설명을 포함합니다.

[VisualStyleRenderer](#)

이 클래스를 설명하고 모든 해당 멤버의 링크를 포함합니다.

[Image](#)

이 클래스를 설명하고 모든 해당 멤버의 링크를 포함합니다.

[Region](#)

이 클래스를 설명하고 모든 해당 멤버의 링크를 포함합니다.

[Color](#)

이 클래스를 설명하고 모든 해당 멤버의 링크를 포함합니다.

## 관련 섹션

[Windows Forms의 크기 및 배율 조정](#)

폼의 크기와 배율을 변경하는 방법을 보여 주는 항목의 링크를 포함합니다.

[Windows Forms의 그래픽 및 그리기](#)

Windows Forms에서 사용자 지정 그리기를 수행하는 방법을 설명하는 항목의 링크를 포함합니다.

[소유자 그리기 지원이 기본 제공되는 컨트롤](#)

Windows Forms 컨트롤의 소유자 그리기 지원을 나열합니다.



# 방법: Windows Forms의 테두리 변경

2020-02-03 • 2 minutes to read • [Edit Online](#)

Windows Forms의 모양과 동작을 결정할 때 선택할 수 있는 여러 테두리 스타일이 있습니다. [FormBorderStyle](#) 속성을 변경하여 폼의 크기 조정 동작을 제어할 수 있습니다. 또한 [FormBorderStyle](#)를 설정하면 캡션 표시줄 표시 방식 및 표시되는 단추에 영향을 줍니다. 자세한 내용은 [FormBorderStyle](#)를 참조하세요.

Visual Studio에서는 이 작업이 광범위하게 지원됩니다.

방법: 디자이너를 사용 하여 Windows Forms 테두리 변경도 참조 하세요.

**Windows Forms**의 테두리 스타일을 프로그래밍 방식으로 설정하려면

- [FormBorderStyle](#) 속성을 원하는 스타일로 설정합니다. 다음 코드 예제에서는 `DlgBx1` 폼의 테두리 스타일을 [FixedDialog](#)로 설정 합니다.

```
DlgBx1.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedDialog
```

```
DlgBx1.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedDialog;
```

```
DlgBx1->FormBorderStyle =  
System::Windows::Forms::FormBorderStyle::FixedDialog;
```

또한 [방법: 디자인 타임에 대화 상자 만들기를](#) 참조 하세요.

또한 선택적 최소화 및 최대화 단추를 제공 하는 폼에 대 한 테두리 스타일을 선택한 경우 이러한 단추 중 하나 또는 둘 다를 사용할지 여부를 지정할 수 있습니다. 이러한 단추는 사용자 환경을 긴밀히 제어하려는 경우에 유용합니다. 최소화 및 최대화 단추는 기본적으로 사용 하도록 설정 되어 있으며 [속성 창](#)을 통해 해당 기능을 조작할 수 있습니다.

## 참고 항목

- [FormBorderStyle](#)
- [FixedDialog](#)
- [Windows Forms 시작](#)

# Windows Forms 컨트롤

2020-02-03 • 3 minutes to read • [Edit Online](#)

Windows Forms 애플리케이션의 사용자 인터페이스를 디자인하고 수정할 때는 컨트롤을 추가하고 정렬하고 배치해야 합니다. 컨트롤은 폼 개체에 포함된 개체입니다. 각 컨트롤 형식은 컨트롤을 특정 목적에 적합하게 만들어 주는 자체적인 속성, 메서드 및 이벤트 집합을 갖습니다. 디자이너에서 컨트롤을 조작하고 런타임에 동적으로 컨트롤을 추가하는 코드를 작성할 수 있습니다.

## 섹션 내용

### [Windows Forms에 컨트롤 넣기](#)

폼의 컨트롤 배치와 관련된 링크를 제공합니다.

### [Windows Forms\에서 컨트롤 정렬](#)

[개별 Windows Forms 컨트롤에 레이블을 지정 하 고](#)이에 대한 바로 가기를 제공  
사용 하는 바로 가기 키, 컨트롤의 텍스트 레이블 및 보조키에 대해 설명합니다.

### [Windows Forms\에서 사용할 컨트롤](#)

### [.NET Framework에서 사용자 지정 Windows Forms 컨트롤 개발](#)

사용자 지정 Windows Forms 컨트롤을 개발하는 데 도움이 되는 배경 정보 및 샘플을 제공합니다.

### [디자인할 때 Windows Forms 컨트롤 개발](#)

디자인 및 상속을 통해 사용자 지정 컨트롤을 만드는 방법을 설명합니다.

## 관련 단원

### [클라이언트 응용 프로그램](#)

Windows 기반 애플리케이션을 개발하는 방법을 개괄적으로 설명합니다.

# Windows Forms에 사용자 입력

2020-02-03 • 3 minutes to read • [Edit Online](#)

Windows Forms에는 관련된 Windows 메시지를 처리하는 동안 발생한 이벤트를 기반으로 하는 사용자 입력 모델이 포함됩니다. 이 섹션의 항목에서는 특정 작업을 수행하는 방법을 보여 주는 코드 예제를 포함하여 마우스 및 키보드 사용자 입력에 대한 정보를 제공합니다.

## 섹션 내용

### [Windows Forms 애플리케이션의 사용자 입력](#)

Windows 메시지를 처리하는 메서드와 사용자 입력 이벤트를 개괄적으로 설명합니다.

### [Windows Forms 애플리케이션의 키보드 입력](#)

키보드 메시지 처리, 다양한 유형의 키 및 키보드 이벤트에 대한 정보를 제공합니다.

### [Windows Forms 애플리케이션의 마우스 입력](#)

마우스 이벤트와 마우스 커서 및 마우스 캡처를 비롯한 다른 마우스 관련 기능에 대한 정보를 제공합니다.

### [방법: 코드에서 마우스 및 키보드 이벤트 시뮬레이션](#)

마우스 및 키보드 입력을 프로그래밍 방식으로 시뮬레이션하는 여러 가지 방법을 보여 줍니다.

### [방법: Windows Forms 컨트롤에서 사용자 입력 이벤트 처리](#)

대부분의 사용자 입력 이벤트를 처리하고 각 이벤트에 대한 정보를 보고하는 코드 예제를 제공합니다.

### [Windows Forms에서 사용자 입력 유효성 검사](#)

Windows Forms 애플리케이션에서 사용자 입력의 유효성을 검사하는 방법을 설명합니다.

## 관련 섹션

또한 [Windows Forms에서 이벤트 처리기 만들기를](#) 참조 하세요.

# Windows Forms 애플리케이션의 사용자 입력

2020-02-03 • 6 minutes to read • [Edit Online](#)

Windows Forms에서 사용자 입력은 Windows 메시지 형식으로 응용 프로그램에 전송 됩니다. 일련의 재정의 가능한 메서드는 응용 프로그램, 폼 및 컨트롤 수준에서 이러한 메시지를 처리 합니다. 이러한 메서드는 마우스 및 키보드 메시지를 받을 때 마우스 또는 키보드 입력에 대한 정보를 가져오기 위해 처리할 수 있는 이벤트를 발생 시킵니다. 대부분의 경우 Windows Forms 응용 프로그램은 이러한 이벤트를 처리 하여 모든 사용자 입력을 처리할 수 있습니다. 응용 프로그램, 폼 또는 컨트롤이 메시지를 수신 하기 전에 특정 메시지를 가로채기 위해 메시지를 처리 하는 메서드 중 하나를 재정의 해야 하는 경우도 있습니다.

## 마우스 및 키보드 이벤트

모든 Windows Forms 컨트롤은 마우스 및 키보드 입력과 관련 된 이벤트 집합을 상속 합니다. 예를 들어, 컨트롤은 [KeyPress](#) 이벤트를 처리 하여 누른 키의 문자 코드를 확인 하거나 컨트롤이 [MouseClick](#) 이벤트를 처리 하여 마우스 클릭 위치를 확인할 수 있습니다. 마우스 및 키보드 이벤트에 대한 자세한 내용은 Windows Forms [키보드 이벤트](#) 및 [마우스 이벤트](#) 사용을 참조 하세요.

## 사용자 입력 메시지를 처리 하는 메서드

폼과 컨트롤에는 메시지 큐의 서로 다른 지점에서 Windows 메시지를 처리 하는 재정의 가능한 메서드 집합과 [IMessageFilter](#) 인터페이스에 대한 액세스 권한이 있습니다. 이러한 메서드에는 모두 Windows 메시지의 하위 수준 세부 정보를 캡슐화 하는 [Message](#) 매개 변수가 있습니다. 이러한 메서드를 구현 하거나 재정의 하여 메시지를 검사 한 다음 메시지를 사용 하거나 메시지 큐의 다음 소비자에 게 전달할 수 있습니다. 다음 표에서는 Windows Forms의 모든 Windows 메시지를 처리 하는 메서드를 보여 줍니다.

방법	참고 사항
<a href="#">PreFilterMessage</a>	이 메서드는 응용 프로그램 수준에서 지연 된 (게시 된) Windows 메시지를 가로칩니다.
<a href="#">PreProcessMessage</a>	이 메서드는 처리 되기 전에 폼 및 컨트롤 수준에서 Windows 메시지를 가로칩니다.
<a href="#">WndProc</a>	이 메서드는 폼 및 컨트롤 수준에서 Windows 메시지를 처리 합니다.
<a href="#">DefWndProc</a>	이 메서드는 폼 및 컨트롤 수준에서 Windows 메시지의 기본 처리를 수행 합니다. 이는 창의 최소 기능을 제공 합니다.
<a href="#">OnNotifyMessage</a>	이 메서드는 처리 된 후 폼 및 컨트롤 수준에서 메시지를 가로칩니다. 이 메서드를 호출 하려면 <a href="#">EnableNotifyMessage</a> 스타일 비트를 설정 해야 합니다.

키보드 및 마우스 메시지는 이러한 메시지 형식과 관련 된 재정의 가능한 추가 메서드 집합에 의해 처리 됩니다. 자세한 내용은 [키보드 입력 작동 방식](#) 및 [Windows Forms에서 마우스 입력이 작동하는 방식](#)을 참조 하세요.

## 참고 항목

- [Windows Forms에 사용자 입력](#)
- [Windows Forms 애플리케이션의 키보드 입력](#)

- Windows Forms 애플리케이션의 마우스 입력

# Windows Forms 애플리케이션의 키보드 입력

2020-02-03 • 2 minutes to read • [Edit Online](#)

Windows Forms는 특정 키 누름에 응답 하는 데 사용할 수 있는 표준 키보드 이벤트를 포함 하며, 응용 프로그램, 폼 및 컨트롤 수준에서 키 누름을 가로채 고 수정 하 고 사용할 수 있는 방법도 제공 합니다.

## 섹션 내용

### 키보드 입력 작동 방식

키보드 메시지를 처리 하 고 키보드 이벤트로 변환 하는 방법을 설명 합니다.

### 키보드 이벤트 사용

키보드 이벤트 유형과 키보드 이벤트 처리기에서 수신 하는 정보에 대 한 정보를 제공 합니다.

### 방법: 표준 컨트롤로 키보드 입력 수정

키 값이 컨트롤에 도달 하기 전에 수정 하는 방법을 보여 주는 코드 예제를 제공 합니다.

### 방법: 누른 보조키 확인

다른 키 외에도 SHIFT, ALT 또는 CTRL 키를 눌렀는지 여부를 확인 하는 방법을 보여 줍니다.

### 방법: 폼 수준에서 키보드 입력 처리

키가 컨트롤에 도달 하기 전에 가로채는 방법을 보여 주는 코드 예제를 제공 합니다.

# 키보드 입력 작동 방식

2019-10-23 • 17 minutes to read • [Edit Online](#)

Windows Forms에서는 Windows 메시지에 대한 응답으로 키보드 이벤트를 발생시켜 키보드 입력을 처리합니다. 대부분의 Windows Forms 애플리케이션에서는 키보드 이벤트를 처리하여 키보드 입력을 단독으로 처리합니다. 그러나 키가 컨트롤에 도달하기 전에 키를 가로채는 등의 고급 키보드 입력 시나리오를 구현하려면 키보드 메시지가 작동하는 방식을 알아야 합니다. 이 항목에서는 Windows Forms에서 인식하는 키 데이터 형식을 설명하고 키보드 메시지가 라우팅되는 방법에 대한 개요를 설명합니다. 키보드 이벤트에 대한 자세한 내용은 [키보드 이벤트 사용](#)을 참조하세요.

## 키 형식

Windows Forms는 비트 [Keys](#) 열거형으로 표시 되는 가상 키 코드로 키보드 입력을 식별 합니다. [Keys](#) 열거를 사용하여 일련의 누른 키를 결합 하여 단일 값을 만들 수 있습니다. 이러한 값은 WM\_KEYDOWN 및 WM\_SYSKEYDOWN Windows 메시지와 함께 제공되는 값에 해당합니다. [KeyDown](#) 또는 [KeyUp](#) 이벤트를 처리 하여 대부분의 물리적 키 누름을 검색할 수 있습니다. 문자 키는 [Keys](#) 열거형의 하위 집합이 며 WM\_CHAR 및 WM\_SYSCHAR Windows 메시지와 함께 제공 되는 값에 해당 합니다. 누른 키의 조합이 문자를 반환 하는 경우 이벤트를 [KeyPress](#) 처리 하여 문자를 검색할 수 있습니다. 또는 Visual Basic 프로그래밍 인터페이스를 [Keyboard](#) 통해 노출 되는 키를 검색 하고 키를 보내는 키를 검색 하는 데 사용할 수 있습니다. 자세한 내용은 [키보드에 액세스](#)를 참조하세요.

## 키보드 이벤트의 순서

앞에서 설명한 대로 한 컨트롤에서 발생할 수 있는 키보드 관련 이벤트는 세 가지입니다. 다음 시퀀스는 키보드 이벤트의 일반적인 순서를 보여 줍니다.

1. 사용자가 "a" 키를 푸시합니다. 키가 전처리 되 고 디스패치 되며 [KeyDown](#) 이벤트가 발생 합니다.
2. 사용자가 "a" 키를 보유 하 고 키가 전처리 되 고 디스패치 되며 [KeyPress](#) 이벤트가 발생 합니다.

이 이벤트는 사용자가 키를 누르고 있을 때 여러 차례 발생합니다.

3. 사용자가 "a" 키를 해제 하 고 키가 전처리 되 고 디스패치 된 [KeyUp](#) 후 이벤트가 발생 합니다.

## 키 전처리

다른 메시지와 마찬가지로 키보드 메시지는 폼 이나 컨트롤 [WndProc](#) 의 메서드에서 처리 됩니다. 그러나 키보드 메시지를 처리 하기 전에 메서드는 [PreProcessMessage](#) 특수 문자 키와 물리적 키를 처리 하기 위해 재정의할 수 있는 메서드를 하나 이상 호출 합니다. 이러한 메서드를 재정의하여 컨트롤에서 메시지를 처리하기 전에 특정 키를 감지하고 필터링할 수 있습니다. 다음 표에서는 수행할 작업과 이와 관련하여 발생하는 메서드를 메서드 발생 순서에 따라 보여 줍니다.

### KeyDown 이벤트의 전처리

작업	관련 메서드	노트
엑셀러레이터 키나 메뉴 바로 가기 같은 명령 키를 확인합니다.	<a href="#">ProcessCmdKey</a>	이 메서드는 일반 키보다 우선되는 명령 키를 처리합니다. 이 메서드가 <code>true</code> 를 반환하는 경우 키 메시지가 디스패치되지 않고 키 이벤트도 발생하지 않습니다. <a href="#">IsInputKey</a> 반환 <code>false</code> 되면 가 호출 됩니다. <code>.</code>

작업	관련 메서드	노트
전처리를 필요로 하는 특수 키 나 <a href="#">KeyDown</a> 이벤트를 발생 시키고 컨트롤로 디스패치할 일반 문자 키를 확인 합니다.	<a href="#">IsInputKey</a>	메서드가를 반환 <code>true</code> 하면 컨트롤이 일반 문자이 <a href="#">KeyDown</a> 고 이벤트가 발생 하는 것을 의미 합니다. 인 경우 <code>false</code> 가 호출됩니다 <a href="#">ProcessDialogKey</a> . <b>참고:</b> 컨트롤이 키 또는 키 조합을 가져오기 위해 원하는 키에 <a href="#">PreviewKeyDown</a> 대 한 이벤트 및의 <a href="#">PreviewKeyDownEventArgs</a> 집합 <a href="#">IsInputKey</a> 을 <code>true</code> 처리할 수 있습니다.
탐색 키(Esc, Tab, Enter 또는 화살표 키)를 확인합니다.	<a href="#">ProcessDialogKey</a>	이 메서드는 컨트롤 내에서 컨트롤과 부모 컨트롤 간의 포커스 전환과 같은 특수 기능을 담당하는 실제 키를 처리 합니다. 직접 실행 컨트롤이 키 <a href="#">ProcessDialogKey</a> 를 처리 하지 않는 경우는 부모 컨트롤에서 호출 되 고 계층의 최상위 컨트롤에 대해 호출 됩니다. 이 메서드에서 <code>true</code> 를 반환하면 전처리는 완료되지만 키 이벤트는 생성되지 않습니다. 가 반환 <code>false</code> 되 면 이벤트가 발생합니다. <a href="#">KeyDown</a>

## KeyPress 이벤트의 전처리

작업	관련 메서드	노트
키가 컨트롤에서 전처리되어야 하는 일반 문자인지 여부를 확인합니다.	<a href="#">IsInputChar</a>	문자가 일반 문자 이면이 메서드스를 반환 <code>true</code> 하 고 이벤트는 <a href="#">KeyPress</a> 발생 하며 추가 전처리는 발생 하지 않습니다. 그렇지 <a href="#">ProcessDialogChar</a> 않으면 가 호출 됩니다.
문자가 단추의 &OK와 같은 니모닉인지 확인하려면 선택합니다.	<a href="#">ProcessDialogChar</a>	와 마찬가지로 <a href="#">ProcessDialogKey</a> 이 메서드는 컨트롤 계층 구조를 통해 호출 됩니다. 컨트롤이 컨테이너 컨트롤이면 자체와 해당 자식 컨트롤에서를 호출 <a href="#">ProcessMnemonic</a> 하 여 니모닉이 있는지 확인 합니다. 가 <a href="#">ProcessDialogChar</a> 를 <code>true</code> 반환하면 이벤트가발생하지 않습니다. <a href="#">KeyPress</a>

## 키보드 메시지 전처리

키보드 메시지는 양식이 나 [WndProc](#) 컨트롤의 메서드에 도달한 후 재정의할 수 있는 메서드 집합에 의해 처리 됩니다. 이러한 각 메서드는 컨트롤에서 [Boolean](#) 키보드 메시지를 처리 하 고 사용 했는지 여부를 지정 하는 값을 반환 합니다. 이러한 메서드 중 한 메서드에서 `true` 를 반환하면 메시지가 처리된 것으로 간주됩니다. 그러므로 컨트롤의 기본 또는 부모에 추가 처리를 위한 메시지를 전달하지 않습니다. 그렇지 않은 경우에는 메시지가 메시지 큐에 유지되고 컨트롤의 기본 또는 부모의 다른 메서드에서 처리됩니다. 다음 표에서는 키보드 메시지를 처리하는 메서드를 보여 줍니다.

메서드	노트
<a href="#">ProcessKeyMessage</a>	이 메서드는 <a href="#">WndProc</a> 컨트롤의 메서드에서 받는 모든 키보드 메시지를 처리 합니다.



메서드	노트
<a href="#">ProcessKeyPreview</a>	이 메서드는 키보드 메시지를 컨트롤의 부모에 보냅니다. 가 <a href="#">ProcessKeyPreview</a> 를 <code>true</code> 반환 하면 키 이벤트가 생성 되지 않고, <a href="#">ProcessKeyEventArgs</a> 그렇지 않으면 호출 됩니다.
<a href="#">ProcessKeyEventArgs</a>	이 메서드는 <a href="#">KeyDown</a> , <a href="#">KeyPress</a> 및 <a href="#">KeyUp</a> 이벤트를 적절 하게 발생 시킵니다.

## 키보드 메서드 재정의

키보드 메시지를 전처리하고 처리할 때 재정의할 수 있는 여러 가지 메서드가 있지만 그중 다음과 같은 메서드가 특히 많이 사용됩니다. 다음 표에서는 수행할 작업과 키보드 메서드를 재정의할 수 있는 가장 좋은 방법을 보여 줍니다. 메서드를 재정의 하는 방법에 대 한 자세한 내용은 [파생 클래스의 속성 및 메서드 재정의](#)를 참조 하세요.

작업	메서드
탐색 키를 가로채 고 이벤트를 <a href="#">KeyDown</a> 발생 시킵니다. Tab 키와 Enter 키를 텍스트 상자에서 처리하려는 경우를 예를 들어보겠습니다.	<a href="#">IsInputKey</a> 을 재정의합니다. <b>참고:</b> 또는 <a href="#">PreviewKeyDown</a> 이벤트를 처리 하 고 원하는 키에 <a href="#">IsInputKey</a> 대 한 <a href="#">PreviewKeyDownEventArgs</a> 를 <code>true</code> 로 설정할 수 있습니다.
컨트롤에서 특수 입력 키나 탐색 키에 대한 처리를 수행합니다. 예를 들어 목록 컨트롤에서 화살표 키를 사용하여 선택한 항목을 변경할 수 있습니다.	<a href="#">ProcessDialogKey</a> 를 재정의합니다.
탐색 키를 가로채 고 이벤트를 <a href="#">KeyPress</a> 발생 시킵니다. 예를 들어 스피ن 상자 컨트롤에서 화살표 키를 여러 차례 눌러 항목 전체를 빠르게 진행할 수 있습니다.	<a href="#">IsInputChar</a> 을 재정의합니다.
이벤트 중에 <a href="#">KeyPress</a> 특수 입력 또는 탐색 처리를 수행 합니다. 예를 들어, 목록 컨트롤에서 "r" 키를 누른 채로 r 문자로 시작하는 항목 사이를 건너뛸 수 있습니다.	<a href="#">ProcessDialogChar</a> 를 재정의합니다.
사용자 지정 니모닉 처리를 수행합니다. 예를 들어 도구 모음에 포함된, 소유자가 그린 단추의 니모닉을 처리할 수 있습니다.	<a href="#">ProcessMnemonic</a> 을 재정의합니다.

## 참고자료

- [Keys](#)
- [WndProc](#)
- [PreProcessMessage](#)
- [My.Computer.Keyboard](#) 개체
- [키보드에 액세스](#)
- [키보드 이벤트 사용](#)

# 키보드 이벤트 사용

2019-10-23 • 6 minutes to read • [Edit Online](#)

대부분 Windows Forms 프로그램에서는 키보드 이벤트를 처리하는 방식으로 키보드 입력을 처리합니다. 이 항목에서는 각 이벤트를 사용하는 시기 및 각 이벤트에 대해 제공되는 데이터에 대한 세부 정보를 포함하여 키보드 이벤트에 대한 개요를 제공합니다. 도 참조 하세요 [이벤트 처리기 개요 \(Windows Forms\)](#) 하 고 [이벤트 개요 \(Windows Forms\)](#)합니다.

## 키보드 이벤트

Windows Forms에서는 사용자가 키보드 키를 누를 때 발생하는 두 가지 이벤트와 사용자가 키보드 키를 놓을 때 발생하는 한 가지 이벤트를 제공합니다.

- 한 번 발생하는 [KeyDown](#) 이벤트.
- 사용자가 같은 키를 누르고 있을 때 여러 번 발생할 수 있는 [KeyPress](#) 이벤트.
- 사용자가 키를 놓을 때 한 번 발생하는 [KeyUp](#) 이벤트.

사용자가 키를 누를 때 Windows Forms에서는 키보드 메시지가 문자 키 또는 물리적 키를 지정하는지에 따라 발생할 이벤트를 결정합니다. 문자 및 물리적 키에 대 한 자세한 내용은 참조 하세요. [키보드 입력 작동 방식](#)합니다.

다음 표에서는 세 가지 키보드 이벤트에 대해 설명합니다.

키보드 이벤트	설명	결과
<a href="#">KeyDown</a>	이 이벤트는 사용자가 물리적 키를 누를 때 발생합니다.	<a href="#">KeyDown</a> 에 대한 처리기는 다음을 수신합니다. <ul style="list-style-type: none"><li>• 물리적 키보드 단추를 지정하는 <a href="#">KeyCode</a> 속성을 제공하는 <a href="#">KeyEventArgs</a> 매개 변수.</li><li>• <a href="#">Modifiers</a> 속성(SHIFT, CTRL 또는 ALT).</li><li>• 키 코드 및 한정자를 결합하는 <a href="#">KeyData</a> 속성. <a href="#">KeyEventArgs</a> 매개 변수는 다음을 제공합니다.<ul style="list-style-type: none"><li>◦ 기본 컨트롤의 키 수신을 방지하도록 설정될 수 있는 <a href="#">Handled</a> 속성.</li><li>◦ 해당 키 입력에 대한 <a href="#">KeyPress</a> 및 <a href="#">KeyUp</a> 이벤트를 억제하는 사용될 수 있는 <a href="#">SuppressKeyPress</a> 속성.</li></ul></li></ul>

키보드 이벤트	설명	결과
KeyPress	이 이벤트는 키를 하나 이상 눌러서 문자가 표시될 때 발생합니다. 예를 들어 사용자가 SHIFT 키와 소문자 "a" 키를 누르면 대문자 "A" 문자가 표시됩니다.	<p>KeyPress는 KeyDown 뒤에 발생합니다.</p> <ul style="list-style-type: none"> <li>KeyPress에 대한 처리기는 다음을 수신합니다.</li> <li>눌린 키의 문자 코드가 포함된 KeyPressEventArgs 매개 변수. 이 문자 코드는 모든 문자 키 및 한정자 키 조합에 대해 고유합니다.</li> </ul> <p>예를 들어 "A" 키는 다음을 생성합니다.</p> <ul style="list-style-type: none"> <li>문자 코드 65, SHIFT 키와 함께 누른 경우</li> <li>또는 CAPS LOCK 키, 97, 키 자체를 누른 경우</li> <li>및 1, CTRL 키와 함께 누른 경우.</li> </ul>
KeyUp	이 이벤트는 사용자가 물리적 키를 놓을 때 발생합니다.	<p>KeyUp에 대한 처리기는 다음을 수신합니다.</p> <ul style="list-style-type: none"> <li>KeyEventArgs 매개 변수: <ul style="list-style-type: none"> <li>물리적 키보드 단추를 지정하는 KeyCode 속성을 제공.</li> <li>Modifiers 속성(SHIFT, CTRL 또는 ALT).</li> <li>키 코드 및 한정자를 결합하는 KeyData 속성.</li> </ul> </li> </ul>

## 참고자료

- Windows Forms 응용 프로그램의 키보드 입력
- 키보드 입력 작동 방식
- Windows Forms 애플리케이션의 마우스 입력

# 방법: 표준 컨트롤로 키보드 입력 수정

2019-10-23 • 11 minutes to read • [Edit Online](#)

Windows Forms는 키보드 입력을 사용 및 수정할 수 있는 기능을 제공합니다. 키 사용은 메시지 큐 아래의 다른 메서드 및 이벤트가 키 값을 수신하지 않도록 메서드 또는 이벤트 처리기 내에서 키를 처리하는 것을 가리킵니다. 키 수정은 메시지 큐 아래의 메서드 및 이벤트 처리기가 다른 키 값을 수신하도록 키 값을 수정하는 것을 가리킵니다. 이 항목에서는 이러한 작업을 수행하는 방법을 보여 줍니다.

키를 사용하려면

- **KeyPress** 이벤트 처리기에서 **KeyPressEventArgs** 클래스의 **Handled** 속성을 `true`로 설정합니다.

또는

**KeyDown** 이벤트 처리기에서 **KeyEventArgs** 클래스의 **Handled** 속성을 `true`로 설정합니다.

## NOTE

**KeyDown** 이벤트 처리기에서 **Handled** 속성을 설정해도 **KeyPress** 및 **KeyUp** 이벤트가 현재 키 입력에 대해 발생하지 않도록 차단되지는 않습니다. 이렇게 하려면 **SuppressKeyPress** 속성을 사용합니다.

다음 예제는 **KeyPress** 이벤트 처리기가 수신한 **KeyPressEventArgs**의 **KeyChar** 속성을 검사하는 `switch` 문에서 발췌한 내용입니다. 이 코드는 'A' 및 'a' 문자 키를 사용합니다.

```
// Consume 'A' and 'a'.
case (char)65:
case (char)97:
    MessageBox.Show("Control.KeyPress: '" +
        e.KeyChar.ToString() + "' consumed.");
    e.Handled = true;
    break;
```

```
' Consume 'A' and 'a'.
Case ChrW(65), ChrW(97)
    MessageBox.Show(("Control.KeyPress: '" + _
        e.KeyChar.ToString() + "' consumed.))
    e.Handled = True
```

표준 문자 키를 수정하려면

- **KeyPress** 이벤트 처리기에서 **KeyPressEventArgs** 클래스의 **KeyChar** 속성을 새 문자 키의 값으로 설정합니다.

다음 예제는 'B'를 'A'로 수정하고 'b'를 'a'로 수정하는 `switch` 문에서 발췌한 내용입니다. 새로운 키 값이 메시지 큐의 다른 메서드 및 이벤트에 전파되도록 **KeyPressEventArgs** 매개 변수의 **Handled** 속성은 `false`로 설정됩니다.

```
// Detect 'B', modify it to 'A', and forward the key.
case (char)66:
    MessageBox.Show("Control.KeyPress: '" +
        e.KeyChar.ToString() + "' replaced by 'A'.");
    e.KeyChar = (char)65;
    e.Handled = false;
    break;

// Detect 'b', modify it to 'a', and forward the key.
case (char)98:
    MessageBox.Show("Control.KeyPress: '" +
        e.KeyChar.ToString() + "' replaced by 'a'.");
    e.KeyChar = (char)97;
    e.Handled = false;
    break;
```

```
' Modify 'B' to 'A' and forward the key.
Case ChrW(66)
    MessageBox.Show(("Control.KeyPress: '" + _
        e.KeyChar.ToString() + "' replaced by 'A'."))
    e.KeyChar = ChrW(65)
    e.Handled = False

' Modify 'b' to 'a' and forward the key.
Case ChrW(98)
    MessageBox.Show(("Control.KeyPress: '" + _
        e.KeyChar.ToString() + "' replaced by 'a'."))
    e.KeyChar = ChrW(97)
    e.Handled = False
```

문자가 아닌 키를 수정하려면

- Windows 메시지를 처리하는 [Control](#) 메서드를 재정의하고, WM\_KEYDOWN 또는 WM\_SYSKEYDOWN 메시지를 검색하고, [Message](#) 매개 변수의 [WParam](#) 속성을 문자가 아닌 새 키를 나타내는 [Keys](#) 값으로 설정합니다.

다음 코드 예제에서는 컨트롤의 [PreProcessMessage](#) 메서드를 재정의하여 F1-F9 키를 검색하고 F3 키 누름을 F1로 수정하는 방법을 보여 줍니다. 키보드 메시지를 가로채기 [Control](#) 위해 재정의할 수 있는 메서드에 대한 자세한 내용은 [Windows Forms 응용 프로그램의 사용자 입력](#) 및 [키보드 입력 작동 방식](#)을 참조 하세요.

```

// Detect F1 through F9 during preprocessing and modify F3.
public override bool PreProcessMessage(ref Message m)
{
    if (m.Msg == WM_KEYDOWN)
    {
        Keys keyCode = (Keys)m.WParam & Keys.KeyCode;

        // Detect F1 through F9.
        switch (keyCode)
        {
            case Keys.F1:
            case Keys.F2:
            case Keys.F3:
            case Keys.F4:
            case Keys.F5:
            case Keys.F6:
            case Keys.F7:
            case Keys.F8:
            case Keys.F9:

                MessageBox.Show("Control.PreProcessMessage: '" +
                    keyCode.ToString() + "' pressed.");

                // Replace F3 with F1, so that ProcessKeyMessage will
                // receive F1 instead of F3.
                if (keyCode == Keys.F3)
                {
                    m.WParam = (IntPtr)Keys.F1;
                    MessageBox.Show("Control.PreProcessMessage: '" +
                        keyCode.ToString() + "' replaced by F1.");
                }
                break;
        }
    }

    // Send all other messages to the base method.
    return base.PreProcessMessage(ref m);
}

```

```

' Detect F1 through F9 during preprocessing and modify F3.
Public Overrides Function PreProcessMessage(ByRef m As Message) _
    As Boolean

    If m.Msg = WM_KEYDOWN Then
        Dim keyCode As Keys = CType(m.WParam, Keys) And Keys.KeyCode

        ' Detect F1 through F9.
        Select Case keyCode
            Case Keys.F1, Keys.F2, Keys.F3, Keys.F4, Keys.F5, _
                Keys.F6, Keys.F7, Keys.F8, Keys.F9

                MessageBox.Show(("Control.PreProcessMessage: '" + _
                    keyCode.ToString() + "' pressed."))

                ' Replace F3 with F1, so that ProcessKeyMessage will
                ' receive F1 instead of F3.
                If keyCode = Keys.F3 Then
                    m.WParam = CType(Keys.F1, IntPtr)
                    MessageBox.Show(("Control.PreProcessMessage: '" + _
                        keyCode.ToString() + "' replaced by F1."))
                End If
            End Select
        End If

        ' Send all other messages to the base method.
        Return MyBase.PreProcessMessage(m)
    End Function

```

## 예제

다음 코드 예제는 이전 섹션의 코드 예제에 대한 전체 애플리케이션입니다. 응용 프로그램은 [TextBox](#) 클래스에서 파생된 사용자 지정 컨트롤을 통해 키보드 입력을 사용 및 수정합니다.

```

using System;
using System.Drawing;
using System.Windows.Forms;

namespace KeyboardInput
{
    [System.Security.Permissions.PermissionSet(System.Security.Permissions.SecurityAction.Demand,
        Name="FullTrust")]
    class Form1 : Form
    {
        // The following Windows message value is defined in Winuser.h.
        private int WM_KEYDOWN = 0x100;
        CustomTextBox CustomTextBox1 = new CustomTextBox();

        [STAThread]
        public static void Main()
        {
            Application.EnableVisualStyles();
            Application.Run(new Form1());
        }

        public Form1()
        {
            this.AutoSize = true;
            this.Controls.Add(CustomTextBox1);
            CustomTextBox1.KeyPress +=
                new KeyPressEventHandler(CustomTextBox1_KeyPress);
        }

        // Consume and modify several character keys.
        void CustomTextBox1_KeyPress(object sender, KeyPressEventArgs e)

```

```

void CustomTextBox_KeyPress(object sender, KeyEventArgs e)
{
    switch (e.KeyChar)
    {
        // Consume 'A' and 'a'.
        case (char)65:
        case (char)97:
            MessageBox.Show("Control.KeyPress: '" +
                e.KeyChar.ToString() + "' consumed.");
            e.Handled = true;
            break;

        // Detect 'B', modify it to 'A', and forward the key.
        case (char)66:
            MessageBox.Show("Control.KeyPress: '" +
                e.KeyChar.ToString() + "' replaced by 'A'.");
            e.KeyChar = (char)65;
            e.Handled = false;
            break;

        // Detect 'b', modify it to 'a', and forward the key.
        case (char)98:
            MessageBox.Show("Control.KeyPress: '" +
                e.KeyChar.ToString() + "' replaced by 'a'.");
            e.KeyChar = (char)97;
            e.Handled = false;
            break;
    }
}

[System.Security.Permissions.PermissionSet(System.Security.Permissions.SecurityAction.Demand,
Name="FullTrust")]
public class CustomTextBox : TextBox
{
    // The following Windows message value is defined in Winuser.h.
    private int WM_KEYDOWN = 0x100;

    public CustomTextBox()
    {
        this.Size = new Size(100, 100);
        this.AutoSize = false;
    }

    // Detect F1 through F9 during preprocessing and modify F3.
    public override bool PreProcessMessage(ref Message m)
    {
        if (m.Msg == WM_KEYDOWN)
        {
            Keys keyCode = (Keys)m.WParam & Keys.KeyCode;

            // Detect F1 through F9.
            switch (keyCode)
            {
                case Keys.F1:
                case Keys.F2:
                case Keys.F3:
                case Keys.F4:
                case Keys.F5:
                case Keys.F6:
                case Keys.F7:
                case Keys.F8:
                case Keys.F9:

                    MessageBox.Show("Control.PreProcessMessage: '" +
                        keyCode.ToString() + "' pressed.");

                    // Replace F3 with F1, so that ProcessKeyMessage will
                    // receive F1 instead of F3.
                    if (keyCode == Keys.F3)
                    {

```



```

        {
            m.WParam = (IntPtr)Keys.F1;
            MessageBox.Show("Control.PreProcessMessage: '" +
                keyCode.ToString() + "' replaced by F1.");
        }
        break;
    }
}

// Send all other messages to the base method.
return base.PreProcessMessage(ref m);
}

// Detect F1 through F9 during processing.
protected override bool ProcessKeyMessage(ref Message m)
{
    if (m.Msg == WM_KEYDOWN)
    {
        Keys keyCode = (Keys)m.WParam & Keys.KeyCode;

        // Detect F1 through F9.
        switch (keyCode)
        {
            case Keys.F1:
            case Keys.F2:
            case Keys.F3:
            case Keys.F4:
            case Keys.F5:
            case Keys.F6:
            case Keys.F7:
            case Keys.F8:
            case Keys.F9:

                MessageBox.Show("Control.ProcessKeyMessage: '" +
                    keyCode.ToString() + "' pressed.");
                break;
        }
    }

    // Send all messages to the base method.
    return base.ProcessKeyMessage(ref m);
}
}
}

```

```

Imports System.Drawing
Imports System.Security
Imports System.Security.Permissions
Imports System.Windows.Forms

Namespace KeyboardInput
<System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.Demand,
Name:="FullTrust")> _
    Class Form1
        Inherits Form

        ' The following Windows message value is defined in Winuser.h.
        Private WM_KEYDOWN As Integer = &H100
        Private WithEvents CustomTextBox1 As New CustomTextBox()

        <STAThread> _
        Public Shared Sub Main()
            Application.EnableVisualStyles()
            Application.Run(New Form1())
        End Sub

        Public Sub New()

```

```

Public Sub New()
    Me.AutoSize = True
    Me.Controls.Add(CustomTextBox1)
End Sub

' Consume and modify several character keys.
Sub CustomTextBox1_KeyPress(ByVal sender As Object, _
    ByVal e As KeyPressEventArgs) Handles CustomTextBox1.KeyPress

    Select Case e.KeyChar

        ' Consume 'A' and 'a'.
        Case ChrW(65), ChrW(97)
            MessageBox.Show(("Control.KeyPress: '" + _
                e.KeyChar.ToString() + "' consumed."))
            e.Handled = True

        ' Modify 'B' to 'A' and forward the key.
        Case ChrW(66)
            MessageBox.Show(("Control.KeyPress: '" + _
                e.KeyChar.ToString() + "' replaced by 'A'."))
            e.KeyChar = ChrW(65)
            e.Handled = False

        ' Modify 'b' to 'a' and forward the key.
        Case ChrW(98)
            MessageBox.Show(("Control.KeyPress: '" + _
                e.KeyChar.ToString() + "' replaced by 'a'."))
            e.KeyChar = ChrW(97)
            e.Handled = False

    End Select
End Sub
End Class

<System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.Demand,
Name:="FullTrust")> _
Public Class CustomTextBox
    Inherits TextBox

    ' The following Windows message value is defined in Winuser.h.
    Private WM_KEYDOWN As Integer = &H100

    Public Sub New()
        Me.Size = New Size(100, 100)
        Me.AutoSize = False
    End Sub

    ' Detect F1 through F9 during preprocessing and modify F3.
    Public Overrides Function PreProcessMessage(ByRef m As Message) _
        As Boolean

        If m.Msg = WM_KEYDOWN Then
            Dim keyCode As Keys = CType(m.WParam, Keys) And Keys.KeyCode

            ' Detect F1 through F9.
            Select Case keyCode
                Case Keys.F1, Keys.F2, Keys.F3, Keys.F4, Keys.F5, _
                    Keys.F6, Keys.F7, Keys.F8, Keys.F9

                    MessageBox.Show(("Control.PreProcessMessage: '" + _
                        keyCode.ToString() + "' pressed."))

                    ' Replace F3 with F1, so that ProcessKeyMessage will
                    ' receive F1 instead of F3.
                    If keyCode = Keys.F3 Then
                        m.WParam = CType(Keys.F1, IntPtr)
                        MessageBox.Show(("Control.PreProcessMessage: '" + _
                            keyCode.ToString() + "' replaced by F1."))
                    End If
                End Select
            End If
        End If
    End Function
End Class

```

```

        End Select
    End If

    ' Send all other messages to the base method.
    Return MyBase.PreProcessMessage(m)
End Function

' Detect F1 through F9 during processing.
Protected Overrides Function ProcessKeyMessage(ByRef m As Message) _
    As Boolean

    If m.Msg = WM_KEYDOWN Then
        Dim keyCode As Keys = CType(m.WParam, Keys) And Keys.KeyCode

        ' Detect F1 through F9.
        Select Case keyCode
            Case Keys.F1, Keys.F2, Keys.F3, Keys.F4, Keys.F5, _
                Keys.F6, Keys.F7, Keys.F8, Keys.F9

                MessageBox.Show(("Control.ProcessKeyMessage: '" + _
                    keyCode.ToString() + "' pressed."))
        End Select
    End If

    ' Send all messages to the base method.
    Return MyBase.ProcessKeyMessage(m)
End Function

End Class
End Namespace

```

## 코드 컴파일

이 예제에는 다음 사항이 필요합니다.

- System, System.Drawing 및 System.Windows.Forms 어셈블리에 대한 참조

## 참고자료

- [Windows Forms 응용 프로그램의 키보드 입력](#)
- [Windows Forms 응용 프로그램의 사용자 입력](#)
- [키보드 입력 작동 방식](#)

# 방법: 누른 보조 키 확인

2019-10-23 • 4 minutes to read • [Edit Online](#)

사용자의 키 입력을 허용 하는 응용 프로그램을 만들 때 SHIFT, ALT, CTRL 키 등 보조키를 모니터링 해야 할 수도 있습니다. 보조키를 다른 키와 함께 누르거나 마우스를 클릭 하면 응용 프로그램이 적절 하게 응답할 수 있습니다. 예를 들어 문자 S를 누르면 "s"가 화면에 표시 될 수 있지만 CTRL + S 키를 누르면 현재 문서가 저장 될 수 있습니다. `KeyDown` 이벤트를 처리 하는 경우 이벤트 `Modifiers` 처리기에서 `KeyEventArgs` 받은 속성은 키를 누르는 보조키를 지정 합니다. 또는의 `KeyData KeyEventArgs` 속성은 누른 문자 뿐만 아니라 비트 or로 결합 된 모든 보조키를 지정 합니다. 그러나 `KeyPress` 이벤트 또는 마우스 이벤트를 처리 하는 경우 이벤트 처리기는이 정보를 받지 않습니다. 이 경우 `ModifierKeys Control` 클래스의 속성을 사용 해야 합니다. 두 경우 모두 적절 `Keys` 한 값 및 테스트 하는 값의 비트 and를 수행 해야 합니다. 열거형 `Keys` 은 각 보조키의 변형을 제공 하므로 올바른 값으로 비트 and를 수행 하는 것이 중요 합니다. 예를 들어 shift 키는 `Shift RShiftKey` , `ShiftKey` 및 `LShiftKey` 로 표시 되며 shift 키 `Shift`를 보조키로 테스트 하는 데 올바른 값은입니다. 마찬가지로 CTRL 및 ALT 키를 보조키로 테스트 하려면 각각 `Control` 및 `Alt` 값을 사용 해야 합니다.

## NOTE

Visual Basic 프로그래머는 속성을 통해 키 정보에 `Keyboard` 액세스할 수도 있습니다.

누른 보조키를 확인 하려면

- `ModifierKeys` 속성 `AND` 및 열거형 값과 함께 비트 연산자를 사용하여 특정 보조키를 눌렀는지 여부를 확인 합니다. `Keys` 다음 코드 예제에서는 `KeyPress` 이벤트 처리기 내에서 SHIFT 키를 눌렀는지 여부를 확인 하는 방법을 보여 줍니다.

```
private:
    void textBox1_KeyPress(Object^ sender, KeyPressEventArgs^ e)
    {
        if ((Control::ModifierKeys & Keys::Shift) == Keys::Shift)
        {
            MessageBox::Show("Pressed " + Keys::Shift.ToString());
        }
    }
}
```

```
public void TextBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    if ((Control.ModifierKeys & Keys.Shift) == Keys.Shift)
    {
        MessageBox.Show("Pressed " + Keys.Shift);
    }
}
```

```
Public Sub TextBox1_KeyPress(ByVal sender As Object, _
    ByVal e As KeyPressEventArgs) Handles TextBox1.KeyPress

    If ((Control.ModifierKeys And Keys.Shift) = Keys.Shift) Then
        MsgBox("Pressed " + Keys.Shift.ToString())
    End If
End Sub
```

## 참고자료

- [Keys](#)
- [ModifierKeys](#)
- [Windows Forms 응용 프로그램의 키보드 입력](#)
- 방법: Visual Basic에서 CapsLock가 설정 되어 있는지 확인

# 방법: 양식 수준에서 키보드 입력 처리

2019-10-23 • 8 minutes to read • [Edit Online](#)

Windows Forms에서는 메시지가 컨트롤에 도달하기 전에 폼 수준에서 키보드 메시지를 처리하는 기능을 제공합니다. 이 항목에서는 다음 작업을 수행하는 방법에 대해 설명합니다.

폼 수준에서 키보드 메시지를 처리하려면 다음을 수행합니다.

- 시작 폼의 [KeyPress](#) 또는 [KeyDown](#) 이벤트를 처리하고 키보드 메시지가 폼의 컨트롤에 도달하기 전에 폼이 이 메시지를 수신하도록 폼의 [KeyPreview](#) 속성을 `true`로 설정합니다. 다음 코드 예제에서는 모든 숫자 키를 감지하고 '1', '4', '7'을 사용하여 the [KeyPress](#) 이벤트를 처리합니다.

```
// Detect all numeric characters at the form level and consume 1,
// 4, and 7. Note that Form.KeyPreview must be set to true for this
// event handler to be called.
private:
void Form1_KeyPress(Object^ sender, KeyPressEventArgs^ e)
{
    if ((e->KeyChar >= '0') && (e->KeyChar <= '9'))
    {
        MessageBox::Show("Form.KeyPress: '" +
            e->KeyChar.ToString() + "' pressed.");

        switch (e->KeyChar)
        {
            case '1':
            case '4':
            case '7':
                MessageBox::Show("Form.KeyPress: '" +
                    e->KeyChar.ToString() + "' consumed.");
                e->Handled = true;
                break;
        }
    }
}
```

```
// Detect all numeric characters at the form level and consume 1,
// 4, and 7. Note that Form.KeyPreview must be set to true for this
// event handler to be called.
void Form1_KeyPress(object sender, KeyPressEventArgs e)
{
    if (e.KeyChar >= 48 && e.KeyChar <= 57)
    {
        MessageBox.Show("Form.KeyPress: '" +
            e.KeyChar.ToString() + "' pressed.");

        switch (e.KeyChar)
        {
            case (char)49:
            case (char)52:
            case (char)55:
                MessageBox.Show("Form.KeyPress: '" +
                    e.KeyChar.ToString() + "' consumed.");
                e.Handled = true;
                break;
        }
    }
}
```

```

' Detect all numeric characters at the form level and consume 1,
' 4, and 7. Note that Form.KeyPreview must be set to true for this
' event handler to be called.
Sub Form1_KeyPress(ByVal sender As Object, _
    ByVal e As KeyPressEventArgs) Handles Me.KeyPress

    If e.KeyChar >= ChrW(48) And e.KeyChar <= ChrW(57) Then
        MessageBox.Show(("Form.KeyPress: '" + _
            e.KeyChar.ToString() + "' pressed."))

        Select Case e.KeyChar
            Case ChrW(49), ChrW(52), ChrW(55)
                MessageBox.Show(("Form.KeyPress: '" + _
                    e.KeyChar.ToString() + "' consumed."))
                e.Handled = True
        End Select
    End If
End Sub

```

## 예제

다음 코드 예제는 위 예제에 대한 전체 애플리케이션입니다. 애플리케이션에는 [TextBox](#)에서 포커스를 이동할 수 있도록 하는 여러 가지 다른 컨트롤과 함께 [TextBox](#)가 포함됩니다. 기본 [Form](#)의 [KeyPress](#) 이벤트는 '1', '4', '7'을 사용하고 [TextBox](#)의 [KeyPress](#) 이벤트는 '2', '5', '8'을 사용하면서 나머지 키를 표시합니다. 포커스가 [TextBox](#)에 있을 때 [MessageBox](#) 출력과 포커스가 다른 컨트롤의 하나에 있는 동안 숫자 키를 누를 때 [MessageBox](#) 출력을 비교합니다.

```

#using <System.Drawing.dll>
#using <System.Windows.Forms.dll>
#using <System.dll>

using namespace System;
using namespace System::Drawing;
using namespace System::Windows::Forms;
using namespace System::Security::Permissions;

namespace KeyboardInputForm
{
    public ref class Form1 sealed: public Form, public IMessageFilter
    {
        // The following Windows message value is defined in Winuser.h.
    private:
        static const int WM_KEYDOWN = 0x100;
    private:
        TextBox^ inputTextBox;
    public:
        Form1()
        {
            inputTextBox = gcnew TextBox();
            this->AutoSize = true;
            Application::AddMessageFilter(this);
            FlowLayoutPanel^ panel = gcnew FlowLayoutPanel();
            panel->AutoSize = true;
            panel->FlowDirection = FlowDirection::TopDown;
            panel->Controls->Add(gcnew Button());
            panel->Controls->Add(gcnew RadioButton());
            panel->Controls->Add(inputTextBox);
            this->Controls->Add(panel);
            this->KeyPreview = true;
            this->KeyPress +=
                gcnew KeyPressEventHandler(this, &Form1::Form1_KeyPress);
            inputTextBox->KeyPress +=
                gcnew KeyPressEventHandler(this,
                    &Form1::inputTextBox_KeyPress);
        }
    }
}

```

```

// Detect all numeric characters at the
// application level and consume 0.
[SecurityPermission(SecurityAction::LinkDemand,
    Flags=SecurityPermissionFlag::UnmanagedCode)]
virtual bool PreFilterMessage(Message% m)
{
    // Detect key down messages.
    if (m.Msg == WM_KEYDOWN)
    {
        Keys keyCode = (Keys)((int)m.WParam) & Keys::KeyCode;
        // Determine whether the keystroke is a number from the top of
        // the keyboard, or a number from the keypad.
        if (((keyCode >= Keys::D0) && (keyCode <= Keys::D9))
            || ((keyCode >= Keys::NumPad0)
                && (keyCode <= Keys::NumPad9)))
        {
            MessageBox::Show(
                "IMessageFilter.PreFilterMessage: '" +
                keyCode.ToString() + "' pressed.");

            if ((keyCode == Keys::D0) || (keyCode == Keys::NumPad0))
            {
                MessageBox::Show(
                    "IMessageFilter.PreFilterMessage: '" +
                    keyCode.ToString() + "' consumed.");
                return true;
            }
        }
    }

    // Forward all other messages.
    return false;
}

// Detect all numeric characters at the form level and consume 1,
// 4, and 7. Note that Form.KeyPreview must be set to true for this
// event handler to be called.
private:
void Form1_KeyPress(Object^ sender, KeyPressEventArgs^ e)
{
    if ((e->KeyChar >= '0') && (e->KeyChar <= '9'))
    {
        MessageBox::Show("Form.KeyPress: '" +
            e->KeyChar.ToString() + "' pressed.");

        switch (e->KeyChar)
        {
            case '1':
            case '4':
            case '7':
                MessageBox::Show("Form.KeyPress: '" +
                    e->KeyChar.ToString() + "' consumed.");
                e->Handled = true;
                break;
        }
    }
}

// Detect all numeric characters at the TextBox level and consume
// 2, 5, and 8.
private:
void inputTextBox_KeyPress(Object^ sender, KeyPressEventArgs^ e)
{
    if ((e->KeyChar >= '0') && (e->KeyChar <= '9'))
    {
        MessageBox::Show("Control.KeyPress: '" +
            e->KeyChar.ToString() + "' pressed.");
    }
}

```



```

        switch (e->KeyChar)
        {
            case '2':
            case '5':
            case '8':
                MessageBox::Show("Control.KeyPress: '" +
                    e->KeyChar.ToString() + "' consumed.");
                e->Handled = true;
                break;
        }
    }
}

};

}

[STAThread]
int main()
{
    Application::EnableVisualStyles();
    Application::Run(gcnew KeyboardInputForm::Form1());
}

```

```

using System;
using System.Drawing;
using System.Windows.Forms;

namespace KeyboardInputForm
{
    class Form1 : Form
    {
        TextBox TextBox1 = new TextBox();

        [STAThread]
        public static void Main()
        {
            Application.EnableVisualStyles();
            Application.Run(new Form1());
        }

        public Form1()
        {
            this.AutoSize = true;

            FlowLayoutPanel panel = new FlowLayoutPanel();
            panel.AutoSize = true;
            panel.FlowDirection = FlowDirection.TopDown;
            panel.Controls.Add(TextBox1);
            this.Controls.Add(panel);

            this.KeyPreview = true;
            this.KeyPress +=
                new KeyPressEventHandler(Form1_KeyPress);
            TextBox1.KeyPress +=
                new KeyPressEventHandler(TextBox1_KeyPress);
        }

        // Detect all numeric characters at the form level and consume 1,
        // 4, and 7. Note that Form.KeyPreview must be set to true for this
        // event handler to be called.
        void Form1_KeyPress(object sender, KeyPressEventArgs e)
        {
            if (e.KeyChar >= 48 && e.KeyChar <= 57)
            {
                MessageBox.Show("Form.KeyPress: '" +
                    e.KeyChar.ToString() + "' pressed.");

                switch (e.KeyChar)

```

```

        {
            case (char)49:
            case (char)52:
            case (char)55:
                MessageBox.Show("Form.KeyPress: '" +
                    e.KeyChar.ToString() + "' consumed.");
                e.Handled = true;
                break;
        }
    }
}

// Detect all numeric characters at the TextBox level and consume
// 2, 5, and 8.
void TextBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    if (e.KeyChar >= 48 && e.KeyChar <= 57)
    {
        MessageBox.Show("Control.KeyPress: '" +
            e.KeyChar.ToString() + "' pressed.");

        switch (e.KeyChar)
        {
            case (char)50:
            case (char)53:
            case (char)56:
                MessageBox.Show("Control.KeyPress: '" +
                    e.KeyChar.ToString() + "' consumed.");
                e.Handled = true;
                break;
        }
    }
}
}
}
}
}

```

```

Imports System.Drawing
Imports System.Windows.Forms

Namespace KeyboardInputForm

    Class Form1
        Inherits Form

        Private WithEvents TextBox1 As New TextBox()

        <STAThread()> _
        Public Shared Sub Main()
            Application.EnableVisualStyles()
            Application.Run(New Form1())
        End Sub

        Public Sub New()
            Me.AutoSize = True

            Dim panel As New FlowLayoutPanel()
            panel.AutoSize = True
            panel.FlowDirection = FlowDirection.TopDown
            panel.Controls.Add(TextBox1)
            Me.Controls.Add(panel)

            Me.KeyPreview = True
        End Sub

        ' Detect all numeric characters at the form level and consume 1,
        ' 4, and 7. Note that Form.KeyPreview must be set to true for this
        ' event handler to be called.
    End Class
End Namespace

```

```

Sub Form1_KeyPress(ByVal sender As Object, _
    ByVal e As KeyEventArgs) Handles Me.KeyPress

    If e.KeyChar >= ChrW(48) And e.KeyChar <= ChrW(57) Then
        MessageBox.Show(("Form.KeyPress: '" + _
            e.KeyChar.ToString() + "' pressed."))

        Select Case e.KeyChar
            Case ChrW(49), ChrW(52), ChrW(55)
                MessageBox.Show(("Form.KeyPress: '" + _
                    e.KeyChar.ToString() + "' consumed."))
                e.Handled = True
        End Select
    End If
End Sub

' Detect all numeric characters at the TextBox level and consume
' 2, 5, and 8.
Sub TextBox1_KeyPress(ByVal sender As Object, _
    ByVal e As KeyEventArgs) Handles TextBox1.KeyPress

    If e.KeyChar >= ChrW(48) And e.KeyChar <= ChrW(57) Then
        MessageBox.Show(("Control.KeyPress: '" + _
            e.KeyChar.ToString() + "' pressed."))

        Select Case e.KeyChar
            Case ChrW(50), ChrW(53), ChrW(56)
                MessageBox.Show(("Control.KeyPress: '" + _
                    e.KeyChar.ToString() + "' consumed."))
                e.Handled = True
        End Select
    End If
End Sub

End Class
End Namespace

```

## 코드 컴파일

이 예제에는 다음 사항이 필요합니다.

- System, System.Drawing 및 System.Windows.Forms 어셈블리에 대한 참조

## 참고자료

- [Windows Forms 응용 프로그램의 키보드 입력](#)

# Windows Forms 애플리케이션의 마우스 입력

2020-02-03 • 2 minutes to read • [Edit Online](#)

Windows Forms에는 다양한 마우스 이벤트 및 사용자 지정된 마우스 커서, 마우스 캡처 및 끌어서 놓기 동작에 대한 추가 지원이 포함되어 있습니다.

## 섹션 내용

### [Windows Forms에서 마우스 입력이 작동하는 방식](#)

마우스 이벤트에 대한 정보 및 마우스에 대한 현재 정보와 시스템 설정을 가져오는 방법을 제공합니다.

### [Windows Forms의 마우스 이벤트](#)

마우스 이벤트가 발생하는 순서 및 특정 컨트롤 내에서 마우스 이벤트가 발생하는 방법에 대한 정보를 제공합니다.

### [방법: 클릭과 두 번 클릭 간 구별](#)

한 번 클릭과 두 번 클릭을 사용하여 호환되지 않는 작업을 시작하는 방법을 보여 줍니다.

### [Windows Forms의 마우스 포인터](#)

마우스 커서를 변경하는 방법을 설명합니다.

### [Windows Forms의 마우스 캡처](#)

컨트롤이 마우스를 캡처할 수 있는 방법을 설명합니다.

### [Windows Forms에서의 끌어서 놓기 기능](#)

끌어서 놓기 동작을 구현하는 방법을 설명합니다.

## 관련 섹션

### [마우스에 액세스](#)

Visual Basic을 사용하여 마우스에 액세스에 대한 항목을 나열합니다.

# Windows Forms에서 마우스 입력이 작동하는 방식

2020-02-03 • 14 minutes to read • [Edit Online](#)

마우스 입력 수신 및 처리는 모든 Windows 응용 프로그램에서 중요 한 부분입니다. 마우스 이벤트를 처리 하여 응용 프로그램에서 작업을 수행 하거나 마우스 위치 정보를 사용 하여 적중 테스트 또는 기타 작업을 수행할 수 있습니다. 또한 응용 프로그램의 컨트롤이 마우스 입력을 처리 하는 방식을 변경할 수 있습니다. 이 항목에서는 이러한 마우스 이벤트에 대해 자세히 설명 하고 마우스에 대 한 시스템 설정을 가져오고 변경 하는 방법을 설명 합니다. 마우스 이벤트와 함께 제공 되는 데이터 및 마우스 클릭 이벤트가 발생 하는 순서에 대 한 자세한 내용은 [Windows Forms의 마우스 이벤트를 참조 하세요.](#)

## 마우스 위치 및 적중 테스트

사용자가 마우스를 움직이면 운영 체제가 마우스 포인터를 이동 합니다. 마우스 포인터에는 운영 체제가 추적 하고 포인터의 위치로 인식 하는 핫 스팟 이라고 하는 단일 픽셀이 있습니다. 사용자가 마우스를 이동 하거나 마우스 단추를 누르면 [HotSpot](#)를 포함 하는 [Control](#)에서 적절 한 마우스 이벤트를 발생 시킵니다. 마우스 이벤트를 처리 할 때 또는 [Cursor](#) 클래스의 [Position](#) 속성을 사용 하여 [MouseEventArgs](#)의 [Location](#) 속성을 사용 하여 현재 마우스 위치를 가져올 수 있습니다. 이후에는 마우스 위치 정보를 사용 하여 적중 테스트를 수행한 다음 마우스 위치에 따라 동작을 수행할 수 있습니다. 적중 테스트 기능은 [ListView](#), [TreeView](#), [MonthCalendar](#) 및 [DataGridView](#) 컨트롤과 같은 Windows Forms의 여러 컨트롤에 기본 제공 됩니다. 적절한 마우스 [MouseHover](#) 이벤트와 함께 사용 됩니다. 예를 들어 적중 테스트는 응용 프로그램에서 특정 작업을 수행 해야 하는 시기를 결정 하는 데 매우 유용 합니다.

## 마우스 이벤트

마우스 입력에 응답 하는 기본적인 방법은 마우스 이벤트를 처리 하는 것입니다. 다음 표에서는 마우스 이벤트를 보여 주고 이벤트가 발생 하는 시기를 설명 합니다.

마우스 이벤트	DESCRIPTION
<a href="#">Click</a>	이 이벤트는 마우스 단추를 놓을 때, 일반적으로 <a href="#">MouseUp</a> 이벤트 이전에 발생 합니다. 이 이벤트의 처리기는 <a href="#">EventArgs</a> 형식의 인수를 받습니다. 클릭이 발생 한 경우를 결정 해야 하는 경우 이 이벤트를 처리 합니다.
<a href="#">MouseClicked</a>	이 이벤트는 사용자가 마우스로 컨트롤을 클릭할 때 발생 합니다. 이 이벤트의 처리기는 <a href="#">MouseEventArgs</a> 형식의 인수를 받습니다. 클릭이 발생할 때 마우스에 대 한 정보를 가져와야 하는 경우 이 이벤트를 처리 합니다.
<a href="#">DoubleClick</a>	이 이벤트는 컨트롤을 두 번 클릭할 때 발생 합니다. 이 이벤트의 처리기는 <a href="#">EventArgs</a> 형식의 인수를 받습니다. 두 번 클릭이 발생 한 경우를 결정 해야 하는 경우 이 이벤트를 처리 합니다.
<a href="#">MouseDoubleClick</a>	이 이벤트는 사용자가 마우스로 컨트롤을 두 번 클릭할 때 발생 합니다. 이 이벤트의 처리기는 <a href="#">MouseEventArgs</a> 형식의 인수를 받습니다. 두 번 클릭이 발생할 때 마우스에 대 한 정보를 가져와야 할 때 이 이벤트를 처리 합니다.

마우스 이벤트	DESCRIPTION
<a href="#">MouseDown</a>	이 이벤트는 마우스 포인터가 컨트롤 위에 있을 때 사용자가 마우스 단추를 누를 때 발생 합니다. 이 이벤트의 처리기는 <a href="#">MouseEventArgs</a> 형식의 인수를 받습니다.
<a href="#">MouseEnter</a>	이 이벤트는 컨트롤의 형식에 따라 마우스 포인터가 컨트롤의 테두리 또는 클라이언트 영역에 들어가면 발생 합니다. 이 이벤트의 처리기는 <a href="#">EventArgs</a> 형식의 인수를 받습니다.
<a href="#">MouseHover</a>	이 이벤트는 마우스 포인터를 컨트롤 위에 놓았을 때 발생 합니다. 이 이벤트의 처리기는 <a href="#">EventArgs</a> 형식의 인수를 받습니다.
<a href="#">MouseLeave</a>	이 이벤트는 컨트롤의 형식에 따라 마우스 포인터가 컨트롤의 테두리 또는 클라이언트 영역을 벗어날 때 발생 합니다. 이 이벤트의 처리기는 <a href="#">EventArgs</a> 형식의 인수를 받습니다.
<a href="#">MouseMove</a>	이 이벤트는 컨트롤 위에 있는 동안 마우스 포인터를 움직이면 발생 합니다. 이 이벤트의 처리기는 <a href="#">MouseEventArgs</a> 형식의 인수를 받습니다.
<a href="#">MouseUp</a>	이 이벤트는 마우스 포인터가 컨트롤 위에 있을 때 사용자가 마우스 단추를 놓을 때 발생 합니다. 이 이벤트의 처리기는 <a href="#">MouseEventArgs</a> 형식의 인수를 받습니다.
<a href="#">MouseWheel</a>	이 이벤트는 컨트롤에 포커스가 있는 동안 사용자가 마우스 휠을 돌릴 때 발생 합니다. 이 이벤트의 처리기는 <a href="#">MouseEventArgs</a> 형식의 인수를 받습니다. <a href="#">MouseEventArgs</a> 의 <a href="#">Delta</a> 속성을 사용 하여 마우스가 스크롤 된 정도를 확인할 수 있습니다.

## 마우스 입력 변경 및 시스템 설정 검색

컨트롤에서 파생 하고 [GetStyle](#) 및 [SetStyle](#) 메서드를 사용 하여 컨트롤에서 마우스 입력을 처리 하는 방법을 검색 하고 변경할 수 있습니다. [SetStyle](#) 메서드는 [ControlStyles](#) 값의 비트 조합을 사용 하여 컨트롤이 표준 클릭 또는 두 번 클릭 동작을 수행 하거나 컨트롤이 자체 마우스 처리를 처리할지 여부를 결정 합니다. 또한 [SystemInformation](#) 클래스에는 마우스의 기능을 설명 하고 마우스가 운영 체제와 상호 작용 하는 방식을 지정 하는 속성이 포함 되어 있습니다. 다음 표에서는 이러한 속성을 요약 합니다.

속성	DESCRIPTION
<a href="#">DoubleClickSize</a>	운영 체제에서 두 번 클릭을 두 번 클릭 하는 것으로 간주 하도록 사용자가 두 번 클릭 해야 하는 영역의 크기 (픽셀 단위)를 가져옵니다.
<a href="#">DoubleClickTime</a>	마우스 작업을 두 번 클릭 하는 것으로 간주 하기 위해 운영 체제에서 첫 번째 클릭과 두 번째 클릭 사이에 경과할 수 있는 최대 시간 (밀리초)을 가져옵니다.
<a href="#">MouseButtons</a>	마우스의 단추 수를 가져옵니다.
<a href="#">MouseButtonsSwapped</a>	마우스 왼쪽 단추와 오른쪽 단추의 기능이 바뀌었는지 여부를 나타내는 값을 가져옵니다.

속성	DESCRIPTION
<a href="#">MouseHoverSize</a>	마우스 호버 메시지가 생성되기 전에 마우스 포인터가 마우스 호버 시간 동안 머물러야 하는 사각형의 크기를 픽셀 단위로 가져옵니다.
<a href="#">MouseHoverTime</a>	마우스 호버 메시지가 생성되기 전에 마우스 포인터가 호버 사각형에 머물러야 하는 시간을 밀리초 단위로 가져옵니다.
<a href="#">MousePresent</a>	마우스가 설치 되어 있는지 여부를 나타내는 값을 가져옵니다.
<a href="#">MouseSpeed</a>	현재 마우스 속도를 나타내는 값 (1 ~ 20)을 가져옵니다.
<a href="#">MouseWheelPresent</a>	휠 마우스가 설치되어 있는지 여부를 나타내는 값을 가져옵니다.
<a href="#">MouseWheelScrollDelta</a>	단일 마우스 휠 회전의 증가값에 대한 델타 값 크기를 가져옵니다.
<a href="#">MouseWheelScrollLines</a>	마우스 휠을 돌릴 때 스크롤되는 줄 수를 가져옵니다.

## 참고 항목

- [Windows Forms 애플리케이션의 마우스 입력](#)
- [Windows Forms의 마우스 캡처](#)
- [Windows Forms의 마우스 포인터](#)

# Windows Forms의 마우스 이벤트

2020-02-03 • 13 minutes to read • [Edit Online](#)

마우스 입력을 처리하는 경우 일반적으로 마우스 포인터의 위치와 마우스 단추의 상태를 알아야 합니다. 이 항목에서는 마우스 이벤트에서 이 정보를 가져오는 방법을 자세히 설명하고 Windows Forms 컨트롤에서 마우스 클릭 이벤트가 발생하는 순서를 설명합니다. 모든 마우스 이벤트에 대한 한 목록 및 설명은 [Windows Forms에서 마우스 입력이 작동하는 방식](#)을 참조하세요. [이벤트 처리기 개요 \(Windows Forms\)](#) 및 [이벤트 개요 \(Windows Forms\)](#)도 참조하세요.

## 마우스 정보

[MouseEventArgs](#)는 마우스 단추 클릭 및 마우스 움직임 추적과 관련된 마우스 이벤트 처리기로 전송됩니다. [MouseEventArgs](#)는 클라이언트 좌표에서 마우스 포인터의 위치, 누른 마우스 단추, 마우스 휠의 스크롤 여부를 포함하여 마우스의 현재 상태에 대한 정보를 제공합니다. 마우스 포인터가 컨트롤의 범위로 들어오거나 나갈 때 단 순히 알리는 이벤트와 같은 여러 마우스 이벤트가 추가 정보 없이 [EventArgs](#)를 이벤트 처리기로 전송합니다.

마우스 이벤트를 처리하지 않고 마우스 단추의 현재 상태나 마우스 포인터의 위치를 확인하려는 경우 [MouseButtons](#) 클래스의 [MousePosition](#) 및 [Control](#) 속성을 사용할 수도 있습니다. [MouseButtons](#)는 현재 누른 마우스 단추에 대한 정보를 반환합니다. [MousePosition](#)은 마우스 포인터의 화면 좌표를 반환하며 [Position](#)에서 반환되는 값과 같습니다.

## 화면 좌표와 클라이언트 좌표 간의 변환

일부 마우스 위치 정보는 클라이언트 좌표로 표시되고 일부 정보는 화면 좌표로 표시되므로 좌표계 간에 지점을 변환해야 할 수도 있습니다. [PointToClient](#) 클래스에서 사용할 수 있는 [PointToScreen](#) 및 [Control](#) 메서드를 사용하면 이 작업을 쉽게 수행할 수 있습니다.

## 표준 클릭 이벤트 동작

마우스 클릭 이벤트를 적절한 순서로 처리하려는 경우 Windows Forms 컨트롤에서 클릭 이벤트가 발생하는 순서를 알아야 합니다. 개별 컨트롤에 대한 다음 목록에 명시된 경우를 제외하고 모든 Windows Forms 컨트롤은 어떤 마우스 단추인지에 관계없이 마우스 단추를 눌렀다 놓는 순서대로 클릭 이벤트를 발생시킵니다. 다음 목록에서는 마우스 단추 한 번 클릭에 대해 발생하는 이벤트 순서를 보여 줍니다.

1. [MouseDown](#) 이벤트
2. [Click](#) 이벤트
3. [MouseClick](#) 이벤트
4. [MouseUp](#) 이벤트

다음은 마우스 단추를 두 번 클릭할 때 발생 하는 이벤트의 순서입니다.

1. [MouseDown](#) 이벤트
2. [Click](#) 이벤트
3. [MouseClick](#) 이벤트
4. [MouseUp](#) 이벤트
5. [MouseDown](#) 이벤트



6. [DoubleClick](#) 이벤트 해당 컨트롤에 대한 [StandardDoubleClick](#) 스타일 비트가 `true`로 설정되었는지 여부에 따라 달라질 수 있습니다. [ControlStyles](#)를 설정하는 방법에 대한 자세한 내용은 [SetStyle](#) 메서드를 참조하세요.

7. [MouseDown](#) 이벤트

8. [MouseUp](#) 이벤트

마우스 클릭 이벤트의 순서를 보여 주는 코드 예제는 [방법: Windows Forms 컨트롤에서 사용자 입력 이벤트 처리](#)를 참조하세요.

개별 컨트롤

다음 컨트롤은 표준 마우스 클릭 이벤트 동작을 준수하지 않습니다.

- [Button](#)
- [CheckBox](#)
- [ComboBox](#)
- [RadioButton](#)

#### NOTE

[ComboBox](#) 컨트롤의 경우 사용자가 편집 필드, 단추 또는 목록 내의 항목을 클릭하면 나중에 자세히 설명하는 이벤트 동작이 발생합니다.

- 마우스 왼쪽 단추 클릭: [Click](#), [MouseDown](#)
  - 마우스 오른쪽 단추 클릭: 클릭 이벤트가 발생하지 않음
  - 마우스 왼쪽 단추 두 번 클릭: [Click](#), [MouseDown](#), [Click](#), [MouseDown](#)
  - 마우스 오른쪽 단추 두 번 클릭: 클릭 이벤트가 발생하지 않음
- [TextBox](#), [RichTextBox](#), [ListBox](#), [MaskedTextBox](#) 및 [CheckedListBox](#) 컨트롤

#### NOTE

사용자가 이러한 컨트롤 내의 아무 곳이나 클릭하면 나중에 자세히 설명하는 이벤트 동작이 발생합니다.

- 마우스 왼쪽 단추 클릭: [Click](#), [MouseDown](#)
  - 마우스 오른쪽 단추 클릭: 클릭 이벤트가 발생하지 않음
  - 마우스 왼쪽 단추 두 번 클릭: [Click](#), [MouseDown](#), [DoubleClick](#), [MouseDown](#)
  - 마우스 오른쪽 단추 두 번 클릭: 클릭 이벤트가 발생하지 않음
- [ListView](#) 컨트롤

#### NOTE

사용자가 [ListView](#) 컨트롤의 항목을 클릭하는 경우에만 나중에 자세히 설명하는 이벤트 동작이 발생합니다. 컨트롤의 다른 곳을 클릭하면 이벤트가 발생하지 않습니다. 나중에 설명하는 이벤트 외에도 [BeforeLabelEdit](#) 컨트롤에 유효성 검사를 사용하려는 경우 중요할 수 있는 [AfterLabelEdit](#) 및 [ListView](#) 이벤트가 있습니다.

- 마우스 왼쪽 단추 클릭: [Click](#), [MouseDown](#)

- 마우스 오른쪽 단추 클릭: [Click](#), [MouseDown](#)
- 마우스 왼쪽 단추 두 번 클릭: [Click](#), [MouseDown](#), [DoubleClick](#), [MouseDownDoubleClick](#)
- 마우스 오른쪽 단추 두 번 클릭: [Click](#), [MouseDown](#), [DoubleClick](#), [MouseDownDoubleClick](#)

- [TreeView](#) 컨트롤

#### NOTE

사용자가 [TreeView](#) 컨트롤에서 항목 자체나 항목 오른쪽을 클릭하는 경우에만 나중에 자세히 설명하는 이벤트 동작이 발생합니다. 컨트롤의 다른 곳을 클릭하면 이벤트가 발생하지 않습니다. 나중에 설명하는 이벤트 외에도 [BeforeCheck](#) 컨트롤에 유효성 검사를 사용하려는 경우 중요할 수 있는 [BeforeSelect](#), [BeforeLabelEdit](#), [AfterSelect](#), [AfterCheck](#), [AfterLabelEdit](#) 및 [TreeView](#) 이벤트가 있습니다.

- 마우스 왼쪽 단추 클릭: [Click](#), [MouseDown](#)
- 마우스 오른쪽 단추 클릭: [Click](#), [MouseDown](#)
- 마우스 왼쪽 단추 두 번 클릭: [Click](#), [MouseDown](#), [DoubleClick](#), [MouseDownDoubleClick](#)
- 마우스 오른쪽 단추 두 번 클릭: [Click](#), [MouseDown](#), [DoubleClick](#), [MouseDownDoubleClick](#)

#### 토글 컨트롤의 그리기 동작

[ButtonBase](#) 클래스에서 파생되는 컨트롤과 같은 토글 컨트롤은 마우스 클릭 이벤트와 결합되어 다음과 같은 고유한 그리기 동작을 제공합니다.

1. 사용자가 마우스 단추를 누릅니다.
2. 컨트롤이 눌린 상태로 그려집니다.
3. [MouseDown](#) 이벤트가 발생합니다.
4. 사용자가 마우스 단추를 놓습니다.
5. 컨트롤이 올려진 상태로 그려집니다.
6. [Click](#) 이벤트가 발생합니다.
7. [MouseDown](#) 이벤트가 발생합니다.
8. [MouseUp](#) 이벤트가 발생합니다.

#### NOTE

사용자가 마우스 단추를 누른 동안 토글 컨트롤에서 포인터를 이동하는 경우(예: 누른 동안 [Button](#) 컨트롤에서 마우스 이동) 토글 컨트롤이 올려진 상태로 그려지고 [MouseUp](#) 이벤트만 발생합니다. 이런 상황에서는 [Click](#) 또는 [MouseDown](#) 이벤트가 발생하지 않습니다.

## 참고 항목

- [Windows Forms 애플리케이션의 마우스 입력](#)

# 방법: 클릭과 두 번 클릭 간 구별

2019-10-23 • 12 minutes to read • [Edit Online](#)

일반적으로 단일 *클릭*은 UI(사용자 인터페이스) 동작을 시작하고 *두 번 클릭*은 동작을 확장합니다 예를 들어 한 번 클릭은 대개 항목을 선택하고 두 번 클릭은 선택된 항목을 편집합니다. 그러나 Windows Forms 클릭 이벤트는 한 번 클릭과 두 번 클릭이 호환되지 않는 동작을 수행하는 시나리오에는 쉽게 적용되지 않습니다. [Click](#) 또는 [MouseClick](#) 이벤트에 연결된 동작이 [DoubleClick](#) 또는 [MouseDoubleClick](#) 이벤트에 연결된 동작 앞에 수행되기 때문입니다. 이 항목에서는 이 문제에 대한 두 가지 솔루션을 보여 줍니다. 한 솔루션은 두 번 클릭 이벤트를 처리하고 클릭 이벤트 처리 시 작업을 롤백하는 것입니다. 드물지만 [MouseDown](#) 이벤트를 처리하고 [SystemInformation](#) 클래스의 [DoubleClickTime](#) 및 [DoubleClickSize](#) 속성을 사용하여 클릭 및 두 번 클릭 동작을 시뮬레이션해야 할 수 있습니다. 클릭 사이의 시간을 측정하고, [DoubleClickTime](#) 값에 도달하기 전에 두 번째 클릭이 발생하고 클릭이 [DoubleClickSize](#)에서 정의된 사각형 내에 있으면 두 번 클릭 동작을 수행하고, 그렇지 않으면 클릭 동작을 수행합니다.

클릭 동작을 롤백하려면 다음을 수행합니다.

- 작업하고 있는 컨트롤에 표준 두 번 클릭 동작이 있는지 확인합니다. 그렇지 않으면 [SetStyle](#) 메서드로 컨트롤을 사용하도록 설정합니다. 두 번 클릭 이벤트를 처리하고 두 번 클릭 동작과 클릭 동작을 롤백합니다. 다음 코드 예제에서는 두 번 클릭이 사용되는 사용자 지정 단추를 만드는 방법과 두 번 클릭 이벤트 처리 코드에서 클릭 동작을 롤백하는 방법을 보여 줍니다.

```

using System;
using System.ComponentModel;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace MouseRollBackSingleClick
{
    public class Form1 : Form
    {
        private DoubleClickButton button1;
        private FormBorderStyle initialStyle;

        public Form1()
        {
            initialStyle = this.FormBorderStyle;
            this.ClientSize = new System.Drawing.Size(292, 266);
            button1 = new DoubleClickButton();
            button1.Location = new Point (40,40);
            button1.Click += new EventHandler(button1_Click);
            button1.AutoSize = true;
            this.AllowDrop = true;
            button1.Text = "Click or Double Click";
            button1.DoubleClick += new EventHandler(button1_DoubleClick);
            this.Controls.Add(button1);
        }

        // Handle the double click event.
        void button1_DoubleClick(object sender, EventArgs e)
        {
            // Change the border style back to the initial style.
            this.FormBorderStyle = initialStyle;
            MessageBox.Show("Rolled back single click change.");
        }

        // Handle the click event.
        void button1_Click(object sender, EventArgs e)
        {
            this.FormBorderStyle = FormBorderStyle.FixedToolWindow;
        }

        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.Run(new Form1());
        }
    }

    public class DoubleClickButton : Button
    {
        public DoubleClickButton() : base()
        {
            // Set the style so a double click event occurs.
            SetStyle(ControlStyles.StandardClick |
                ControlStyles.StandardDoubleClick, true);
        }
    }
}

```

```

Imports System.ComponentModel
Imports System.Drawing
Imports System.Text
Imports System.Windows.Forms

Public Class Form1
    Inherits Form
    Private WithEvents button1 As DoubleClickButton
    Private initialStyle As FormBorderStyle

    Public Sub New()
        Me.SuspendLayout()
        initialStyle = Me.FormBorderStyle
        Me.ClientSize = New System.Drawing.Size(292, 266)
        button1 = New DoubleClickButton()
        button1.Location = New Point(40, 40)
        button1.AutoSize = True
        button1.Text = "Click or Double Click"
        Me.Controls.Add(button1)
        Me.Name = "Form1"
        Me.ResumeLayout(False)
        Me.PerformLayout()

    End Sub

    ' Handle the double click event.
    Private Sub button1_DoubleClick(ByVal sender As Object, ByVal e As EventArgs) _
        Handles button1.DoubleClick

        ' Change the border style back to the initial style.
        Me.FormBorderStyle = initialStyle
        MessageBox.Show("Rolled back single click change.")

    End Sub

    ' Handle the click event.
    Private Sub button1_Click(ByVal sender As Object, ByVal e As EventArgs) _
        Handles button1.Click

        Me.FormBorderStyle = FormBorderStyle.FixedToolWindow

    End Sub

    <SThread()> _
    Shared Sub Main()
        Application.EnableVisualStyles()
        Application.Run(New Form1())

    End Sub
End Class

Public Class DoubleClickButton
    Inherits Button

    Public Sub New()
        ' Set the style so a double click event occurs.
        SetStyle(ControlStyles.StandardClick Or ControlStyles.StandardDoubleClick, True)

    End Sub
End Class

```

**MouseDown** 이벤트에서 클릭을 구분하려면 다음을 수행합니다.

- **MouseDown** 이벤트를 처리하고 적절한 **SystemInformation** 속성 및 **Timer** 구성 요소를 사용하여 클릭 사이의 위치 및 시간 차이를 결정합니다. 클릭 또는 두 번 클릭이 수행되는지에 따라 적절한 동작을 수행합니다. 다음 코드 예제에서는 이 작업을 수행하는 방법을 보여 줍니다.

```
#using <System.Drawing.dll>
#using <System.Windows.Forms.dll>
#using <System.dll>

using namespace System;
using namespace System::Drawing;
using namespace System::Windows::Forms;

namespace SingleVersusDoubleClick
{
    public ref class Form1 : public Form
    {
    private:
        Rectangle hitTestRectangle;
    private:
        Rectangle doubleClickRectangle;
    private:
        TextBox^ outputBox;
    private:
        Timer^ doubleClickTimer;
    private:
        ProgressBar^ doubleClickBar;
    private:
        Label^ hitTestLabel;
    private:
        Label^ timerLabel;
    private:
        bool isFirstClick;
    private:
        bool isDoubleClick;
    private:
        int milliseconds;

    public:
        Form1()
        {
            hitTestRectangle = Rectangle();
            hitTestRectangle.Location = Point(30, 20);
            hitTestRectangle.Size = System::Drawing::Size(100, 40);

            doubleClickRectangle = Rectangle();

            outputBox = gcnew TextBox();
            outputBox->Location = Point(30, 120);
            outputBox->Size = System::Drawing::Size(200, 100);
            outputBox->AutoSize = false;
            outputBox->Multiline = true;

            doubleClickTimer = gcnew Timer();
            doubleClickTimer->Interval = 100;
            doubleClickTimer->Tick +=
                gcnew EventHandler(this, &Form1::doubleClickTimer_Tick);

            doubleClickBar = gcnew ProgressBar();
            doubleClickBar->Location = Point(30, 85);
            doubleClickBar->Minimum = 0;
            doubleClickBar->Maximum = SystemInformation::DoubleClickTime;

            hitTestLabel = gcnew Label();
            hitTestLabel->Location = Point(30, 5);
            hitTestLabel->Size = System::Drawing::Size(100, 15);
            hitTestLabel->Text = "Hit test rectangle:";
```

```

        timerLabel = gcnew Label();
        timerLabel->Location = Point(30, 70);
        timerLabel->Size = System::Drawing::Size(100, 15);
        timerLabel->Text = "Double click timer:";

        isFirstClick = true;

        this->Paint += gcnew PaintEventHandler(this, &Form1::Form1_Paint);
        this->MouseDown +=
            gcnew MouseEventHandler(this, &Form1::Form1_MouseDown);
        this->Controls->
            AddRange(gcnew array<Control^> { doubleClickBar, outputBox,
                hitTestLabel, timerLabel });
    }

    // Detect a valid single click or double click.
private:
    void Form1_MouseDown(Object^ sender, MouseEventArgs^ e)
    {
        // Verify that the mouse click is in the main hit
        // test rectangle.
        if (!hitTestRectangle.Contains(e->Location))
        {
            return;
        }

        // This is the first mouse click.
        if (isFirstClick)
        {
            isFirstClick = false;

            // Determine the location and size of the double click
            // rectangle area to draw around the cursor point.
            doubleClickRectangle = Rectangle(
                e->X - (SystemInformation::DoubleClickSize.Width / 2),
                e->Y - (SystemInformation::DoubleClickSize.Height / 2),
                SystemInformation::DoubleClickSize.Width,
                SystemInformation::DoubleClickSize.Height);
            Invalidate();

            // Start the double click timer.
            doubleClickTimer->Start();
        }

        // This is the second mouse click.
        else
        {
            // Verify that the mouse click is within the double click
            // rectangle and is within the system-defined double
            // click period.
            if (doubleClickRectangle.Contains(e->Location) &&
                milliseconds < SystemInformation::DoubleClickTime)
            {
                isDoubleClick = true;
            }
        }
    }

private:
    void doubleClickTimer_Tick(Object^ sender, EventArgs^ e)
    {
        milliseconds += 100;
        doubleClickBar->Increment(100);

        // The timer has reached the double click time limit.
        if (milliseconds >= SystemInformation::DoubleClickTime)
        {
            doubleClickTimer->Stop();
        }
    }
}

```

```

        doubleClickTimer->Stop();

        if (isDoubleClick)
        {
            outputBox->AppendText("Perform double click action");
            outputBox->AppendText(Environment::NewLine);
        }
        else
        {
            outputBox->AppendText("Perform single click action");
            outputBox->AppendText(Environment::NewLine);
        }

        // Allow the MouseDown event handler to process clicks again.
        isFirstClick = true;
        isDoubleClick = false;
        milliseconds = 0;
        doubleClickBar->Value = 0;
    }
}

// Paint the hit test and double click rectangles.
private:
void Form1_Paint(Object^ sender, PaintEventArgs^ e)
{
    // Draw the border of the main hit test rectangle.
    e->Graphics->DrawRectangle(Pens::Black, hitTestRectangle);

    // Fill in the double click rectangle.
    e->Graphics->FillRectangle(Brushes::Blue, doubleClickRectangle);
}
};
}

[STAThread]
int main()
{
    Application::EnableVisualStyles();
    Application::Run(gcnew SingleVersusDoubleClick::Form1);
}

```

```

using System;
using System.Drawing;
using System.Windows.Forms;

namespace SingleVersusDoubleClick
{
    class Form1 : Form
    {
        private Rectangle hitTestRectangle = new Rectangle();
        private Rectangle doubleClickRectangle = new Rectangle();
        private TextBox textBox1 = new TextBox();
        private Timer doubleClickTimer = new Timer();
        private ProgressBar doubleClickBar = new ProgressBar();
        private Label label1 = new Label();
        private Label label2 = new Label();
        private bool isFirstClick = true;
        private bool isDoubleClick = false;
        private int milliseconds = 0;

        [STAThread]
        public static void Main()
        {
            Application.EnableVisualStyles();
            Application.Run(new Form1());
        }
    }
}

```



```

public Form1()
{
    label1.Location = new Point(30, 5);
    label1.Size = new Size(100, 15);
    label1.Text = "Hit test rectangle:";

    label2.Location = new Point(30, 70);
    label2.Size = new Size(100, 15);
    label2.Text = "Double click timer:";

    hitTestRectangle.Location = new Point(30, 20);
    hitTestRectangle.Size = new Size(100, 40);

    doubleClickTimer.Interval = 100;
    doubleClickTimer.Tick +=
        new EventHandler(doubleClickTimer_Tick);

    doubleClickBar.Location = new Point(30, 85);
    doubleClickBar.Minimum = 0;
    doubleClickBar.Maximum = SystemInformation.DoubleClickTime;

    textBox1.Location = new Point(30, 120);
    textBox1.Size = new Size(200, 100);
    textBox1.AutoSize = false;
    textBox1.Multiline = true;

    this.Paint += new PaintEventHandler(Form1_Paint);
    this.MouseDown += new MouseEventHandler(Form1_MouseDown);
    this.Controls.AddRange(new Control[] { doubleClickBar, textBox1,
        label1, label2 });
}

// Detect a valid single click or double click.
void Form1_MouseDown(object sender, MouseEventArgs e)
{
    // Verify that the mouse click is in the main hit
    // test rectangle.
    if (!hitTestRectangle.Contains(e.Location))
    {
        return;
    }

    // This is the first mouse click.
    if (isFirstClick)
    {
        isFirstClick = false;

        // Determine the location and size of the double click
        // rectangle area to draw around the cursor point.
        doubleClickRectangle = new Rectangle(
            e.X - (SystemInformation.DoubleClickSize.Width / 2),
            e.Y - (SystemInformation.DoubleClickSize.Height / 2),
            SystemInformation.DoubleClickSize.Width,
            SystemInformation.DoubleClickSize.Height);
        Invalidate();

        // Start the double click timer.
        doubleClickTimer.Start();
    }

    // This is the second mouse click.
    else
    {
        // Verify that the mouse click is within the double click
        // rectangle and is within the system-defined double
        // click period.
        if (doubleClickRectangle.Contains(e.Location) &&
            milliseconds < SystemInformation.DoubleClickTime)
        ,

```

```

        {
            isDoubleClick = true;
        }
    }
}

void doubleClickTimer_Tick(object sender, EventArgs e)
{
    milliseconds += 100;
    doubleClickBar.Increment(100);

    // The timer has reached the double click time limit.
    if (milliseconds >= SystemInformation.DoubleClickTime)
    {
        doubleClickTimer.Stop();

        if (isDoubleClick)
        {
            textBox1.AppendText("Perform double click action");
            textBox1.AppendText(Environment.NewLine);
        }
        else
        {
            textBox1.AppendText("Perform single click action");
            textBox1.AppendText(Environment.NewLine);
        }

        // Allow the MouseDown event handler to process clicks again.
        isFirstClick = true;
        isDoubleClick = false;
        milliseconds = 0;
        doubleClickBar.Value = 0;
    }
}

// Paint the hit test and double click rectangles.
void Form1_Paint(object sender, PaintEventArgs e)
{
    // Draw the border of the main hit test rectangle.
    e.Graphics.DrawRectangle(Pens.Black, hitTestRectangle);

    // Fill in the double click rectangle.
    e.Graphics.FillRectangle(Brushes.Blue, doubleClickRectangle);
}
}
}

```

```

Imports System.Drawing
Imports System.Windows.Forms

Namespace SingleVersusDoubleClick

    Class Form1
        Inherits Form
        Private hitTestRectangle As New Rectangle()
        Private doubleClickRectangle As New Rectangle()
        Private textBox1 As New TextBox()
        Private WithEvents doubleClickTimer As New Timer()
        Private doubleClickBar As New ProgressBar()
        Private label1 As New Label()
        Private label2 As New Label()
        Private isFirstClick As Boolean = True
        Private isDoubleClick As Boolean = False
        Private milliseconds As Integer = 0

        <STAThread(> _
        Public Shared Sub Main()
            Application.EnableVisualStyles()

```

```

Application.EnableVisualStyles()
Application.Run(New Form1())
End Sub

Public Sub New()
    label1.Location = New Point(30, 5)
    label1.Size = New Size(100, 15)
    label1.Text = "Hit test rectangle:"

    label2.Location = New Point(30, 70)
    label2.Size = New Size(100, 15)
    label2.Text = "Double click timer:"

    hitTestRectangle.Location = New Point(30, 20)
    hitTestRectangle.Size = New Size(100, 40)
    doubleClickTimer.Interval = 100

    doubleClickBar.Location = New Point(30, 85)
    doubleClickBar.Minimum = 0
    doubleClickBar.Maximum = SystemInformation.DoubleClickTime

    textBox1.Location = New Point(30, 120)
    textBox1.Size = New Size(200, 100)
    textBox1.AutoSize = False
    textBox1.Multiline = True

    Me.Controls.Add(doubleClickBar)
    Me.Controls.Add(textBox1)
    Me.Controls.Add(label1)
    Me.Controls.Add(label2)
End Sub

' Detect a valid single click or double click.
Sub Form1_MouseDown(ByVal sender As Object, _
    ByVal e As MouseEventArgs) Handles Me.MouseDown

    ' Verify that the mouse click is in the main hit
    ' test rectangle.
    If Not hitTestRectangle.Contains(e.Location) Then
        Return
    End If

    ' This is the first mouse click.
    If isFirstClick = True Then
        isFirstClick = False

        ' Determine the location and size of the double click
        ' rectangle to draw around the cursor point.
        doubleClickRectangle = New Rectangle( _
            e.X - (SystemInformation.DoubleClickSize.Width / 2), _
            e.Y - (SystemInformation.DoubleClickSize.Height / 2), _
            SystemInformation.DoubleClickSize.Width, _
            SystemInformation.DoubleClickSize.Height)
        Invalidate()

        ' Start the double click timer.
        doubleClickTimer.Start()

    ' This is the second mouse click.
    Else
        ' Verify that the mouse click is within the double click
        ' rectangle and is within the system-defined double
        ' click period.
        If doubleClickRectangle.Contains(e.Location) And _
            milliseconds < SystemInformation.DoubleClickTime Then
            isDoubleClick = True
        End If
    End If
End Sub

```

```

Sub doubleClickTimer_Tick(ByVal sender As Object, _
    ByVal e As EventArgs) Handles doubleClickTimer.Tick

    milliseconds += 100
    doubleClickBar.Increment(100)

    ' The timer has reached the double click time limit.
    If milliseconds >= SystemInformation.DoubleClickTime Then
        doubleClickTimer.Stop()

        If isDoubleClick Then
            textBox1.AppendText("Perform double click action")
            textBox1.AppendText(Environment.NewLine)
        Else
            textBox1.AppendText("Perform single click action")
            textBox1.AppendText(Environment.NewLine)
        End If

        ' Allow the MouseDown event handler to process clicks again.
        isFirstClick = True
        isDoubleClick = False
        milliseconds = 0
        doubleClickBar.Value = 0
    End If
End Sub

' Paint the hit test and double click rectangles.
Sub Form1_Paint(ByVal sender As Object, _
    ByVal e As PaintEventArgs) Handles Me.Paint

    ' Draw the border of the main hit test rectangle.
    e.Graphics.DrawRectangle(Pens.Black, hitTestRectangle)

    ' Fill in the double click rectangle.
    e.Graphics.FillRectangle(Brushes.Blue, doubleClickRectangle)
End Sub
End Class
End Namespace

```

## 코드 컴파일

이러한 예제에는 다음이 필요합니다.

- System, System.Drawing 및 System.Windows.Forms 어셈블리에 대한 참조

## 참고자료

- [Windows Forms 애플리케이션의 마우스 입력](#)

# Windows Forms의 마우스 포인터

2020-02-03 • 5 minutes to read • [Edit Online](#)

커서 라고도 하는 마우스 *포인터*는 마우스를 사용 하여 사용자 입력에 대 한 화면에 포커스 지점을 지정 하는 비트 맵입니다. 이 항목에서는 Windows Forms의 마우스 포인터에 대 한 개요를 제공 하고 마우스 포인터를 수정 하고 제어 하는 몇 가지 방법에 대해 설명 합니다.

## 마우스 포인터에 액세스

마우스 포인터는 [Cursor](#) 클래스로 표시 되 고 각 [Control](#)에는 해당 컨트롤에 대 한 포인터를 지정 하는 [Control.Cursor](#) 속성이 있습니다. [Cursor](#) 클래스에는 포인터를 설명 하는 속성 (예: [Position](#) 및 [HotSpot](#) 속성)과 포 인터의 모양 (예: [Show](#), [Hide](#) 및 [DrawStretched](#) 메서드)을 수정할 수 있는 메서드가 포함 되어 있습니다.

## 마우스 포인터 제어

마우스 포인터를 사용할 수 있는 영역을 제한 하거나 마우스 위치를 변경 하는 경우가 있습니다. [Cursor](#)의 [Position](#) 속성을 사용 하여 마우스의 현재 위치를 가져오거나 설정할 수 있습니다. 또한 마우스 포인터를 사용 하여 [Clip](#) 속 성을 설정할 수 있는 영역을 제한할 수 있습니다. 기본적으로 클립 영역은 전체 화면입니다.

## 마우스 포인터 변경

마우스 포인터를 변경 하는 것은 사용자에게 피드백을 제공 하는 중요 한 방법입니다. 예를 들어 [MouseEnter](#) 처리 기에서 마우스 포인터를 수정 하고 [MouseLeave](#) 이벤트를 사용 하여 사용자에게 계산을 알리고 컨트롤에서 사용 자 상호 작용을 제한할 수 있습니다. 경우에 따라 응용 프로그램이 끌어서 놓기 작업에 관여 하는 경우와 같은 시스 템 이벤트로 인해 마우스 포인터가 변경 됩니다.

마우스 포인터를 변경 하는 기본 방법은 컨트롤의 [Control.Cursor](#) 또는 [DefaultCursor](#) 속성을 새 [Cursor](#) 설정 하는 것입니다. 마우스 포인터를 변경 하는 예제는 [Cursor](#) 클래스의 코드 예제를 참조 하세요. 또한 [Cursors](#) 클래스는 손 모양의 포인터와 같이 다양 한 형식의 포인터에 대 한 [Cursor](#) 개체 집합을 노출 합니다. 마우스 포인터가 컨트롤 에 있을 때마다 모래 시계와 유사한 대기 포인터를 표시 하려면 [Control](#) 클래스의 [UseWaitCursor](#) 속성을 사용 합니 다.

## 참고 항목

- [Cursor](#)
- [Windows Forms 애플리케이션의 마우스 입력](#)
- [Windows Forms에서의 끌어서 놓기 기능](#)

# Windows Forms의 마우스 캡처

2020-02-03 • 2 minutes to read • [Edit Online](#)

*마우스 캡처*는 컨트롤이 모든 마우스 입력의 명령을 사용 하는 경우를 참조 합니다. 컨트롤은 마우스를 캡처 했을 때 포인터가 테두리 내에 있는지 여부에 관계 없이 마우스 입력을 수신 합니다.

## 마우스 캡처 설정

Windows Forms 사용자가 컨트롤에서 마우스 단추를 누를 때 컨트롤이 캡처되고 마우스 단추를 놓을 때 컨트롤이 컨트롤에 의해 해제 됩니다.

[Control](#) 클래스의 [Capture](#) 속성은 컨트롤이 마우스를 캡처 했는지 여부를 지정 합니다. 컨트롤이 마우스 캡처를 손실 하는 경우를 확인 하려면 [MouseCaptureChanged](#) 이벤트를 처리 합니다.

전경 창이 마우스를 캡처할 수 있습니다. 배경 창이 마우스를 캡처하도록 시도 하면 창에서 마우스 포인터가 창의 표시 부분 안에 있을 때 발생 하는 마우스 이벤트에 대 한 메시지만 수신 합니다. 또한 전경 창이 마우스를 캡처한 경우에 여전히 클릭할 수 다른 창 전경으로 상태로 전환 합니다. 마우스를 캡처하면 바로 가기 키가 작동 하지 않습니다.

## 참고 항목

- [Windows Forms 애플리케이션의 마우스 입력](#)

# Windows Forms에서의 끌어서 놓기 기능

2020-02-03 • 6 minutes to read • [Edit Online](#)

Windows Forms에는 끌어서 놓기 동작을 구현하는 메서드, 이벤트 및 클래스 집합이 포함되어 있습니다. 이 항목에서는 Windows Forms의 끌어서 놓기 지원에 대해 개괄적으로 설명합니다. 또한 [끌어서 놓기 작업 및 클립보드 지원](#)을 참조하세요.

## 끌어서 놓기 작업 수행

끌어서 놓기 작업을 수행하려면 [DoDragDrop](#) 클래스의 [Control](#) 메서드를 사용합니다. 끌어서 놓기 작업을 수행하는 방법에 대한 자세한 내용은 [DoDragDrop](#)을 참조하세요. 끌어서 놓기 작업이 시작되기 전에 마우스 포인터를 위로 끌어와야 하는 사각형을 가져오려면 [DragSize](#) 클래스의 [SystemInformation](#) 속성을 사용합니다.

## 끌어서 놓기 작업과 관련된 이벤트

끌어서 놓기 작업에는 두 가지 범주의 이벤트가 있습니다. 하나는 끌어서 놓기 작업의 현재 대상에서 발생하는 이벤트이고, 다른 하나는 끌어서 놓기 작업의 소스에서 발생하는 이벤트입니다.

### 현재 대상의 이벤트

다음 표에서는 끌어서 놓기 작업의 현재 대상에서 발생하는 이벤트를 보여 줍니다.

마우스 이벤트	DESCRIPTION
<a href="#">DragEnter</a>	이 이벤트는 개체를 컨트롤의 범위로 끌어올 때 발생합니다. 이 이벤트의 처리기는 <a href="#">DragEventArgs</a> 형식의 인수를 받습니다.
<a href="#">DragOver</a>	이 이벤트는 마우스 포인터가 컨트롤의 범위 내에 있는 동안 개체를 끌 때 발생합니다. 이 이벤트의 처리기는 <a href="#">DragEventArgs</a> 형식의 인수를 받습니다.
<a href="#">DragDrop</a>	이 이벤트는 끌어서 놓기 작업이 완료될 때 발생합니다. 이 이벤트의 처리기는 <a href="#">DragEventArgs</a> 형식의 인수를 받습니다.
<a href="#">DragLeave</a>	이 이벤트는 컨트롤의 범위 밖으로 개체를 끌 때 발생합니다. 이 이벤트의 처리기는 <a href="#">EventArgs</a> 형식의 인수를 받습니다.

[DragEventArgs](#) 클래스는 마우스 포인터의 위치, 마우스 단추의 현재 상태 및 키보드의 한정자 키, 끄는 데이터, 끌기 이벤트의 소스에서 허용되는 작업과 작업의 대상 놓기 효과를 지정하는 [DragDropEffects](#) 값을 제공합니다.

### 소스의 이벤트

다음 표에서는 끌어서 놓기 작업의 소스에서 발생하는 이벤트를 보여 줍니다.

마우스 이벤트	DESCRIPTION
<a href="#">GiveFeedback</a>	이 이벤트는 끌기 작업 중에 발생합니다. 마우스 포인터 변경 등 끌어서 드롭 작업이 발생하고 있음을 알리는 시각 신호를 사용자에게 제공할 수 있습니다. 이 이벤트의 처리기는 <a href="#">GiveFeedbackEventArgs</a> 형식의 인수를 받습니다.

마우스 이벤트	DESCRIPTION
<a href="#">QueryContinueDrag</a>	이 이벤트는 끌어서 놓기 작업 중에 발생하며 끌기 소스가 끌어서 놓기 작업을 취소해야 할지를 결정하도록 합니다. 이 이벤트의 처리기는 <a href="#">QueryContinueDragEventArgs</a> 형식의 인수를 받습니다.

[QueryContinueDragEventArgs](#) 클래스는 마우스 단추의 현재 상태 및 키보드의 한정자 키, Esc 키를 눌렀는지 여부를 지정하는 값, 끌어서 놓기 작업을 계속할지 여부를 지정하기 위해 설정할 수 있는 [DragAction](#) 값을 제공합니다.

## 참고 항목

- [Windows Forms 애플리케이션의 마우스 입력](#)



# 방법: 코드에서 마우스 및 키보드 이벤트 시뮬레이션

2019-12-10 • 16 minutes to read • [Edit Online](#)

Windows Forms는 프로그래밍 방식으로 마우스 및 키보드 입력을 시뮬레이션하기 위한 여러 가지 옵션을 제공합니다. 이 항목에서는 이러한 옵션에 대해 간략하게 설명합니다.

## 마우스 입력 시뮬레이션

마우스 이벤트를 시뮬레이션하는 가장 좋은 방법은 시뮬레이션하려는 마우스 이벤트를 발생시키는 `On` *EventName* 메서드를 호출하는 것입니다. 이벤트를 발생시키는 메서드는 보호되며 컨트롤이나 폼 외부에서 액세스할 수 없기 때문에 이 옵션은 일반적으로 사용자 지정 컨트롤 및 폼 내에서만 가능합니다. 예를 들어 다음 단계에서 코드에서 마우스 오른쪽 단추를 클릭하여 시뮬레이션하는 방법을 보여 줍니다.

프로그래밍 방식으로 마우스 오른쪽 단추를 클릭하려면

1. `MouseEventArgs` 속성이 `Button` 값으로 설정된 `MouseButtons.Right` 를 만듭니다.
2. 이 `OnMouseClick` 를 인수로 사용하여 `MouseEventArgs` 메서드를 호출합니다.

사용자 지정 컨트롤에 대한 자세한 내용은 [디자인 타임에 Windows Forms 컨트롤 개발](#)을 참조하세요.

마우스 입력을 시뮬레이션하는 다른 방법이 있습니다. 예를 들어 일반적으로 마우스 입력을 통해 설정되는 상태를 나타내는 컨트롤 속성(예: `Checked` 컨트롤의 `CheckBox` 속성)을 프로그래밍 방식으로 설정하거나 시뮬레이션하려는 이벤트에 연결된 대리자를 직접 호출할 수 있습니다.

## 키보드 입력 시뮬레이션

마우스 입력에 대해 위에서 설명한 전략을 사용하여 키보드 입력을 시뮬레이션할 수도 있지만 Windows Forms에서는 활성 애플리케이션에 키 입력을 전송하기 위한 `SendKeys` 클래스도 제공합니다.

### Caution

다양한 키보드를 통해 전 세계에서 사용하기 위한 애플리케이션인 경우 `SendKeys.Send` 를 사용하면 예기치 않은 결과가 발생할 수 있으며 피해야 합니다.

## NOTE

[SendKeys](#) 클래스는 Windows Vista에서 실행되는 애플리케이션에서 사용할 수 있도록 .NET Framework 3.0에서 업데이트되었습니다. Windows Vista의 향상된 보안(사용자 계정 컨트롤 또는 UAC라고 함) 때문에 이전 구현이 예상대로 작동하지 않습니다.

[SendKeys](#) 클래스는 타이밍 문제에 취약하며, 이를 해결하기 위해 일부 개발자가 노력해야 했습니다. 업데이트된 구현도 타이밍 문제에 취약하지만 약간 더 빠르게 해결 방법에 대한 변경이 필요할 수도 있습니다. [SendKeys](#) 클래스는 먼저 이전 구현을 사용하려고 시도하며, 실패할 경우 새 구현을 사용합니다. 따라서 [SendKeys](#) 클래스는 운영 체제마다 다르게 동작할 수 있습니다. 또한 [SendKeys](#) 클래스가 새 구현을 사용하는 경우 [SendWait](#) 메서드는 다른 프로세스로 전송된 메시지가 처리될 때까지 기다리지 않습니다.

애플리케이션이 운영 체제와 관계없이 일관된 동작에 의존하는 경우 app.config 파일에 다음 애플리케이션 설정을 추가하여 [SendKeys](#) 클래스에서 새 구현을 사용하도록 강제할 수 있습니다.

```
<appSettings>
  <add key="SendKeys" value="SendInput"/>
</appSettings>
```

[SendKeys](#) 클래스에서 이전 구현을 사용하도록 강제하려면 `"JournalHook"` 값을 대신 사용합니다.

동일한 애플리케이션에 키 입력을 보내려면

1. [Send](#) 클래스의 [SendWait](#) 또는 [SendKeys](#) 메서드를 호출합니다. 애플리케이션의 활성 컨트롤이 지정된 키 입력을 받습니다. 다음 코드 예제에서는 [Send](#) 를 사용하여 사용자가 폼의 화면을 두 번 클릭할 때 Enter 키 누름을 시뮬레이션합니다. 이 예제에서는 탭 인덱스가 0인 단일 [Form](#) 컨트롤이 있는 [Button](#) 을 가정합니다.

```
// Send a key to the button when the user double-clicks anywhere
// on the form.
private:
void Form1_DoubleClick(Object^ sender, EventArgs^ e)
{
    // Send the enter key to the button, which triggers the click
    // event for the button. This works because the tab stop of
    // the button is 0.
    SendKeys::Send("{ENTER}");
}
```

```
// Send a key to the button when the user double-clicks anywhere
// on the form.
private void Form1_DoubleClick(object sender, EventArgs e)
{
    // Send the enter key to the button, which raises the click
    // event for the button. This works because the tab stop of
    // the button is 0.
    SendKeys.Send("{ENTER}");
}
```

```
' Send a key to the button when the user double-clicks anywhere
' on the form.
Private Sub Form1_DoubleClick(ByVal sender As Object, _
    ByVal e As EventArgs) Handles Me.DoubleClick

    ' Send the enter key to the button, which raises the click
    ' event for the button. This works because the tab stop of
    ' the button is 0.
    SendKeys.Send("{ENTER}")
End Sub
```

다른 애플리케이션에 키 입력을 보내려면

1. 키 입력을 수신할 애플리케이션 창을 활성화한 다음 [Send](#) 또는 [SendWait](#) 메서드를 호출합니다. 다른 애플리케이션을 활성화할 관리되는 메서드가 없으므로 네이티브 Windows 메서드를 사용하여 다른 애플리케이션에 포커스를 강제로 설정해야 합니다. 다음 코드 예제에서는 플랫폼 호출을 통해 `FindWindow` 및 `SetForegroundWindow` 메서드를 호출하여 계산기 애플리케이션 창을 활성화한 다음 [SendWait](#) 를 호출하여 계산기 애플리케이션에 일련의 계산을 실행합니다.

#### NOTE

계산기 애플리케이션을 찾는 `FindWindow` 호출의 올바른 매개 변수는 Windows 버전에 따라 달라집니다. 다음 코드는 Windows 7에서 계산기 응용 프로그램을 찾습니다. Windows Vista에서 첫 번째 매개 변수를 "SciCalc"로 변경 합니다. Visual Studio에 포함된 Spy++ 도구를 사용하여 올바른 매개 변수를 확인할 수 있습니다.

```
// Get a handle to an application window.
public:
    [DllImport("USER32.DLL", CharSet = CharSet::Unicode)]
    static IntPtr FindWindow(String^ lpClassName, String^ lpWindowName);
public:
    // Activate an application window.
    [DllImport("USER32.DLL")]
    static bool SetForegroundWindow(IntPtr hWnd);

    // Send a series of key presses to the Calculator application.
private:
    void button1_Click(Object^ sender, EventArgs^ e)
    {
        // Get a handle to the Calculator application. The window class
        // and window name were obtained using the Spy++ tool.
        IntPtr calculatorHandle = FindWindow("CalcFrame", "Calculator");

        // Verify that Calculator is a running process.
        if (calculatorHandle == IntPtr::Zero)
        {
            MessageBox::Show("Calculator is not running.");
            return;
        }

        // Make Calculator the foreground application and send it
        // a set of calculations.
        SetForegroundWindow(calculatorHandle);
        SendKeys::SendWait("111");
        SendKeys::SendWait("*");
        SendKeys::SendWait("11");
        SendKeys::SendWait("=");
    }
}
```

```

// Get a handle to an application window.
[DllImport("USER32.DLL", CharSet = CharSet.Unicode)]
public static extern IntPtr FindWindow(string lpClassName,
    string lpWindowName);

// Activate an application window.
[DllImport("USER32.DLL")]
public static extern bool SetForegroundWindow(IntPtr hWnd);

// Send a series of key presses to the Calculator application.
private void button1_Click(object sender, EventArgs e)
{
    // Get a handle to the Calculator application. The window class
    // and window name were obtained using the Spy++ tool.
    IntPtr calculatorHandle = FindWindow("CalcFrame", "Calculator");

    // Verify that Calculator is a running process.
    if (calculatorHandle == IntPtr.Zero)
    {
        MessageBox.Show("Calculator is not running.");
        return;
    }

    // Make Calculator the foreground application and send it
    // a set of calculations.
    SetForegroundWindow(calculatorHandle);
    SendKeys.SendWait("111");
    SendKeys.SendWait("*");
    SendKeys.SendWait("11");
    SendKeys.SendWait("=");
}

```

```

' Get a handle to an application window.
Declare Auto Function FindWindow Lib "USER32.DLL" ( _
    ByVal lpClassName As String, _
    ByVal lpWindowName As String) As IntPtr

' Activate an application window.
Declare Auto Function SetForegroundWindow Lib "USER32.DLL" _
    (ByVal hWnd As IntPtr) As Boolean

' Send a series of key presses to the Calculator application.
Private Sub button1_Click(ByVal sender As Object, _
    ByVal e As EventArgs) Handles button1.Click

    ' Get a handle to the Calculator application. The window class
    ' and window name were obtained using the Spy++ tool.
    Dim calculatorHandle As IntPtr = FindWindow("CalcFrame", "Calculator")

    ' Verify that Calculator is a running process.
    If calculatorHandle = IntPtr.Zero Then
        MsgBox("Calculator is not running.")
        Return
    End If

    ' Make Calculator the foreground application and send it
    ' a set of calculations.
    SetForegroundWindow(calculatorHandle)
    SendKeys.SendWait("111")
    SendKeys.SendWait("*")
    SendKeys.SendWait("11")
    SendKeys.SendWait("=")
End Sub

```

## 예제

다음 코드 예제는 이전 코드 예제에 대한 전체 애플리케이션입니다.

```
#using <System.Drawing.dll>
#using <System.Windows.Forms.dll>
#using <System.dll>

using namespace System;
using namespace System::Runtime::InteropServices;
using namespace System::Drawing;
using namespace System::Windows::Forms;

namespace SimulateKeyPress
{
    public ref class Form1 : public Form
    {
    public:
        Form1()
        {
            Button^ button1 = gcnew Button();
            button1->Location = Point(10, 10);
            button1->TabIndex = 0;
            button1->Text = "Click to automate Calculator";
            button1->AutoSize = true;
            button1->Click += gcnew EventHandler(this, &Form1::button1_Click);

            this->DoubleClick += gcnew EventHandler(this,
                &Form1::Form1_DoubleClick);
            this->Controls->Add(button1);
        }

        // Get a handle to an application window.
    public:
        [DllImport("USER32.DLL", CharSet = CharSet::Unicode)]
        static IntPtr FindWindow(String^ lpClassName, String^ lpWindowName);
    public:
        // Activate an application window.
        [DllImport("USER32.DLL")]
        static bool SetForegroundWindow(IntPtr hWnd);

        // Send a series of key presses to the Calculator application.
    private:
        void button1_Click(Object^ sender, EventArgs^ e)
        {
            // Get a handle to the Calculator application. The window class
            // and window name were obtained using the Spy++ tool.
            IntPtr calculatorHandle = FindWindow("CalcFrame", "Calculator");

            // Verify that Calculator is a running process.
            if (calculatorHandle == IntPtr::Zero)
            {
                MessageBox::Show("Calculator is not running.");
                return;
            }

            // Make Calculator the foreground application and send it
            // a set of calculations.
            SetForegroundWindow(calculatorHandle);
            SendKeys::SendWait("111");
            SendKeys::SendWait("*");
            SendKeys::SendWait("11");
            SendKeys::SendWait("=");
        }

        // Send a key to the button when the user double-clicks anywhere
```

```

        // on the form.
private:
    void Form1_DoubleClick(Object^ sender, EventArgs^ e)
    {
        // Send the enter key to the button, which triggers the click
        // event for the button. This works because the tab stop of
        // the button is 0.
        SendKeys::Send("{ENTER}");
    }
};

}

[STAThread]
int main()
{
    Application::EnableVisualStyles();
    Application::Run(gcnew SimulateKeyPress::Form1());
}

```

```

using System;
using System.Runtime.InteropServices;
using System.Drawing;
using System.Windows.Forms;

namespace SimulateKeyPress
{
    class Form1 : Form
    {
        private Button button1 = new Button();

        [STAThread]
        public static void Main()
        {
            Application.EnableVisualStyles();
            Application.Run(new Form1());
        }

        public Form1()
        {
            button1.Location = new Point(10, 10);
            button1.TabIndex = 0;
            button1.Text = "Click to automate Calculator";
            button1.AutoSize = true;
            button1.Click += new EventHandler(button1_Click);

            this.DoubleClick += new EventHandler(Form1_DoubleClick);
            this.Controls.Add(button1);
        }

        // Get a handle to an application window.
        [DllImport("USER32.DLL", CharSet = CharSet.Unicode)]
        public static extern IntPtr FindWindow(string lpClassName,
            string lpWindowName);

        // Activate an application window.
        [DllImport("USER32.DLL")]
        public static extern bool SetForegroundWindow(IntPtr hWnd);

        // Send a series of key presses to the Calculator application.
        private void button1_Click(object sender, EventArgs e)
        {
            // Get a handle to the Calculator application. The window class
            // and window name were obtained using the Spy++ tool.
            IntPtr calculatorHandle = FindWindow("CalcFrame", "Calculator");

            // Verify that Calculator is a running process.
            if (calculatorHandle == IntPtr.Zero)
            {

```

```

        {
            MessageBox.Show("Calculator is not running.");
            return;
        }

        // Make Calculator the foreground application and send it
        // a set of calculations.
        SetForegroundWindow(calculatorHandle);
        SendKeys.SendWait("111");
        SendKeys.SendWait("*");
        SendKeys.SendWait("11");
        SendKeys.SendWait("=");
    }

    // Send a key to the button when the user double-clicks anywhere
    // on the form.
    private void Form1_DoubleClick(object sender, EventArgs e)
    {
        // Send the enter key to the button, which raises the click
        // event for the button. This works because the tab stop of
        // the button is 0.
        SendKeys.Send("{ENTER}");
    }
}

```

```

Imports System.Runtime.InteropServices
Imports System.Drawing
Imports System.Windows.Forms

Namespace SimulateKeyPress

    Class Form1
        Inherits Form
        Private WithEvents button1 As New Button()

        <STAThread()> _
        Public Shared Sub Main()
            Application.EnableVisualStyles()
            Application.Run(New Form1())
        End Sub

        Public Sub New()
            button1.Location = New Point(10, 10)
            button1.TabIndex = 0
            button1.Text = "Click to automate Calculator"
            button1.AutoSize = True
            Me.Controls.Add(button1)
        End Sub

        ' Get a handle to an application window.
        Declare Auto Function FindWindow Lib "USER32.DLL" ( _
            ByVal lpClassName As String, _
            ByVal lpWindowName As String) As IntPtr

        ' Activate an application window.
        Declare Auto Function SetForegroundWindow Lib "USER32.DLL" _
            (ByVal hWnd As IntPtr) As Boolean

        ' Send a series of key presses to the Calculator application.
        Private Sub button1_Click(ByVal sender As Object, _
            ByVal e As EventArgs) Handles button1.Click

            ' Get a handle to the Calculator application. The window class
            ' and window name were obtained using the Spy++ tool.
            Dim calculatorHandle As IntPtr = FindWindow("CalcFrame", "Calculator")

            ' Verify that Calculator is a running process

```

```

        verify that calculator is a running process.
    If calculatorHandle = IntPtr.Zero Then
        MsgBox("Calculator is not running.")
        Return
    End If

    ' Make Calculator the foreground application and send it
    ' a set of calculations.
    SetForegroundWindow(calculatorHandle)
    SendKeys.SendWait("111")
    SendKeys.SendWait("*")
    SendKeys.SendWait("11")
    SendKeys.SendWait("=")
End Sub

' Send a key to the button when the user double-clicks anywhere
' on the form.
Private Sub Form1_DoubleClick(ByVal sender As Object, _
    ByVal e As EventArgs) Handles Me.DoubleClick

    ' Send the enter key to the button, which raises the click
    ' event for the button. This works because the tab stop of
    ' the button is 0.
    SendKeys.Send("{ENTER}")
End Sub

End Class
End Namespace

```

## 코드 컴파일

이 예제에는 다음 사항이 필요합니다.

- System, System.Drawing 및 System.Windows.Forms 어셈블리에 대한 참조

## 참조

- [Windows Forms에 사용자 입력](#)



# 방법: Windows Forms 컨트롤에서 사용자 입력 이벤트 처리

2020-02-03 • 27 minutes to read • [Edit Online](#)

이 예제에서는 Windows Forms 컨트롤에서 발생할 수 있는 대부분의 키보드, 마우스, 포커스 및 유효성 검사 이벤트를 처리하는 방법을 보여 줍니다. 포커스가 있을 경우 `TextBoxInput` 이라는 텍스트 상자가 이벤트를 수신하며, 각 이벤트에 대한 정보가 이벤트 발생 순서대로 `TextBoxOutput` 이라는 텍스트 상자에 기록됩니다. 애플리케이션에는 보고할 이벤트를 필터링하는 데 사용할 수 있는 확인란 집합도 포함되어 있습니다.

## 예제

```
#using <System.Drawing.dll>
#using <System.Windows.Forms.dll>
#using <System.dll>

using namespace System;
using namespace System::Drawing;
using namespace System::ComponentModel;
using namespace System::Windows::Forms;

namespace UserInputWalkthrough
{
    public ref class Form1 : public Form
    {
    {
        Label^ lblEvent;
        Label^ lblInput;

        TextBox^ TextBoxOutput;
        TextBox^ TextBoxInput;
        GroupBox^ GroupBoxEvents;
        Button^ ButtonClear;
        LinkLabel^ LinkLabelDrag;

        CheckBox^ CheckBoxToggleAll;
        CheckBox^ CheckBoxMouse;
        CheckBox^ CheckBoxMouseEnter;
        CheckBox^ CheckBoxMouseMove;
        CheckBox^ CheckBoxMousePoints;
        CheckBox^ CheckBoxMouseDrag;
        CheckBox^ CheckBoxMouseDragOver;
        CheckBox^ CheckBoxKeyboard;
        CheckBox^ CheckBoxKeyUpDown;
        CheckBox^ CheckBoxFocus;
        CheckBox^ CheckBoxValidation;

    public:
        Form1() : Form()
        {
            this->Load += gcnew EventHandler(this, &Form1::Form1_Load);

            lblEvent = gcnew Label();
            lblInput = gcnew Label();

            TextBoxOutput = gcnew TextBox();
            TextBoxInput = gcnew TextBox();
            GroupBoxEvents = gcnew GroupBox();
            ButtonClear = gcnew Button();
            LinkLabelDrag = gcnew LinkLabel();
```

```

        CheckBoxToggleAll = gcnew CheckBox();
        CheckBoxMouse = gcnew CheckBox();
        CheckBoxMouseEnter = gcnew CheckBox();
        CheckBoxMouseMove = gcnew CheckBox();
        CheckBoxMousePoints = gcnew CheckBox();
        CheckBoxMouseDrag = gcnew CheckBox();
        CheckBoxMouseDragOver = gcnew CheckBox();
        CheckBoxKeyboard = gcnew CheckBox();
        CheckBoxKeyUpDown = gcnew CheckBox();
        CheckBoxFocus = gcnew CheckBox();
        CheckBoxValidation = gcnew CheckBox();
    }

private:
    void Form1_Load(Object^ sender, EventArgs^ e)
    {
        this->GroupBoxEvents->SuspendLayout();
        this->SuspendLayout();

        lblEvent->Location = Point(232, 12);
        lblEvent->Size = System::Drawing::Size(98, 14);
        lblEvent->AutoSize = true;
        lblEvent->Text = "Generated Events:";

        lblInput->Location = Point(13, 12);
        lblInput->Size = System::Drawing::Size(95, 14);
        lblInput->AutoSize = true;
        lblInput->Text = "User Input Target:";

        TextBoxInput->Location = Point(13, 34);
        TextBoxInput->Size = System::Drawing::Size(200, 200);
        TextBoxInput->AllowDrop = true;
        TextBoxInput->AutoSize = false;
        TextBoxInput->Cursor = Cursors::Cross;
        TextBoxInput->Multiline = true;
        TextBoxInput->TabIndex = 1;

        LinkLabelDrag->AllowDrop = true;
        LinkLabelDrag->AutoSize = true;
        LinkLabelDrag->Location = Point(13, 240);
        LinkLabelDrag->Size = System::Drawing::Size(175, 14);
        LinkLabelDrag->TabIndex = 2;
        LinkLabelDrag->TabStop = true;
        LinkLabelDrag->Text = "Click here to use as a drag source";
        LinkLabelDrag->Links->Add(gcnew LinkLabel::Link(0,
            LinkLabelDrag->Text->Length));

        GroupBoxEvents->Location = Point(13, 281);
        GroupBoxEvents->Size = System::Drawing::Size(200, 302);
        GroupBoxEvents->Text = "Event Filter:";
        GroupBoxEvents->TabStop = true;
        GroupBoxEvents->TabIndex = 3;
        GroupBoxEvents->Controls->Add(CheckBoxMouseEnter);
        GroupBoxEvents->Controls->Add(CheckBoxToggleAll);
        GroupBoxEvents->Controls->Add(CheckBoxMousePoints);
        GroupBoxEvents->Controls->Add(CheckBoxKeyUpDown);
        GroupBoxEvents->Controls->Add(CheckBoxMouseDragOver);
        GroupBoxEvents->Controls->Add(CheckBoxMouseDrag);
        GroupBoxEvents->Controls->Add(CheckBoxValidation);
        GroupBoxEvents->Controls->Add(CheckBoxMouseMove);
        GroupBoxEvents->Controls->Add(CheckBoxFocus);
        GroupBoxEvents->Controls->Add(CheckBoxKeyboard);
        GroupBoxEvents->Controls->Add(CheckBoxMouse);

        CheckBoxToggleAll->AutoSize = true;
        CheckBoxToggleAll->Location = Point(7, 20);
        CheckBoxToggleAll->Size = System::Drawing::Size(122, 17);
        CheckBoxToggleAll->TabIndex = 4;
        CheckBoxToggleAll->Text = "Toggle All Events":

```

CheckBoxMouseEnter->Text = "Mouse Enter/Hover/Leave";

```
CheckBoxMouse->AutoSize = true;
CheckBoxMouse->Location = Point(7, 45);
CheckBoxMouse->Size = System::Drawing::Size(137, 17);
CheckBoxMouse->TabIndex = 5;
CheckBoxMouse->Text = "Mouse and Click Events";
```

```
CheckBoxMouseEnter->AutoSize = true;
CheckBoxMouseEnter->Location = Point(26, 69);
CheckBoxMouseEnter->Margin =
    System::Windows::Forms::Padding(3, 3, 3, 1);
CheckBoxMouseEnter->Size = System::Drawing::Size(151, 17);
CheckBoxMouseEnter->TabIndex = 6;
CheckBoxMouseEnter->Text = "Mouse Enter/Hover/Leave";
```

```
CheckBoxMouseMove->AutoSize = true;
CheckBoxMouseMove->Location = Point(26, 89);
CheckBoxMouseMove->Margin =
    System::Windows::Forms::Padding(3, 2, 3, 3);
CheckBoxMouseMove->Size = System::Drawing::Size(120, 17);
CheckBoxMouseMove->TabIndex = 7;
CheckBoxMouseMove->Text = "Mouse Move Events";
```

```
CheckBoxMousePoints->AutoSize = true;
CheckBoxMousePoints->Location = Point(26, 112);
CheckBoxMousePoints->Margin =
    System::Windows::Forms::Padding(3, 3, 3, 1);
CheckBoxMousePoints->Size = System::Drawing::Size(141, 17);
CheckBoxMousePoints->TabIndex = 8;
CheckBoxMousePoints->Text = "Draw Mouse Points";
```

```
CheckBoxMouseDrag->AutoSize = true;
CheckBoxMouseDrag->Location = Point(26, 135);
CheckBoxMouseDrag->Margin =
    System::Windows::Forms::Padding(3, 1, 3, 3);
CheckBoxMouseDrag->Size = System::Drawing::Size(151, 17);
CheckBoxMouseDrag->TabIndex = 9;
CheckBoxMouseDrag->Text = "Mouse Drag && Drop Events";
```

```
CheckBoxMouseDragOver->AutoSize = true;
CheckBoxMouseDragOver->Location = Point(44, 159);
CheckBoxMouseDragOver->Size = System::Drawing::Size(142, 17);
CheckBoxMouseDragOver->TabIndex = 10;
CheckBoxMouseDragOver->Text = "Mouse Drag Over Events";
```

```
CheckBoxKeyboard->AutoSize = true;
CheckBoxKeyboard->Location = Point(8, 184);
CheckBoxKeyboard->Size = System::Drawing::Size(103, 17);
CheckBoxKeyboard->TabIndex = 11;
CheckBoxKeyboard->Text = "Keyboard Events";
```

```
CheckBoxKeyUpDown->AutoSize = true;
CheckBoxKeyUpDown->Location = Point(26, 207);
CheckBoxKeyUpDown->Margin =
    System::Windows::Forms::Padding(3, 3, 3, 1);
CheckBoxKeyUpDown->Size = System::Drawing::Size(133, 17);
CheckBoxKeyUpDown->TabIndex = 12;
CheckBoxKeyUpDown->Text = "Key Up && Down Events";
```

```
CheckBoxFocus->AutoSize = true;
CheckBoxFocus->Location = Point(8, 233);
CheckBoxFocus->Margin =
    System::Windows::Forms::Padding(3, 2, 3, 3);
CheckBoxFocus->Size = System::Drawing::Size(146, 17);
CheckBoxFocus->TabIndex = 13;
CheckBoxFocus->Text = "Focus && Activation Events";
```

```
CheckBoxValidation->AutoSize = true;
CheckBoxValidation->Location = Point(8, 257);
```

```

CheckBoxValidation->Location = Point(8, 25);
CheckBoxValidation->Size = System::Drawing::Size(104, 17);
CheckBoxValidation->TabIndex = 14;
CheckBoxValidation->Text = "Validation Events";

TextBoxOutput->Location = Point(232, 34);
TextBoxOutput->Size = System::Drawing::Size(308, 510);
TextBoxOutput->Multiline = true;
TextBoxOutput->CausesValidation = false;
TextBoxOutput->ReadOnly = true;
TextBoxOutput->ScrollBars = ScrollBars::Vertical;
TextBoxOutput->TabIndex = 15;
TextBoxOutput->WordWrap = false;

ButtonClear->Location = Point(232, 560);
ButtonClear->Size = System::Drawing::Size(308, 23);
ButtonClear->TabIndex = 16;
ButtonClear->Text = "Clear Event List";

this->ClientSize = System::Drawing::Size(552, 595);
this->Controls->Add(LinkLabelDrag);
this->Controls->Add(ButtonClear);
this->Controls->Add(GroupBoxEvents);
this->Controls->Add(lblEvent);
this->Controls->Add(lblInput);
this->Controls->Add(TextBoxInput);
this->Controls->Add(TextBoxOutput);
this->Text = "User Input Events";

ButtonClear->Click +=
    gcnew EventHandler(this, &Form1::ButtonClear_Click);
TextBoxInput->KeyDown +=
    gcnew KeyEventHandler(this, &Form1::TextBoxInput_KeyDown);
TextBoxInput->KeyPress +=
    gcnew KeyPressEventHandler(this,
        &Form1::TextBoxInput_KeyPress);
TextBoxInput->KeyUp +=
    gcnew KeyEventHandler(this, &Form1::TextBoxInput_KeyUp);
TextBoxInput->Click +=
    gcnew EventHandler(this, &Form1::TextBoxInput_Click);
TextBoxInput->DoubleClick +=
    gcnew EventHandler(this, &Form1::TextBoxInput_DoubleClick);
TextBoxInput->MouseClick +=
    gcnew MouseEventHandler(this, &Form1::TextBoxInput_MouseClick);
TextBoxInput->MouseDoubleClick +=
    gcnew MouseEventHandler(this,
        &Form1::TextBoxInput_MouseDoubleClick);
TextBoxInput->MouseDown +=
    gcnew MouseEventHandler(this, &Form1::TextBoxInput_MouseDown);
TextBoxInput->MouseUp +=
    gcnew MouseEventHandler(this, &Form1::TextBoxInput_MouseUp);
TextBoxInput->MouseEnter +=
    gcnew EventHandler(this, &Form1::TextBoxInput_MouseEnter);
TextBoxInput->MouseHover +=
    gcnew EventHandler(this, &Form1::TextBoxInput_MouseHover);
TextBoxInput->MouseLeave +=
    gcnew EventHandler(this, &Form1::TextBoxInput_MouseLeave);
TextBoxInput->MouseWheel +=
    gcnew MouseEventHandler(this, &Form1::TextBoxInput_MouseWheel);
TextBoxInput->MouseMove +=
    gcnew MouseEventHandler(this, &Form1::TextBoxInput_MouseMove);
TextBoxInput->MouseCaptureChanged +=
    gcnew EventHandler(this,
        &Form1::TextBoxInput_MouseCaptureChanged);
TextBoxInput->DragEnter +=
    gcnew DragEventHandler(this, &Form1::TextBoxInput_DragEnter);
TextBoxInput->DragDrop +=
    gcnew DragEventHandler(this, &Form1::TextBoxInput_DragDrop);
TextBoxInput->DragOver +=

```

```

        gcnew DragEventHandler(this, &Form1::TextBoxInput_DragOver);
        TextBoxInput->DragLeave +=
            gcnew EventHandler(this, &Form1::TextBoxInput_DragLeave);
        TextBoxInput->Enter +=
            gcnew EventHandler(this, &Form1::TextBoxInput_Enter);
        TextBoxInput->Leave +=
            gcnew EventHandler(this, &Form1::TextBoxInput_Leave);
        TextBoxInput->GotFocus +=
            gcnew EventHandler(this, &Form1::TextBoxInput_GotFocus);
        TextBoxInput->LostFocus +=
            gcnew EventHandler(this, &Form1::TextBoxInput_LostFocus);
        TextBoxInput->Validated +=
            gcnew EventHandler(this, &Form1::TextBoxInput_Validated);
        TextBoxInput->Validating +=
            gcnew CancelEventHandler(this,
                &Form1::TextBoxInput_Validating);

        LinkLabelDrag->MouseDown +=
            gcnew MouseEventHandler(this, &Form1::LinkLabelDrag_MouseDown);
        LinkLabelDrag->GiveFeedback +=
            gcnew GiveFeedbackEventHandler(this,
                &Form1::LinkLabelDrag_GiveFeedback);

        CheckBoxToggleAll->CheckedChanged +=
            gcnew EventHandler(this,
                &Form1::CheckBoxToggleAll_CheckedChanged);
        CheckBoxMouse->CheckedChanged +=
            gcnew EventHandler(this, &Form1::CheckBoxMouse_CheckedChanged);
        CheckBoxMouseDrag->CheckedChanged +=
            gcnew EventHandler(this,
                &Form1::CheckBoxMouseDrag_CheckedChanged);
        CheckBoxMouseEnter->CheckedChanged +=
            gcnew EventHandler(this,
                &Form1::CheckBoxMouseMove_CheckedChanged);
        CheckBoxMouseMove->CheckedChanged +=
            gcnew EventHandler(this,
                &Form1::CheckBoxMouseMove_CheckedChanged);
        CheckBoxKeyboard->CheckedChanged +=
            gcnew EventHandler(this,
                &Form1::CheckBoxKeyboard_CheckedChanged);

        this->GroupBoxEvents->ResumeLayout(false);
        this->GroupBoxEvents->PerformLayout();
        this->ResumeLayout(false);
        this->PerformLayout();
        CheckAllChildCheckBoxes(this, true);
    }

    // Recursively search the form for all contained checkboxes and
    // initially check them
private:
    void CheckAllChildCheckBoxes(Control^ parent, bool value)
    {
        CheckBox^ box;
        for each (Control^ currentControl in parent->Controls)
        {
            if (dynamic_cast<CheckBox^>(currentControl))
            {
                box = (CheckBox^)currentControl;
                box->Checked = value;
            }

            // Recurse if control contains other controls
            if (currentControl->Controls->Count > 0)
            {
                CheckAllChildCheckBoxes(currentControl, value);
            }
        }
    }
}

```

```

        // All-purpose method for displaying a line of text in one of the
        // text boxes.
private:
    void DisplayLine(String^ line)
    {
        TextBoxOutput->AppendText(line);
        TextBoxOutput->AppendText(Environment::NewLine);
    }

    // Click event handler for the button that clears the text box.
private:
    void ButtonClear_Click(Object^ sender, EventArgs^ e)
    {
        TextBoxOutput->Invalidate();
        TextBoxOutput->Clear();
    }

private:
    void TextBoxInput_KeyDown(Object^ sender, KeyEventArgs^ e)
    {
        if (CheckBoxKeyUpDown->Checked)
        {
            DisplayLine("KeyDown: " + e->KeyData.ToString());
        }
    }

private:
    void TextBoxInput_KeyUp(Object^ sender, KeyEventArgs^ e)
    {
        if (CheckBoxKeyUpDown->Checked)
        {
            DisplayLine("KeyUp: " + e->KeyData.ToString());
        }
    }

private:
    void TextBoxInput_KeyPress(Object^ sender,
        KeyPressEventArgs^ e)
    {
        if (CheckBoxKeyboard->Checked)
        {
            if (Char::IsWhiteSpace(e->KeyChar))
            {
                DisplayLine("KeyPress: WS");
            }
            else
            {
                DisplayLine("KeyPress: " + e->KeyChar.ToString());
            }
        }
    }

private:
    void TextBoxInput_Click(Object^ sender, EventArgs^ e)
    {
        if (CheckBoxMouse->Checked)
        {
            DisplayLine("Click event");
        }
    }

private:
    void TextBoxInput_DoubleClick(Object^ sender, EventArgs^ e)
    {
        if (CheckBoxMouse->Checked)
        {
            DisplayLine("DoubleClick event");
        }
    }

```

```

    }

private:
    void TextBoxInput_MouseClick(Object^ sender, MouseEventArgs^ e)
    {
        if (CheckBoxMouse->Checked)
        {
            DisplayLine("MouseClicked: " + e->Button.ToString() +
                " " + e->Location.ToString());
        }
    }

private:
    void TextBoxInput_MouseDoubleClick(Object^ sender,
        MouseEventArgs^ e)
    {
        if (CheckBoxMouse->Checked)
        {
            DisplayLine("MouseDoubleClick: " + e->Button.ToString() +
                " " + e->Location.ToString());
        }
    }

private:
    void TextBoxInput_MouseDown(Object^ sender,
        MouseEventArgs^ e)
    {
        if (CheckBoxMouse->Checked)
        {
            DisplayLine("MouseDown: " + e->Button.ToString() +
                " " + e->Location.ToString());
        }
    }

private:
    void TextBoxInput_MouseUp(Object^ sender,
        MouseEventArgs^ e)
    {
        if (CheckBoxMouse->Checked)
        {
            DisplayLine("MouseUp: " + e->Button.ToString() +
                " " + e->Location.ToString());
        }

        // The TextBox control was designed to change focus only on
        // the primary click, so force focus to avoid user confusion.
        if (!TextBoxInput->Focused)
        {
            TextBoxInput->Focus();
        }
    }

private:
    void TextBoxInput_MouseEnter(Object^ sender, EventArgs^ e)
    {
        if (CheckBoxMouseEnter->Checked)
        {
            DisplayLine("MouseEnter event");
        }
    }

private:
    void TextBoxInput_MouseHover(Object^ sender, EventArgs^ e)
    {
        if (CheckBoxMouseEnter->Checked)
        {
            DisplayLine("MouseHover event");
        }
    }
}

```

```

private:
    void TextBoxInput_MouseLeave(Object^ sender, EventArgs^ e)
    {
        if (CheckBoxMouseEnter->Checked)
        {
            DisplayLine("MouseLeave event");
        }
    }

private:
    void TextBoxInput_MouseWheel(Object^ sender,
        MouseEventArgs^ e)
    {
        if (CheckBoxMouse->Checked)
        {
            DisplayLine("MouseWheel: " + e->Delta.ToString() +
                " detents at " + e->Location.ToString());
        }
    }

private:
    void TextBoxInput_MouseMove(Object^ sender,
        MouseEventArgs^ e)
    {
        if (CheckBoxMouseMove->Checked)
        {
            DisplayLine("MouseMove: " + e->Button.ToString() + " " +
                e->Location.ToString());
        }

        if (CheckBoxMousePoints->Checked)
        {
            Graphics^ g = TextBoxInput->CreateGraphics();
            g->FillRectangle(Brushes::Black, e->Location.X,
                e->Location.Y, 1, 1);
            delete g;
        }
    }

private:
    void TextBoxInput_MouseCaptureChanged(Object^ sender,
        EventArgs^ e)
    {
        if (CheckBoxMouseDrag->Checked)
        {
            DisplayLine("MouseCaptureChanged event");
        }
    }

private:
    void TextBoxInput_DragEnter(Object^ sender, DragEventArgs^ e)
    {
        if (CheckBoxMouseDrag->Checked)
        {
            Point^ pt = gcnew Point(e->X, e->Y);
            DisplayLine("DragEnter: " +
                CovertKeyStateToString(e->KeyState)
                + " at " + pt->ToString());
        }
    }

private:
    void TextBoxInput_DragDrop(Object^ sender, DragEventArgs^ e)
    {
        if (CheckBoxMouseDrag->Checked)
        {
            Point^ pt = gcnew Point(e->X, e->Y);
            DisplayLine("DragDrop: " +

```



```

        CovertKeyStateToString(e->KeyState)
        + " at " + pt->ToString());
    }
}

private:
void TextBoxInput_DragOver(Object^ sender, DragEventArgs^ e)
{
    if (CheckBoxMouseDragOver->Checked)
    {
        Point^ pt = gcnew Point(e->X, e->Y);
        DisplayLine("DragOver: " +
            CovertKeyStateToString(e->KeyState)
            + " at " + pt->ToString());
    }

    // Allow if drop data is of type string.
    if (!e->Data->GetDataPresent(String::typeid))
    {
        e->Effect = DragDropEffects::None;
    }
    else
    {
        e->Effect = DragDropEffects::Copy;
    }
}

private:
void TextBoxInput_DragLeave(Object^ sender,
    EventArgs^ e)
{
    if (CheckBoxMouseDrag->Checked)
    {
        DisplayLine("DragLeave event");
    }
}

private:
static String^ CovertKeyStateToString(int keyState)
{
    String^ keyString = "None";

    // Which button was pressed?
    if ((keyState & 1) == 1)
    {
        keyString = "Left";
    }
    else if ((keyState & 2) == 2)
    {
        keyString = "Right";
    }
    else if ((keyState & 16) == 16)
    {
        keyString = "Middle";
    }

    // Are one or more modifier keys also pressed?
    if ((keyState & 4) == 4)
    {
        keyString += "+SHIFT";
    }

    if ((keyState & 8) == 8)
    {
        keyString += "+CTRL";
    }

    if ((keyState & 32) == 32)
    {

```

```

        keyString += "+ALT";
    }

    return keyString;
}

private:
void TextBoxInput_Enter(Object^ sender, EventArgs^ e)
{
    if (CheckBoxFocus->Checked)
    {
        DisplayLine("Enter event");
    }
}

private:
void TextBoxInput_Leave(Object^ sender, EventArgs^ e)
{
    if (CheckBoxFocus->Checked)
    {
        DisplayLine("Leave event");
    }
}

private:
void TextBoxInput_GotFocus(Object^ sender, EventArgs^ e)
{
    if (CheckBoxFocus->Checked)
    {
        DisplayLine("GotFocus event");
    }
}

private:
void TextBoxInput_LostFocus(Object^ sender, EventArgs^ e)
{
    if (CheckBoxFocus->Checked)
    {
        DisplayLine("LostFocus event");
    }
}

private:
void TextBoxInput_Validated(Object^ sender, EventArgs^ e)
{
    if (CheckBoxValidation->Checked)
    {
        DisplayLine("Validated event");
    }
}

private:
void TextBoxInput_Validating(
    Object^ sender, CancelEventArgs^ e)
{
    if (CheckBoxValidation->Checked)
    {
        DisplayLine("Validating event");
    }
}

private:
void CheckBoxToggleAll_CheckedChanged(
    Object^ sender, EventArgs^ e)
{
    if (dynamic_cast<CheckBox^>(sender))
    {
        CheckAllChildCheckBoxes(this, ((CheckBox^)sender)->Checked);
    }
}

```

```

    }
}

private:
    void CheckBoxMouse_CheckedChanged(
        Object^ sender, EventArgs^ e)
    {
        ConfigureCheckBoxSettings();
    }

private:
    void CheckBoxMouseDrag_CheckedChanged(
        Object^ sender, EventArgs^ e)
    {
        ConfigureCheckBoxSettings();
    }

private:
    void CheckBoxKeyboard_CheckedChanged(
        Object^ sender, EventArgs^ e)
    {
        ConfigureCheckBoxSettings();
    }

private:
    void CheckBoxMouseMove_CheckedChanged(
        Object^ sender, EventArgs^ e)
    {
        ConfigureCheckBoxSettings();
    }

    // Reconcile dependencies between the check box
    // selection choices.
private:
    void ConfigureCheckBoxSettings()
    {
        // CheckBoxMouse is a top-level check box.
        if (!CheckBoxMouse->Checked)
        {
            CheckBoxMouseEnter->Enabled = false;
            CheckBoxMouseMove->Enabled = false;
            CheckBoxMouseDrag->Enabled = false;
            CheckBoxMouseDragOver->Enabled = false;
            CheckBoxMousePoints->Enabled = false;
        }
        else
        {
            CheckBoxMouseEnter->Enabled = true;
            CheckBoxMouseMove->Enabled = true;
            CheckBoxMouseDrag->Enabled = true;
            CheckBoxMousePoints->Enabled = true;

            // Enable children depending on the state of the parent.
            if (!CheckBoxMouseDrag->Checked)
            {
                CheckBoxMouseDragOver->Enabled = false;
            }
            else
            {
                CheckBoxMouseDragOver->Enabled = true;
            }
        }

        if (!CheckBoxKeyboard->Checked)
        {
            CheckBoxKeyUpDown->Enabled = false;
        }
        else
        {

```

```

        CheckBoxKeyUpDown->Enabled = true;
    }
}

private:
    void LinkLabelDrag_MouseDown(Object^ sender, MouseEventArgs^ e)
    {
        String^ data = "Sample Data";
        LinkLabelDrag->DoDragDrop(data, DragDropEffects::All);
    }

private:
    void LinkLabelDrag_GiveFeedback(Object^ sender,
        GiveFeedbackEventArgs^ e)
    {
        if ((e->Effect & DragDropEffects::Copy) ==
            DragDropEffects::Copy)
        {
            LinkLabelDrag->Cursor = Cursors::HSplit;
        }
        else
        {
            LinkLabelDrag->Cursor = Cursors::Default;
        }
    }
};

[STAThread]
int main()
{
    Application::EnableVisualStyles();
    Application::Run(gcnew UserInputWalkthrough::Form1());
}

```

```

using System;
using System.Drawing;
using System.ComponentModel;
using System.Windows.Forms;

namespace UserInputWalkthrough
{
    public class Form1 : Form
    {
        Label Label1 = new Label();
        Label Label2 = new Label();
        TextBox TextBoxOutput = new TextBox();
        TextBox TextBoxInput = new TextBox();
        GroupBox GroupBoxEvents = new GroupBox();
        Button ButtonClear = new Button();
        LinkLabel LinkLabelDrag = new LinkLabel();

        CheckBox CheckBoxToggleAll = new CheckBox();
        CheckBox CheckBoxMouse = new CheckBox();
        CheckBox CheckBoxMouseEnter = new CheckBox();
        CheckBox CheckBoxMouseMove = new CheckBox();
        CheckBox CheckBoxMousePoints = new CheckBox();
        CheckBox CheckBoxMouseDrag = new CheckBox();
        CheckBox CheckBoxMouseDragOver = new CheckBox();
        CheckBox CheckBoxKeyboard = new CheckBox();
        CheckBox CheckBoxKeyUpDown = new CheckBox();
        CheckBox CheckBoxFocus = new CheckBox();
        CheckBox CheckBoxValidation = new CheckBox();

        [STAThread]
        static void Main()
        {

```

```

        Application.EnableVisualStyles();
        Application.Run(new Form1());
    }

    public Form1()
        : base()
    {
        this.Load += new EventHandler(Form1_Load);
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        this.GroupBoxEvents.SuspendLayout();
        this.SuspendLayout();

        Label1.Location = new Point(232, 12);
        Label1.Size = new Size(98, 14);
        Label1.AutoSize = true;
        Label1.Text = "Generated Events:";

        Label2.Location = new Point(13, 12);
        Label2.Size = new Size(95, 14);
        Label2.AutoSize = true;
        Label2.Text = "User Input Target:";

        TextBoxInput.Location = new Point(13, 34);
        TextBoxInput.Size = new Size(200, 200);
        TextBoxInput.AllowDrop = true;
        TextBoxInput.AutoSize = false;
        TextBoxInput.Cursor = Cursors.Cross;
        TextBoxInput.Multiline = true;
        TextBoxInput.TabIndex = 1;

        LinkLabelDrag.AllowDrop = true;
        LinkLabelDrag.AutoSize = true;
        LinkLabelDrag.Location = new Point(13, 240);
        LinkLabelDrag.Size = new Size(175, 14);
        LinkLabelDrag.TabIndex = 2;
        LinkLabelDrag.TabStop = true;
        LinkLabelDrag.Text = "Click here to use as a drag source";
        LinkLabelDrag.Links.Add(new LinkLabel.Link(0,
            LinkLabelDrag.Text.Length));

        GroupBoxEvents.Location = new Point(13, 281);
        GroupBoxEvents.Size = new Size(200, 302);
        GroupBoxEvents.Text = "Event Filter:";
        GroupBoxEvents.TabStop = true;
        GroupBoxEvents.TabIndex = 3;
        GroupBoxEvents.Controls.Add(CheckBoxMouseEnter);
        GroupBoxEvents.Controls.Add(CheckBoxToggleAll);
        GroupBoxEvents.Controls.Add(CheckBoxMousePoints);
        GroupBoxEvents.Controls.Add(CheckBoxKeyUpDown);
        GroupBoxEvents.Controls.Add(CheckBoxMouseDragOver);
        GroupBoxEvents.Controls.Add(CheckBoxMouseDrag);
        GroupBoxEvents.Controls.Add(CheckBoxValidation);
        GroupBoxEvents.Controls.Add(CheckBoxMouseMove);
        GroupBoxEvents.Controls.Add(CheckBoxFocus);
        GroupBoxEvents.Controls.Add(CheckBoxKeyboard);
        GroupBoxEvents.Controls.Add(CheckBoxMouse);

        CheckBoxToggleAll.AutoSize = true;
        CheckBoxToggleAll.Location = new Point(7, 20);
        CheckBoxToggleAll.Size = new Size(122, 17);
        CheckBoxToggleAll.TabIndex = 4;
        CheckBoxToggleAll.Text = "Toggle All Events";

        CheckBoxMouse.AutoSize = true;
        CheckBoxMouse.Location = new Point(7, 45);
        CheckBoxMouse.Size = new Size(137, 17);
    }

```

```
CheckBoxMouse.TabIndex = 5;
CheckBoxMouse.Text = "Mouse and Click Events";

CheckBoxMouseEnter.AutoSize = true;
CheckBoxMouseEnter.Location = new Point(26, 69);
CheckBoxMouseEnter.Margin = new Padding(3, 3, 3, 1);
CheckBoxMouseEnter.Size = new System.Drawing.Size(151, 17);
CheckBoxMouseEnter.TabIndex = 6;
CheckBoxMouseEnter.Text = "Mouse Enter/Hover/Leave";

CheckBoxMouseMove.AutoSize = true;
CheckBoxMouseMove.Location = new Point(26, 89);
CheckBoxMouseMove.Margin = new Padding(3, 2, 3, 3);
CheckBoxMouseMove.Size = new Size(120, 17);
CheckBoxMouseMove.TabIndex = 7;
CheckBoxMouseMove.Text = "Mouse Move Events";

CheckBoxMousePoints.AutoSize = true;
CheckBoxMousePoints.Location = new Point(26, 112);
CheckBoxMousePoints.Margin = new Padding(3, 3, 3, 1);
CheckBoxMousePoints.Size = new Size(141, 17);
CheckBoxMousePoints.TabIndex = 8;
CheckBoxMousePoints.Text = "Draw Mouse Points";

CheckBoxMouseDrag.AutoSize = true;
CheckBoxMouseDrag.Location = new Point(26, 135);
CheckBoxMouseDrag.Margin = new Padding(3, 1, 3, 3);
CheckBoxMouseDrag.Size = new Size(151, 17);
CheckBoxMouseDrag.TabIndex = 9;
CheckBoxMouseDrag.Text = "Mouse Drag && Drop Events";

CheckBoxMouseDragOver.AutoSize = true;
CheckBoxMouseDragOver.Location = new Point(44, 159);
CheckBoxMouseDragOver.Size = new Size(142, 17);
CheckBoxMouseDragOver.TabIndex = 10;
CheckBoxMouseDragOver.Text = "Mouse Drag Over Events";

CheckBoxKeyboard.AutoSize = true;
CheckBoxKeyboard.Location = new Point(8, 184);
CheckBoxKeyboard.Size = new Size(103, 17);
CheckBoxKeyboard.TabIndex = 11;
CheckBoxKeyboard.Text = "Keyboard Events";

CheckBoxKeyUpDown.AutoSize = true;
CheckBoxKeyUpDown.Location = new Point(26, 207);
CheckBoxKeyUpDown.Margin = new Padding(3, 3, 3, 1);
CheckBoxKeyUpDown.Size = new Size(133, 17);
CheckBoxKeyUpDown.TabIndex = 12;
CheckBoxKeyUpDown.Text = "Key Up && Down Events";

CheckBoxFocus.AutoSize = true;
CheckBoxFocus.Location = new Point(8, 233);
CheckBoxFocus.Margin = new Padding(3, 2, 3, 3);
CheckBoxFocus.Size = new Size(146, 17);
CheckBoxFocus.TabIndex = 13;
CheckBoxFocus.Text = "Focus && Activation Events";

CheckBoxValidation.AutoSize = true;
CheckBoxValidation.Location = new Point(8, 257);
CheckBoxValidation.Size = new Size(104, 17);
CheckBoxValidation.TabIndex = 14;
CheckBoxValidation.Text = "Validation Events";

TextBoxOutput.Location = new Point(232, 34);
TextBoxOutput.Size = new Size(308, 510);
TextBoxOutput.Multiline = true;
TextBoxOutput.CausesValidation = false;
TextBoxOutput.ReadOnly = true;
TextBoxOutput.ScrollBars = ScrollBars.Vertical;
```

```

TextBoxOutput.TabIndex = 15;
TextBoxOutput.WordWrap = false;

ButtonClear.Location = new Point(232, 560);
ButtonClear.Size = new Size(308, 23);
ButtonClear.TabIndex = 16;
ButtonClear.Text = "Clear Event List";

this.ClientSize = new Size(552, 595);
this.Controls.Add(LinkLabelDrag);
this.Controls.Add(ButtonClear);
this.Controls.Add(GroupBoxEvents);
this.Controls.Add(Label1);
this.Controls.Add(Label2);
this.Controls.Add(TextBoxInput);
this.Controls.Add(TextBoxOutput);
this.Text = "User Input Events";

ButtonClear.Click +=
    new EventHandler(ButtonClear_Click);
TextBoxInput.KeyDown +=
    new KeyEventHandler(TextBoxInput_KeyDown);
TextBoxInput.KeyPress +=
    new KeyPressEventHandler(TextBoxInput_KeyPress);
TextBoxInput.KeyUp +=
    new KeyEventHandler(TextBoxInput_KeyUp);
TextBoxInput.Click +=
    new EventHandler(TextBoxInput_Click);
TextBoxInput.DoubleClick +=
    new EventHandler(TextBoxInput_DoubleClick);
TextBoxInput.MouseClick +=
    new MouseEventArgs(TextBoxInput_MouseClick);
TextBoxInput.MouseDoubleClick +=
    new MouseEventArgs(TextBoxInput_MouseDoubleClick);
TextBoxInput.MouseDown +=
    new MouseEventArgs(TextBoxInput_MouseDown);
TextBoxInput.MouseUp +=
    new MouseEventArgs(TextBoxInput_MouseUp);
TextBoxInput.MouseEnter +=
    new EventHandler(TextBoxInput_MouseEnter);
TextBoxInput.MouseHover +=
    new EventHandler(TextBoxInput_MouseHover);
TextBoxInput.MouseLeave +=
    new EventHandler(TextBoxInput_MouseLeave);
TextBoxInput.MouseWheel +=
    new MouseEventArgs(TextBoxInput_MouseWheel);
TextBoxInput.MouseMove +=
    new MouseEventArgs(TextBoxInput_MouseMove);
TextBoxInput.MouseCaptureChanged +=
    new EventHandler(TextBoxInput_MouseCaptureChanged);
TextBoxInput.DragEnter +=
    new DragEventHandler(TextBoxInput_DragEnter);
TextBoxInput.DragDrop +=
    new DragEventHandler(TextBoxInput_DragDrop);
TextBoxInput.DragOver +=
    new DragEventHandler(TextBoxInput_DragOver);
TextBoxInput.DragLeave +=
    new EventHandler(TextBoxInput_DragLeave);
TextBoxInput.Enter +=
    new EventHandler(TextBoxInput_Enter);
TextBoxInput.Leave +=
    new EventHandler(TextBoxInput_Leave);
TextBoxInput.GotFocus +=
    new EventHandler(TextBoxInput_GotFocus);
TextBoxInput.LostFocus +=
    new EventHandler(TextBoxInput_LostFocus);
TextBoxInput.Validated +=
    new EventHandler(TextBoxInput_Validated);
TextBoxInput.Validating +=

```

```

        new CancelEventHandler(TextBoxInput_Validating);

LinkLabelDrag.MouseDown +=
    new MouseEventHandler(LinkLabelDrag_MouseDown);
LinkLabelDrag.GiveFeedback +=
    new GiveFeedbackEventHandler(LinkLabelDrag_GiveFeedback);

CheckBoxToggleAll.CheckedChanged +=
    new EventHandler(CheckBoxToggleAll_CheckedChanged);
CheckBoxMouse.CheckedChanged +=
    new EventHandler(CheckBoxMouse_CheckedChanged);
CheckBoxMouseDrag.CheckedChanged +=
    new EventHandler(CheckBoxMouseDrag_CheckedChanged);
CheckBoxMouseEnter.CheckedChanged +=
    new EventHandler(CheckBoxMouseMove_CheckedChanged);
CheckBoxMouseMove.CheckedChanged +=
    new EventHandler(CheckBoxMouseMove_CheckedChanged);
CheckBoxKeyboard.CheckedChanged +=
    new EventHandler(CheckBoxKeyboard_CheckedChanged);

this.GroupBoxEvents.ResumeLayout(false);
this.GroupBoxEvents.PerformLayout();
this.ResumeLayout(false);
this.PerformLayout();
CheckAllChildCheckBoxes(this, true);
}

// Recursively search the form for all contained checkboxes and
// initially check them
private void CheckAllChildCheckBoxes(Control parent, bool value)
{
    CheckBox box;
    foreach (Control currentControl in parent.Controls)
    {
        if (currentControl is CheckBox)
        {
            box = (CheckBox)currentControl;
            box.Checked = value;
        }

        // Recurse if control contains other controls
        if (currentControl.Controls.Count > 0)
        {
            CheckAllChildCheckBoxes(currentControl, value);
        }
    }
}

// All-purpose method for displaying a line of text in one of the
// text boxes.
private void DisplayLine(string line)
{
    TextBoxOutput.AppendText(line);
    TextBoxOutput.AppendText(Environment.NewLine);
}

// Click event handler for the button that clears the text box.
private void ButtonClear_Click(object sender, EventArgs e)
{
    TextBoxOutput.Invalidate();
    TextBoxOutput.Clear();
}

private void TextBoxInput_KeyDown(object sender, KeyEventArgs e)
{
    if (CheckBoxKeyUpDown.Checked)
    {
        DisplayLine("KeyDown: " + e.KeyData.ToString());
    }
}

```



```

    }

    private void TextBoxInput_KeyUp(object sender, KeyEventArgs e)
    {
        if (CheckBoxKeyUpDown.Checked)
        {
            DisplayLine("KeyUp: " + e.KeyData.ToString());
        }
    }

    private void TextBoxInput_KeyPress(object sender,
        KeyPressEventArgs e)
    {
        if (CheckBoxKeyboard.Checked)
        {
            if (Char.IsWhiteSpace(e.KeyChar))
            {
                DisplayLine("KeyPress: WS");
            }
            else
            {
                DisplayLine("KeyPress: " + e.KeyChar.ToString());
            }
        }
    }

    private void TextBoxInput_Click(object sender, EventArgs e)
    {
        if (CheckBoxMouse.Checked)
        {
            DisplayLine("Click event");
        }
    }

    private void TextBoxInput_DoubleClick(object sender, EventArgs e)
    {
        if (CheckBoxMouse.Checked)
        {
            DisplayLine("DoubleClick event");
        }
    }

    private void TextBoxInput_MouseClick(object sender, MouseEventArgs e)
    {
        if (CheckBoxMouse.Checked)
        {
            DisplayLine("MouseClicked: " + e.Button.ToString() +
                " " + e.Location.ToString());
        }
    }

    private void TextBoxInput_MouseDoubleClick(object sender,
        MouseEventArgs e)
    {
        if (CheckBoxMouse.Checked)
        {
            DisplayLine("MouseDoubleClick: " + e.Button.ToString() +
                " " + e.Location.ToString());
        }
    }

    private void TextBoxInput_MouseDown(object sender,
        MouseEventArgs e)
    {
        if (CheckBoxMouse.Checked)
        {
            DisplayLine("MouseDown: " + e.Button.ToString() +
                " " + e.Location.ToString());
        }
    }

```

```

    }

private void TextBoxInput_MouseUp(object sender,
    MouseEventArgs e)
{
    if (CheckBoxMouse.Checked)
    {
        DisplayLine("MouseUp: " + e.Button.ToString() +
            " " + e.Location.ToString());
    }

    // The TextBox control was designed to change focus only on
    // the primary click, so force focus to avoid user confusion.
    if (!TextBoxInput.Focused)
    {
        TextBoxInput.Focus();
    }
}

private void TextBoxInput_MouseEnter(object sender, EventArgs e)
{
    if (CheckBoxMouseEnter.Checked)
    {
        DisplayLine("MouseEnter event");
    }
}

private void TextBoxInput_MouseHover(object sender, EventArgs e)
{
    if (CheckBoxMouseEnter.Checked)
    {
        DisplayLine("MouseHover event");
    }
}

private void TextBoxInput_MouseLeave(object sender, EventArgs e)
{
    if (CheckBoxMouseEnter.Checked)
    {
        DisplayLine("MouseLeave event");
    }
}

private void TextBoxInput_MouseWheel(object sender,
    MouseEventArgs e)
{
    if (CheckBoxMouse.Checked)
    {
        DisplayLine("MouseWheel: " + e.Delta.ToString() +
            " detents at " + e.Location.ToString());
    }
}

private void TextBoxInput_MouseMove(object sender,
    MouseEventArgs e)
{
    if (CheckBoxMouseMove.Checked)
    {
        DisplayLine("MouseMove: " + e.Button.ToString() + " " +
            e.Location.ToString());
    }

    if (CheckBoxMousePoints.Checked)
    {
        Graphics g = TextBoxInput.CreateGraphics();
        g.FillRectangle(Brushes.Black, e.Location.X,
            e.Location.Y, 1, 1);
        g.Dispose();
    }
}

```

```

    }
}

private void TextBoxInput_MouseCaptureChanged(object sender,
    EventArgs e)
{
    if (CheckBoxMouseDrag.Checked)
    {
        DisplayLine("MouseCaptureChanged event");
    }
}

private void TextBoxInput_DragEnter(object sender,
    DragEventArgs e)
{
    if (CheckBoxMouseDrag.Checked)
    {
        Point pt = new Point(e.X, e.Y);
        DisplayLine("DragEnter: " +
            CovertKeyStateToString(e.KeyState)
            + " at " + pt.ToString());
    }
}

private void TextBoxInput_DragDrop(object sender,
    DragEventArgs e)
{
    if (CheckBoxMouseDrag.Checked)
    {
        Point pt = new Point(e.X, e.Y);
        DisplayLine("DragDrop: " +
            CovertKeyStateToString(e.KeyState)
            + " at " + pt.ToString());
    }
}

private void TextBoxInput_DragOver(object sender,
    DragEventArgs e)
{
    if (CheckBoxMouseDragOver.Checked)
    {
        Point pt = new Point(e.X, e.Y);
        DisplayLine("DragOver: " +
            CovertKeyStateToString(e.KeyState)
            + " at " + pt.ToString());
    }

    // Allow if drop data is of type string.
    if (!e.Data.GetDataPresent(typeof(String)))
    {
        e.Effect = DragDropEffects.None;
    }
    else
    {
        e.Effect = DragDropEffects.Copy;
    }
}

private void TextBoxInput_DragLeave(object sender,
    EventArgs e)
{
    if (CheckBoxMouseDrag.Checked)
    {
        DisplayLine("DragLeave event");
    }
}

private string CovertKeyStateToString(int keyState)
{

```

```

        string keyString = "None";

        // Which button was pressed?
        if ((keyState & 1) == 1)
        {
            keyString = "Left";
        }
        else if ((keyState & 2) == 2)
        {
            keyString = "Right";
        }
        else if ((keyState & 16) == 16)
        {
            keyString = "Middle";
        }

        // Are one or more modifier keys also pressed?
        if ((keyState & 4) == 4)
        {
            keyString += "+SHIFT";
        }

        if ((keyState & 8) == 8)
        {
            keyString += "+CTRL";
        }

        if ((keyState & 32) == 32)
        {
            keyString += "+ALT";
        }

        return keyString;
    }

    private void TextBoxInput_Enter(object sender, EventArgs e)
    {
        if (CheckBoxFocus.Checked)
        {
            DisplayLine("Enter event");
        }
    }

    private void TextBoxInput_Leave(object sender, EventArgs e)
    {
        if (CheckBoxFocus.Checked)
        {
            DisplayLine("Leave event");
        }
    }

    private void TextBoxInput_GotFocus(object sender, EventArgs e)
    {
        if (CheckBoxFocus.Checked)
        {
            DisplayLine("GotFocus event");
        }
    }

    private void TextBoxInput_LostFocus(object sender, EventArgs e)
    {
        if (CheckBoxFocus.Checked)
        {
            DisplayLine("LostFocus event");
        }
    }

    private void TextBoxInput_Validated(object sender, EventArgs e)
    {

```

```

        if (CheckBoxValidation.Checked)
        {
            DisplayLine("Validated event");
        }
    }

    private void TextBoxInput_Validating(
        object sender, CancelEventArgs e)
    {
        if (CheckBoxValidation.Checked)
        {
            DisplayLine("Validating event");
        }
    }

    private void CheckBoxToggleAll_CheckedChanged(
        object sender, EventArgs e)
    {
        if (sender is CheckBox)
        {
            CheckAllChildCheckBoxes(this, ((CheckBox)sender).Checked);
        }
    }

    private void CheckBoxMouse_CheckedChanged(
        object sender, EventArgs e)
    {
        ConfigureCheckBoxSettings();
    }

    private void CheckBoxMouseDrag_CheckedChanged(
        object sender, EventArgs e)
    {
        ConfigureCheckBoxSettings();
    }

    private void CheckBoxKeyboard_CheckedChanged(
        object sender, EventArgs e)
    {
        ConfigureCheckBoxSettings();
    }

    private void CheckBoxMouseMove_CheckedChanged(
        object sender, EventArgs e)
    {
        ConfigureCheckBoxSettings();
    }

    // Reconcile dependencies between the check box
    // selection choices.
    private void ConfigureCheckBoxSettings()
    {
        // CheckBoxMouse is a top-level check box.
        if (!CheckBoxMouse.Checked)
        {
            CheckBoxMouseEnter.Enabled = false;
            CheckBoxMouseMove.Enabled = false;
            CheckBoxMouseDrag.Enabled = false;
            CheckBoxMouseDragOver.Enabled = false;
            CheckBoxMousePoints.Enabled = false;
        }
        else
        {
            CheckBoxMouseEnter.Enabled = true;
            CheckBoxMouseMove.Enabled = true;
            CheckBoxMouseDrag.Enabled = true;
            CheckBoxMousePoints.Enabled = true;

            // Enable children depending on the state of the parent.

```

```

        if (!CheckBoxMouseDrag.Checked)
        {
            CheckBoxMouseDragOver.Enabled = false;
        }
        else
        {
            CheckBoxMouseDragOver.Enabled = true;
        }
    }

    if (!CheckBoxKeyboard.Checked)
    {
        CheckBoxKeyUpDown.Enabled = false;
    }
    else
    {
        CheckBoxKeyUpDown.Enabled = true;
    }
}

private void LinkLabelDrag_MouseDown(object sender,
    MouseEventArgs e)
{
    string data = "Sample Data";
    LinkLabelDrag.DoDragDrop(data, DragDropEffects.All);
}

private void LinkLabelDrag_GiveFeedback(object sender,
    GiveFeedbackEventArgs e)
{
    if ((e.Effect & DragDropEffects.Copy) ==
        DragDropEffects.Copy)
    {
        LinkLabelDrag.Cursor = Cursors.HSplit;
    }
    else
    {
        LinkLabelDrag.Cursor = Cursors.Default;
    }
}
}
}

```

```

Imports System.Drawing
Imports System.ComponentModel
Imports System.Windows.Forms

Namespace UserInputWalkthrough

    Public Class Form1
        Inherits Form

        Dim Label1 As New Label
        Dim Label2 As New Label
        Dim TextBoxOutput As New TextBox
        Dim WithEvents TextBoxInput As New TextBox
        Dim WithEvents GroupBoxEvents As New GroupBox
        Dim WithEvents ButtonClear As New Button
        Dim WithEvents LinkLabelDrag As New LinkLabel

        Dim WithEvents CheckBoxToggleAll As New CheckBox
        Dim WithEvents CheckBoxMouse As New CheckBox
        Dim WithEvents CheckBoxMouseEnter As New CheckBox
        Dim WithEvents CheckBoxMouseMove As New CheckBox
        Dim WithEvents CheckBoxMousePoints As New CheckBox
        Dim WithEvents CheckBoxMouseDrag As New CheckBox
        Dim WithEvents CheckBoxMouseDragOver As New CheckBox
    End Class
End Namespace

```

```

Dim WithEvents CheckBoxKeyboard As New CheckBox
Dim WithEvents CheckBoxKeyUpDown As New CheckBox
Dim WithEvents CheckBoxFocus As New CheckBox
Dim WithEvents CheckBoxValidation As New CheckBox

<STAThread()> _
Shared Sub Main()
    Application.EnableVisualStyles()
    Application.Run(New Form1())
End Sub

Public Sub New()
    MyBase.New()
End Sub

Private Sub Form1_Load(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load

    Me.GroupBoxEvents.SuspendLayout()
    Me.SuspendLayout()

    Label1.Location = New Point(232, 12)
    Label1.Size = New Size(98, 14)
    Label1.AutoSize = True
    Label1.Text = "Generated Events:"

    Label2.Location = New Point(13, 12)
    Label2.Size = New Size(95, 14)
    Label2.AutoSize = True
    Label2.Text = "User Input Target:"

    TextBoxInput.Location = New Point(13, 34)
    TextBoxInput.Size = New Size(200, 200)
    TextBoxInput.AllowDrop = True
    TextBoxInput.AutoSize = False
    TextBoxInput.Cursor = Cursors.Cross
    TextBoxInput.Multiline = True
    TextBoxInput.TabIndex = 1

    LinkLabelDrag.AllowDrop = True
    LinkLabelDrag.AutoSize = True
    LinkLabelDrag.Location = New Point(13, 240)
    LinkLabelDrag.Size = New Size(175, 14)
    LinkLabelDrag.TabIndex = 2
    LinkLabelDrag.TabStop = True
    LinkLabelDrag.Text = "Click here to use as a drag source"
    LinkLabelDrag.Links.Add(New LinkLabel.Link(0, _
        LinkLabelDrag.Text.Length))

    GroupBoxEvents.Location = New Point(13, 281)
    GroupBoxEvents.Size = New Size(200, 302)
    GroupBoxEvents.TabIndex = 3
    GroupBoxEvents.TabStop = False
    GroupBoxEvents.Text = "Event Filter:"
    GroupBoxEvents.Controls.Add(CheckBoxMouseEnter)
    GroupBoxEvents.Controls.Add(CheckBoxToggleAll)
    GroupBoxEvents.Controls.Add(CheckBoxMousePoints)
    GroupBoxEvents.Controls.Add(CheckBoxKeyUpDown)
    GroupBoxEvents.Controls.Add(CheckBoxMouseDragOver)
    GroupBoxEvents.Controls.Add(CheckBoxMouseDrag)
    GroupBoxEvents.Controls.Add(CheckBoxValidation)
    GroupBoxEvents.Controls.Add(CheckBoxMouseMove)
    GroupBoxEvents.Controls.Add(CheckBoxFocus)
    GroupBoxEvents.Controls.Add(CheckBoxKeyboard)
    GroupBoxEvents.Controls.Add(CheckBoxMouse)

    CheckBoxToggleAll.AutoSize = True
    CheckBoxToggleAll.Location = New Point(7, 20)
    CheckBoxToggleAll.Size = New Size(122, 17)

```

```
CheckBoxToggleAll.TabIndex = 4
CheckBoxToggleAll.Text = "Toggle All Events"

CheckBoxMouse.AutoSize = True
CheckBoxMouse.Location = New Point(7, 45)
CheckBoxMouse.Size = New Size(137, 17)
CheckBoxMouse.TabIndex = 5
CheckBoxMouse.Text = "Mouse and Click Events"

CheckBoxMouseEnter.AutoSize = True
CheckBoxMouseEnter.Location = New Point(26, 69)
CheckBoxMouseEnter.Margin = New Padding(3, 3, 3, 1)
CheckBoxMouseEnter.Size = New System.Drawing.Size(151, 17)
CheckBoxMouseEnter.TabIndex = 6
CheckBoxMouseEnter.Text = "Mouse Enter/Hover/Leave"

CheckBoxMouseMove.AutoSize = True
CheckBoxMouseMove.Location = New Point(26, 89)
CheckBoxMouseMove.Margin = New Padding(3, 2, 3, 3)
CheckBoxMouseMove.Size = New Size(120, 17)
CheckBoxMouseMove.TabIndex = 7
CheckBoxMouseMove.Text = "Mouse Move Events"

CheckBoxMousePoints.AutoSize = True
CheckBoxMousePoints.Location = New Point(26, 112)
CheckBoxMousePoints.Margin = New Padding(3, 3, 3, 1)
CheckBoxMousePoints.Size = New Size(141, 17)
CheckBoxMousePoints.TabIndex = 8
CheckBoxMousePoints.Text = "Draw Mouse Points"

CheckBoxMouseDrag.AutoSize = True
CheckBoxMouseDrag.Location = New Point(26, 135)
CheckBoxMouseDrag.Margin = New Padding(3, 1, 3, 3)
CheckBoxMouseDrag.Size = New Size(151, 17)
CheckBoxMouseDrag.TabIndex = 9
CheckBoxMouseDrag.Text = "Mouse Drag && Drop Events"

CheckBoxMouseDragOver.AutoSize = True
CheckBoxMouseDragOver.Location = New Point(44, 159)
CheckBoxMouseDragOver.Size = New Size(142, 17)
CheckBoxMouseDragOver.TabIndex = 10
CheckBoxMouseDragOver.Text = "Mouse Drag Over Events"

CheckBoxKeyboard.AutoSize = True
CheckBoxKeyboard.Location = New Point(8, 184)
CheckBoxKeyboard.Size = New Size(103, 17)
CheckBoxKeyboard.TabIndex = 11
CheckBoxKeyboard.Text = "Keyboard Events"

CheckBoxKeyUpDown.AutoSize = True
CheckBoxKeyUpDown.Location = New Point(26, 207)
CheckBoxKeyUpDown.Margin = New Padding(3, 3, 3, 1)
CheckBoxKeyUpDown.Size = New Size(133, 17)
CheckBoxKeyUpDown.TabIndex = 12
CheckBoxKeyUpDown.Text = "Key Up && Down Events"

CheckBoxFocus.AutoSize = True
CheckBoxFocus.Location = New Point(8, 233)
CheckBoxFocus.Margin = New Padding(3, 2, 3, 3)
CheckBoxFocus.Size = New Size(146, 17)
CheckBoxFocus.TabIndex = 13
CheckBoxFocus.Text = "Focus && Activation Events"

CheckBoxValidation.AutoSize = True
CheckBoxValidation.Location = New Point(8, 257)
CheckBoxValidation.Size = New Size(104, 17)
CheckBoxValidation.TabIndex = 14
CheckBoxValidation.Text = "Validation Events"
```



```

        TextBoxOutput.Location = New Point(232, 34)
        TextBoxOutput.Size = New Size(308, 510)
        TextBoxOutput.Multiline = True
        TextBoxOutput.CausesValidation = False
        TextBoxOutput.ReadOnly = True
        TextBoxOutput.ScrollBars = ScrollBars.Vertical
        TextBoxOutput.TabIndex = 15
        TextBoxOutput.WordWrap = False

        ButtonClear.Location = New Point(232, 560)
        ButtonClear.Size = New Size(308, 23)
        ButtonClear.TabIndex = 16
        ButtonClear.Text = "Clear Event List"

        Me.ClientSize = New Size(552, 595)
        Me.Controls.Add(LinkLabelDrag)
        Me.Controls.Add(ButtonClear)
        Me.Controls.Add(GroupBoxEvents)
        Me.Controls.Add(Label1)
        Me.Controls.Add(Label2)
        Me.Controls.Add(TextBoxInput)
        Me.Controls.Add(TextBoxOutput)
        Me.Text = "User Input Events"

        Me.GroupBoxEvents.ResumeLayout(False)
        Me.GroupBoxEvents.PerformLayout()
        Me.ResumeLayout(False)
        Me.PerformLayout()
        CheckAllChildCheckBoxes(Me, True)
End Sub

' Recursively search the form for all contained checkboxes and
' initially check them
Private Sub CheckAllChildCheckBoxes(ByRef parent As Control, _
    ByVal value As Boolean)

    Dim currentControl As Control
    Dim box As CheckBox
    For Each currentControl In parent.Controls
        box = TryCast(currentControl, CheckBox)
        If box IsNot Nothing Then
            box.Checked = value
        End If

        'Recurse if control contains other controls
        If currentControl.Controls.Count > 0 Then
            CheckAllChildCheckBoxes(currentControl, value)
        End If
    Next
End Sub

' All-purpose method for displaying a line of text in one of the
' text boxes.
Private Sub DisplayLine(ByRef line As String)

    TextBoxOutput.AppendText(line)
    TextBoxOutput.AppendText(ControlChars.NewLine)
End Sub

' Click event handler for the button that clears the text box.
Private Sub ButtonClear_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles ButtonClear.Click

    TextBoxOutput.Invalidate()
    TextBoxOutput.Clear()
End Sub

Private Sub TextBoxInput_KeyDown(ByVal sender As Object, _
    ByVal e As KeyEventArgs) Handles TextBoxInput.KeyDown

```

```

        ByVal e As KeyEventArgs) Handles TextBoxInput.KeyDown

        If CheckBoxKeyUpDown.Checked = True Then
            DisplayLine("KeyDown: " + e.KeyData.ToString())
        End If
    End Sub

    Private Sub TextBoxInput_KeyUp(ByVal sender As Object, _
        ByVal e As KeyEventArgs) Handles TextBoxInput.KeyUp

        If CheckBoxKeyUpDown.Checked = True Then
            DisplayLine("KeyUp: " + e.KeyData.ToString())
        End If
    End Sub

    Private Sub TextBoxInput_KeyPress(ByVal sender As Object, _
        ByVal e As KeyPressEventArgs) Handles TextBoxInput.KeyPress

        If CheckBoxKeyboard.Checked = True Then

            If Char.IsWhiteSpace(e.KeyChar) Then
                Dim keyVal As Integer
                keyVal = AscW(e.KeyChar)
                DisplayLine("KeyPress: WS-" + keyVal.ToString())
            Else
                DisplayLine("KeyPress: " + e.KeyChar.ToString())
            End If
        End If
    End Sub

    Private Sub TextBoxInput_Click(ByVal sender As Object, _
        ByVal e As EventArgs) Handles TextBoxInput.Click

        If CheckBoxMouse.Checked = True Then
            DisplayLine("Click event")
        End If
    End Sub

    Private Sub TextBoxInput_DoubleClick(ByVal sender As Object, _
        ByVal e As EventArgs) Handles TextBoxInput.DoubleClick

        If CheckBoxMouse.Checked = True Then
            DisplayLine("DoubleClick event")
        End If
    End Sub

    Private Sub TextBoxInput_MouseClick(ByVal sender As Object, _
        ByVal e As MouseEventArgs) Handles TextBoxInput.MouseClick

        If CheckBoxMouse.Checked = True Then
            DisplayLine("MouseClicked: " + e.Button.ToString() + _
                " " + e.Location.ToString())
        End If
    End Sub

    Private Sub TextBoxInput_MouseDoubleClick(ByVal sender As Object, _
        ByVal e As MouseEventArgs) Handles TextBoxInput.MouseDoubleClick

        If CheckBoxMouse.Checked = True Then
            DisplayLine("MouseDoubleClick: " + e.Button.ToString() + _
                " " + e.Location.ToString())
        End If
    End Sub

    Private Sub TextBoxInput_MouseDown(ByVal sender As System.Object, _
        ByVal e As MouseEventArgs) Handles TextBoxInput.MouseDown

        If CheckBoxMouse.Checked = True Then
            DisplayLine("MouseDown: " + e.Button.ToString() + _
                " " + e.Location.ToString())
        End If
    End Sub

```

```

        "" + e.Location.ToString())
    End If
End Sub

Private Sub TextBoxInput_MouseUp(ByVal sender As Object, _
    ByVal e As MouseEventArgs) Handles TextBoxInput.MouseUp

    If CheckBoxMouse.Checked = True Then
        DisplayLine("MouseUp: " + e.Button.ToString() + _
            "" + e.Location.ToString())
    End If

    ' The TextBox control was designed to change focus only on
    ' the primary click, so force focus to avoid user confusion.
    If TextBoxInput.Focused = False Then
        TextBoxInput.Focus()
    End If
End Sub

Private Sub TextBoxInput_MouseEnter(ByVal sender As Object, _
    ByVal e As EventArgs) Handles TextBoxInput.MouseEnter

    If CheckBoxMouseEnter.Checked = True Then
        DisplayLine("MouseEnter event")
    End If
End Sub

Private Sub TextBoxInput_MouseHover(ByVal sender As Object, _
    ByVal e As EventArgs) Handles TextBoxInput.MouseHover

    If CheckBoxMouseEnter.Checked = True Then
        DisplayLine("MouseHover event")
    End If
End Sub

Private Sub TextBoxInput_MouseLeave(ByVal sender As Object, _
    ByVal e As EventArgs) Handles TextBoxInput.MouseLeave

    If CheckBoxMouseEnter.Checked = True Then
        DisplayLine("MouseLeave event")
    End If
End Sub

Private Sub TextBoxInput_MouseWheel(ByVal sender As Object, _
    ByVal e As MouseEventArgs) Handles TextBoxInput.MouseWheel

    If CheckBoxMouse.Checked = True Then
        DisplayLine("MouseWheel: " + e.Delta.ToString() + _
            "" detents at " + e.Location.ToString())
    End If
End Sub

Private Sub TextBoxInput_MouseMove(ByVal sender As Object, _
    ByVal e As MouseEventArgs) Handles TextBoxInput.MouseMove

    If CheckBoxMouseMove.Checked = True Then
        DisplayLine("MouseMove: " + e.Button.ToString() + " " + _
            e.Location.ToString())
    End If

    If CheckBoxMousePoints.Checked = True Then
        Dim g As Graphics = TextBoxInput.CreateGraphics()
        g.FillRectangle(Brushes.Black, e.Location.X, _
            e.Location.Y, 1, 1)
        g.Dispose()
    End If
End Sub

Private Sub TextBoxInput_MouseCaptureChanged( _

```

```

    ByVal sender As Object, ByVal e As EventArgs) _
    Handles TextBoxInput.MouseCaptureChanged

    If CheckBoxMouseDrag.Checked = True Then
        DisplayLine("MouseCaptureChanged event")
    End If
End Sub

Private Sub TextBoxInput_DragEnter(ByVal sender As Object, _
    ByVal e As DragEventArgs) Handles TextBoxInput.DragEnter

    Dim pt As Point
    If CheckBoxMouseDrag.Checked = True Then
        pt = New Point(e.X, e.Y)
        DisplayLine("DragEnter: " + _
            CovertKeyStateToString(e.KeyState) _
            + " at " + pt.ToString())
    End If
End Sub

Private Sub TextBoxInput_DragDrop(ByVal sender As Object, _
    ByVal e As DragEventArgs) Handles TextBoxInput.DragDrop

    If CheckBoxMouseDrag.Checked = True Then
        Dim pt As Point
        pt = New Point(e.X, e.Y)

        DisplayLine("DragDrop: " + _
            CovertKeyStateToString(e.KeyState) _
            + " at " + pt.ToString())
    End If
End Sub

Private Sub TextBoxInput_DragOver(ByVal sender As Object, _
    ByVal e As DragEventArgs) Handles TextBoxInput.DragOver

    If CheckBoxMouseDragOver.Checked = True Then
        Dim pt As Point
        pt = New Point(e.X, e.Y)
        DisplayLine("DragOver: " + _
            CovertKeyStateToString(e.KeyState) _
            + " at " + pt.ToString())
    End If

    ' Allow if drop data is of type string.
    If Not (e.Data.GetDataPresent(GetType(String))) Then
        e.Effect = DragDropEffects.None
    Else
        e.Effect = DragDropEffects.Copy
    End If
End Sub

Private Sub TextBoxInput_DragLeave(ByVal sender As Object, _
    ByVal e As EventArgs) Handles TextBoxInput.DragLeave

    If CheckBoxMouseDrag.Checked = True Then
        DisplayLine("DragLeave event")
    End If
End Sub

Private Function CovertKeyStateToString(ByVal keyState As Integer) _
    As String

    Dim keyString As String = "None"

    ' Which button was pressed?
    If ((keyState And 1) = 1) Then
        keyString = "Left"
    ElseIf ((keyState And 2) = 2) Then

```

```

        keyString = "Right"
    ElseIf ((keyState And 16) = 16) Then
        keyString = "Middle"
    End If

    ' Are one or more modifier keys also pressed?
    If ((keyState And 4) = 4) Then
        keyString += "+SHIFT"
    End If
    If ((keyState And 8) = 8) Then
        keyString += "+CTRL"
    End If
    If ((keyState And 32) = 32) Then
        keyString += "+ALT"
    End If

    Return keyString
End Function

Private Sub TextBoxInput_Enter(ByVal sender As Object, _
    ByVal e As EventArgs) Handles TextBoxInput.Enter

    If CheckBoxFocus.Checked = True Then
        DisplayLine("Enter event")
    End If
End Sub

Private Sub TextBoxInput_Leave(ByVal sender As Object, _
    ByVal e As EventArgs) Handles TextBoxInput.Leave

    If CheckBoxFocus.Checked = True Then
        DisplayLine("Leave event")
    End If
End Sub

Private Sub TextBoxInput_GotFocus(ByVal sender As Object, _
    ByVal e As EventArgs) Handles TextBoxInput.GotFocus

    If CheckBoxFocus.Checked = True Then
        DisplayLine("GotFocus event")
    End If
End Sub

Private Sub TextBoxInput_LostFocus(ByVal sender As Object, _
    ByVal e As EventArgs) Handles TextBoxInput.LostFocus

    If CheckBoxFocus.Checked = True Then
        DisplayLine("LostFocus event")
    End If
End Sub

Private Sub TextBoxInput_Validated(ByVal sender As Object, _
    ByVal e As EventArgs) Handles TextBoxInput.Validated

    If CheckBoxValidation.Checked = True Then
        DisplayLine("Validated event")
    End If
End Sub

Private Sub TextBoxInput_Validating(ByVal sender As Object, _
    ByVal e As CancelEventArgs) _
    Handles TextBoxInput.Validating

    If CheckBoxValidation.Checked = True Then
        DisplayLine("Validating event")
    End If
End Sub

Private Sub CheckBoxToggleAll_CheckedChanged( _

```

```

ByVal sender As Object, ByVal e As EventArgs) _
Handles CheckBoxToggleAll.CheckedChanged

Dim cb As CheckBox
cb = TryCast(sender, CheckBox)
If cb IsNot Nothing Then
    CheckAllChildCheckBoxes(Me, cb.Checked)
End If
End Sub

Private Sub CheckBoxMouse_CheckedChanged( _
    ByVal sender As Object, ByVal e As EventArgs) _
Handles CheckBoxMouse.CheckedChanged

    ConfigureCheckBoxSettings()
End Sub

Private Sub CheckBoxMouseDrag_CheckedChanged( _
    ByVal sender As Object, ByVal e As EventArgs) _
Handles CheckBoxMouseDrag.CheckedChanged

    ConfigureCheckBoxSettings()
End Sub

Private Sub CheckBoxKeyboard_CheckedChanged( _
    ByVal sender As Object, ByVal e As EventArgs) _
Handles CheckBoxKeyboard.CheckedChanged

    ConfigureCheckBoxSettings()
End Sub

Private Sub CheckBoxMouseMove_CheckedChanged( _
    ByVal sender As Object, ByVal e As EventArgs) _
Handles CheckBoxMouseEnter.CheckedChanged, _
    CheckBoxMouseMove.CheckedChanged

    ConfigureCheckBoxSettings()
End Sub

' Reconcile dependencies between the check box
' selection choices.
Private Sub ConfigureCheckBoxSettings()

    ' CheckBoxMouse is a top-level check box.
    If CheckBoxMouse.Checked = False Then
        CheckBoxMouseEnter.Enabled = False
        CheckBoxMouseMove.Enabled = False
        CheckBoxMouseDrag.Enabled = False
        CheckBoxMouseDragOver.Enabled = False
        CheckBoxMousePoints.Enabled = False
    Else
        CheckBoxMouseEnter.Enabled = True
        CheckBoxMouseMove.Enabled = True
        CheckBoxMouseDrag.Enabled = True
        CheckBoxMousePoints.Enabled = True

        ' Enable children depending on the state of the parent.
        If CheckBoxMouseDrag.Checked = False Then
            CheckBoxMouseDragOver.Enabled = False
        Else
            CheckBoxMouseDragOver.Enabled = True
        End If
    End If

    If CheckBoxKeyboard.Checked = False Then
        CheckBoxKeyUpDown.Enabled = False
    Else
        CheckBoxKeyUpDown.Enabled = True
    End If

```

```

End Sub

Private Sub LinkLabelDrag_MouseDown(ByVal sender As Object, _
    ByVal e As MouseEventArgs) Handles LinkLabelDrag.MouseDown

    Dim data As String = "Sample Data"
    LinkLabelDrag.DoDragDrop(data, DragDropEffects.All)
End Sub

Private Sub LinkLabelDrag_GiveFeedback(ByVal sender As Object, _
    ByVal e As GiveFeedbackEventArgs) _
    Handles LinkLabelDrag.GiveFeedback

    If ((e.Effect And DragDropEffects.Copy) = _
        DragDropEffects.Copy) Then
        LinkLabelDrag.Cursor = Cursors.HSplit
    Else
        LinkLabelDrag.Cursor = Cursors.Default
    End If
End Sub

End Class
End Namespace

```

## 코드 컴파일

이 예제에는 다음 사항이 필요합니다.

- System, System.Drawing 및 System.Windows.Forms 어셈블리에 대한 참조

## 참고 항목

- [Windows Forms에 사용자 입력](#)

# Windows Forms에서 사용자 입력 유효성 검사

2020-04-24 • 20 minutes to read • [Edit Online](#)

사용자가 응용 프로그램에 데이터를 입력할 때 응용 프로그램에서 데이터를 사용하기 전에 데이터가 유효한지 확인할 수 있습니다. 특정 텍스트 필드가 0길이가 아니거나, 필드가 전화 번호 또는 다른 형식의 데이터로 서식을 지정하거나, 데이터베이스 보안을 손상시키는 데 사용할 수 있는 안전하지 않은 문자가 문자열에 포함되지 않도록 요구할 수 있습니다. Windows Forms는 응용 프로그램의 입력 유효성을 검사하는 여러 가지 방법을 제공합니다.

## 마스크된 텍스트 상자 컨트롤을 통한 유효성 검사

사용자가 전화 번호나 부품 번호와 같이 잘 정의된 형식으로 데이터를 입력하도록 요구하는 경우 컨트롤을 [MaskedTextBox](#) 사용하여 최소한의 코드로 이 작업을 신속하게 수행할 수 있습니다. 마스크는 텍스트 상자의 지정된 위치에서 입력할 수 있는 문자를 지정하는 마스킹 언어의 문자로 구성된 문자열입니다. 컨트롤은 사용자에게 프롬프트 집합을 표시합니다. 예를 들어 사용자가 잘못된 항목을 입력하는 경우(예: 숫자가 필요할 때 사용자가 문자를 입력하면 컨트롤이 자동으로 입력을 거부합니다.)

사용되는 마스킹 [MaskedTextBox](#) 언어는 매우 유연합니다. 필요한 문자, 선택적 문자, 하이픈 및 괄호, 통화 문자 및 날짜 구분 기호와 같은 리터럴 문자를 지정할 수 있습니다. 컨트롤은 데이터 원본에 바인딩할 때도 잘 작동합니다. 데이터 [Format](#) 바인딩의 이벤트는 마스크를 준수하기 위해 들어오는 데이터를 다시 포진하는 [Parse](#) 데 사용할 수 있으며, 이벤트는 데이터 필드의 사양을 준수하기 위해 나가는 데이터를 다시 포진하는 데 사용할 수 있습니다.

자세한 내용은 [마스크된 텍스트 상자 컨트롤](#)을 참조하십시오.

## 이벤트 기반 유효성 검사

유효성 검사에 대한 전체 프로그래밍 방식으로 제어하거나 복잡한 유효성 검사를 수행해야 하는 경우 대부분의 Windows Forms 컨트롤에 기본 제공된 유효성 검사 이벤트를 사용해야 합니다. 자유 형식 사용자 입력을 허용하는 각 [Validating](#) 컨트롤에는 컨트롤에 데이터 유효성 검사가 필요할 때마다 발생하는 이벤트가 있습니다. 이벤트 [Validating](#) 처리 메서드에서 여러 가지 방법으로 사용자 입력의 유효성을 검사할 수 있습니다. 예를 들어 우편 번호를 포함해야 하는 텍스트 상자가 있는 경우 다음과 같은 방법으로 유효성 검사를 수행할 수 있습니다.

- 우편 번호가 특정 우편 번호 그룹에 속해야 하는 경우 입력에서 문자열 비교를 수행하여 사용자가 입력한 데이터의 유효성을 검사할 수 있습니다. 예를 들어 우편 번호가 {10001, 10002, 10003} 집합에 있어야 하는 경우 문자열 비교를 사용하여 데이터의 유효성을 검사할 수 있습니다.
- 우편 번호가 특정 양식에 있어야 하는 경우 정규식을 사용하여 사용자가 입력한 데이터의 유효성을 검사할 수 있습니다. 예를 들어 품의 ##### 유효성을 검사하거나 #####-#### 정규식을 `^(\d{5})(-\d{4})?$` 사용할 수 있습니다. 품의 A#A #A# 유효성을 검사하려면 정규식을 `[A-Z]\d[A-Z] \d[A-Z]\d` 사용할 수 있습니다. 정규식에 대한 자세한 내용은 [.NET Framework 정규식](#) 및 [정규식 예제](#)를 참조하십시오.
- 우편 번호가 유효한 미국 우편 번호여야 하는 경우 우편 번호 웹 서비스를 호출하여 사용자가 입력한 데이터의 유효성을 검사할 수 있습니다.

이벤트는 [Validating](#) 형식의 [CancelEventArgs](#) 개체를 제공 합니다. 컨트롤의 데이터가 유효하지 않다고 판단되면 이 개체의 [Validating Cancel](#) 속성을 로 설정하여 이벤트를 `true` 취소할 수 있습니다. [Cancel](#) 속성을 설정하지 않으면 Windows Forms는 해당 컨트롤에 대해 유효성 검사가 [Validated](#) 성공했다고 가정하고 이벤트를 발생시킵니다.

에서 전자 메일 주소의 유효성을 [TextBox](#) 검사하는 [Validating](#) 코드 예제는 을 참조하십시오.

### 데이터 바인딩 및 이벤트 기반 유효성 검사

유효성 검사는 데이터베이스 테이블과 같은 데이터 원본에 컨트롤을 바인딩할 때 매우 유용합니다. 유효성 검사를 사용하면 컨트롤의 데이터가 데이터 원본에 필요한 형식을 정하고 안전하지 않을 수 있는 따옴표 및 백 슬래시와



같은 특수 문자를 포함하지 않도록 할 수 있습니다.

데이터 바인딩을 사용하면 **Validating** 컨트롤의 데이터가 이벤트 실행 중에 데이터 원본과 동기화됩니다.

**Validating** 이벤트를 취소하면 데이터가 데이터 원본과 동기화되지 않습니다.

#### IMPORTANT

**Validating** 이벤트 이후에 수행되는 사용자 지정 유효성 검사가 있는 경우 데이터 바인딩에 영향을 주지 않습니다. 예를 들어 데이터 바인딩을 **Validated** 취소하려고 시도하는 이벤트에 코드가 있는 경우 데이터 바인딩이 계속 발생합니다. 이 경우 **Validated** 이벤트에서 유효성 검사를 수행하려면 컨트롤의 **데이터 원본 업데이트 모드** 속성(데이터바인딩)\(고급)에서 **Onvalidation**에서 절대로 변경하고 유효성 검사 코드에 `<yourFIELD>* "].WriteValue() 제어를 .DataBindings[" 추가합니다.`

#### 암시적 및 명시적 유효성 검사

그렇다면 컨트롤의 데이터는 언제 검증됩니까? 이것은 개발자에게 달려 있습니다. 응용 프로그램의 요구 사항에 따라 암시적 또는 명시적 유효성 검사를 사용할 수 있습니다.

##### 암시적 유효성 검사

암시적 유효성 검사 방법은 사용자가 데이터를 입력할 때 데이터의 유효성을 검사합니다. 키가 누를 때 데이터를 읽거나 사용자가 입력 포커스를 한 컨트롤에서 멀리 가져와 다음 컨트롤로 이동할 때마다 데이터를 컨트롤에 입력할 때 데이터의 유효성을 검사할 수 있습니다. 이 방법은 사용자가 작업하는 데이터에 대한 즉각적인 피드백을 제공하려는 경우에 유용합니다.

컨트롤에 대한 암시적 유효성 검사를 사용하려면 해당 컨트롤의 **AutoValidate EnablePreventFocusChange** 속성을 **EnableAllowFocusChange** 또는 로 설정해야 합니다. **Validating** 이벤트를 취소하면 컨트롤의 동작은 **AutoValidate**에 할당된 값에 따라 결정됩니다. 할당된 **EnablePreventFocusChange** 경우 이벤트를 취소하면 **Validated** 이벤트가 발생하지 않습니다. 입력 포커스는 사용자가 데이터를 올바른 입력으로 변경할 때까지 현재 컨트롤에 유지됩니다. 을 지정하면 **EnableAllowFocusChange Validated** 이벤트를 취소할 때 이벤트가 발생하지 않지만 포커스는 다음 컨트롤로 계속 변경됩니다.

속성에 **Disable AutoValidate** 할당하면 암시적 유효성 검사가 완전히 방지됩니다. 컨트롤의 유효성을 검사하려면 명시적 유효성 검사를 사용해야 합니다.

##### 명시적 유효성 검사

명시적 유효성 검사 방법은 한 번에 데이터의 유효성을 검사합니다. 저장 단추 또는 다음 링크를 클릭하는 등 사용자 작업에 대한 응답으로 데이터의 유효성을 검사할 수 있습니다. 사용자 작업이 발생하면 다음 방법 중 하나로 명시적 유효성 검사를 트리거할 수 있습니다.

- 포커스가 손실된 마지막 컨트롤의 유효성을 검사하기 위해 호출합니다. **Validate**
- 폼 **ValidateChildren** 또는 컨테이너 컨트롤의 모든 자식 컨트롤의 유효성을 검사하기 위해 호출합니다.
- 사용자 지정 메서드를 호출하여 컨트롤의 데이터의 유효성을 수동으로 확인합니다.

#### Windows 양식 컨트롤에 대한 기본 암시적 유효성 검사 동작

Windows Forms 컨트롤마다 **AutoValidate** 해당 속성에 대한 기본값이 다릅니다. 다음 표에서는 가장 일반적인 컨트롤과 해당 기본값을 보여 주었습니다.

제어	기본 유효성 검사 동작
<b>ContainerControl</b>	<b>Inherit</b>
<b>Form</b>	<b>EnableAllowFocusChange</b>
<b>PropertyGrid</b>	비주얼 스튜디오에서 노출되지 않은 속성

제어	기본 유효성 검사 동작
<a href="#">ToolStripContainer</a>	비주얼 스튜디오에서 노출되지 않은 속성
<a href="#">SplitContainer</a>	<a href="#">Inherit</a>
<a href="#">UserControl</a>	<a href="#">EnableAllowFocusChange</a>

## 양식 닫기 및 유효성 검사 재정의

컨트롤에 포함된 데이터가 유효하지 않으므로 포커스를 유지 관리하는 경우 일반적인 방법 중 하나로 상위 양식을 닫을 수 없습니다.

- 닫기 버튼을 클릭합니다.
- 시스템 메뉴에서 닫기(close)를 선택합니다.
- 프로그래밍 방식으로 [Close](#) 메서드를 호출합니다.

그러나 경우에 따라 컨트롤의 값이 유효한지 여부에 관계없이 사용자가 폼을 닫도록 할 수 있습니다. 유효성 검사를 재정의하고 양식의 이벤트에 대한 처리기를 만들어 잘못된 데이터가 [FormClosing](#) 여전히 포함된 양식을 닫을 수 있습니다. 이 경우 [Cancel](#) 속성을 로 `false` 설정합니다. 이렇게 하면 폼이 닫힙니다. 자세한 내용과 예제는 [Form.FormClosing](#)를 참조하세요.

### NOTE

이러한 방식으로 양식을 강제로 닫으면 아직 저장되지 않은 양식 컨트롤의 모든 데이터가 손실됩니다. 또한 모달 양식은 컨트롤이 닫혀 있을 때 컨트롤의 내용을 확인하지 않습니다. 컨트롤 유효성 검사를 사용하여 컨트롤에 포커스를 잠글 수 있지만 폼 닫기와 관련된 동작에 대해 걱정할 필요는 없습니다.

## 참고 항목

- [Control.Validating](#)
- [Form.FormClosing](#)
- [System.Windows.Forms.FormClosingEventArgs](#)
- [MaskedTextBox](#) 컨트롤
- [정규 표현식 예제](#)

# Windows Forms 대화 상자

2020-02-03 • 2 minutes to read • [Edit Online](#)

대화 상자는 사용자와 상호 작용하고 정보를 검색하는 데 사용됩니다. 간단히 말해서 대화 상자는 [FormBorderStyle](#) 열거형 속성이 `FixedDialog`로 설정된 폼입니다. Visual Studio에서 Windows Forms 디자이너를 사용하여 사용자 지정 대화 상자를 직접 생성할 수 있습니다. `Label`, `Textbox` 및 `Button`과 같은 컨트롤을 추가하여 특정 필요에 맞게 대화 상자를 사용자 지정합니다. 이 .NET Framework에는 파일 열기 및 메시지 상자와 같이 사용자 고유의 응용 프로그램에 맞게 조정할 수 있는 미리 정의된 대화 상자도 포함되어 있습니다. 자세한 내용은 [대화 상자 컨트롤 및 구성 요소](#)를 참조 하세요.

## 섹션 내용

방법: [Windows Forms 대화 상자 표시](#)

대화 상자 표시에 대한 지침을 제공합니다.

## 관련 섹션

[대화 상자 컨트롤 및 구성 요소](#)

미리 정의된 대화 상자 컨트롤을 나열합니다.

[Windows Forms의 모양 변경](#)

Windows Forms 애플리케이션의 모양을 변경하는 방법에 대해 설명하는 항목의 링크가 들어 있습니다.

[TabControl 컨트롤 개요](#)

탭 컨트롤을 대화 상자에 통합하는 방법을 설명합니다.

# 방법: Windows Forms 대화 상자 표시

2020-03-21 • 2 minutes to read • [Edit Online](#)

응용 프로그램에 다른 양식을 표시하는 것과 동일한 방식으로 대화 상자를 표시합니다. 시작 양식은 응용 프로그램을 실행할 때 자동으로 로드됩니다. 응용 프로그램에 두 번째 양식 또는 대화 상자를 표시하려면 코드를 작성하여 로드하고 표시합니다. 마찬가지로 양식이나 대화 상자를 사라지게 하려면 코드를 작성하여 언로드하거나 숨깁니다.

대화 상자를 표시하려면

1. 대화 상자를 열려는 이벤트 처리기로 이동합니다. 메뉴 명령을 선택하거나 단추를 클릭할 때 또는 다른 이벤트가 발생할 때 발생할 수 있습니다.
2. 이벤트 처리기에서 코드를 추가하여 대화 상자를 엽니다. 이 예제에서는 단추 클릭 이벤트가 대화 상자를 표시하는 데 사용됩니다.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    Dim dlg1 as new Form()
    dlg1.ShowDialog()
End Sub
```

```
private void button1_Click(object sender, System.EventArgs e)
{
    Form dlg1 = new Form();
    dlg1.ShowDialog();
}
```

```
private:
void button1_Click(System::Object ^ sender,
    System::EventArgs ^ e)
{
    Form ^ dlg1 = gcnew Form();
    dlg1->ShowDialog();
}
```

# Windows Forms 데이터 바인딩

2020-02-03 • 6 minutes to read • [Edit Online](#)

Windows Forms의 데이터 바인딩은 폼의 컨트롤에서 데이터 소스의 정보를 표시하고 변경하는 방법을 제공합니다. 기존의 데이터 소스뿐 아니라 데이터를 포함하는 거의 모든 구조에 바인딩할 수 있습니다.

## 섹션 내용

### [데이터 바인딩 및 Windows Forms](#)

Windows Forms의 데이터 바인딩에 대한 개요를 제공합니다.

### [Windows Forms에서 지원하는 데이터 소스](#)

Windows Forms에서 사용할 수 있는 데이터 소스를 설명합니다.

### [데이터 바인딩과 관련된 인터페이스](#)

Windows Forms 데이터 바인딩과 함께 사용되는 여러 인터페이스를 설명합니다.

### [방법: Windows Forms에서 데이터 탐색](#)

데이터 소스의 항목을 탐색하는 방법을 보여 줍니다.

### [Windows Forms 데이터 바인딩의 변경 알림](#)

Windows Forms 데이터 바인딩에 대한 다양한 유형의 변경 알림을 설명합니다.

### [방법: INotifyPropertyChanged 인터페이스 구현](#)

[INotifyPropertyChanged](#) 인터페이스를 구현하는 방법을 보여 줍니다. 인터페이스는 비즈니스 개체의 속성 변경 내용을 바인딩된 컨트롤에 전달합니다.

### [방법: PropertyChanged 패턴 적용](#)

Windows Forms 사용자 정의 컨트롤의 속성에 *PropertyName*변경 패턴을 적용 하는 방법을 보여 줍니다.

### [방법: IListed 인터페이스 구현](#)

[IListed](#) 인터페이스를 구현하여 바인딩 가능한 목록에 대한 스키마 검색을 사용하는 방법을 보여 줍니다.

### [방법: IListSource 인터페이스 구현](#)

[IListSource](#) 인터페이스를 구현하여 [IList](#)를 구현하지 않지만 다른 위치에서 목록을 제공하는 바인딩 가능한 클래스를 만드는 방법을 보여 줍니다.

### [방법: 동일한 데이터 소스에 바인딩된 여러 컨트롤의 동기화 상태가 유지되도록 설정](#)

[BindingComplete](#) 이벤트를 처리하여 데이터 소스에 바인딩된 모든 컨트롤의 동기화 상태가 유지되도록 하는 방법을 보여 줍니다.

### [방법: 자식 테이블에서 선택된 행이 올바른 위치에 유지되도록 설정](#)

부모 테이블의 필드가 변경될 때 자식 테이블의 선택된 행이 변경되지 않도록 하는 방법을 보여 줍니다.

또한 데이터 바인딩과 관련 된 인터페이스, [방법: Windows Forms 데이터 탐색](#) 및 [방법: Windows Form에 단순 바인딩된 컨트롤 만들기](#)를 참조 하세요.

## 참조

### [System.Windows.Forms.Binding](#)

바인딩 가능한 구성 요소와 데이터 소스 간의 바인딩을 나타내는 클래스를 설명합니다.

### [System.Windows.Forms.BindingSource](#)

컨트롤에 바인딩하기 위해 데이터 소스를 캡슐화하는 클래스를 설명합니다.

## 관련 섹션

[BindingSource 구성 요소](#)

[BindingSource](#) 구성 요소를 사용하는 방법을 보여 주는 항목 목록을 포함합니다.

[DataGridView 컨트롤](#)

바인딩 가능한 datagrid 컨트롤을 사용하는 방법을 보여 주는 항목 목록을 제공합니다.

또한 [Visual Studio에서 데이터 액세스](#)를 참조하세요.

# 데이터 바인딩 및 Windows Forms

2020-02-03 • 13 minutes to read • [Edit Online](#)

Windows Forms에서는 기존의 데이터 소스뿐 아니라 데이터를 포함하는 거의 모든 구조에 바인딩할 수 있습니다. 런타임에 계산하거나 파일에서 읽거나 다른 컨트롤의 값에서 파생하는 값 배열에 바인딩할 수 있습니다.

또한 컨트롤의 속성을 데이터 소스에 바인딩할 수도 있습니다. 일반적인 데이터 바인딩에서는 대개 표시 속성(예: [Text](#) 컨트롤의 [TextBox](#) 속성)을 데이터 소스에 바인딩합니다. .NET Framework를 사용하여 바인딩을 통해서도 다른 속성을 설정할 수도 있습니다. 바인딩을 사용하여 수행할 수 있는 작업은 다음과 같습니다.

- 이미지 컨트롤의 그래픽 설정
- 컨트롤 하나 이상의 배경색 설정
- 컨트롤의 크기 설정

기본적으로 데이터 바인딩은 폼에 있는 컨트롤의 런타임 액세스 가능 속성을 자동으로 설정하는 방법입니다.

## 데이터 바인딩의 형식

Windows Forms는 단순 바인딩과 복합 바인딩의 두 가지 데이터 바인딩 형식을 사용할 수 있습니다. 각 바인딩 형식은 서로 다른 이점을 제공합니다.

데이터 바인딩 형식	DESCRIPTION
단순 데이터 바인딩	<p>컨트롤이 데이터 세트 테이블의 열 값과 같은 단일 데이터 요소에 바인딩하는 기능입니다. 이러한 형식의 바인딩은 일반적으로 <a href="#">TextBox</a> 컨트롤 또는 <a href="#">Label</a> 컨트롤과 같이 보통 단일 값만 표시하는 컨트롤에 사용됩니다. 실제로 컨트롤의 모든 속성은 데이터베이스의 필드에 바인딩할 수 있습니다. Visual Studio에서 이 기능에 대해 폭넓게 지원이 됩니다.</p> <p>자세한 내용은 다음을 참조하세요.</p> <p><a href="#">데이터 바인딩과 관련 된 - 인터페이스</a></p> <ul style="list-style-type: none"><li>- 방법: <a href="#">Windows Forms 데이터 탐색</a></li><li>- 방법: <a href="#">Windows Form에 단순 바인딩된 컨트롤 만들기</a></li></ul>
복합 데이터 바인딩	<p>둘 이상의 데이터 요소(일반적으로 데이터베이스 내 둘 이상의 레코드)에 바인딩하는 컨트롤의 기능입니다. 복합 바인딩은 목록 기반 바인딩이라고도 합니다. 복합 바인딩을 지원하는 컨트롤의 예로는 <a href="#">DataGridView</a>, <a href="#">ListBox</a> 및 <a href="#">ComboBox</a> 컨트롤이 있습니다. 복합 데이터 바인딩의 예제는 <a href="#">방법: 데이터에 Windows Forms ComboBox 또는 ListBox 컨트롤 바인딩을 참조 하세요.</a></p>

## BindingSource 구성 요소

Windows Forms에서는 데이터 바인딩을 간소화하기 위해 데이터 소스를 [BindingSource](#) 구성 요소에 바인딩한 다음 컨트롤을 [BindingSource](#)에 바인딩할 수 있습니다. 단순 또는 복합 바인딩 시나리오에서 [BindingSource](#)를 사용할 수 있습니다. 두 경우 모두 [BindingSource](#)는 데이터 소스와 바인딩되는 컨트롤 간의 매개 역할을 하며 변경 알림, 현재 항목 관리 및 기타 서비스를 제공합니다.

## 데이터 바인딩을 사용하는 일반적인 시나리오

거의 모든 상업용 애플리케이션에서는 대개 데이터 바인딩을 통해 특정 형식의 데이터 소스에서 읽은 정보를 사용합니다. 다음 목록에는 데이터 표시 및 조작 방법으로 데이터 바인딩을 사용하는 가장 일반적인 몇 가지 시나리오가 나와 있습니다.

시나리오	DESCRIPTION
보고	보고서를 사용하면 데이터를 인쇄된 문서에 유동적으로 표시 및 요약할 수 있습니다. 일반적으로는 데이터 소스의 선택한 콘텐츠를 화면이나 프린터에 인쇄하는 보고서를 만듭니다. 일반적인 보고서에는 목록, 송장, 요약 등이 포함됩니다. 항목은 보통 목록의 열로 서식이 지정되며 각 목록 항목 아래에는 하위 항목이 구성되지만 데이터에 가장 적합한 레이아웃을 선택해야 합니다.
데이터 입력	일반적으로는 데이터 입력 양식을 통해 많은 양의 관련 데이터를 입력하거나 사용자에게 정보를 표시합니다. 사용자는 텍스트 상자, 옵션 단추, 드롭다운 목록 및 확인란을 사용하여 정보를 입력하거나 항목을 선택할 수 있습니다. 그러면 정보가 전송되어 데이터베이스에 저장됩니다. 이 데이터베이스의 구조는 입력된 정보를 기반으로 합니다.
마스터/세부 관계	마스터/세부 애플리케이션은 관련된 데이터를 확인하기 위한 하나의 형식입니다. 이 응용 프로그램에는 두 데이터 테이블이 관계로 연결됩니다. 기존의 비즈니스 예제에서는 "Customers" 테이블과 "Orders" 테이블 간의 관계가 고객과 해당 주문을 연결합니다. 두 개의 Windows Forms <a href="#">DataGridView</a> 컨트롤을 사용하여 마스터/세부 응용 프로그램을 만드는 방법에 대한 자세한 내용은 <a href="#">방법: 두 개의 Windows Forms DataGridView 컨트롤을 사용하여 마스터/세부 폼 만들기</a> 를 참조하세요.
조회 테이블	또 다른 일반적인 데이터 표시/조작 시나리오는 테이블 조회입니다. <a href="#">ComboBox</a> 컨트롤은 대규모 데이터 표시의 일부으로 데이터를 표시하고 조작하는 경우가 많습니다. 여기서 중요한 점은, <a href="#">ComboBox</a> 컨트롤에 표시되는 데이터와 데이터베이스에 기록되는 데이터가 다르다는 점입니다. 예를 들어 식료품점에서 구입할 수 있는 품목을 표시하는 <a href="#">ComboBox</a> 컨트롤을 사용하는 경우에는 빵, 우유, 계란 등의 제품 이름을 표시하는 경우가 많습니다. 그러나 데이터베이스 내의 정보를 쉽게 검색하고 데이터베이스를 표준화하기 위해 지정된 주문의 특정 품목 정보를 #501, #603 등의 품목 번호로 저장할 수 있습니다. 이처럼 폼에 있는 <a href="#">ComboBox</a> 컨트롤의 식료품 "이름"과 주문에 포함된 관련 항목 번호 간에는 암시적 연결이 설정됩니다. 그리고 이러한 연결이 테이블 조회의 핵심 요소입니다. 자세한 내용은 <a href="#">방법: Windows Forms BindingSource 구성 요소를 사용하여 조회 테이블 만들기</a> 를 참조하세요.

## 참고 항목

- [Binding](#)
- [Windows Forms 데이터 바인딩](#)
- [방법: 데이터 소스에 Windows Forms DataGridView 컨트롤 바인딩](#)
- [BindingSource 구성 요소](#)



# Windows Forms에서 지원하는 데이터 소스

2020-02-03 • 11 minutes to read • [Edit Online](#)

일반적으로 데이터 바인딩은 응용 프로그램 내에서 데이터베이스에 저장된 데이터를 활용하는 데 사용되었습니다. Windows Forms 데이터 바인딩을 사용하면 최소한의 특정 요구 사항이 충족되는 한, 배열 및 컬렉션과 같은 다른 구조의 데이터 뿐만 아니라 데이터베이스의 데이터에 액세스할 수 있습니다.

## 바인딩할 구조체

Windows Forms에서는 단순 개체 (단순 바인딩)에서 ADO.NET data tables (복합 바인딩)와 같은 복합 목록으로 다양한 구조에 바인딩할 수 있습니다. 단순 바인딩의 경우 Windows Forms 단순 개체의 공용 속성에 대한 바인딩을 지원합니다. 목록 기반 바인딩에는 일반적으로 개체가 [IList](#) 인터페이스 또는 [IListSource](#) 인터페이스를 지원해야 합니다. 또한 [BindingSource](#) 구성 요소를 통해 바인딩하는 경우에는 [IEnumerable](#) 인터페이스를 지원하는 개체에 바인딩할 수 있습니다. 데이터 바인딩과 관련된 인터페이스에 대한 자세한 내용은 [데이터 바인딩과 관련된 인터페이스](#)를 참조하세요.

다음 목록에서는 Windows Forms에서 바인딩할 수 있는 구조를 보여 줍니다.

### BindingSource

[BindingSource](#)은 가장 일반적인 Windows Forms 데이터 소스이며, 데이터 소스와 Windows Forms 컨트롤 간의 프록시 역할을 합니다. 일반적인 [BindingSource](#) 사용 패턴은 [BindingSource](#)에 컨트롤을 바인딩하고

[BindingSource](#) 데이터 원본 (예: ADO.NET 데이터 테이블 또는 비즈니스 개체)에 바인딩하는 것입니다.

[BindingSource](#)는 데이터 바인딩 지원 수준을 설정하고 개선하는 서비스를 제공합니다. 예를 들어 [DataGridView](#) 및 [ComboBox](#)와 같은 Windows Forms 목록 기반 컨트롤은 [IEnumerable](#) 데이터 원본에 대한 바인딩을 직접 지원하지 않지만 [BindingSource](#)를 통해 바인딩하여 시나리오를 사용하도록 설정할 수 있습니다. 이 경우 [BindingSource](#)는 데이터 원본을 [IList](#) 변환합니다.

### 단순 개체

Windows Forms는 [Binding](#) 유형을 사용하여 개체 인스턴스의 공용 속성에 대한 데이터 바인딩 컨트롤 속성을 지원합니다. Windows Forms는 [BindingSource](#)를 사용할 때 개체 인스턴스에 대한 [ListControl](#)과 같은 목록 기반 컨트롤의 바인딩도 지원합니다.

### 배열 또는 컬렉션

데이터 소스 역할을 하려면 목록에서 [IList](#) 인터페이스를 구현해야 합니다. 한 가지 예는 [Array](#) 클래스의 인스턴스인 배열입니다. 배열에 대한 자세한 내용은 [방법: 개체 배열 만들기 \(Visual Basic\)](#)를 참조하세요.

일반적으로 데이터 바인딩을 위한 개체 목록을 만들 때 [BindingList<T>](#)를 사용해야 합니다. [BindingList<T>](#)은 [IBindingList](#) 인터페이스의 제네릭 버전입니다. [IBindingList](#) 인터페이스는 양방향 데이터 바인딩에 필요한 속성, 메서드 및 이벤트를 추가하여 [IList](#) 인터페이스를 확장합니다.

### IEnumerable

Windows Forms 컨트롤은 [BindingSource](#) 구성 요소를 통해 바인딩되는 경우에만 [IEnumerable](#) 인터페이스를 지원하는 데이터 소스에 바인딩할 수 있습니다.

### ADO.NET data 개체

ADO.NET에는 바인딩하는 데 적합한 여러 데이터 구조를 제공합니다. 각각은 복잡성과 복잡도에 따라 다릅니다.

- [DataColumn](#)입니다. [DataColumn](#)은 테이블을 구성하는 여러 열이 있는 [DataTable](#)의 필수 구성 요소입니다. 각 [DataColumn](#)에는 열이 보유하는 데이터의 종류를 결정하는 [DataType](#) 속성이 있습니다. 예를 들어 자동차를 설명하는 테이블의 자동차를 만듭니다. 컨트롤 (예: [TextBox](#) 컨트롤의 [Text](#) 속성)을 데이터 테이블 내의 열에 간단하게 바인딩할 수 있습니다.
- [DataTable](#)입니다. [DataTable](#)은 ADO.NET의 행과 열이 있는 테이블의 표현입니다. 데이터 테이블에는 두 개

의 컬렉션이 포함 되어 있습니다. 즉, 지정 된 테이블의 데이터 열 (궁극적으로는 해당 테이블에 입력할 수 있는 데이터의 종류를 결정 함)을 나타내고 지정 된 테이블의 데이터 행을 나타내는 [DataRow](#)를 [DataColumn](#)합니다. 데이터 테이블에 포함 된 정보에 컨트롤을 복잡 하게 바인딩할 수 있습니다. 예를 들어 [DataGridView](#) 컨트롤을 데이터 테이블에 바인딩할 수 있습니다. 그러나 [DataTable](#)에 바인딩하는 경우에는 테이블의 기본 뷰에 대 한 실제 바인딩입니다.

- [DataView](#)입니다. [DataView](#)은 필터링 하거나 정렬할 수 있는 단일 데이터 테이블의 사용자 지정 된 뷰입니다. 데이터 뷰는 복합 바인딩된 컨트롤에서 사용 하는 데이터 "스냅샷"입니다. 데이터 뷰 내에서 데이터를 단순 하게 바인딩하거나 복잡 하게 바인딩할 수 있지만 데이터 원본이 깨끗 하 고 업데이트 되는 것이 아니라 데이터의 고정 "그림"에 바인딩합니다.
- [DataSet](#)입니다. [DataSet](#)은 데이터베이스의 데이터에 대 한 테이블, 관계 및 제약 조건의 컬렉션입니다. 데이터 집합 내에서 데이터를 단순 하게 바인딩하거나 복잡 하게 바인딩할 수 있지만 [DataSet](#)의 기본 [DataViewManager](#)에 바인딩합니다 (다음 글머리 기호 참조).
- [DataViewManager](#)입니다. [DataViewManager](#)은 [DataView](#)와 유사 하지만 관계가 포함 된 전체 [DataSet](#)의 사용자 지정 된 뷰입니다. [DataViewSettings](#) 컬렉션을 사용 하여 지정 된 테이블에 대 한 [DataViewManager](#)에 있는 모든 뷰에 대해 기본 필터 및 정렬 옵션을 설정할 수 있습니다.

## 참고 항목

- [Windows Forms 데이터 바인딩의 변경 알림](#)
- [데이터 바인딩 및 Windows Forms](#)
- [Windows Forms 데이터 바인딩](#)

# 데이터 바인딩과 관련된 인터페이스

2019-11-23 • 25 minutes to read • [Edit Online](#)

ADO.NET를 사용 하면 응용 프로그램의 바인딩 요구 사항과 작업 중인 데이터에 맞게 다양 한 데이터 구조를 만들 수 있습니다. Windows Forms에서 데이터를 제공하거나 사용하는 고유 클래스를 만들고 싶을 수도 있습니다. 이러한 개체는 기본 데이터 바인딩부터 디자인 타임 지원, 오류 확인, 변경 사항 알림, 심지어 데이터 자체에 적용된 변경 내용의 구조화된 롤백 지원에 이르기까지 다양한 수준의 복합 기능을 제공할 수 있습니다.

## 데이터 바인딩 인터페이스의 소비자

다음 섹션에서는 인터페이스 개체의 두 그룹에 대해 설명 합니다. 첫 번째 그룹에서는 데이터 소스 작성자에 의해 데이터 소스에 구현된 인터페이스를 나열합니다. 이러한 인터페이스는 데이터 소스 소비자가 사용하며, 이러한 소비자의 대부분은 Windows Forms 컨트롤 또는 구성 요소입니다. 두 번째 그룹에서는 구성 요소 작성자를 위해 설계된 인터페이스를 나열합니다. 구성 요소 작성자는 Windows Forms 데이터 바인딩 엔진에서 사용할 데이터 바인딩을 지원하는 구성 요소를 만들 때 이러한 인터페이스를 사용합니다. 양식과 연결된 클래스 내에 이러한 인터페이스를 구현하여 데이터 바인딩을 사용할 수 있습니다. 각 경우에 데이터와의 상호 작용이 가능한 인터페이스를 구현하는 클래스가 제공됩니다. Visual Studio RAD (신속한 응용 프로그램 개발) 데이터 디자인 환경 도구는 이 기능을 이미 활용 하고 있습니다.

데이터 소스 작성자가 구현할 수 있는 인터페이스

다음 인터페이스는 Windows Forms 컨트롤에서 사용할 수 있도록 설계되었습니다.

- **ICollection** 인터페이스

**ICollection** 인터페이스를 구현 하는 클래스는 **Array**, **ArrayList**또는 **CollectionBase**일 수 있습니다. **Object**형식의 항목에 대한 인덱싱된 목록입니다. 인덱스의 첫 번째 항목에 따라 형식이 결정되기 때문에 이러한 목록에는 같은 형식이 포함되어 있어야 합니다. **ICollection**는 런타임에만 바인딩에 사용할 수 있습니다.

### NOTE

Windows Forms 바인딩할 비즈니스 개체 목록을 만들려는 경우 **BindingList<T>**를 사용 하는 것이 좋습니다. **BindingList<T>**은 양방향 Windows Forms 데이터 바인딩에 필요한 기본 인터페이스를 구현 하는 확장 가능한 클래스입니다.

- **IBindingList** 인터페이스

**IBindingList** 인터페이스를 구현 하는 클래스는 훨씬 높은 수준의 데이터 바인딩 기능을 제공 합니다. 이 구현은 목록 자체가 변경된 경우(예: 목록 항목 개수의 증가/감소)뿐 아니라 목록 항목이 변경된 경우(예: 고객 목록의 세 번째 항목에서 주소 필드가 변경됨)에도 기본적인 정렬 기능과 변경 알림을 제공합니다. 변경 알림은 여러 컨트롤을 같은 데이터에 바인딩할 때 한 컨트롤의 데이터 변경을 다른 바인딩된 컨트롤에 전파하려는 경우에 중요합니다.

### NOTE

**SupportsChangeNotification** 속성을 통해 **IBindingList** 인터페이스에 대해 변경 알림을 사용 하도록 설정 합니다. 이 속성은 `true` 경우 변경 된 목록이 나 목록의 항목이 변경 되었음을 나타내는 **ListChanged** 이벤트를 발생 시킵니다.

변경 형식은 **ListChangedEventArgs** 매개 변수의 **ListChangedType** 속성으로 설명 됩니다. 그러므로 데이터 모델이 업데이트될 때마다 같은 데이터 소스에 바인딩된 다른 컨트롤과 같은 모든 종속 뷰도 업데이트됨

니다. 그러나 목록에 포함 된 개체는 목록에서 [ListChanged](#) 이벤트를 발생 시킬 수 있도록 변경 될 때이를 알려야 합니다.

#### NOTE

[BindingList<T>](#)는 [IBindingList](#) 인터페이스의 제네릭 구현을 제공 합니다.

- [IBindingListView](#) 인터페이스

[IBindingListView](#) 인터페이스를 구현 하는 클래스는 필터링 및 고급 정렬 기능 뿐만 아니라 [IBindingList](#) 구현에 대 한 모든 기능을 제공 합니다. 이 구현은 문자열 기반 필터링 기능과 속성 설명자 방향 쌍을 사용한 여러 열 정렬 기능을 제공합니다.

- [IEditableObject](#) 인터페이스

[IEditableObject](#) 인터페이스를 구현 하는 클래스를 사용 하면 개체에서 해당 개체에 대 한 변경 내용이 영구적으로 적용 되는 시기를 제어할 수 있습니다. 이 구현에서는 [BeginEdit](#), [EndEdit](#) 및 [CancelEdit](#) 메서드를 사용하여 개체에 대 한 변경 내용을 롤백할 수 있습니다. 다음은 [BeginEdit](#), [EndEdit](#) 및 [CancelEdit](#) 방법의 기능에 대해 간략하게 설명 하고 서로 함께 작동 하여 데이터에 대 한 변경 내용을 롤백할 수 있도록 하는 방법입니다.

- [BeginEdit](#) 메서드는 개체에 대 한 편집을 시작 하도록 신호를 보냅니다. 이 인터페이스를 구현 하는 개체는 [CancelEdit](#) 메서드가 호출 되는 경우 업데이트를 삭제할 수 있는 방법으로 [BeginEdit](#) 메서드 호출 후 업데이트를 저장 해야 합니다. 데이터 바인딩 Windows Forms에서는 단일 편집 트랜잭션 (예: [BeginEdit](#), [BeginEdit](#), [EndEdit](#))의 범위 내에서 여러 번 [BeginEdit](#)를 호출할 수 있습니다. [IEditableObject](#) 구현은 [BeginEdit](#) 이미 호출 되었는지 여부를 추적 하고 [BeginEdit](#)에 대 한 후속 호출을 무시 해야 합니다. 이 메서드를 여러 번 호출할 수 있기 때문에 후속 호출에서 비파괴를 호출 하는 것이 중요 합니다. 즉, 이후의 [BeginEdit](#) 호출은 생성 된 업데이트를 제거 하거나 첫 번째 [BeginEdit](#) 호출에서 저장 된 데이터를 변경할 수 없습니다.
- 개체가 현재 편집 모드에 있는 경우 [EndEdit](#) 메서드는 [BeginEdit](#)를 내부 개체에 호출한 이후 변경 내용을 무시합니다.
- [CancelEdit](#) 메서드는 개체에 대 한 변경 내용을 모두 삭제 합니다.

[BeginEdit](#), [EndEdit](#) 및 [CancelEdit](#) 메서드가 작동 하는 방법에 대 한 자세한 내용은 [데이터베이스에 데이터 다시 저장](#)을 참조 하세요.

데이터 기능의이 트랜잭션 개념은 [DataGridView](#) 컨트롤에서 사용 됩니다.

- [ICancelAddNew](#) 인터페이스

[ICancelAddNew](#) 인터페이스를 구현 하는 클래스는 일반적으로 [IBindingList](#) 인터페이스를 구현 하며, [AddNew](#) 메서드를 사용 하여 추가 된 데이터 소스에 대 한 추가 작업을 롤백할 수 있습니다. 데이터 소스가 [IBindingList](#) 인터페이스를 구현 하는 경우에도 [ICancelAddNew](#) 인터페이스를 구현 해야 합니다.

- [IDataErrorInfo](#) 인터페이스

[IDataErrorInfo](#) 인터페이스를 구현 하는 클래스를 통해 개체가 바인딩된 컨트롤에 사용자 지정 오류 정보를 제공할 수 있습니다.

- [Error](#) 속성은 일반 오류 메시지 텍스트 (예: "오류가 발생 했습니다.")를 반환 합니다.
- [Item\[\]](#) 속성은 열에서 특정 오류 메시지를 포함 하는 문자열을 반환 합니다 (예: "[State](#) 열의 값이 잘못 되었습니다.").

- [IEnumerable](#) 인터페이스

[IEnumerable](#) 인터페이스를 구현 하는 클래스는 일반적으로 ASP.NET에서 사용 됩니다. 이 인터페이스에 대

한 Windows Forms 지원은 [BindingSource](#) 구성 요소를 통해서만 사용할 수 있습니다.

#### NOTE

[BindingSource](#) 구성 요소는 바인딩을 위해 모든 [IEnumerable](#) 항목을 별도의 목록에 복사 합니다.

- [ITypedList](#) 인터페이스

[ITypedList](#) 인터페이스를 구현 하는 [collections](#) 클래스는 바인딩된 컨트롤에 노출 되는 속성 집합 및 순서를 제어 하는 기능을 제공 합니다.

#### NOTE

[GetItemProperties](#) 메서드를 구현 하고 [PropertyDescriptor](#) 배열이 null이 아닌 경우 배열의 마지막 항목은 다른 항목 목록에 해당 하는 목록 속성을 설명 하는 속성 설명자가 됩니다.

- [ICustomTypeDescriptor](#) 인터페이스

[ICustomTypeDescriptor](#) 인터페이스를 구현 하는 클래스는 자체에 대 한 동적 정보를 제공 합니다. 이 인터페이스는 [ITypedList](#)와 비슷하지만 목록 대신 개체에 사용 됩니다. 이 인터페이스는 [DataRowView](#)에서 기본 행의 스키마를 프로젝션 하는 데 사용 됩니다. [ICustomTypeDescriptor](#)의 간단한 구현은 [CustomTypeDescriptor](#) 클래스에서 제공 합니다.

#### NOTE

[ICustomTypeDescriptor](#)를 구현 하는 형식에 대 한 디자인 타임 바인딩을 지원 하려면 형식도 [IComponent](#)를 구현 해야 하며 폼의 인스턴스로 존재 해야 합니다.

- [IListSource](#) 인터페이스

[IListSource](#) 인터페이스를 구현 하는 클래스는 목록에 없는 개체에 대 한 목록 기반 바인딩을 활성화 합니다. [IListSource](#)의 [GetList](#) 메서드는 [IList](#)에서 상속 되지 않는 개체에서 바인딩할 수 있는 목록을 반환 하는 데 사용 됩니다. [IListSource](#) 사용 되는 [DataSet](#) 클래스입니다.

- [IRaiseItemChangedEvents](#) 인터페이스

[IRaiseItemChangedEvents](#) 인터페이스를 구현 하는 클래스는 [IBindingList](#) 인터페이스를 구현 하는 바인딩 가능한 목록입니다. 이 인터페이스는 [RaisesItemChangedEvents](#) 속성을 통해 형식이 [ItemChanged](#) 형식의 [ListChanged](#) 이벤트를 발생 시키는 지 여부를 나타내는 데 사용 됩니다.

#### NOTE

데이터 소스가 위에 설명 된 이벤트 변환을 나열 하기 위해 속성을 제공 하고 [BindingSource](#) 구성 요소와 상호 작용 하는 경우에는 [IRaiseItemChangedEvents](#)를 구현 해야 합니다. 그렇지 않으면 [BindingSource](#)는 이벤트 변환을 나열 하는 속성을 수행 하 여 성능이 저하 됩니다.

- [ISupportInitialize](#) 인터페이스

[ISupportInitialize](#) 인터페이스를 구현 하는 구성 요소는 속성을 설정 하고 공동 종속 속성을 초기화 하기 위해 일괄 처리 최적화를 활용 합니다. [ISupportInitialize](#)는 두 가지 메서드를 포함 합니다.

- 개체 초기화가 시작 됨을 [BeginInit](#) 신호를 보냅니다.
- 개체 초기화를 완료 하는 [EndInit](#) 신호를 보냅니다.

- [ISupportInitializeNotification](#) 인터페이스

또한 `ISupportInitializeNotification` 인터페이스를 구현 하는 구성 요소는 `ISupportInitialize` 인터페이스를 구현 합니다. 이 인터페이스를 사용 하면 초기화가 완료 되었음을 다른 `ISupportInitialize` 구성 요소에 알릴 수 있습니다. `ISupportInitializeNotification` 인터페이스에는 두 개의 멤버가 포함 되어 있습니다.

- `IsInitialized`는 구성 요소가 초기화 되었는지 여부를 나타내는 `boolean` 값을 반환 합니다.
- `EndInit`를 호출 하면 `Initialized` 발생 합니다.

- `INotifyPropertyChanged` 인터페이스

이 인터페이스를 구현하는 클래스는 해당 속성 값이 변경될 때 이벤트를 발생시키는 형식입니다. 이 인터페이스는 각 컨트롤 속성에서 변경 이벤트를 갖는 패턴을 바꾸는 데 사용됩니다. `BindingList<T>`에서 사용하는 경우 비즈니스 개체는 `INotifyPropertyChanged` 인터페이스를 구현 해야 하고, `BindingList`는 `PropertyChanged` 이벤트를 `ListChanged` 형식의 이벤트로 변환 합니다. `ItemChanged`

#### NOTE

바인딩된 클라이언트와 데이터 소스 간의 바인딩에서 변경 알림이 발생 하도록 하려면 바인딩된 데이터 소스 형식이 `INotifyPropertyChanged` 인터페이스 (기본 설정)를 구현 하거나 바인딩 형식에 대해 `propertyName.Changed` 이벤트를 제공할 수 있지만 둘 다 수행할 수는 없습니다.

구성 요소 작성자가 구현할 수 있는 인터페이스

다음 인터페이스는 Windows Forms 데이터 바인딩 엔진에서 사용됩니다.

- `IBindableComponent` 인터페이스

이 인터페이스를 구현하는 클래스는 컨트롤이 아닌 구성 요소이며 데이터 바인딩을 지원합니다. 이 클래스는 이 인터페이스의 `DataBindings` 및 `BindingContext` 속성을 통해 구성 요소의 데이터 바인딩 및 바인딩 컨텍스트를 반환 합니다.

#### NOTE

구성 요소가 `Control`에서 상속 하는 경우 `IBindableComponent` 인터페이스를 구현할 필요가 없습니다.

- `ICurrencyManagerProvider` 인터페이스

`ICurrencyManagerProvider` 인터페이스를 구현 하는 클래스는 이 특정 구성 요소와 연결 된 바인딩을 관리 하는 자체 `CurrencyManager`를 제공 하는 구성 요소입니다. 사용자 지정 `CurrencyManager`에 대한 액세스 는 `CurrencyManager` 속성에 의해 제공 됩니다.

#### NOTE

`Control`에서 상속 하는 클래스는 `BindingContext` 속성을 통해 바인딩을 자동으로 관리 하므로 `ICurrencyManagerProvider`를 구현 해야 하는 경우가 매우 드뭅니다.

## 참고 항목

- 데이터 바인딩 및 Windows Forms
- 방법: Windows Form에 단순 바인딩된 컨트롤 만들기
- Windows Forms 데이터 바인딩

# Windows Forms 데이터 바인딩의 변경 알림

2020-02-03 • 7 minutes to read • [Edit Online](#)

Windows Forms 데이터 바인딩의 가장 중요 한 개념 중 하나는 *변경 알림*입니다. 데이터 원본 및 바인딩된 컨트롤에 항상 최신 데이터가 포함 되도록 하려면 데이터 바인딩에 대 한 변경 알림을 추가 해야 합니다. 특히 바인딩된 컨트롤이 데이터 소스에 적용 된 변경 내용에 대 한 알림 메시지를 표시 하 고 컨트롤의 바인딩된 속성에 적용 된 변경 내용에 대 한 알림이 데이터 소스에 표시 되도록 합니다.

데이터 바인딩의 종류에 따라 다음과 같은 다양 한 종류의 변경 알림이 있습니다.

- 단일 컨트롤 속성이 개체의 단일 인스턴스에 바인딩되는 단순 바인딩입니다.
- 목록에 있는 항목의 속성 또는 개체 목록에 바인딩된 컨트롤 속성에 바인딩된 단일 컨트롤 속성을 포함할 수 있는 목록 기반 바인딩입니다.

또한 데이터 바인딩에 사용 하려는 Windows Forms 컨트롤을 만드는 경우 컨트롤의 바인딩된 속성에 대 한 변경 내용이 데이터 소스에 전파 되도록 *PropertyName* 변경 패턴을 컨트롤에 적용 해야 합니다.

## 단순 바인딩에 대 한 변경 알림

단순 바인딩의 경우 비즈니스 개체는 바인딩된 속성 값이 변경 될 때 변경 알림을 제공 해야 합니다. 이렇게 하려면 비즈니스 개체의 각 속성에 대해 *PropertyNameChanged* 이벤트를 표시 하 고 비즈니스 개체를 [BindingSource](#) 또는 비즈니스 개체가 [INotifyPropertyChanged](#) 인터페이스를 구현 하는 기본 설정 방법으로 컨트롤에 바인딩한 다음 속성 값이 변경 될 때 *PropertyChanged* 이벤트를 발생 시킵니다. 자세한 내용은 [방법: INotifyPropertyChanged 인터페이스 구현](#)을 참조 하세요.

[INotifyPropertyChanged 인터페이스 구현](#)을 참조 하세요. [INotifyPropertyChanged](#) 인터페이스를 구현 하는 개체를 사용 하는 경우 [BindingSource](#)를 사용 하여 개체를 컨트롤에 바인딩할 필요가 없지만 [BindingSource](#)를 사용 하는 것이 좋습니다.

## 목록 기반 바인딩에 대 한 변경 알림

Windows Forms은 바인딩된 목록에 의존 하여 속성 변경 (목록 항목 속성 값 변경) 및 변경 된 목록 (항목을 삭제 하거나 목록에 추가) 정보를 바인딩된 컨트롤에 제공 합니다. 따라서 데이터 바인딩에 사용 되는 목록은 두 형식의 변경 알림을 모두 제공 하는 [IBindingList](#)를 구현 해야 합니다. [BindingList<T>](#)은 [IBindingList](#)의 제네릭 구현 이며 Windows Forms 데이터 바인딩과 함께 사용 하도록 설계 되었습니다. [INotifyPropertyChanged](#)를 구현 하는 비즈니스 개체 유형을 포함 하는 [BindingList<T>](#)를 만들 수 있으며,이 목록에서 *PropertyChanged* 이벤트를 *ListChanged* 이벤트로 자동으로 변환 합니다. 바인딩된 목록이 [IBindingList](#)아닌 경우 [BindingSource](#) 구성 요소를 사용 하여 개체 목록을 Windows Forms 컨트롤에 바인딩해야 합니다. [BindingSource](#) 구성 요소는 [BindingList<T>](#)와 유사한 속성-목록 변환 기능을 제공 합니다. 자세한 내용은 [방법: BindingSource와 INotifyPropertyChanged 인터페이스를 사용 하여 변경 알림 발생](#)을 참조 하세요.

## 사용자 지정 컨트롤에 대 한 변경 알림

마지막으로, 컨트롤 쪽에서 데이터에 바인딩하기 위해 디자인 된 각 속성에 대해 *PropertyNameChanged* 이벤트를 노출 해야 합니다. 그러면 컨트롤 속성에 대 한 변경 내용이 바인딩된 데이터 소스에 전파 됩니다. 자세한 내용은 [방법: PropertyChanged 패턴 적용](#)을 참조 하세요.

## 참고 항목

- [BindingSource](#)
- [INotifyPropertyChanged](#)
- [BindingList<T>](#)

- [Windows Forms 데이터 바인딩](#)
- [Windows Forms에서 지원하는 데이터 소스](#)
- [데이터 바인딩 및 Windows Forms](#)



# 방법: PropertyChanged 패턴 적용

2019-10-23 • 2 minutes to read • [Edit Online](#)

다음 코드 예제를 적용 하는 방법에 설명 합니다 *PropertyName* 패턴을 사용자 지정 컨트롤을 합니다. Windows Forms 데이터 바인딩 엔진에서 사용 되는 사용자 지정 컨트롤을 구현 하는 경우이 패턴을 적용 합니다.

## 예제

```

// This class implements a simple user control
// that demonstrates how to apply the propertyNameChanged pattern.
[ComplexBindingProperties("DataSource", "DataMember")]
public class CustomerControl : UserControl
{
    private DataGridView dataGridView1;
    private Label label1;
    private DateTime lastUpdate = DateTime.Now;

    public EventHandler DataSourceChanged;

    public object DataSource
    {
        get
        {
            return this.dataGridView1.DataSource;
        }
        set
        {
            if (DataSource != value)
            {
                this.dataGridView1.DataSource = value;
                OnDataSourceChanged();
            }
        }
    }

    public string DataMember
    {
        get { return this.dataGridView1.DataMember; }

        set { this.dataGridView1.DataMember = value; }
    }

    private void OnDataSourceChanged()
    {
        if (DataSourceChanged != null)
        {
            DataSourceChanged(this, new EventArgs());
        }
    }

    public CustomerControl()
    {
        this.dataGridView1 = new System.Windows.Forms.DataGridView();
        this.label1 = new System.Windows.Forms.Label();
        this.dataGridView1.ColumnHeadersHeightSizeMode =
            System.Windows.Forms.DataGridViewColumnHeadersHeightSizeMode.AutoSize;
        this.dataGridView1.ImeMode = System.Windows.Forms.ImeMode.Disable;
        this.dataGridView1.Location = new System.Drawing.Point(19, 55);
        this.dataGridView1.Size = new System.Drawing.Size(350, 150);
        this.dataGridView1.TabIndex = 1;
        this.label1.AutoSize = true;
        this.label1.Location = new System.Drawing.Point(19, 23);
        this.label1.Name = "label1";
        this.label1.Size = new System.Drawing.Size(76, 13);
        this.label1.TabIndex = 2;
        this.label1.Text = "Customer List:";
        this.Controls.Add(this.label1);
        this.Controls.Add(this.dataGridView1);
        this.Size = new System.Drawing.Size(350, 216);
    }
}

```

```

' This class implements a simple user control
' that demonstrates how to apply the propertyNameChanged pattern.
<ComplexBindingProperties("DataSource", "DataMember")> _
Public Class CustomerControl
    Inherits UserControl
    Private dataGridView1 As DataGridView
    Private label1 As Label
    Private lastUpdate As DateTime = DateTime.Now

    Public DataSourceChanged As EventHandler

    Public Property DataSource() As Object
        Get
            Return Me.dataGridView1.DataSource
        End Get
        Set
            If DataSource IsNot Value Then
                Me.dataGridView1.DataSource = Value
                OnDataSourceChanged()
            End If
        End Set
    End Property

    Public Property DataMember() As String
        Get
            Return Me.dataGridView1.DataMember
        End Get
        Set
            Me.dataGridView1.DataMember = value
        End Set
    End Property

    Private Sub OnDataSourceChanged()
        If (DataSourceChanged IsNot Nothing) Then
            DataSourceChanged(Me, New EventArgs())
        End If
    End Sub

    Public Sub New()
        Me.dataGridView1 = New System.Windows.Forms.DataGridView()
        Me.label1 = New System.Windows.Forms.Label()
        Me.dataGridView1.ColumnHeadersHeightSizeMode = _
            System.Windows.Forms.DataGridViewColumnHeadersHeightSizeMode.AutoSize
        Me.dataGridView1.ImeMode = System.Windows.Forms.ImeMode.Disable
        Me.dataGridView1.Location = New System.Drawing.Point(19, 55)
        Me.dataGridView1.Size = New System.Drawing.Size(350, 150)
        Me.dataGridView1.TabIndex = 1
        Me.label1.AutoSize = True
        Me.label1.Location = New System.Drawing.Point(19, 23)
        Me.label1.Name = "label1"
        Me.label1.Size = New System.Drawing.Size(76, 13)
        Me.label1.TabIndex = 2
        Me.label1.Text = "Customer List:"
        Me.Controls.Add(Me.label1)
        Me.Controls.Add(Me.dataGridView1)
        Me.Size = New System.Drawing.Size(350, 216)
    End Sub
End Class

```

이전 코드 예제를 컴파일하려면:

- 빈 코드 파일에 코드를 붙여 넣습니다. 포함 된 Windows Form에 사용자 지정 컨트롤을 사용 해야 하는 `Main` 메서드.

## 참고자료

- [방법: INotifyPropertyChanged 인터페이스 구현](#)
- [Windows Forms 데이터 바인딩의 변경 알림](#)
- [Windows Forms 데이터 바인딩](#)

# 방법: 바인딩된 컨트롤 만들기 및 표시된 데이터 서식 지정

2019-10-23 • 6 minutes to read • [Edit Online](#)

Windows Forms 데이터 바인딩을 사용 하면 서식 지정 및 고급 바인딩 대화 상자를 사용 하여 데이터 바인딩된 컨트롤에 표시 되는 데이터의 서식을 지정할 수 있습니다.

컨트롤을 바인딩하고 표시된 데이터의 서식을 지정하려면 다음을 수행합니다.

1. 데이터 소스에 연결합니다. 자세한 내용은 [데이터 원본에 연결](#)을 참조 하세요.
2. Visual Studio의 폼에서 컨트롤을 선택한 다음 속성 창을 엽니다.
3. (데이터 바인딩) 속성을 확장 한 다음 (고급) 상자에서 줄임표 단추 (...|)속성 창의 줄임표 단추 (...)를 클릭 하여 서식 지정 및 고급을 표시 합니다. 바인딩 대화 상자-해당 컨트롤에 대 한 속성의 전체 목록을 포함 합니다.
4. 바인딩하려는 속성을 선택 하고 바인딩 화살표를 선택 합니다.

사용 가능한 데이터 소스 목록이 표시됩니다.

5. 원하는 단일 데이터 요소를 찾을 때까지 바인딩할 데이터 소스를 확장합니다.

예를 들어 데이터 세트의 테이블에서 열 값에 바인딩할 경우 데이터 세트 이름을 확장하고 나서 테이블을 이름을 확장하여 열 이름을 표시합니다.

6. 바인딩할 요소의 이름을 선택 합니다.
7. 형식 형식 상자에서 컨트롤에 표시 되는 데이터에 적용 하려는 형식을 선택 합니다.

모든 경우에 데이터 소스에 DBNull이 있으면 컨트롤에 표시된 값을 지정할 수 있습니다. 그러지 않으면 선택한 형식 유형에 따라 옵션이 약간 달라집니다. 다음 표에서는 형식 유형 및 옵션을 보여 줍니다.

형식 유형	서식 지정 옵션
서식 없음	옵션 없음.
숫자	소수 자릿수 up-down 컨트롤을 사용 하여 소수 자릿수를 지정 합니다.
통화	소수 자릿수 up-down 컨트롤을 사용 하여 소수 자릿수를 지정 합니다.
날짜 시간	유형 선택 상자에서 항목 중 하나를 선택 하여 날짜 및 시간을 표시 하는 방법을 선택 합니다.
지수	소수 자릿수 up-down 컨트롤을 사용 하여 소수 자릿수를 지정 합니다.

형식 유형	서식 지정 옵션
사용자 지정	<p>사용자 지정 서식 문자열 사용을 지정합니다.</p> <p>자세한 내용은 <a href="#">서식 지정 형식</a>을 참조하세요. <b>참고:</b> 사용자 지정 서식 문자열은 데이터 소스와 바인딩된 컨트롤 간에 성공적으로 왕복된다고 보장할 수 없습니다. 대신에 바인딩에 대한 <a href="#">Parse</a> 또는 <a href="#">Format</a> 이벤트를 처리하고 이벤트 처리 코드에 사용자 지정 서식 지정을 적용합니다.</p>

8. **확인** 을 선택 하 여 서식 지정 및 고급 바인딩 대화 상자를 닫고 속성 창로 돌아갑니다.

## 참고자료

- 방법: [Windows Form에 단순 바인딩된 컨트롤 만들기](#)
- [Windows Forms에서 사용자 입력 유효성 검사](#)
- [Windows Forms 데이터 바인딩](#)

# 방법: Windows Form에 단순 바인딩된 컨트롤 만들기


2019-10-23 • 4 minutes to read • [Edit Online](#)

단순 바인딩을 사용하면 데이터 집합 테이블의 열 값과 같은 단일 데이터 요소를 컨트롤에 표시할 수 있습니다. 컨트롤의 모든 속성을 데이터 값에 단순하게 바인딩할 수 있습니다.

## 컨트롤을 간단하게 바인딩하려면

1. 데이터 소스에 연결합니다. 자세한 내용은 [데이터 원본에 연결](#)을 참조하세요.
2. Visual Studio의 폼에서 컨트롤을 선택하고 속성 창을 표시합니다.
3. (데이터 바인딩) 속성을 확장합니다.

가장 자주 바인딩되는 속성은 (데이터 바인딩) 속성 아래에 표시됩니다. 예를 들어 대부분의 컨트롤에서 **Text** 속성은 가장 자주 바인딩됩니다.

4. 바인딩하려는 속성이 일반적으로 바인딩되는 속성 중 하나가 아닌 경우 (고급) 상자의 줄임표 를 클릭하여 다음을 표시합니다. 서식 지정 및 고급 바인딩 대화 상자를 사용하여 해당 컨트롤의 전체 속성 목록을 표시합니다.
5. 바인딩할 속성을 선택하고 바인딩 아래의 드롭다운 화살표를 클릭합니다.  
사용 가능한 데이터 소스 목록이 표시됩니다.
6. 원하는 단일 데이터 요소를 찾을 때까지 바인딩할 데이터 소스를 확장합니다. 예를 들어 데이터 세트의 테이블에서 열 값에 바인딩할 경우 데이터 세트 이름을 확장하고 나서 테이블을 이름을 확장하여 열 이름을 표시합니다.
7. 바인딩할 요소 이름을 클릭합니다.
8. 서식 지정 및 고급 바인딩 대화 상자에서 작업하는 경우 확인을 클릭하여 속성 창으로 돌아갑니다.
9. 컨트롤의 추가 속성을 바인딩하려면 3 ~ 7 단계를 반복합니다.

### NOTE

단순 바인딩된 컨트롤은 단일 데이터 요소만 표시하기 때문에 단순 바인딩된 컨트롤을 사용하여 Windows Form에 탐색 논리를 포함하는 것이 매우 일반적입니다.

## 참고자료

- [Binding](#)
- [Windows Forms 데이터 바인딩](#)
- [데이터 바인딩 및 Windows Forms](#)

# 방법: 동일한 데이터 소스에 바인딩된 여러 컨트롤의 동기화 상태가 유지되도록 설정

2019-10-23 • 6 minutes to read • [Edit Online](#)

Windows Forms에서 데이터 바인딩을 사용 하여 작업을 하는 경우에 종종 여러 컨트롤은 동일한 데이터 소스에 바인딩됩니다. 일부 경우에는 컨트롤의 바인딩된 속성이 서로 데이터 소스를 사용 하여 동기화 된 상태로 유지 되도록 추가 단계를 수행 해야 할 수도 있습니다. 이러한 단계는 두 가지 상황에서 필요 합니다.

- 데이터 소스를 구현 하지 않는 경우 `IBindingList`를 생성 하므로 `ListChanged` 유형의 이벤트 `ItemChanged`합니다.
- 데이터 원본 구현 `IEditableObject`합니다.

전자의 경우에 사용할 수는 `BindingSource` 데이터 소스 컨트롤에 바인딩할 합니다. 후자의 경우 사용 하여는 `BindingSource` 하고 처리 합니다 `BindingComplete` 이벤트 및 호출 `EndCurrentEdit` 연결 된 `BindingManagerBase`합니다.

## 예제

다음 코드 예제에서는 세 가지 컨트롤을 바인딩하는 방법을 보여 줍니다-두 개의 텍스트 상자 컨트롤 및 `DataGridView` 컨트롤-동일한 열에 `DataSet` 사용 하여 `BindingSource` 구성 요소입니다. 이 예제에서는 처리 하는 방법에 설명 합니다 `BindingComplete` 이벤트 하나의 입력란의 텍스트 값 변경 되 면 추가 텍스트 상자를 확인 및 `DataGridView` 컨트롤은 올바른 값으로 업데이트 됩니다.

이 예제에서는 사용을 `BindingSource` 데이터 원본과 컨트롤을 바인딩할 합니다. 또는 데이터 원본에 직접 컨트롤을 바인딩할 수를 검색 합니다 `BindingManagerBase` 품의 바인딩에 대 한 `BindingContext` 처리 및 `BindingComplete` 에 대 한 이벤트는 `BindingManagerBase`합니다. 이 작업을 수행 하는 방법의 예에 대 한 도움말 페이지를 참조 합니다 `BindingComplete` 이벤트를 `BindingManagerBase`입니다.

```
// Declare the controls to be used.
private BindingSource bindingSource1;
private TextBox textBox1;
private TextBox textBox2;
private DataGridView dataGridView1;

private void InitializeControlsAndDataSource()
{
    // Initialize the controls and set location, size and
    // other basic properties.
    this.dataGridView1 = new DataGridView();
    this.bindingSource1 = new BindingSource();
    this.textBox1 = new TextBox();
    this.textBox2 = new TextBox();
    this.dataGridView1.ColumnHeadersHeightSizeMode =
        DataGridViewColumnHeadersHeightSizeMode.AutoSize;
    this.dataGridView1.Dock = DockStyle.Top;
    this.dataGridView1.Location = new Point(0, 0);
    this.dataGridView1.Size = new Size(292, 150);
    this.textBox1.Location = new Point(132, 156);
    this.textBox1.Size = new Size(100, 20);
    this.textBox2.Location = new Point(12, 156);
    this.textBox2.Size = new Size(100, 20);
    this.ClientSize = new Size(292, 266);
    this.Controls.Add(this.textBox2);
    this.Controls.Add(this.textBox1);
}
```



```

this.Controls.Add(this.dataGridView1);

// Declare the DataSet and add a table and column.
DataSet set1 = new DataSet();
set1.Tables.Add("Menu");
set1.Tables[0].Columns.Add("Beverages");

// Add some rows to the table.
set1.Tables[0].Rows.Add("coffee");
set1.Tables[0].Rows.Add("tea");
set1.Tables[0].Rows.Add("hot chocolate");
set1.Tables[0].Rows.Add("milk");
set1.Tables[0].Rows.Add("orange juice");

// Set the data source to the DataSet.
bindingSource1.DataSource = set1;

//Set the DataMember to the Menu table.
bindingSource1.DataMember = "Menu";

// Add the control data bindings.
dataGridView1.DataSource = bindingSource1;
textBox1.DataBindings.Add("Text", bindingSource1,
    "Beverages", true, DataSourceUpdateMode.OnPropertyChanged);
textBox2.DataBindings.Add("Text", bindingSource1,
    "Beverages", true, DataSourceUpdateMode.OnPropertyChanged);
bindingSource1.BindingComplete +=
    new BindingCompleteEventHandler(bindingSource1_BindingComplete);
}

private void bindingSource1_BindingComplete(object sender, BindingCompleteEventArgs e)
{
    // Check if the data source has been updated, and that no error has occurred.
    if (e.BindingCompleteContext ==
        BindingCompleteContext.DataSourceUpdate && e.Exception == null)

        // If not, end the current edit.
        e.Binding.BindingManagerBase.EndCurrentEdit();
}

```

```

' Declare the controls to be used.
Private WithEvents bindingSource1 As BindingSource
Private WithEvents textBox1 As TextBox
Private WithEvents textBox2 As TextBox
Private WithEvents dataGridView1 As DataGridView

Private Sub InitializeControlsAndDataSource()
    ' Initialize the controls and set location, size and
    ' other basic properties.
    Me.dataGridView1 = New DataGridView()
    Me.bindingSource1 = New BindingSource()
    Me.textBox1 = New TextBox()
    Me.textBox2 = New TextBox()
    Me.dataGridView1.ColumnHeadersHeightSizeMode = DataGridViewColumnHeadersHeightSizeMode.AutoSize
    Me.dataGridView1.Dock = DockStyle.Top
    Me.dataGridView1.Location = New Point(0, 0)
    Me.dataGridView1.Size = New Size(292, 150)
    Me.textBox1.Location = New Point(132, 156)
    Me.textBox1.Size = New Size(100, 20)
    Me.textBox2.Location = New Point(12, 156)
    Me.textBox2.Size = New Size(100, 20)
    Me.ClientSize = New Size(292, 266)
    Me.Controls.Add(Me.textBox2)
    Me.Controls.Add(Me.textBox1)
    Me.Controls.Add(Me.dataGridView1)

```

```

' Declare the DataSet and add a table and column.
Dim set1 As New DataSet()
set1.Tables.Add("Menu")
set1.Tables(0).Columns.Add("Beverages")

' Add some rows to the table.
set1.Tables(0).Rows.Add("coffee")
set1.Tables(0).Rows.Add("tea")
set1.Tables(0).Rows.Add("hot chocolate")
set1.Tables(0).Rows.Add("milk")
set1.Tables(0).Rows.Add("orange juice")

' Set the data source to the DataSet.
bindingSource1.DataSource = set1

'Set the DataMember to the Menu table.
bindingSource1.DataMember = "Menu"

' Add the control data bindings.
dataGridView1.DataSource = bindingSource1
textBox1.DataBindings.Add("Text", bindingSource1, "Beverages", _
    True, DataSourceUpdateMode.OnPropertyChanged)
textBox2.DataBindings.Add("Text", bindingSource1, "Beverages", _
    True, DataSourceUpdateMode.OnPropertyChanged)

End Sub

Private Sub bindingSource1_BindingComplete(ByVal sender As Object, _
    ByVal e As BindingCompleteEventArgs) Handles bindingSource1.BindingComplete

    ' Check if the data source has been updated, and that no error has occurred.
    If e.BindingCompleteContext = BindingCompleteContext.DataSourceUpdate _
        AndAlso e.Exception Is Nothing Then

        ' If not, end the current edit.
        e.Binding.BindingManagerBase.EndCurrentEdit()
    End If

End Sub

End Sub

```

## 코드 컴파일

- 이 코드 예제에서는
- [System](#), [System.Windows.Forms](#) 및 [System.Drawing](#) 어셈블리에 대한 참조
- 사용 하여 폼을 [Load](#) 이벤트를 처리 하 고 호출 하는 `InitializeControlsAndDataSource` 폼의 예제에서 메서드 [Load](#) 이벤트 처리기입니다.

## 참고자료

- 방법: [BindingSource](#) 구성 요소를 사용 하여 양식 간에 바인딩된 데이터 공유
- [Windows Forms 데이터 바인딩의 변경 알림](#)
- [데이터 바인딩과 관련된 인터페이스](#)
- [Windows Forms 데이터 바인딩](#)

# 방법: 자식 테이블에서 선택된 행이 올바른 위치에 유지되도록 설정

2019-10-23 • 16 minutes to read • [Edit Online](#)

Windows Forms에서 데이터 바인딩을 사용할 때 부모/자식 또는 마스터/세부 정보 뷰에 데이터를 표시하는 경우가 많습니다. 이는 동일한 소스의 데이터가 두 컨트롤에 표시되는 데이터 바인딩 시나리오를 가리킵니다. 한 컨트롤에서 선택 항목을 변경하면 두 번째 컨트롤에 표시되는 데이터가 변경됩니다. 예를 들어 첫 번째 컨트롤에는 고객 목록이 포함되고 두 번째 컨트롤에는 첫 번째 컨트롤에서 선택한 고객과 관련된 주문 목록이 포함될 수 있습니다.

.NET Framework 버전 2.0부터 부모/자식 뷰에 데이터를 표시하는 경우 자식 테이블에서 현재 선택된 행이 테이블의 첫 번째 행으로 다시 설정되지 않도록 추가 단계를 수행해야 할 수도 있습니다. 이 작업을 수행하려면 자식 테이블 위치를 캐시하고 부모 테이블이 변경된 후 다시 설정해야 합니다. 일반적으로 자식 다시 설정은 부모 테이블의 행에 있는 필드가 처음 변경될 때 발생합니다.

현재 자식 위치를 캐시하려면

1. 정수 변수를 선언하여 자식 목록 위치를 저장하고, 부울 변수를 선언하여 자식 위치를 캐시할지 여부를 저장합니다.

```
private int cachedPosition = -1;
private bool cacheChildPosition = true;
```

```
Private cachedPosition As Integer = - 1
Private cacheChildPosition As Boolean = True
```

2. 바인딩의 [CurrencyManager](#)에 대한 [ListChanged](#) 이벤트를 처리하고 [Reset](#)의 [ListChangedType](#)을 확인합니다.
3. [CurrencyManager](#)의 현재 위치를 확인합니다. 목록의 첫 번째 항목(일반적으로 0)보다 크면 변수에 저장합니다.

```
void relatedCM_ListChanged(object sender, ListChangedEventArgs e)
{
    // Check to see if this is a caching situation.
    if (cacheChildPosition && cachePositionCheckBox.Checked)
    {
        // If so, check to see if it is a reset situation, and the current
        // position is greater than zero.
        CurrencyManager relatedCM = sender as CurrencyManager;
        if (e.ListChangedType == ListChangedType.Reset && relatedCM.Position > 0)

            // If so, cache the position of the child table.
            cachedPosition = relatedCM.Position;
    }
}
```

```

Private Sub relatedCM_ListChanged(ByVal sender As Object, _
    ByVal e As ListChangedEventArgs)
    ' Check to see if this is a caching situation.
    If cacheChildPosition AndAlso cachePositionCheckBox.Checked Then
        ' If so, check to see if it is a reset situation, and the current
        ' position is greater than zero.
        Dim relatedCM As CurrencyManager = sender
        If e.ListChangedType = ListChangedType.Reset _
            AndAlso relatedCM.Position > 0 Then

            ' If so, cache the position of the child table.
            cachedPosition = relatedCM.Position
        End If
    End If

End Sub

```

4. 부모 통화 관리자에 대한 부모 목록의 [CurrentChanged](#) 이벤트를 처리합니다. 처리기에서 캐싱 시나리오가 아님을 나타내는 부울 값을 설정합니다. [CurrentChanged](#)가 발생하는 경우 부모에 대한 변경 내용은 항목 값 변경이 아니라 목록 위치 변경입니다.

```

void bindingSource1_CurrentChanged(object sender, EventArgs e)
{
    // If the CurrentChanged event occurs, this is not a caching
    // situation.
    cacheChildPosition = false;
}

```

```

' Handle the current changed event. This event occurs when
' the current item is changed, but not when a field of the current
' item is changed.
Private Sub bindingSource1_CurrentChanged(ByVal sender As Object, _
    ByVal e As EventArgs) Handles bindingSource1.CurrentChanged
    ' If the CurrentChanged event occurs, this is not a caching
    ' situation.
    cacheChildPosition = False

End Sub

```

자식 위치를 다시 설정하려면

1. 자식 바인딩의 [CurrencyManager](#)에 대한 [PositionChanged](#) 이벤트를 처리합니다.
2. 자식 테이블 위치를 이전 절차에서 저장된 캐시된 위치로 다시 설정합니다.

```

void relatedCM_PositionChanged(object sender, EventArgs e)
{
    // Check to see if this is a caching situation.
    if (cacheChildPosition && cachePositionCheckBox.Checked)
    {
        CurrencyManager relatedCM = sender as CurrencyManager;

        // If so, check to see if the current position is
        // not equal to the cached position and the cached
        // position is not out of bounds.
        if (relatedCM.Position != cachedPosition && cachedPosition
            > 0 && cachedPosition < relatedCM.Count)
        {
            relatedCM.Position = cachedPosition;
            cachedPosition = -1;
        }
    }
}

```

```

Private Sub relatedCM_PositionChanged(ByVal sender As Object, ByVal e As EventArgs)
    ' Check to see if this is a caching situation.
    If cacheChildPosition AndAlso cachePositionCheckBox.Checked Then
        Dim relatedCM As CurrencyManager = sender

        ' If so, check to see if the current position is
        ' not equal to the cached position and the cached
        ' position is not out of bounds.
        If relatedCM.Position <> cachedPosition AndAlso _
            cachedPosition > 0 AndAlso cachedPosition < _
            relatedCM.Count Then
            relatedCM.Position = cachedPosition
            cachedPosition = -1
        End If
    End If
End Sub

```

## 예제

다음 예제에서는 [CurrencyManager](#)에서 자식 테이블의 현재 위치를 저장하고 부모 테이블에서 편집이 완료된 후 위치를 다시 설정하는 방법을 보여 줍니다. 이 예제에는 [BindingSource](#) 구성 요소를 사용하여 [DataSet](#)의 두 테이블에 바인딩된 두 개의 [DataGridView](#) 컨트롤이 포함되어 있습니다. 두 테이블 간에 관계가 설정되고 [DataSet](#)에 관계가 추가됩니다. 자식 테이블의 위치는 데모용으로 초기에 세 번째 행으로 설정되어 있습니다.

```

using System;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace BT2
{
    public class Form1 : Form
    {
        public Form1()
        {
            InitializeControlsAndDataSource();
        }

        // Declare the controls to be used.
        private BindingSource bindingSource1;
        private DataGridView dataGridView1;

```

```

private Button button1;
private DataGridView dataGridView2;
private CheckBox cachePositionCheckBox;
public DataSet set1;

private void InitializeControlsAndDataSource()
{
    // Initialize the controls and set location, size and
    // other basic properties.
    this.dataGridView1 = new DataGridView();
    this.bindingSource1 = new BindingSource();
    this.button1 = new Button();
    this.dataGridView2 = new DataGridView();
    this.cachePositionCheckBox = new System.Windows.Forms.CheckBox();
    this.dataGridView1.ColumnHeadersHeightSizeMode =
        DataGridViewColumnHeadersHeightSizeMode.AutoSize;
    this.dataGridView1.Dock = DockStyle.Top;
    this.dataGridView1.Location = new Point(0, 20);
    this.dataGridView1.Size = new Size(292, 170);
    this.button1.Location = new System.Drawing.Point(18, 175);
    this.button1.Size = new System.Drawing.Size(125, 23);

    button1.Text = "Clear Parent Field";
    this.button1.Click += new System.EventHandler(this.button1_Click);
    this.dataGridView2.ColumnHeadersHeightSizeMode =
        System.Windows.Forms.DataGridViewColumnHeadersHeightSizeMode.AutoSize;
    this.dataGridView2.Location = new System.Drawing.Point(0, 225);
    this.dataGridView2.Size = new System.Drawing.Size(309, 130);
    this.cachePositionCheckBox.AutoSize = true;
    this.cachePositionCheckBox.Checked = true;
    this.cachePositionCheckBox.Location = new System.Drawing.Point(150, 175);
    this.cachePositionCheckBox.Name = "radioButton1";
    this.cachePositionCheckBox.Size = new System.Drawing.Size(151, 17);
    this.cachePositionCheckBox.Text = "Cache and restore position";
    this.ClientSize = new System.Drawing.Size(325, 420);
    this.Controls.Add(this.dataGridView1);
    this.Controls.Add(this.cachePositionCheckBox);
    this.Controls.Add(this.dataGridView2);
    this.Controls.Add(this.button1);

    // Initialize the data.
    set1 = InitializeDataSet();

    // Set the data source to the DataSet.
    bindingSource1.DataSource = set1;

    //Set the DataMember to the Menu table.
    bindingSource1.DataMember = "Customers";

    // Add the control data bindings.
    dataGridView1.DataSource = bindingSource1;

    // Set the data source and member for the second DataGridView.
    dataGridView2.DataSource = bindingSource1;
    dataGridView2.DataMember = "custOrders";

    // Get the currency manager for the customer orders binding.
    CurrencyManager relatedCM =
        bindingSource1.GetRelatedCurrencyManager("custOrders");

    // Set the position in the child table for demonstration purposes.
    relatedCM.Position = 3;

    // Handle the current changed event. This event occurs when
    // the current item is changed, but not when a field of the current
    // item is changed.
    bindingSource1.CurrentChanged +=
        new EventHandler(bindingSource1_CurrentChanged);
}

```

```

        // Handle the two events for caching and resetting the position.
        relatedCM.ListChanged += new ListChangedEventHandler(relatedCM_ListChanged);
        relatedCM.PositionChanged
            += new EventHandler(relatedCM_PositionChanged);

        // Set caching to true in case current changed event
        // occurred on set up.
        cacheChildPosition = true;
    }

    // Establish the data set with two tables and a relationship
    // between them.
    private DataSet InitializeDataSet()
    {
        set1 = new DataSet();
        // Declare the DataSet and add a table and column.
        set1.Tables.Add("Customers");
        set1.Tables[0].Columns.Add("CustomerID");
        set1.Tables[0].Columns.Add("Customer Name");
        set1.Tables[0].Columns.Add("Contact Name");

        // Add some rows to the table.
        set1.Tables["Customers"].Rows.Add("c1", "Fabrikam, Inc.", "Ellen Adams");
        set1.Tables[0].Rows.Add("c2", "Lucerne Publishing", "Don Hall");
        set1.Tables[0].Rows.Add("c3", "Northwind Traders", "Lori Penor");
        set1.Tables[0].Rows.Add("c4", "Tailspin Toys", "Michael Patten");
        set1.Tables[0].Rows.Add("c5", "Woodgrove Bank", "Jyothi Pai");

        // Declare the DataSet and add a table and column.
        set1.Tables.Add("Orders");
        set1.Tables[1].Columns.Add("CustomerID");
        set1.Tables[1].Columns.Add("OrderNo");
        set1.Tables[1].Columns.Add("OrderDate");

        // Add some rows to the table.
        set1.Tables[1].Rows.Add("c1", "119", "10/04/2006");
        set1.Tables[1].Rows.Add("c1", "149", "10/10/2006");
        set1.Tables[1].Rows.Add("c1", "159", "10/12/2006");
        set1.Tables[1].Rows.Add("c2", "169", "10/10/2006");
        set1.Tables[1].Rows.Add("c2", "179", "10/10/2006");
        set1.Tables[1].Rows.Add("c2", "189", "10/12/2006");
        set1.Tables[1].Rows.Add("c3", "122", "10/04/2006");
        set1.Tables[1].Rows.Add("c4", "130", "10/10/2006");
        set1.Tables[1].Rows.Add("c5", "1.29", "10/14/2006");

        DataRelation dr = new DataRelation("custOrders",
            set1.Tables["Customers"].Columns["CustomerID"],
            set1.Tables["Orders"].Columns["CustomerID"]);
        set1.Relations.Add(dr);
        return set1;
    }

    private int cachedPosition = -1;
    private bool cacheChildPosition = true;

    void relatedCM_ListChanged(object sender, ListChangedEventArgs e)
    {
        // Check to see if this is a caching situation.
        if (cacheChildPosition && cachePositionCheckBox.Checked)
        {
            // If so, check to see if it is a reset situation, and the current
            // position is greater than zero.
            CurrencyManager relatedCM = sender as CurrencyManager;
            if (e.ListChangedType == ListChangedType.Reset && relatedCM.Position > 0)

                // If so, cache the position of the child table.
                cachedPosition = relatedCM.Position;
        }
    }

    void bindingSource1_CurrentChanged(object sender, EventArgs e)

```

```

    {
        // If the CurrentChanged event occurs, this is not a caching
        // situation.
        cacheChildPosition = false;
    }
    void relatedCM_PositionChanged(object sender, EventArgs e)
    {
        // Check to see if this is a caching situation.
        if (cacheChildPosition && cachePositionCheckBox.Checked)
        {
            CurrencyManager relatedCM = sender as CurrencyManager;

            // If so, check to see if the current position is
            // not equal to the cached position and the cached
            // position is not out of bounds.
            if (relatedCM.Position != cachedPosition && cachedPosition
                > 0 && cachedPosition < relatedCM.Count)
            {
                relatedCM.Position = cachedPosition;
                cachedPosition = -1;
            }
        }
    }
    int count = 0;
    private void button1_Click(object sender, EventArgs e)
    {
        // For demo purposes--modifies a value in the first row of the
        // parent table.
        DataRow row1 = set1.Tables[0].Rows[0];
        row1[1] = DBNull.Value;
    }

    [STAThread]
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);
        Application.Run(new Form1());
    }
}

```

```

Imports System.ComponentModel
Imports System.Data
Imports System.Drawing
Imports System.Text
Imports System.Windows.Forms

Public Class Form1
    Inherits Form

    Public Sub New()
        InitializeControlsAndDataSource()

    End Sub

    ' Declare the controls to be used.
    Private WithEvents bindingSource1 As BindingSource
    Private dataGridView1 As DataGridView
    Private WithEvents button1 As Button
    Private dataGridView2 As DataGridView
    Private cachePositionCheckBox As CheckBox
    Public set1 As DataSet

```



```

Private Sub InitializeControlsAndDataSource()
    ' Initialize the controls and set location, size and
    ' other basic properties.
    Me.dataGridView1 = New DataGridView()
    Me.bindingSource1 = New BindingSource()
    Me.button1 = New Button()
    Me.dataGridView2 = New DataGridView()
    Me.cachePositionCheckBox = New System.Windows.Forms.CheckBox()
    Me.dataGridView1.ColumnHeadersHeightSizeMode = _
        DataGridViewColumnHeadersHeightSizeMode.AutoSize
    Me.dataGridView1.Dock = DockStyle.Top
    Me.dataGridView1.Location = New Point(0, 20)
    Me.dataGridView1.Size = New Size(292, 170)
    Me.button1.Location = New System.Drawing.Point(18, 175)
    Me.button1.Size = New System.Drawing.Size(125, 23)

    button1.Text = "Clear Parent Field"

    Me.dataGridView2.ColumnHeadersHeightSizeMode = _
        System.Windows.Forms.DataGridViewColumnHeadersHeightSizeMode.AutoSize
    Me.dataGridView2.Location = New System.Drawing.Point(0, 225)
    Me.dataGridView2.Size = New System.Drawing.Size(309, 130)
    Me.cachePositionCheckBox.AutoSize = True
    Me.cachePositionCheckBox.Checked = True
    Me.cachePositionCheckBox.Location = New System.Drawing.Point(150, 175)
    Me.cachePositionCheckBox.Name = "radioButton1"
    Me.cachePositionCheckBox.Size = New System.Drawing.Size(151, 17)
    Me.cachePositionCheckBox.Text = "Cache and restore position"
    Me.ClientSize = New System.Drawing.Size(325, 420)
    Me.Controls.Add(Me.dataGridView1)
    Me.Controls.Add(Me.cachePositionCheckBox)
    Me.Controls.Add(Me.dataGridView2)
    Me.Controls.Add(Me.button1)

    ' Initialize the data.
    set1 = InitializeDataSet()

    ' Set the data source to the DataSet.
    bindingSource1.DataSource = set1

    'Set the DataMember to the Menu table.
    bindingSource1.DataMember = "Customers"

    ' Add the control data bindings.
    dataGridView1.DataSource = bindingSource1

    ' Set the data source and member for the second DataGridView.
    dataGridView2.DataSource = bindingSource1
    dataGridView2.DataMember = "custOrders"

    ' Get the currency manager for the customer orders binding.
    Dim relatedCM As CurrencyManager = _
        bindingSource1.GetRelatedCurrencyManager("custOrders")

    ' Handle the two events for caching and resetting the position.
    AddHandler relatedCM.ListChanged, AddressOf relatedCM_ListChanged
    AddHandler relatedCM.PositionChanged, AddressOf relatedCM_PositionChanged

    ' Set the position in the child table for demonstration purposes.
    relatedCM.Position = 3

    ' Set cacheing to true in case current changed event
    ' occurred on set up.
    cacheChildPosition = True

```

End Sub

```

' Establish the data set with two tables and a relationship
' between them.
Private Function InitializeDataSet() As DataSet
    set1 = New DataSet()
    ' Declare the DataSet and add a table and column.
    set1.Tables.Add("Customers")
    set1.Tables(0).Columns.Add("CustomerID")
    set1.Tables(0).Columns.Add("Customer Name")
    set1.Tables(0).Columns.Add("Contact Name")

    ' Add some rows to the table.
    set1.Tables("Customers").Rows.Add("c1", "Fabrikam, Inc.", _
        "Ellen Adams")
    set1.Tables(0).Rows.Add("c2", "Lucerne Publishing", "Don Hall")
    set1.Tables(0).Rows.Add("c3", "Northwind Traders", "Lori Penor")
    set1.Tables(0).Rows.Add("c4", "Tailspin Toys", "Michael Patten")
    set1.Tables(0).Rows.Add("c5", "Woodgrove Bank", "Jyothi Pai")

    ' Declare the DataSet and add a table and column.
    set1.Tables.Add("Orders")
    set1.Tables(1).Columns.Add("CustomerID")
    set1.Tables(1).Columns.Add("OrderNo")
    set1.Tables(1).Columns.Add("OrderDate")

    ' Add some rows to the table.
    set1.Tables(1).Rows.Add("c1", "119", "10/04/2006")
    set1.Tables(1).Rows.Add("c1", "149", "10/10/2006")
    set1.Tables(1).Rows.Add("c1", "159", "10/12/2006")
    set1.Tables(1).Rows.Add("c2", "169", "10/10/2006")
    set1.Tables(1).Rows.Add("c2", "179", "10/10/2006")
    set1.Tables(1).Rows.Add("c2", "189", "10/12/2006")
    set1.Tables(1).Rows.Add("c3", "122", "10/04/2006")
    set1.Tables(1).Rows.Add("c4", "130", "10/10/2006")
    set1.Tables(1).Rows.Add("c5", "1.29", "10/14/2006")

    Dim dr As New DataRelation("custOrders", _
        set1.Tables("Customers").Columns("CustomerID"), _
        set1.Tables("Orders").Columns("CustomerID"))
    set1.Relations.Add(dr)
    Return set1
End Function '
Private cachedPosition As Integer = - 1
Private cacheChildPosition As Boolean = True

Private Sub relatedCM_ListChanged(ByVal sender As Object, _
    ByVal e As ListChangedEventArgs)
    ' Check to see if this is a caching situation.
    If cacheChildPosition AndAlso cachePositionCheckBox.Checked Then
        ' If so, check to see if it is a reset situation, and the current
        ' position is greater than zero.
        Dim relatedCM As CurrencyManager = sender
        If e.ListChangedType = ListChangedType.Reset _
            AndAlso relatedCM.Position > 0 Then

            ' If so, cache the position of the child table.
            cachedPosition = relatedCM.Position
        End If
    End If
End Sub

End Sub

' Handle the current changed event. This event occurs when
' the current item is changed, but not when a field of the current
' item is changed.
Private Sub bindingSource1_CurrentChanged(ByVal sender As Object, _
    ByVal e As EventArgs) Handles bindingSource1.CurrentChanged
    ' If the CurrentChanged event occurs, this is not a caching

```

```

        ' situation.
        cacheChildPosition = False

    End Sub

    Private Sub relatedCM_PositionChanged(ByVal sender As Object, ByVal e As EventArgs)
        ' Check to see if this is a caching situation.
        If cacheChildPosition AndAlso cachePositionCheckBox.Checked Then
            Dim relatedCM As CurrencyManager = sender

            ' If so, check to see if the current position is
            ' not equal to the cached position and the cached
            ' position is not out of bounds.
            If relatedCM.Position <> cachedPosition AndAlso _
                cachedPosition > 0 AndAlso cachedPosition < _
                relatedCM.Count Then
                relatedCM.Position = cachedPosition
                cachedPosition = -1
            End If
        End If
    End Sub

    Private count As Integer = 0

    Private Sub button1_Click(ByVal sender As Object, _
        ByVal e As EventArgs) Handles button1.Click
        ' For demo purposes--modifies a value in the first row of the
        ' parent table.
        Dim row1 As DataRow = set1.Tables(0).Rows(0)
        row1(1) = DBNull.Value
    End Sub

    <STAThread> _
    Shared Sub Main()
        Application.EnableVisualStyles()
        Application.SetCompatibleTextRenderingDefault(False)
        Application.Run(New Form1())
    End Sub
End Class

```

코드 예제를 테스트하려면 다음 단계를 수행합니다.

1. 예제를 실행합니다.
2. **Cache and reset position** 확인란이 선택되었는지 확인합니다.
3. **Clear parent field** 단추를 클릭하여 부모 테이블의 필드를 변경합니다. 자식 테이블에서 선택한 행은 변경되지 않습니다.
4. 예제를 닫고 다시 실행합니다. 다시 설정 동작이 부모 행을 처음 변경할 때만 발생하기 때문에 이 작업을 수행해야 합니다.
5. **Cache and reset position** 확인란을 선택 취소합니다.
6. **Clear parent field** 단추를 클릭합니다. 자식 테이블에서 선택한 행이 첫 번째 행으로 변경됩니다.

## 코드 컴파일

이 예제에는 다음 사항이 필요합니다.

- System, System.Data, System.Drawing, System.Windows.Forms and System.XML 어셈블리에 대한 참조

## 참고자료

- 방법: 여러 컨트롤을 확인 동일한 데이터 소스에 바인딩된 동기화 된 상태로 유지
- [BindingSource](#) 구성 요소
- [데이터 바인딩 및 Windows Forms](#)

# 방법: IListSource 인터페이스 구현

2019-10-23 • 8 minutes to read • [Edit Online](#)

구현된 [IListSource](#) 인터페이스를 구현하지 않는 바인딩 가능한 클래스를 만들려면 [IList](#) 않지만 다른 위치에서 목록을 제공 합니다.

## 예제

다음 코드 예제를 구현하는 방법에 설명합니다 [IListSource](#) 인터페이스입니다. 이라는 구성 요소

`EmployeeListSource` 노출을 [IList](#) 구현하여 데이터 바인딩에 [GetList](#) 메서드.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.ComponentModel;

namespace IListSourceCS
{
    public class EmployeeListSource : Component, IListSource
    {
        public EmployeeListSource() {}

        public EmployeeListSource(IContainer container)
        {
            container.Add(this);
        }

        #region IListSource Members

        bool IListSource.ContainsListCollection
        {
            get { return false; }
        }

        System.Collections.IList IListSource.GetList()
        {
            BindingList<Employee> ble = new BindingList<Employee>();

            if (!this.DesignMode)
            {
                ble.Add(new Employee("Aaberg, Jesper", 26000000));
                ble.Add(new Employee("Cajhen, Janko", 19600000));
                ble.Add(new Employee("Furse, Kari", 19000000));
                ble.Add(new Employee("Langhorn, Carl", 16000000));
                ble.Add(new Employee("Todorov, Teodor", 15700000));
                ble.Add(new Employee("Verebélyi, Ágnes", 15700000));
            }

            return ble;
        }

        #endregion
    }
}
```

Imports System.ComponentModel

Public Class EmployeeListSource  
Inherits Component

```

Implements IListSource

<System.Diagnostics.DebuggerNonUserCode()> _
Public Sub New(ByVal Container As System.ComponentModel.IContainer)
    MyClass.New()

    'Required for Windows.Forms Class Composition Designer support
    Container.Add(Me)

End Sub

<System.Diagnostics.DebuggerNonUserCode()> _
Public Sub New()
    MyBase.New()

    'This call is required by the Component Designer.
    InitializeComponent()

End Sub

'Component overrides dispose to clean up the component list.
<System.Diagnostics.DebuggerNonUserCode()> _
Protected Overrides Sub Dispose(ByVal disposing As Boolean)
    If disposing AndAlso components IsNot Nothing Then
        components.Dispose()
    End If
    MyBase.Dispose(disposing)
End Sub

'Required by the Component Designer
Private components As System.ComponentModel.IContainer

'NOTE: The following procedure is required by the Component Designer
'It can be modified using the Component Designer.
'Do not modify it using the code editor.
<System.Diagnostics.DebuggerStepThrough()> _
Private Sub InitializeComponent()
    components = New System.ComponentModel.Container()
End Sub

#Region "IListSource Members"

    Public ReadOnly Property ContainsListCollection() As Boolean Implements
System.ComponentModel.IListSource.ContainsListCollection
        Get
            Return False
        End Get
    End Property

    Public Function GetList() As System.Collections.IList Implements System.ComponentModel.IListSource.GetList

        Dim ble As New BindingList(Of Employee)

        If Not Me.DesignMode Then
            ble.Add(New Employee("Aaberg, Jesper", 26000000))
            ble.Add(New Employee("Cajhen, Janko", 19600000))
            ble.Add(New Employee("Furse, Kari", 19000000))
            ble.Add(New Employee("Langhorn, Carl", 16000000))
            ble.Add(New Employee("Todorov, Teodor", 15700000))
            ble.Add(New Employee("Verebelyi, Ágnes", 15700000))
        End If

        Return ble

    End Function

#End Region

End Class

```

```

using System;
using System.Collections.Generic;
using System.Text;
using System.ComponentModel;

namespace IListSourceCS
{
    public class Employee : BusinessObjectBase
    {
        private string      _id;
        private string      _name;
        private Decimal      parkingId;

        public Employee() : this(string.Empty, 0) {}
        public Employee(string name) : this(name, 0) {}

        public Employee(string name, Decimal parkingId) : base()
        {
            this._id = System.Guid.NewGuid().ToString();

            // Set values
            this.Name = name;
            this.ParkingID = parkingId;
        }

        public string ID
        {
            get { return _id; }
        }

        const string NAME = "Name";
        public string Name
        {
            get { return _name; }
            set
            {
                if (_name != value)
                {
                    _name = value;

                    // Raise the PropertyChanged event.
                    OnPropertyChanged(NAME);
                }
            }
        }

        const string PARKING_ID = "Salary";
        public Decimal ParkingID
        {
            get { return parkingId; }
            set
            {
                if (parkingId != value)
                {
                    parkingId = value;

                    // Raise the PropertyChanged event.
                    OnPropertyChanged(PARKING_ID);
                }
            }
        }
    }
}

```

```

Imports System.ComponentModel

Public Class Employee
    Inherits BusinessObjectBase

    Private _id As String
    Private _name As String
    Private _parkingId As Decimal

    Public Sub New(ByVal name As String, ByVal parkId As Decimal)
        MyBase.New()
        Me._id = System.Guid.NewGuid().ToString()
        ' Set values
        Me.Name = name
        Me.ParkingID = parkId
    End Sub

    Public ReadOnly Property ID() As String
        Get
            Return _id
        End Get
    End Property

    Const NAME_Const As String = "Name"

    Public Property Name() As String
        Get
            Return _name
        End Get
        Set(ByVal value As String)
            If _name <> value Then
                _name = value
                OnPropertyChanged(NAME_Const)
            End If
        End Set
    End Property

    Const PARKINGID_Const As String = "ParkingID"

    Public Property ParkingID() As Decimal
        Get
            Return _parkingId
        End Get
        Set(ByVal value As Decimal)
            If _parkingId <> value Then
                _parkingId = value
                OnPropertyChanged(PARKINGID_Const)
            End If
        End Set
    End Property

End Class

```



```

using System;
using System.Collections.Generic;
using System.Text;
using System.ComponentModel;
using System.Diagnostics;

namespace IListSourceCS
{
    public class BusinessObjectBase : INotifyPropertyChanged
    {
        #region INotifyPropertyChanged Members

        public event PropertyChangedEventHandler PropertyChanged;

        protected virtual void OnPropertyChanged(string propertyName)
        {
            OnPropertyChanged(new PropertyChangedEventArgs(propertyName));
        }

        private void OnPropertyChanged(PropertyChangedEventArgs e)
        {
            if (null != PropertyChanged)
            {
                PropertyChanged(this, e);
            }
        }

        #endregion
    }
}

```

```

Imports System.ComponentModel

Public Class BusinessObjectBase
    Implements INotifyPropertyChanged

    #Region "INotifyPropertyChanged Members"

        Public Event PropertyChanged(ByVal sender As Object, ByVal e As
System.ComponentModel.PropertyChangedEventArgs) Implements
System.ComponentModel.INotifyPropertyChanged.PropertyChanged

        Protected Overridable Sub OnPropertyChanged(ByVal propertyName As String)
            OnPropertyChanged(New PropertyChangedEventArgs(propertyName))
        End Sub

        Private Sub OnPropertyChanged(ByVal e As PropertyChangedEventArgs)
            RaiseEvent PropertyChanged(Me, e)
        End Sub

    #End Region

End Class

```

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace IListSourceCS
{
    .....
}

```

```

public class Form1 : Form
{
    private System.ComponentModel.IContainer components = null;
    private FlowLayoutPanel flowLayoutPanel1;
    private Label label2;
    private DataGridView dataGridView1;
    private DataGridViewTextBoxColumn nameDataGridViewTextBoxColumn;
    private DataGridViewTextBoxColumn salaryDataGridViewTextBoxColumn;
    private DataGridViewTextBoxColumn idDataGridViewTextBoxColumn;
    private EmployeeListSource employeeListSource1;

    public Form1()
    {
        InitializeComponent();
    }

    protected override void Dispose(bool disposing)
    {
        if (disposing && (components != null))
        {
            components.Dispose();
        }
        base.Dispose(disposing);
    }

    #region Windows Form Designer generated code

    private void InitializeComponent()
    {
        this.components = new System.ComponentModel.Container();
        System.Windows.Forms.DataGridViewCellStyle dataGridViewCellStyle1 = new
System.Windows.Forms.DataGridViewCellStyle();
        System.Windows.Forms.DataGridViewCellStyle dataGridViewCellStyle2 = new
System.Windows.Forms.DataGridViewCellStyle();
        this.flowLayoutPanel1 = new System.Windows.Forms.FlowLayoutPanel();
        this.label2 = new System.Windows.Forms.Label();
        this.dataGridView1 = new System.Windows.Forms.DataGridView();
        this.nameDataGridViewTextBoxColumn = new System.Windows.Forms.DataGridViewTextBoxColumn();
        this.salaryDataGridViewTextBoxColumn = new System.Windows.Forms.DataGridViewTextBoxColumn();
        this.idDataGridViewTextBoxColumn = new System.Windows.Forms.DataGridViewTextBoxColumn();
        this.employeeListSource1 = new EmployeeListSource(this.components);
        this.flowLayoutPanel1.SuspendLayout();
        ((System.ComponentModel.ISupportInitialize)(this.dataGridView1)).BeginInit();
        this.SuspendLayout();
        //
        // flowLayoutPanel1
        //
        this.flowLayoutPanel1.AutoSize = true;
        this.flowLayoutPanel1.Controls.Add(this.label2);
        this.flowLayoutPanel1.Dock = System.Windows.Forms.DockStyle.Top;
        this.flowLayoutPanel1.Location = new System.Drawing.Point(0, 0);
        this.flowLayoutPanel1.Name = "flowLayoutPanel1";
        this.flowLayoutPanel1.Size = new System.Drawing.Size(416, 51);
        this.flowLayoutPanel1.TabIndex = 11;
        //
        // label2
        //
        this.label2.AutoSize = true;
        this.label2.Location = new System.Drawing.Point(3, 6);
        this.label2.Margin = new System.Windows.Forms.Padding(3, 6, 3, 6);
        this.label2.Name = "label2";
        this.label2.Size = new System.Drawing.Size(408, 39);
        this.label2.TabIndex = 0;
        this.label2.Text = "This sample demonstrates how to implement the IListSource interface. In this
sam" +
        "ple, a DataGridView is bound at design time to a Component (employeeListSource1)" +
        " that implements IListSource.";
        //
        // dataGridView1

```

```

//
this.dataGridView1.AllowUserToAddRows = false;
this.dataGridView1.AllowUserToDeleteRows = false;
dataGridViewCellStyle1.BackColor = System.Drawing.Color.FromArgb(((int)(((byte)(255)))), ((int)
(((byte)(255)))), ((int)(((byte)(192))))));
this.dataGridView1.AlternatingRowsDefaultCellStyle = dataGridViewCellStyle1;
this.dataGridView1.AutoGenerateColumns = false;
this.dataGridView1.ColumnHeadersHeightSizeMode =
System.Windows.Forms.DataGridViewColumnHeadersHeightSizeMode.AutoSize;
this.dataGridView1.Columns.AddRange(new System.Windows.Forms.DataGridViewColumn[] {
this.nameDataGridViewTextBoxColumn,
this.salaryDataGridViewTextBoxColumn,
this.iDDataGridViewTextBoxColumn});
this.dataGridView1.DataSource = this.employeeListSource1;
this.dataGridView1.Dock = System.Windows.Forms.DockStyle.Fill;
this.dataGridView1.Location = new System.Drawing.Point(0, 51);
this.dataGridView1.Name = "dataGridView1";
this.dataGridView1.RowHeadersVisible = false;
this.dataGridView1.SelectionMode = DataGridViewSelectionMode.FullRowSelect;
this.dataGridView1.Size = new System.Drawing.Size(416, 215);
this.dataGridView1.TabIndex = 12;
//
// nameDataGridViewTextBoxColumn
//
this.nameDataGridViewTextBoxColumn.DataPropertyName = "Name";
this.nameDataGridViewTextBoxColumn.FillWeight = 131.7987F;
this.nameDataGridViewTextBoxColumn.HeaderText = "Name";
this.nameDataGridViewTextBoxColumn.Name = "nameDataGridViewTextBoxColumn";
//
// salaryDataGridViewTextBoxColumn
//
this.salaryDataGridViewTextBoxColumn.DataPropertyName = "ParkingID";
this.salaryDataGridViewTextBoxColumn.DefaultCellStyle = dataGridViewCellStyle2;
this.salaryDataGridViewTextBoxColumn.FillWeight = 121.8274F;
this.salaryDataGridViewTextBoxColumn.HeaderText = "Parking ID";
this.salaryDataGridViewTextBoxColumn.Name = "salaryDataGridViewTextBoxColumn";
//
// iDDataGridViewTextBoxColumn
//
this.iDDataGridViewTextBoxColumn.AutoSizeMode =
System.Windows.Forms.DataGridViewAutoSizeColumnMode.Fill;
this.iDDataGridViewTextBoxColumn.DataPropertyName = "ID";
this.iDDataGridViewTextBoxColumn.FillWeight = 46.37391F;
this.iDDataGridViewTextBoxColumn.HeaderText = "ID";
this.iDDataGridViewTextBoxColumn.Name = "iDDataGridViewTextBoxColumn";
this.iDDataGridViewTextBoxColumn.ReadOnly = true;
//
// Form1
//
this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
this.ClientSize = new System.Drawing.Size(416, 266);
this.Controls.Add(this.dataGridView1);
this.Controls.Add(this.flowLayoutPanel1);
this.Name = "Form1";
this.Text = "IListSource Sample";
this.flowLayoutPanel1.ResumeLayout(false);
this.flowLayoutPanel1.PerformLayout();
((System.ComponentModel.ISupportInitialize)(this.dataGridView1)).EndInit();
this.ResumeLayout(false);
this.PerformLayout();
}

#endregion

}

static class Program
{
    [STAThread]

```

```

        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}

```

```

Imports System.ComponentModel
Imports System.Windows.Forms

```

```
Public Class Form1
```

```
    Inherits System.Windows.Forms.Form
```

```
    Friend WithEvents flowLayoutPanel1 As FlowLayoutPanel
```

```
    Friend WithEvents label2 As Label
```

```
    Friend WithEvents dataGridView1 As DataGridView
```

```
    Friend WithEvents nameDataGridViewTextBoxColumn As DataGridViewTextBoxColumn
```

```
    Friend WithEvents salaryDataGridViewTextBoxColumn As DataGridViewTextBoxColumn
```

```
    Friend WithEvents idDataGridViewTextBoxColumn As DataGridViewTextBoxColumn
```

```
    Friend WithEvents employeeListSource1 As EmployeeListSource
```

```
    Public Sub New()
```

```
        MyBase.New()
```

```
        Me.InitializeComponent()
```

```
    End Sub
```

```
    'Form overrides dispose to clean up the component list.
```

```
    <System.Diagnostics.DebuggerNonUserCode()> _
```

```
    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
```

```
        If disposing AndAlso components IsNot Nothing Then
            components.Dispose()

```

```
        End If
```

```
        MyBase.Dispose(disposing)
```

```
    End Sub
```

```
    'Required by the Windows Form Designer
```

```
    Private components As System.ComponentModel.IContainer
```

```
    'NOTE: The following procedure is required by the Windows Form Designer
```

```
    'It can be modified using the Windows Form Designer.
```

```
    'Do not modify it using the code editor.
```

```
    <System.Diagnostics.DebuggerStepThrough()> _
```

```
    Private Sub InitializeComponent()
```

```
        components = New System.ComponentModel.Container()
```

```
        Dim dataGridViewCellStyle1 = New System.Windows.Forms.DataGridViewCellStyle()
```

```
        Dim dataGridViewCellStyle2 = New System.Windows.Forms.DataGridViewCellStyle()
```

```
        Me.flowLayoutPanel1 = New System.Windows.Forms.FlowLayoutPanel()
```

```
        Me.label2 = New System.Windows.Forms.Label()
```

```
        Me.dataGridView1 = New System.Windows.Forms.DataGridView()
```

```
        Me.nameDataGridViewTextBoxColumn = New System.Windows.Forms.DataGridViewTextBoxColumn()
```

```
        Me.salaryDataGridViewTextBoxColumn = New System.Windows.Forms.DataGridViewTextBoxColumn()
```

```
        Me.idDataGridViewTextBoxColumn = New System.Windows.Forms.DataGridViewTextBoxColumn()
```

```
        Me.employeeListSource1 = New EmployeeListSource(Me.components)
```

```
        Me.flowLayoutPanel1.SuspendLayout()
```

```
        CType(Me.dataGridView1, System.ComponentModel.ISupportInitialize).BeginInit()
```

```
        Me.SuspendLayout()
```

```
        '
```

```
        ' flowLayoutPanel1
```

```
        '
```

```
        Me.flowLayoutPanel1.AutoSize = True
```

```
        Me.flowLayoutPanel1.Controls.Add(Me.label2)
```

```
        Me.flowLayoutPanel1.Dock = System.Windows.Forms.DockStyle.Top
```

```
        Me.flowLayoutPanel1.Location = New System.Drawing.Point(0, 0)
```

```

Me.flowLayoutPanel1.Name = "flowLayoutPanel1"
Me.flowLayoutPanel1.Size = New System.Drawing.Size(416, 51)
Me.flowLayoutPanel1.TabIndex = 11
'
' label2
'
Me.label2.AutoSize = True
Me.label2.Location = New System.Drawing.Point(3, 6)
Me.label2.Margin = New System.Windows.Forms.Padding(3, 6, 3, 6)
Me.label2.Name = "label2"
Me.label2.Size = New System.Drawing.Size(408, 39)
Me.label2.TabIndex = 0
Me.label2.Text = "This sample demonstrates how to implement the IListSource interface. In this sam" +
-
    "ple, a DataGridView is bound at design time to a Component (employeeListSource1)" + _
    " that implements IListSource."
'
' dataGridView1
'
Me.dataGridView1.AllowUserToAddRows = False
Me.dataGridView1.AllowUserToDeleteRows = False
dataGridViewCellStyle1.BackColor = System.Drawing.Color.FromArgb(255, 255, 192)
Me.dataGridView1.AlternatingRowsDefaultCellStyle = dataGridViewCellStyle1
Me.dataGridView1.AutoGenerateColumns = False
Me.dataGridView1.ColumnHeadersHeightSizeMode =
System.Windows.Forms.DataGridViewColumnHeadersHeightSizeMode.AutoSize
Me.dataGridView1.Columns.AddRange(New System.Windows.Forms.DataGridViewColumn() { _
Me.nameDataGridViewTextBoxColumn, Me.salaryDataGridViewTextBoxColumn, Me.iDDataGridViewTextBoxColumn})
Me.dataGridView1.DataSource = Me.employeeListSource1
Me.dataGridView1.Dock = System.Windows.Forms.DockStyle.Fill
Me.dataGridView1.Location = New System.Drawing.Point(0, 51)
Me.dataGridView1.Name = "dataGridView1"
Me.dataGridView1.RowHeadersVisible = False
Me.dataGridView1.SelectionMode = System.Windows.Forms.DataGridViewSelectionMode.FullRowSelect
Me.dataGridView1.Size = New System.Drawing.Size(416, 215)
Me.dataGridView1.TabIndex = 12
'
' nameDataGridViewTextBoxColumn
'
Me.nameDataGridViewTextBoxColumn.DataPropertyName = "Name"
Me.nameDataGridViewTextBoxColumn.FillWeight = 131.7987F
Me.nameDataGridViewTextBoxColumn.HeaderText = "Name"
Me.nameDataGridViewTextBoxColumn.Name = "nameDataGridViewTextBoxColumn"
'
' salaryDataGridViewTextBoxColumn
'
Me.salaryDataGridViewTextBoxColumn.DataPropertyName = "ParkingID"
Me.salaryDataGridViewTextBoxColumn.DefaultCellStyle = dataGridViewCellStyle2
Me.salaryDataGridViewTextBoxColumn.FillWeight = 121.8274F
Me.salaryDataGridViewTextBoxColumn.HeaderText = "Parking ID"
Me.salaryDataGridViewTextBoxColumn.Name = "salaryDataGridViewTextBoxColumn"
'
' iDDataGridViewTextBoxColumn
'
Me.iDDataGridViewTextBoxColumn.AutoSizeMode = System.Windows.Forms.DataGridViewAutoSizeColumnMode.Fill
Me.iDDataGridViewTextBoxColumn.DataPropertyName = "ID"
Me.iDDataGridViewTextBoxColumn.FillWeight = 46.37391F
Me.iDDataGridViewTextBoxColumn.HeaderText = "ID"
Me.iDDataGridViewTextBoxColumn.Name = "iDDataGridViewTextBoxColumn"
Me.iDDataGridViewTextBoxColumn.ReadOnly = True
'
' Form1
'
Me.AutoScaleDimensions = New System.Drawing.SizeF(6.0F, 13.0F)
Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
Me.ClientSize = New System.Drawing.Size(416, 266)
Me.Controls.Add(Me.dataGridView1)
Me.Controls.Add(Me.flowLayoutPanel1)
Me.Name = "Form1"

```

```
Me.Text = "IListSource Sample"
Me.flowLayoutPanel1.ResumeLayout(False)
Me.flowLayoutPanel1.PerformLayout()
CType(Me.dataGridView1, System.ComponentModel.ISupportInitialize).EndInit()
Me.ResumeLayout(False)
Me.PerformLayout()

End Sub

Shared Sub Main()
    Application.Run(New Form1())
End Sub

End Class
```

## 코드 컴파일

이 예제에는 다음 사항이 필요합니다.

- [System.Drawing](#) 및 [System.Windows.Forms](#) 어셈블리에 대한 참조

## 참고자료

- [IListSource](#)
- [ITypedList](#)
- [BindingList<T>](#)
- [IBindingList](#)
- [데이터 바인딩 및 Windows Forms](#)

# 방법: INotifyPropertyChanged 인터페이스 구현

2019-10-23 • 6 minutes to read • [Edit Online](#)

다음 코드 예제를 구현 하는 방법에 설명 합니다 [INotifyPropertyChanged](#) 인터페이스입니다. Windows Forms 데이터 바인딩에 사용 되는 비즈니스 개체에서 인터페이스를 구현 합니다. 구현 된 인터페이스 비즈니스 개체의 속성 변경 내용이 바인딩된 컨트롤에 통신 합니다.

## 예제

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Runtime.CompilerServices;
using System.Windows.Forms;

// Change the namespace to the project name.
namespace TestNotifyPropertyChangedCS
{
    // This form demonstrates using a BindingSource to bind
    // a list to a DataGridView control. The list does not
    // raise change notifications. However the DemoCustomer type
    // in the list does.
    public partial class Form1 : Form
    {
        // This button causes the value of a list element to be changed.
        private Button changeItemBtn = new Button();

        // This DataGridView control displays the contents of the list.
        private DataGridView customersDataGridView = new DataGridView();

        // This BindingSource binds the list to the DataGridView control.
        private BindingSource customersBindingSource = new BindingSource();

        public Form1()
        {
            InitializeComponent();

            // Set up the "Change Item" button.
            this.changeItemBtn.Text = "Change Item";
            this.changeItemBtn.Dock = DockStyle.Bottom;
            this.changeItemBtn.Click +=
                new EventHandler(changeItemBtn_Click);
            this.Controls.Add(this.changeItemBtn);

            // Set up the DataGridView.
            customersDataGridView.Dock = DockStyle.Top;
            this.Controls.Add(customersDataGridView);

            this.Size = new Size(400, 200);
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            // Create and populate the list of DemoCustomer objects
            // which will supply data to the DataGridView.
            BindingList<DemoCustomer> customerList = new BindingList<DemoCustomer>();
            customerList.Add(DemoCustomer.CreateNewCustomer());
            customerList.Add(DemoCustomer.CreateNewCustomer());
            customerList.Add(DemoCustomer.CreateNewCustomer());
        }
    }
}
```

```

        // Bind the list to the BindingSource.
        this.customersBindingSource.DataSource = customerList;

        // Attach the BindingSource to the DataGridView.
        this.customersDataGridView.DataSource =
            this.customersBindingSource;
    }

    // Change the value of the CompanyName property for the first
    // item in the list when the "Change Item" button is clicked.
    void changeItemBtn_Click(object sender, EventArgs e)
    {
        // Get a reference to the list from the BindingSource.
        BindingList<DemoCustomer> customerList =
            this.customersBindingSource.DataSource as BindingList<DemoCustomer>;

        // Change the value of the CompanyName property for the
        // first item in the list.
        customerList[0].CustomerName = "Tailspin Toys";
        customerList[0].PhoneNumber = "(708)555-0150";
    }
}

// This is a simple customer class that
// implements the IPropertyChange interface.
public class DemoCustomer : INotifyPropertyChanged
{
    // These fields hold the values for the public properties.
    private Guid idValue = Guid.NewGuid();
    private string customerNameValue = String.Empty;
    private string phoneNumberValue = String.Empty;

    public event PropertyChangedEventHandler PropertyChanged;

    // This method is called by the Set accessor of each property.
    // The CallerMemberName attribute that is applied to the optional propertyName
    // parameter causes the property name of the caller to be substituted as an argument.
    private void NotifyPropertyChanged([CallerMemberName] String propertyName = "")
    {
        if (PropertyChanged != null)
        {
            {
                PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
            }
        }
    }

    // The constructor is private to enforce the factory pattern.
    private DemoCustomer()
    {
        customerNameValue = "Customer";
        phoneNumberValue = "(312)555-0100";
    }

    // This is the public factory method.
    public static DemoCustomer CreateNewCustomer()
    {
        return new DemoCustomer();
    }

    // This property represents an ID, suitable
    // for use as a primary key in a database.
    public Guid ID
    {
        get
        {
            {
                return this.idValue;
            }
        }
    }
}

```



```

        public string CustomerName
        {
            get
            {
                return this.customerNameValue;
            }

            set
            {
                if (value != this.customerNameValue)
                {
                    this.customerNameValue = value;
                    NotifyPropertyChanged();
                }
            }
        }

        public string PhoneNumber
        {
            get
            {
                return this.phoneNumberValue;
            }

            set
            {
                if (value != this.phoneNumberValue)
                {
                    this.phoneNumberValue = value;
                    NotifyPropertyChanged();
                }
            }
        }
    }
}

```

```

Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Drawing
Imports System.Runtime.CompilerServices
Imports System.Windows.Forms

' This form demonstrates using a BindingSource to bind
' a list to a DataGridView control. The list does not
' raise change notifications. However the DemoCustomer type
' in the list does.

Public Class Form1
    Inherits System.Windows.Forms.Form

    ' This button causes the value of a list element to be changed.
    Private changeItemBtn As New Button()

    ' This DataGridView control displays the contents of the list.
    Private customersDataGridView As New DataGridView()

    ' This BindingSource binds the list to the DataGridView control.
    Private customersBindingSource As New BindingSource()

    Public Sub New()
        InitializeComponent()

        ' Set up the "Change Item" button.
        Me.changeItemBtn.Text = "Change Item"
        Me.changeItemBtn.Dock = DockStyle.Bottom
        AddHandler Me.changeItemBtn.Click, AddressOf changeItemBtn_Click
        Me.Controls.Add(Me.changeItemBtn)

        ' Set up the DataGridView

```

```

        Set up the DataGridView.
        customersDataGridView.Dock = DockStyle.Top
        Me.Controls.Add(customersDataGridView)

        Me.Size = New Size(400, 200)
    End Sub

    Private Sub Form1_Load(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles Me.Load

        ' Create and populate the list of DemoCustomer objects
        ' which will supply data to the DataGridView.
        Dim customerList As New BindingList(Of DemoCustomer)

        customerList.Add(DemoCustomer.CreateNewCustomer())
        customerList.Add(DemoCustomer.CreateNewCustomer())
        customerList.Add(DemoCustomer.CreateNewCustomer())

        ' Bind the list to the BindingSource.
        Me.customersBindingSource.DataSource = customerList

        ' Attach the BindingSource to the DataGridView.
        Me.customersDataGridView.DataSource = Me.customersBindingSource
    End Sub

    ' This event handler changes the value of the CompanyName
    ' property for the first item in the list.
    Private Sub changeItemBtn_Click(ByVal sender As Object, ByVal e As EventArgs)
        ' Get a reference to the list from the BindingSource.
        Dim customerList As BindingList(Of DemoCustomer) = _
            CType(customersBindingSource.DataSource, BindingList(Of DemoCustomer))

        ' Change the value of the CompanyName property for the
        ' first item in the list.
        customerList(0).CustomerName = "Tailspin Toys"
        customerList(0).PhoneNumber = "(708)555-0150"
    End Sub
End Class

' This class implements a simple customer type
' that implements the IPropertyChange interface.
Public Class DemoCustomer
    Implements INotifyPropertyChanged

    ' These fields hold the values for the public properties.
    Private idValue As Guid = Guid.NewGuid()
    Private customerNameValue As String = String.Empty
    Private phoneNumberValue As String = String.Empty

    Public Event PropertyChanged As PropertyChangedEventHandler _
        Implements INotifyPropertyChanged.PropertyChanged

    ' This method is called by the Set accessor of each property.
    ' The CallerMemberName attribute that is applied to the optional propertyName
    ' parameter causes the property name of the caller to be substituted as an argument.
    Private Sub NotifyPropertyChanged(<CallerMemberName()> Optional ByVal propertyName As String = Nothing)
        RaiseEvent PropertyChanged(Me, New PropertyChangedEventArgs(propertyName))
    End Sub

    ' The constructor is private to enforce the factory pattern.
    Private Sub New()
        customerNameValue = "Customer"
        phoneNumberValue = "(312)555-0100"
    End Sub

    ' This is the public factory method.
    Public Shared Function CreateNewCustomer() As DemoCustomer
        Return New DemoCustomer()
    End Function
End Class

```

```

' This property represents an ID, suitable
' for use as a primary key in a database.
Public ReadOnly Property ID() As Guid
    Get
        Return Me.idValue
    End Get
End Property

Public Property CustomerName() As String
    Get
        Return Me.customerNameValue
    End Get

    Set(ByVal value As String)
        If Not (value = customerNameValue) Then
            Me.customerNameValue = value
            NotifyPropertyChanged()
        End If
    End Set
End Property

Public Property PhoneNumber() As String
    Get
        Return Me.phoneNumberValue
    End Get

    Set(ByVal value As String)
        If Not (value = phoneNumberValue) Then
            Me.phoneNumberValue = value
            NotifyPropertyChanged()
        End If
    End Set
End Property
End Class

```

## 참고자료

- 방법: [PropertyNameChanged](#) 패턴 적용
- [Windows Forms 데이터 바인딩](#)
- 방법: [BindingSource](#)와 [INotifyPropertyChanged](#) 인터페이스를 사용 하여 변경 알림 발생
- [Windows Forms 데이터 바인딩의 변경 알림](#)

# 방법: IListed 인터페이스 구현

2019-10-23 • 10 minutes to read • [Edit Online](#)

구현된 `IListed` 인터페이스 바인딩 가능한 목록에 대한 스키마 검색을 사용하도록 설정합니다.

## 예제

다음 코드 예제를 구현하는 방법에 설명합니다 `IListed` 인터페이스입니다. 라는 제네릭 형식은

`SortableBindingList` 에서 파생되는 `BindingList<T>` 클래스를 구현합니다 `IListed` 인터페이스입니다. 이라는 간단한 클래스 `Customer` 의 헤더에 바인딩되는 데이터를 제공된 `DataGridView` 제어 합니다.

```

using System;
using System.Collections.Generic;
using System.Text;
using System.ComponentModel;
using System.Windows.Forms;
using System.Collections;
using System.Reflection;

namespace ITypedListCS
{
    [Serializable()]
    public class SortableBindingList<T> : BindingList<T>, ITypedList
    {
        [NonSerialized()]
        private PropertyDescriptorCollection properties;

        public SortableBindingList() : base()
        {
            // Get the 'shape' of the list.
            // Only get the public properties marked withBrowsable = true.
            PropertyDescriptorCollection pdc = TypeDescriptor.GetProperties(
                typeof(T),
                new Attribute[] { new BrowsableAttribute(true) });

            // Sort the properties.
            properties = pdc.Sort();
        }

        #region ITypedList Implementation

        public PropertyDescriptorCollection GetItemProperties(PropertyDescriptor[] listAccessors)
        {
            PropertyDescriptorCollection pdc;

            if (listAccessors != null && listAccessors.Length > 0)
            {
                // Return child list shape.
                pdc = ListBindingHelper.GetListItemProperties(listAccessors[0].PropertyType);
            }
            else
            {
                // Return properties in sort order.
                pdc = properties;
            }

            return pdc;
        }

        // This method is only used in the design-time framework
        // and by the obsolete DataGrid control.
        public string GetListName(PropertyDescriptor[] listAccessors)
        {
            return typeof(T).Name;
        }

        #endregion
    }
}

```

```

Imports System.ComponentModel
Imports System.Collections.Generic
Imports System.Windows.Forms

<Serializable()> _
Public Class SortableBindingList(Of TKey)
    Inherits BindingList(Of TKey)
    Implements IList

    <NonSerialized()> _
    Private properties As PropertyDescriptorCollection

    Public Sub New()
        MyBase.New()

        ' Get the 'shape' of the list.
        ' Only get the public properties marked withBrowsable = true.
        Dim pdc As PropertyDescriptorCollection = TypeDescriptor.GetProperties(GetType(TKey), New Attribute()
{New BrowsableAttribute(True)})

        ' Sort the properties.
        properties = pdc.Sort()

    End Sub

#Region "IList Implementation"

    Public Function GetItemProperties(ByVal listAccessors() As System.ComponentModel.PropertyDescriptor) As
System.ComponentModel.PropertyDescriptorCollection Implements
System.ComponentModel.IList.GetItemProperties

        Dim pdc As PropertyDescriptorCollection

        If (Not (listAccessors Is Nothing)) And (listAccessors.Length > 0) Then
            ' Return child list shape
            pdc = ListBindingHelper.GetListItemProperties(listAccessors(0).PropertyType)
        Else
            ' Return properties in sort order
            pdc = properties
        End If

        Return pdc

    End Function

    ' This method is only used in the design-time framework
    ' and by the obsolete DataGridView control.
    Public Function GetListName( _
        ByVal listAccessors() As PropertyDescriptor) As String _
        Implements System.ComponentModel.IList.GetListName

        Return GetType(TKey).Name

    End Function

#End Region

End Class

```

```

using System;
using System.Collections.Generic;
using System.Text;
using System.ComponentModel;

namespace IListCS
{

```

```

class Customer : INotifyPropertyChanged
{
    public Customer() {}

    public Customer(int id, string name, string company, string address, string city, string state, string
zip)
    {
        this._id = id;
        this._name = name;
        this._company = company;
        this._address = address;
        this._city = city;
        this._state = state;
        this._zip = zip;
    }

    #region Public Properties

    private int _id;

    public int ID
    {
        get { return _id; }
        set
        {
            if (_id != value)
            {
                _id = value;
                OnPropertyChanged(new PropertyChangedEventArgs("ID"));
            }
        }
    }

    private string _name;

    public string Name
    {
        get { return _name; }
        set
        {
            if (_name != value)
            {
                _name = value;
                OnPropertyChanged(new PropertyChangedEventArgs("Name"));
            }
        }
    }

    private string _company;

    public string Company
    {
        get { return _company; }
        set
        {
            if (_company != value)
            {
                _company = value;
                OnPropertyChanged(new PropertyChangedEventArgs("Company"));
            }
        }
    }

    private string _address;

    public string Address
    {
        get { return _address; }
        set
    }

```

```

        {
            if (_address != value)
            {
                _address = value;
                OnPropertyChanged(new PropertyChangedEventArgs("Address"));
            }
        }
    }

    private string _city;

    public string City
    {
        get { return _city; }
        set
        {
            if (_city != value)
            {
                _city = value;
                OnPropertyChanged(new PropertyChangedEventArgs("City"));
            }
        }
    }

    private string _state;

    public string State
    {
        get { return _state; }
        set
        {
            if (_state != value)
            {
                _state = value;
                OnPropertyChanged(new PropertyChangedEventArgs("State"));
            }
        }
    }

    private string _zip;

    public string ZipCode
    {
        get { return _zip; }
        set
        {
            if (_zip != value)
            {
                _zip = value;
                OnPropertyChanged(new PropertyChangedEventArgs("ZipCode"));
            }
        }
    }

    #endregion

    #region INotifyPropertyChanged Members

    public event PropertyChangedEventHandler PropertyChanged;

    protected virtual void OnPropertyChanged(PropertyChangedEventArgs e)
    {
        if (null != PropertyChanged)
        {
            PropertyChanged(this, e);
        }
    }

    #endregion

```



```
}  
}
```

```
Imports System.ComponentModel  
  
Public Class Customer  
    Implements INotifyPropertyChanged  
  
    Public Sub New()  
  
    End Sub  
  
    Public Sub New(ByVal id As Integer, ByVal name As String, ByVal company As String, ByVal address As  
String, ByVal city As String, ByVal state As String, ByVal zip As String)  
        Me._id = id  
        Me._name = name  
        Me._company = company  
        Me._address = address  
        Me._city = city  
        Me._state = state  
        Me._zip = zip  
  
    End Sub  
  
    #Region "Public Properties"  
  
    Private _id As Integer  
    Public Property ID() As Integer  
        Get  
            Return _id  
        End Get  
        Set(ByVal value As Integer)  
            If _id <> value Then  
                _id = value  
                OnPropertyChanged(New PropertyChangedEventArgs("ID"))  
            End If  
        End Set  
    End Property  
  
    Private _name As String  
  
    Public Property Name() As String  
        Get  
            Return _name  
        End Get  
        Set(ByVal value As String)  
            If _name <> value Then  
                _name = value  
                OnPropertyChanged(New PropertyChangedEventArgs("Name"))  
            End If  
        End Set  
    End Property  
  
    Private _company As String  
  
    Public Property Company() As String  
        Get  
            Return _company  
        End Get  
        Set(ByVal value As String)  
            If _company <> value Then  
                _company = value  
                OnPropertyChanged(New PropertyChangedEventArgs("Company"))  
            End If  
        End Set  
    End Property
```

```

Private _address As String

Public Property Address() As String
    Get
        Return _address
    End Get
    Set(ByVal value As String)
        If _address <> value Then
            _address = value
            OnPropertyChanged(New PropertyChangedEventArgs("Address"))
        End If
    End Set
End Property

```

```

Private _city As String

Public Property City() As String
    Get
        Return _city
    End Get
    Set(ByVal value As String)
        If _city <> value Then
            _city = value
            OnPropertyChanged(New PropertyChangedEventArgs("City"))
        End If
    End Set
End Property

```

```

Private _state As String

Public Property State() As String
    Get
        Return _state
    End Get
    Set(ByVal value As String)
        If _state <> value Then
            _state = value
            OnPropertyChanged(New PropertyChangedEventArgs("State"))
        End If
    End Set
End Property

```

```

Private _zip As String

Public Property ZipCode() As String
    Get
        Return _zip
    End Get
    Set(ByVal value As String)
        If _zip <> value Then
            _zip = value
            OnPropertyChanged(New PropertyChangedEventArgs("ZipCode"))
        End If
    End Set
End Property

```

```

#End Region

```

```

#Region "INotifyPropertyChanged Members"

```

```

    Public Event PropertyChanged(ByVal sender As Object, ByVal e As
System.ComponentModel.PropertyChangedEventArgs) Implements
System.ComponentModel.INotifyPropertyChanged.PropertyChanged

```

```

    Protected Overridable Sub OnPropertyChanged(ByVal e As PropertyChangedEventArgs)

```

```

        RaiseEvent PropertyChanged(Me, e)
    End Sub

#End Region

End Class

```

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace ITypedListCS
{
    public partial class Form1 : Form
    {
        private SortableBindingList<Customer> sortableBindingListOfCustomers;
        private BindingList<Customer> bindingListOfCustomers;

        private System.ComponentModel.IContainer components = null;
        private System.Windows.Forms.FlowLayoutPanel flowLayoutPanel1;
        private System.Windows.Forms.Label label2;
        private DataGridView dataGridView1;
        private Button button1;
        private Button button2;

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            this.sortableBindingListOfCustomers = new SortableBindingList<Customer>();
            this.bindingListOfCustomers = new BindingList<Customer>();

            this.dataGridView1.DataSource = this.bindingListOfCustomers;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            this.dataGridView1.DataSource = null;
            this.dataGridView1.DataSource = this.sortableBindingListOfCustomers;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            this.dataGridView1.DataSource = null;
            this.dataGridView1.DataSource = this.bindingListOfCustomers;
        }

        protected override void Dispose(bool disposing)
        {
            {
                if (disposing && (components != null))
                {
                    components.Dispose();
                }
                base.Dispose(disposing);
            }
        }

        #region Windows Form Designer generated code

        private void InitializeComponent()
        {

```

```

        System.ComponentModel.ComponentResourceManager resources = new
System.ComponentModel.ComponentResourceManager(typeof(Form1));
        this.flowLayoutPanel1 = new System.Windows.Forms.FlowLayoutPanel();
        this.label2 = new System.Windows.Forms.Label();
        this.dataGridView1 = new System.Windows.Forms.DataGridView();
        this.button1 = new System.Windows.Forms.Button();
        this.button2 = new System.Windows.Forms.Button();
        this.flowLayoutPanel1.SuspendLayout();
        ((System.ComponentModel.ISupportInitialize)(this.dataGridView1)).BeginInit();
        this.SuspendLayout();
        //
        // flowLayoutPanel1
        //
        this.flowLayoutPanel1.AutoSize = true;
        this.flowLayoutPanel1.Controls.Add(this.label2);
        this.flowLayoutPanel1.Dock = System.Windows.Forms.DockStyle.Top;
        this.flowLayoutPanel1.Location = new System.Drawing.Point(0, 0);
        this.flowLayoutPanel1.Name = "flowLayoutPanel1";
        this.flowLayoutPanel1.Size = new System.Drawing.Size(566, 51);
        this.flowLayoutPanel1.TabIndex = 13;
        //
        // label2
        //
        this.label2.AutoSize = true;
        this.label2.Location = new System.Drawing.Point(3, 6);
        this.label2.Margin = new System.Windows.Forms.Padding(3, 6, 3, 6);
        this.label2.Name = "label2";
        this.label2.Size = new System.Drawing.Size(558, 39);
        this.label2.TabIndex = 0;
        this.label2.Text = "This sample demonstrates how to implement the ITypedList interface. Clicking
on the 'Sort Columns' button will bind the DataGridView to a sub-classed BindingList<T> that implements
ITypedList to provide a sorted list of columns. Clicking on the 'Reset' button will bind the DataGridView to
a normal BindingList<T>.";
        //
        // dataGridView1
        //
        this.dataGridView1.AllowUserToAddRows = false;
        this.dataGridView1.AllowUserToDeleteRows = false;
        this.dataGridView1.Anchor = ((System.Windows.Forms.AnchorStyles)
((((System.Windows.Forms.AnchorStyles.Top | System.Windows.Forms.AnchorStyles.Bottom)
| System.Windows.Forms.AnchorStyles.Left)
| System.Windows.Forms.AnchorStyles.Right))));
        this.dataGridView1.AutoSizeColumnsMode =
System.Windows.Forms.DataGridViewAutoSizeColumnsMode.Fill;
        this.dataGridView1.ColumnHeadersHeightSizeMode =
System.Windows.Forms.DataGridViewColumnHeadersHeightSizeMode.AutoSize;
        this.dataGridView1.Location = new System.Drawing.Point(6, 57);
        this.dataGridView1.Name = "dataGridView1";
        this.dataGridView1.ReadOnly = true;
        this.dataGridView1.RowHeadersVisible = false;
        this.dataGridView1.Size = new System.Drawing.Size(465, 51);
        this.dataGridView1.TabIndex = 14;
        //
        // button1
        //
        this.button1.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Top
| System.Windows.Forms.AnchorStyles.Right)));
        this.button1.Location = new System.Drawing.Point(477, 57);
        this.button1.Name = "button1";
        this.button1.Size = new System.Drawing.Size(82, 23);
        this.button1.TabIndex = 15;
        this.button1.Text = "Sort Columns";
        this.button1.UseVisualStyleBackColor = true;
        this.button1.Click += new System.EventHandler(this.button1_Click);
        //
        // button2
        //
        this.button2.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Top
| System.Windows.Forms.AnchorStyles.Right)));

```

```

        this.button2.Location = new System.Drawing.Point(477, 86);
        this.button2.Name = "button2";
        this.button2.Size = new System.Drawing.Size(82, 23);
        this.button2.TabIndex = 16;
        this.button2.Text = "Reset";
        this.button2.UseVisualStyleBackColor = true;
        this.button2.Click += new System.EventHandler(this.button2_Click);
        //
        // Form1
        //
        this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
        this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
        this.ClientSize = new System.Drawing.Size(566, 120);
        this.Controls.Add(this.button2);
        this.Controls.Add(this.button1);
        this.Controls.Add(this.dataGridView1);
        this.Controls.Add(this.flowLayoutPanel1);
        this.Name = "Form1";
        this.Text = "ITypedList Sample";
        this.Load += new System.EventHandler(this.Form1_Load);
        this.flowLayoutPanel1.ResumeLayout(false);
        this.flowLayoutPanel1.PerformLayout();
        ((System.ComponentModel.ISupportInitialize)(this.dataGridView1)).EndInit();
        this.ResumeLayout(false);
        this.PerformLayout();

    }

    #endregion
}

static class Program
{
    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);
        Application.Run(new Form1());
    }
}
}

```

```

Imports System.ComponentModel
Imports System.Windows.Forms

Public Class Form1
    Inherits System.Windows.Forms.Form

    Friend WithEvents flowLayoutPanel1 As FlowLayoutPanel
    Friend WithEvents label2 As System.Windows.Forms.Label
    Friend WithEvents dataGridView1 As DataGridView
    Friend WithEvents button1 As Button
    Friend WithEvents button2 As Button

    Dim sortableBindingListOfCustomers As SortableBindingList(Of Customer)
    Dim bindingListOfCustomers As BindingList(Of Customer)

    Public Sub New()
        MyBase.New()

        Me.InitializeComponent()
    End Sub

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load

```

```

        sortableBindingListOfCustomers = New SortableBindingList(Of Customer)()
        bindingListOfCustomers = New BindingList(Of Customer)()

        Me.dataGridView1.DataSource = bindingListOfCustomers
    End Sub

    Private Sub button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
button1.Click
        Me.dataGridView1.DataSource = Nothing
        Me.dataGridView1.DataSource = sortableBindingListOfCustomers
    End Sub

    Private Sub button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
button2.Click
        Me.dataGridView1.DataSource = Nothing
        Me.dataGridView1.DataSource = bindingListOfCustomers
    End Sub

    'Form overrides dispose to clean up the component list.
    <System.Diagnostics.DebuggerNonUserCode()> _
    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
        If disposing AndAlso components IsNot Nothing Then
            components.Dispose()
        End If
        MyBase.Dispose(disposing)
    End Sub

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()> _
    Private Sub InitializeComponent()
        Dim resources As System.ComponentModel.ComponentResourceManager = New
System.ComponentModel.ComponentResourceManager(GetType(Form1))
        Me.flowLayoutPanel1 = New System.Windows.Forms.FlowLayoutPanel
        Me.label2 = New System.Windows.Forms.Label
        Me.dataGridView1 = New System.Windows.Forms.DataGridview
        Me.button1 = New System.Windows.Forms.Button
        Me.button2 = New System.Windows.Forms.Button
        Me.flowLayoutPanel1.SuspendLayout()
        CType(Me.dataGridView1, System.ComponentModel.ISupportInitialize).BeginInit()
        Me.SuspendLayout()
        '
        'flowLayoutPanel1
        '
        Me.flowLayoutPanel1.AutoSize = True
        Me.flowLayoutPanel1.Controls.Add(Me.label2)
        Me.flowLayoutPanel1.Dock = System.Windows.Forms.DockStyle.Top
        Me.flowLayoutPanel1.Location = New System.Drawing.Point(0, 0)
        Me.flowLayoutPanel1.Name = "flowLayoutPanel1"
        Me.flowLayoutPanel1.Size = New System.Drawing.Size(566, 51)
        Me.flowLayoutPanel1.TabIndex = 13
        '
        'label2
        '
        Me.label2.AutoSize = True
        Me.label2.Location = New System.Drawing.Point(3, 6)
        Me.label2.Margin = New System.Windows.Forms.Padding(3, 6, 3, 6)
        Me.label2.Name = "label2"
        Me.label2.Size = New System.Drawing.Size(558, 39)
        Me.label2.TabIndex = 0
        Me.label2.Text = "This sample demonstrates how to implement the ITypedList interface. Clicking on the
'Sort Columns' button will bind the DataGridView to a sub-classed BindingList<T> that implements ITypedlist to

```

Sort Columns button will bind the DataGridView to a sub-classed BindingList<T> that implements ITypedList to provide a sorted list of columns. Clicking on the 'Reset' button will bind the DataGridView to a normal BindingList<T>."

```
,
    'dataGridView1
    ,
    Me.dataGridView1.AllowUserToAddRows = False
    Me.dataGridView1.AllowUserToDeleteRows = False
    Me.dataGridView1.Anchor = CType((((System.Windows.Forms.AnchorStyles.Top Or
System.Windows.Forms.AnchorStyles.Bottom) _
        Or System.Windows.Forms.AnchorStyles.Left) _
        Or System.Windows.Forms.AnchorStyles.Right), System.Windows.Forms.AnchorStyles)
    Me.dataGridView1.AutoSizeColumnsMode = System.Windows.Forms.DataGridViewAutoSizeColumnsMode.Fill
    Me.dataGridView1.ColumnHeadersHeightSizeMode =
System.Windows.Forms.DataGridViewColumnHeadersHeightSizeMode.AutoSize
    Me.dataGridView1.Location = New System.Drawing.Point(6, 57)
    Me.dataGridView1.Name = "dataGridView1"
    Me.dataGridView1.ReadOnly = True
    Me.dataGridView1.RowHeadersVisible = False
    Me.dataGridView1.Size = New System.Drawing.Size(465, 51)
    Me.dataGridView1.TabIndex = 14
    ,
    'button1
    ,
    Me.button1.Anchor = CType((System.Windows.Forms.AnchorStyles.Top Or
System.Windows.Forms.AnchorStyles.Right), System.Windows.Forms.AnchorStyles)
    Me.button1.Location = New System.Drawing.Point(477, 57)
    Me.button1.Name = "button1"
    Me.button1.Size = New System.Drawing.Size(82, 23)
    Me.button1.TabIndex = 15
    Me.button1.Text = "Sort Columns"
    Me.button1.UseVisualStyleBackColor = True
    ,
    'button2
    ,
    Me.button2.Location = New System.Drawing.Point(477, 86)
    Me.button2.Name = "button2"
    Me.button2.Size = New System.Drawing.Size(82, 23)
    Me.button2.TabIndex = 16
    Me.button2.Text = "Reset"
    Me.button2.UseVisualStyleBackColor = True
    ,
    'Form1
    ,
    Me.AutoScaleDimensions = New System.Drawing.SizeF(6.0!, 13.0!)
    Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
    Me.ClientSize = New System.Drawing.Size(566, 120)
    Me.Controls.Add(Me.button2)
    Me.Controls.Add(Me.button1)
    Me.Controls.Add(Me.dataGridView1)
    Me.Controls.Add(Me.flowLayoutPanel1)
    Me.Name = "Form1"
    Me.Text = "ITypedList Sample"
    Me.flowLayoutPanel1.ResumeLayout(False)
    Me.flowLayoutPanel1.PerformLayout()
    CType(Me.dataGridView1, System.ComponentModel.ISupportInitialize).EndInit()
    Me.ResumeLayout(False)
    Me.PerformLayout()
End Sub

Shared Sub Main()
    Application.Run(New Form1())
End Sub

End Class
```

## 코드 컴파일

이 예제에는 다음 사항이 필요합니다.

- `System.Drawing` 및 `System.Windows.Forms` 어셈블리에 대한 참조

## 참고자료

- [ITypedList](#)
- [BindingList<T>](#)
- [IBindingList](#)
- [데이터 바인딩 및 Windows Forms](#)



# 방법: Windows Forms에서 데이터 탐색

2020-02-03 • 6 minutes to read • [Edit Online](#)

Windows 응용 프로그램에서 데이터 원본의 레코드를 탐색 하는 가장 쉬운 방법은 [BindingSource](#) 구성 요소를 데이터 소스에 바인딩한 다음 [BindingSource](#)에 컨트롤을 바인딩하는 것입니다. 그런 다음 [MoveNext](#), [MoveLast](#), [MovePrevious](#) 및 [MoveFirst](#)같은 [BindingSource](#)에서 기본 제공 탐색 방법을 사용할 수 있습니다. 이러한 메서드를 사용 하여 [BindingSource](#)의 [Position](#) 및 [Current](#) 속성을 적절 하게 조정 합니다. [Position](#) 속성을 설정 하여 항목 을 찾고 현재 항목으로 설정할 수도 있습니다.

데이터 원본에서 위치를 증가 시키려면

- 바인딩된 데이터의 [BindingSource Position](#) 속성을 이동할 레코드 위치로 설정 합니다. 다음 예제에서는 [BindingSource](#)의 [MoveNext](#) 메서드를 사용 하여 `nextButton` 클릭 될 때 [Position](#) 속성을 늘리는 방법을 보여 줍니다. [BindingSource](#)은 `Northwind` 데이터 집합의 `Customers` 테이블과 연결 됩니다.

## NOTE

[Position](#) 속성을 첫 번째 또는 마지막 레코드 이외의 값으로 설정 하면 .NET Framework에서 위치를 목록 범위 밖의 값으로 설정 하는 것이 허용 되지 않으므로 오류가 발생 하지 않습니다. 응용 프로그램에서 첫 번째 또는 마지막 레코드를 넘어가는 지 여부를 확인 하는 것이 중요 한 경우에는 데이터 요소 수를 초과 하는지 여부를 테스트 하는 논리를 포함 합니다.

```
private void nextButton_Click(object sender, System.EventArgs e)
{
    this.customersBindingSource.MoveNext();
}
```

```
Private Sub nextButton_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles nextButton.Click
    Me.customersBindingSource.MoveNext()
End Sub
```

끝을 통과 했는지 여부를 확인 하려면

- [PositionChanged](#) 이벤트에 대한 이벤트 처리기를 만듭니다. 처리기에서 제안 된 위치 값이 실제 데이터 요소 수를 초과 했는지 여부를 테스트할 수 있습니다.

다음 예제에서는 마지막 데이터 요소에 도달 했는지 여부를 테스트 하는 방법을 보여 줍니다. 예제에서 마지막 요소에 있는 경우 폼의 다음 단추를 사용할 수 없습니다.

## NOTE

코드에서 탐색 하는 목록을 변경 하는 경우 사용자가 새 목록의 전체 길이를 검색할 수 있도록 다음 단추를 다시 사용 하도록 설정 해야 합니다. 또한 작업 중인 특정 [BindingSource](#)에 대 한 위의 [PositionChanged](#) 이벤트를 해당 이벤트 처리 메서드와 연결 해야 합니다. 다음은 [PositionChanged](#) 이벤트를 처리 하는 메서드의 예입니다.

```
void customersBindingSource_PositionChanged(object sender, EventArgs e)
{
    if (customersBindingSource.Position == customersBindingSource.Count - 1)
        nextButton.Enabled = false;
    else
        nextButton.Enabled = true;
}
```

```
Sub customersBindingSource_PositionChanged(ByVal sender As Object, _
    ByVal e As EventArgs)

    If customersBindingSource.Position = _
        customersBindingSource.Count - 1 Then
        nextButton.Enabled = False
    Else
        nextButton.Enabled = True
    End If
End Sub
```

항목을 찾아서 현재 항목으로 설정 하려면

1. 현재 항목으로 설정 하려는 레코드를 찾습니다. 데이터 원본이 [IBindingList](#)를 구현 하는 경우 [BindingSource](#)의 [Find](#) 메서드를 사용 하여이 작업을 수행할 수 있습니다. [IBindingList](#)를 구현 하는 데이터 원본의 몇 가지 예는 [BindingList<T>](#) 및 [DataView](#)입니다.

```
void findButton_Click(object sender, EventArgs e)
{
    int foundIndex = customersBindingSource.Find("CustomerID", "ANTON");
    customersBindingSource.Position = foundIndex;
}
```

```
Sub findButton_Click(ByVal sender As Object, ByVal e As EventArgs) _
    Handles findButton.Click
    Dim foundIndex As Integer = customersBindingSource.Find("CustomerID", _
        "ANTON")
    customersBindingSource.Position = foundIndex
End Sub
```

## 참고 항목

- [Windows Forms에서 지원하는 데이터 소스](#)
- [Windows Forms 데이터 바인딩의 변경 알림](#)
- [데이터 바인딩 및 Windows Forms](#)
- [Windows Forms 데이터 바인딩](#)

# Windows Forms 보안

2020-02-03 • 5 minutes to read • [Edit Online](#)

Windows Forms은 코드를 기반으로 하는 보안 모델을 제공 합니다. 코드를 실행 하는 사용자에게 관계 없이 코드에 대해 보안 수준이 설정 됩니다. 이는 컴퓨터 시스템에 이미 포함 되어 있을 수 있는 모든 보안 스키마에 추가 됩니다. 여기에는 브라우저에 있는 영역 기반 보안 (예: Internet Explorer에서 사용 가능한 영역 기반 보안) 또는 운영 체제 (예: Windows NT의 자격 증명 기반 보안)가 포함 될 수 있습니다.

## 섹션 내용

### [Windows Forms의 보안 개요](#)

응용 프로그램의 Windows Forms를 안전 하게 보호 하는 데 필요한 .NET Framework 보안 모델 및 기본 단계에 대해 간략하게 설명 합니다.

### [Windows Forms의 파일 및 데이터 액세스 추가 보안](#)

부분적으로 신뢰할 수 있는 환경에서 파일 및 데이터에 액세스 하는 방법을 설명 합니다.

### [Windows Forms의 인쇄 추가 보안](#)

부분적으로 신뢰할 수 있는 환경에서 인쇄 기능에 액세스 하는 방법을 설명 합니다.

### [Windows Forms의 추가 보안 고려 사항](#)

창 조작 수행, 클립보드 사용 및 부분 신뢰 환경에서 비관리 코드 호출을 설명 합니다.

## 관련 섹션

### [기본 보안 정책](#)

완전 신뢰, 로컬 인트라넷 및 인터넷 권한 집합에 부여 된 기본 사용 권한을 나열 합니다.

### [일반 보안 정책 관리](#)

.NET Framework 보안 정책 관리 및 권한 상승에 대 한 정보를 제공 합니다.

### [위험한 권한 및 정책 관리](#)

잠재적으로 보안 시스템을 우회할 수 있는 the.NET Framework 사용 권한에 대해 설명 합니다.

### [보안 코딩 지침](#)

.NET Framework에 대 한 코드를 안전 하게 작성 하기 위한 모범 사례를 설명 하는 항목으로 연결 합니다.

### [권한 요청](#)

런타임에 코드에서 실행 해야 하는 권한을 알 수 있도록 특성 사용에 대해 설명 합니다.

### [주요 보안 개념](#)

코드 보안의 기본 측면을 다루는 항목에 대 한 링크입니다.

### [코드 액세스 보안 기본 사항](#)

.NET Framework 실행 시간 보안 정책을 사용 하는 기본 사항에 대해 설명 합니다.

### [보안 정책을 수정할 시기 결정](#)

응용 프로그램이 기본 보안 정책에서 분기 되어야 하는 시기를 결정 하는 방법을 설명 합니다.

### [보안 정책 배포](#)

보안 정책 변경 내용을 배포 하는 가장 좋은 방법에 대해 설명 합니다.

# Windows Forms의 보안 개요

2020-02-03 • 27 minutes to read • [Edit Online](#)

.NET Framework 릴리스 전에 사용자의 컴퓨터에서 실행 되는 모든 코드에는 컴퓨터의 사용자에게 있는 리소스에 액세스 하는 것과 동일한 권한 또는 사용 권한이 있었습니다. 예를 들어 사용자가 파일 시스템에 액세스할 수 있는 경우 코드에서 파일 시스템에 액세스할 수 있었습니다. 사용자가 데이터베이스에 액세스할 수 있는 경우 코드에서 해당 데이터베이스에 액세스할 수 있었습니다. 이러한 권한 또는 사용 권한은 사용자가 명시적으로 로컬 컴퓨터에 설치한 실행 파일의 코드에는 적합할 수 있지만 인터넷 또는 로컬 인트라넷에서 들어오는 잠재적 악성 코드에는 적합하지 않을 수 있습니다. 이 코드는 권한 없이 사용자의 컴퓨터 리소스에 액세스할 수 없어야 합니다.

.NET Framework 코드 액세스 보안이라는 인프라를 도입 하여 해당 코드에 사용자가 보유 한 권한 또는 권한을 구별할 수 있도록 합니다. 기본적으로 인터넷 및 인트라넷에서 들어오는 코드는 부분 신뢰에서만 실행할 수 있습니다. 부분 신뢰에서는 애플리케이션에 일련의 제한이 적용됩니다. 특히, 애플리케이션의 로컬 하드 디스크 액세스가 제한되며 비관리 코드를 실행할 수 없습니다. .NET Framework는 해당 코드의 id를 기반으로 코드에 액세스할 수 있는 리소스를 제어 합니다. 여기에는 해당 코드의 출처, **강력한 이름의 어셈블리**가 있는지 여부, 인증서로 서명 되었는지 여부 등이 포함 됩니다.

Windows Forms 응용 프로그램을 배포 하는 데 사용 하는 ClickOnce 기술은 부분 신뢰, 완전 신뢰 또는 높은 권한으로 부분 신뢰에서 실행 되는 응용 프로그램을 보다 쉽게 개발할 수 있도록 도와줍니다. ClickOnce는 권한 상승 및 신뢰할 수 있는 응용 프로그램 배포와 같은 기능을 제공 하므로 응용 프로그램이 책임 방식으로 로컬 사용자로부터 완전 신뢰 또는 높은 권한을 요청할 수 있습니다.

## .NET Framework의 보안 이해

코드 액세스 보안을 통해 코드 발생 위치 및 코드 ID의 다른 측면에 따라 다양한 수준으로 코드를 신뢰할 수 있습니다. 공용 언어 런타임에서 보안 정책을 결정하는 데 사용하는 증거에 대한 자세한 내용은 [증거](#)를 참조하세요. 악성 코드로부터 컴퓨터 시스템을 보호하고 의도적으로 또는 실수로 보안이 손상되지 않도록 신뢰할 수 있는 코드를 보호합니다. 코드 액세스 보안을 통해 애플리케이션에 필요한 권한만 지정할 수 있으므로 애플리케이션이 수행할 수 있는 작업에 대한 제어도 강화됩니다. 코드 액세스 보안은 코드에서 단일 코드 액세스 보안 권한 검사를 수행하지 않는 경우에도 공용 언어 런타임을 대상으로 하는 모든 관리 코드에 영향을 줍니다. .NET Framework 보안에 대한 자세한 내용은 [주요 보안 개념](#) 및 [코드 액세스 보안 기본 사항](#)을 참조 하세요.

사용자가 웹 서버 또는 파일 공유에서 직접 Windows Forms 실행 파일을 실행하는 경우 애플리케이션에 부여되는 신뢰 수준은 코드가 상주하는 위치 및 시작 방법에 따라 달라집니다. 애플리케이션이 실행되면 자동으로 평가되고 공용 언어 런타임으로부터 명명된 권한 집합을 받습니다. 기본적으로 로컬 컴퓨터의 코드에는 완전 신뢰 권한 집합이 부여되고, 로컬 네트워크의 코드에는 로컬 인트라넷 권한 집합이 부여되며, 인터넷의 코드에는 인터넷 권한 집합이 부여됩니다.

### NOTE

.NET Framework 버전 1.0 서비스 팩 1 및 서비스 팩 2에서는 인터넷 영역 코드 그룹이 Nothing 권한 집합을 받습니다. .NET Framework의 다른 모든 릴리스에서는 인터넷 영역 코드 그룹이 인터넷 권한 집합을 받습니다.

각 권한 집합에 부여되는 기본 권한은 [기본 보안 정책](#) 항목에 나열됩니다. 받는 권한에 따라 애플리케이션이 올바르게 실행되거나 보안 예외를 생성합니다.

많은 Windows Forms 응용 프로그램은 ClickOnce를 사용 하여 배포 됩니다. ClickOnce 배포를 생성 하는 데 사용 되는 도구는 앞에서 설명한 것과 다른 보안 기본값을 갖습니다. 자세한 내용은 다음 설명을 참조하세요.

보안 정책을 수정할 수 있기 때문에 애플리케이션에 부여되는 실제 권한은 기본값과 다를 수 있습니다. 즉, 컴퓨터

에 따라 애플리케이션의 권한이 다를 수 있습니다.

## 보다 안전한 Windows Forms 애플리케이션 개발

보안은 애플리케이션 개발의 모든 단계에서 중요합니다. 먼저 [보안 코딩 지침](#)을 검토하고 따릅니다.

그런 다음 애플리케이션을 완전 신뢰로 실행해야 하는지 여부 또는 부분 신뢰로 실행해야 하는지 여부를 결정합니다. 완전 신뢰에서 애플리케이션을 실행하면 로컬 컴퓨터의 리소스에 더 쉽게 액세스할 수 있지만 보안 코딩 지침 항목에 따라 엄격하게 애플리케이션을 디자인 및 개발하지 않을 경우 애플리케이션 및 해당 사용자가 높은 보안 위험에 노출됩니다. 부분 신뢰로 애플리케이션을 실행하면 보다 안전한 애플리케이션을 쉽게 개발할 수 있고 많은 위험이 감소하지만 특정 기능을 구현하는 방법에서 추가 계획이 필요합니다.

부분 신뢰(즉, 인터넷 또는 로컬 인트라넷 권한 집합)를 선택하는 경우 이 환경에서 원하는 애플리케이션의 동작 방식을 결정합니다. Windows Forms는 부분 신뢰 환경에서 기능을 구현하는 보다 안전한 대체 방법을 제공합니다. 데이터 액세스와 같은 애플리케이션의 특정 부분을 부분 신뢰 및 완전 신뢰 환경 둘 다에 대해 다르게 디자인 및 작성할 수 있습니다. 애플리케이션 설정과 같은 일부 Windows Forms 기능은 부분 신뢰로 작동하도록 설계되었습니다. 자세한 내용은 [애플리케이션 설정 개요](#)를 참조하세요.

애플리케이션에 부분 신뢰에서 허용하는 것보다 많은 권한이 필요하지만 완전 신뢰로 실행하지 않으려는 경우 필요한 추가 권한만 어설선하는 동시에 부분 신뢰로 실행할 수 있습니다. 예를 들어 부분 신뢰로 실행하지만 사용자의 파일 시스템에 있는 디렉터리에 대한 읽기 전용 액세스 권한을 애플리케이션에 부여해야 경우 해당 디렉터리에 대해서만 [FileOPermission](#)을 요청할 수 있습니다. 올바르게 사용할 경우 이 접근 방식은 애플리케이션의 기능을 증가시키며 사용자의 보안 위험을 최소화할 수 있습니다.

부분 신뢰로 실행되는 애플리케이션을 개발할 때는 애플리케이션이 실행해야 하는 권한 및 애플리케이션이 선택적으로 사용할 수 있는 권한을 추적합니다. 모든 권한을 알고 난 후 애플리케이션 수준에서 권한에 대한 선언적 요청을 해야 합니다. 권한 요청은 응용 프로그램에 필요한 권한 및 특별히 원하지 않는 권한에 대한 .NET Framework 실행 시간을 알려 줍니다. 권한 요청에 대한 자세한 내용은 [권한 요청](#)을 참조하세요.

선택적 권한을 요청할 때는 애플리케이션에서 부여되지 않은 권한이 필요한 작업을 수행하는 경우 생성되는 보안 예외를 처리해야 합니다. [SecurityException](#)을 적절히 처리하면 애플리케이션이 계속 작동할 수 있습니다. 애플리케이션은 예외를 사용하여 사용자에게 기능에 사용할 수 없도록 설정할지 여부를 결정할 수 있습니다. 예를 들어 필요한 파일 권한이 부여되지 않은 경우 애플리케이션에서 저장 메뉴 옵션을 사용할 수 없습니다.

적절한 권한을 모두 어설선했는지 확인하기 어려운 경우도 있습니다. 예를 들어 화면에서 무해한 것처럼 보이는 메서드 호출이 실행 중 특정 지점에서 파일 시스템에 액세스할 수도 있습니다. 필요한 모든 권한으로 애플리케이션을 배포하지 않을 경우 데스크톱에서 디버깅할 때는 정상적으로 테스트되지만 배포 시 실패할 수 있습니다. .NET Framework 2.0 SDK 및 Visual Studio 2005 모두에는 응용 프로그램에 필요한 권한을 계산 하기 위한 도구인 MTEXE 명령줄 도구와 Visual Studio의 권한 계산 기능이 각각 포함 되어 있습니다.

다음 항목에서는 추가 Windows Forms 보안 기능을 설명합니다.

항목	DESCRIPTION
- <a href="#">Windows Forms의 파일 및 데이터 액세스 추가 보안</a>	부분 신뢰 환경에서 파일 및 데이터에 액세스하는 방법을 설명합니다.
- <a href="#">Windows Forms의 인쇄 추가 보안</a>	부분 신뢰 환경에서 인쇄 기능에 액세스하는 방법을 설명합니다.
- <a href="#">Windows Forms의 추가 보안 고려 사항</a>	창 조작 수행, 클립보드 사용 및 부분 신뢰 환경에서 비관리 코드 호출을 설명합니다.

적절한 권한으로 애플리케이션 배포

클라이언트 컴퓨터에 Windows Forms 응용 프로그램을 배포 하는 가장 일반적인 방법은 응용 프로그램을 실행 하는 데 필요한 모든 구성 요소를 설명 하는 배포 기술인 ClickOnce를 사용 하는 것입니다. ClickOnce는 매니페스트

라는 XML 파일을 사용 하여 응용 프로그램을 구성 하는 어셈블리 및 파일과 응용 프로그램에 필요한 권한을 설명 합니다.

ClickOnce에는 클라이언트 컴퓨터에서 상승 된 권한을 요청 하는 두 가지 기술이 있습니다. 두 기술은 모두 Authenticode 인증서를 사용합니다. 인증서는 애플리케이션이 신뢰할 수 있는 소스에서 제공되었다는 일부 보증을 사용자에게 제공합니다.

다음 표에서는 이러한 기술을 설명합니다.

높은 권한 기술	DESCRIPTION
권한 상승	애플리케이션을 처음 실행할 때 사용자에게 보안 대화 상자를 표시합니다. <b>권한 상승</b> 대화 상자는 사용자가 추가 신뢰를 부여할지 여부에 대해 합리적인 결정을 내릴 수 있도록 애플리케이션을 게시한 사람을 사용자에게 알립니다.
신뢰할 수 있는 애플리케이션 배포	시스템 관리자가 클라이언트 컴퓨터에서 게시자 Authenticode 인증서의 일회성 설치를 수행해야 합니다. 이때 부터 인증서로 서명된 애플리케이션은 신뢰할 수 있는 것으로 간주되며, 추가 확인 메시지 없이 로컬 컴퓨터에서 완전 신뢰로 실행될 수 있습니다.

선택하는 기술은 배포 환경에 따라 달라집니다. 자세한 내용은 [ClickOnce 배포 전략 선택](#)을 참조하세요.

기본적으로 Visual Studio 또는 .NET Framework SDK 도구 (Mage.exe 및 Mageui.exe) 중 하나를 사용 하여 배포 된 ClickOnce 응용 프로그램은 완전 신뢰가 있는 클라이언트 컴퓨터에서 실행 되도록 구성 됩니다. 부분 신뢰를 사용 하거나 일부 추가 권한만 사용하여 애플리케이션을 배포하는 경우 이 기본값을 변경해야 합니다. 배포를 구성할 때 Visual Studio 또는 .NET Framework SDK 도구 Mageui.exe를 사용 하여이 작업을 수행할 수 있습니다.

Mageui.exe를 사용 하는 방법에 대 한 자세한 내용은 [연습: ClickOnce 응용 프로그램 수동 배포](#)를 참조 하세요. 또한 [방법: ClickOnce 애플리케이션에 대한 사용자 지정 권한 설정](#) 또는 [방법: ClickOnce 애플리케이션에 대한 사용자 지정 권한 설정](#)을 참조하세요.

ClickOnce 및 권한 상승의 보안 측면에 대 한 자세한 내용은 [Clickonce 응용 프로그램 보안](#)을 참조 하세요. 신뢰할 수 있는 애플리케이션 배포에 대한 자세한 내용은 [신뢰할 수 있는 애플리케이션 배포 개요](#)를 참조하세요.

#### 애플리케이션 테스트

Visual Studio를 사용 하여 Windows Forms 응용 프로그램을 배포한 경우 개발 환경에서 부분 신뢰 또는 제한 된 권한 집합으로 디버깅을 사용 하도록 설정할 수 있습니다. 또한 [방법: 제한 된 권한으로 ClickOnce 응용 프로그램 디버깅](#)을 참조 하세요.

## 참고 항목

- [Windows Forms 보안](#)
- [코드 액세스 보안 기본 사항](#)
- [ClickOnce 보안 및 배포](#)
- [신뢰할 수 있는 애플리케이션 배포 개요](#)
- [Mage.exe\(매니페스트 생성 및 편집 도구\)](#)
- [MageUI.exe\(매니페스트 생성 및 편집 도구, 그래픽 클라이언트\)](#)

# Windows Forms의 파일 및 데이터 액세스 추가 보안

2020-03-21 • 22 minutes to read • [Edit Online](#)

.NET Framework는 사용 권한을 사용하여 리소스와 데이터를 보호합니다. 애플리케이션이 데이터를 일거나 쓸 수 있는 위치는 애플리케이션에 부여된 권한에 따라 달라집니다. 애플리케이션이 부분 신뢰 환경에서 실행되는 경우 데이터에 대한 액세스 권한이 없거나 데이터에 액세스하는 방법을 변경해야 할 수 있습니다.

보안 제한이 발생할 경우 다음 두가지 옵션이 있습니다. 권한을 어설션(애플리케이션에 부여되었다고 가정)하거나 부분 신뢰에서 작동하도록 작성된 기능 버전을 사용합니다. 다음 섹션에서는 부분 신뢰 환경에서 실행 중인 애플리케이션에서 파일, 데이터베이스 및 레지스트리 액세스 작업을 수행하는 방법을 설명합니다.

## NOTE

기본적으로 ClickOnce 배포를 생성하는 도구는 이러한 배포를 실행하는 컴퓨터에서 전체 신뢰 요청으로 기본설정합니다. 부분 신뢰에서 실행의 추가 보안 이점을 원하는 경우 Visual Studio 또는 Windows SDK 도구(Mage.exe 또는 MageUI.exe) 중 하나에서 이 기본값을 변경해야 합니다. Windows Forms 보안 및 응용 프로그램에 대한 적절한 신뢰 수준을 확인하는 방법에 대한 자세한 내용은 [Windows Forms 개요의 보안](#)을 참조하십시오.

## 파일 액세스

클래스는 [FileIOPermission](#).NET 프레임워크에서 파일 및 폴더 액세스를 제어합니다. 기본적으로 보안 시스템은 로컬 인트라넷 및 인터넷 영역과 같은 부분 신뢰 환경에 [FileIOPermission](#)을 부여하지 않습니다. 그러나 애플리케이션의 디자인을 수정하거나 다른 메서드를 사용하여 파일에 액세스하는 경우 파일 액세스가 필요한 애플리케이션이 여전히 이러한 환경에서 작동할 수 있습니다. 기본적으로 로컬 인트라넷 영역에는 동일한 사이트 액세스 및 동일한 디렉터리 액세스를 포함하고, 원본 사이트에 다시 연결하고, 설치 디렉터리에서 읽을 수 있는 권한이 부여됩니다. 기본적으로 인터넷 영역에는 원본 사이트에 다시 연결할 수 있는 권한만 부여됩니다.

### 사용자 지정 파일

파일 액세스 권한 없음을 처리하는 한 가지 방법은 [OpenFileDialog](#) 또는 [SaveFileDialog](#) 클래스를 통해 특정 파일 정보를 제공하라는 메시지를 사용자에게 표시하는 것입니다. 이 사용자 상호 작용은 애플리케이션이 악의적으로 개인 파일을 로드하거나 중요한 파일을 덮어쓸 수 없도록 합니다. [OpenFile](#) 및 [OpenFile](#) 메서드는 사용자가 지정한 파일에 대한 파일 스트림을 열어 읽기 및 쓰기 파일 액세스를 제공합니다. 메서드는 파일 경로가 표시되지 않도록 하여 사용자 파일 보호를 돕습니다.

## NOTE

이러한 권한은 애플리케이션이 인터넷 영역 또는 인트라넷 영역에 있는지에 따라 달라집니다. 인터넷 영역 애플리케이션은 [OpenFileDialog](#)만 사용할 수 있는 반면 인트라넷 애플리케이션은 무제한 파일 대화 상자 권한이 있습니다.

[FileDialogPermission](#) 클래스는 애플리케이션에서 사용할 수 있는 파일 형식 대화 상자를 지정합니다. 다음 표에서는 각 [FileDialog](#) 클래스를 사용하는 데 필요한 값을 보여 줍니다.

클래스	필요한 액세스 값
<a href="#">OpenFileDialog</a>	<a href="#">Open</a>
<a href="#">SaveFileDialog</a>	<a href="#">Save</a>

#### NOTE

[OpenFile](#) 메서드가 실제로 호출될 때까지 특정 권한이 요청되지 않습니다.

파일 대화 상자를 표시할 수 있는 권한은 애플리케이션에 [FileDialog](#), [OpenFileDialog](#) 및 [SaveFileDialog](#) 클래스의 모든 멤버에 대한 모든 권한을 부여하지 않습니다. 각 메서드를 호출하는 데 필요한 정확한 사용 권한은 .NET Framework 클래스 라이브러리 설명서에서 해당 메서드에 대한 참조 항목을 참조하십시오.

다음 코드 예제에서는 [OpenFile](#) 메서드를 사용하여 사용자 지정 파일을 [RichTextBox](#) 컨트롤에 엽니다. 예제를 사용하려면 [FileDialogPermission](#) 및 연결된 [Open](#) 열거형 값이 필요합니다. 예제에서는 [SecurityException](#)을 처리하여 저장 기능을 사용하지 않도록 설정할지 여부를 확인하는 방법을 보여 줍니다. 이 예제를 사용하려면 [Form](#)에 [ButtonOpen](#)이라는 [Button](#) 컨트롤과 [RtfBoxMain](#)이라는 [RichTextBox](#) 컨트롤이 있어야 합니다.

#### NOTE

저장 기능에 대한 프로그래밍 논리는 예제에 표시되어 있지 않습니다.

```
Private Sub ButtonOpen_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles ButtonOpen.Click

    Dim editingFileName as String = ""
    Dim saveAllowed As Boolean = True

    ' Displays the OpenFileDialog.
    If (OpenFileDialog1.ShowDialog() = DialogResult.OK) Then
        Dim userStream as System.IO.Stream
        Try
            ' Opens the file stream for the file selected by the user.
            userStream =OpenFileDialog1.OpenFile()
            Me.RtfBoxMain.LoadFile(userStream, _
                RichTextBoxStreamType.PlainText)
        Finally
            userStream.Close()
        End Try

        ' Tries to get the file name selected by the user.
        ' Failure means that the application does not have
        ' unrestricted permission to the file.
        Try
            editingFileName = OpenFileDialog1.FileName
        Catch ex As Exception
            If TypeOf ex Is System.Security.SecurityException Then
                ' The application does not have unrestricted permission
                ' to the file so the save feature will be disabled.
                saveAllowed = False
            Else
                Throw ex
            End If
        End Try
    End If
End Sub
```



```

private void ButtonOpen_Click(object sender, System.EventArgs e)
{
    String editingFileName = "";
    Boolean saveAllowed = true;

    // Displays the OpenFileDialog.
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        // Opens the file stream for the file selected by the user.
        using (System.IO.Stream userStream = openFileDialog1.OpenFile())
        {
            this.RtfBoxMain.LoadFile(userStream,
                RichTextBoxStreamType.PlainText);
            userStream.Close();
        }

        // Tries to get the file name selected by the user.
        // Failure means that the application does not have
        // unrestricted permission to the file.
        try
        {
            editingFileName = openFileDialog1.FileName;
        }
        catch (Exception ex)
        {
            if (ex is System.Security.SecurityException)
            {
                // The application does not have unrestricted permission
                // to the file so the save feature will be disabled.
                saveAllowed = false;
            }
            else
            {
                throw ex;
            }
        }
    }
}

```

#### NOTE

Visual C#에서 이벤트 처리기를 사용하도록 코드를 추가해야 합니다. 이전 예제의 코드를 사용하여 다음 코드에서는 이벤트 처리기 `this.ButtonOpen.Click += new System.Windows.Forms.EventHandler(this.ButtonOpen_Click);` 를 사용하도록 설정하는 방법을 보여 줍니다.

### Other Files

애플리케이션 설정을 유지해야 하는 경우와 같이 사용자가 지정하지 않는 파일을 읽거나 써야 하는 경우가 있습니다. 로컬 인트라넷 및 인터넷 영역에서는 로컬 파일에 데이터를 저장할 수 있는 권한이 애플리케이션에 없습니다. 그러나 애플리케이션은 격리된 스토리지에 데이터를 저장할 수 있습니다. 격리된 스토리지는 데이터가 저장된 실제 디렉터리 위치를 포함하는 하나 이상의 격리된 스토리지 파일(스토리지라고 함)이 포함된 추상 데이터 컴파트먼트(특정 스토리지 위치가 아님)입니다. [FileOPermission](#)과 같은 파일 액세스 권한은 필요하지 않습니다. 대신, [IsolatedStoragePermission](#) 클래스는 격리된 스토리지에 대한 권한을 제어합니다. 기본적으로 로컬 인트라넷 및 인터넷 영역에서 실행 중인 애플리케이션은 격리된 스토리지를 사용하여 데이터를 저장할 수 있습니다. 그러나 디스크 할당량과 같은 설정이 달라질 수 있습니다. 격리된 저장소에 대한 자세한 내용은 [격리된 저장소](#)를 참조하십시오.

다음 예제에서는 격리된 스토리지를 사용하여 스토리지에 있는 파일에 데이터를 씁니다. 예제를 사용하려면 [IsolatedStorageFilePermission](#) 및 [DomainIsolationByUser](#) 열거형 값이 필요합니다. 예제에서는 [Button](#) 컨트롤의 특정 속성 값을 읽고 격리된 스토리지의 파일에 쓰는 방법을 보여 줍니다. `Read` 함수는 애플리케이션이 시작된 후에 호출되고 `Write` 함수는 애플리케이션이 종료되기 전에 호출됩니다. 이 예제에서는 `Read` 및 `Write` 함수가

라는 **Form Button** MainButton 컨트롤을 포함하는 a의 멤버로 존재하도록 요구합니다.

```
' Reads the button options from the isolated storage. Uses Default values
' for the button if the options file does not exist.
Public Sub Read()
    Dim isoStore As System.IO.IsolatedStorage.IsolatedStorageFile = _
        System.IO.IsolatedStorage.IsolatedStorageFile. _
            GetUserStoreForDomain()

    Dim filename As String = "options.txt"
    Try
        ' Checks to see if the options.txt file exists.
        If (isoStore.GetFileNames(filename).GetLength(0) <> 0) Then

            ' Opens the file because it exists.
            Dim isos As New System.IO.IsolatedStorage.IsolatedStorageFileStream _
                (filename, IO.FileMode.Open, isoStore)
            Dim reader As System.IO.StreamReader
            Try
                reader = new System.IO.StreamReader(isos)

                ' Reads the values stored.
                Dim converter As System.ComponentModel.TypeConverter
                converter = System.ComponentModel.TypeDescriptor.GetConverter _
                    (GetType(Color))

                Me.MainButton.BackColor = _
                    CType(converter.ConvertFromString _
                        (reader.ReadLine()), Color)
                Me.MainButton.ForeColor = _
                    CType(converter.ConvertFromString _
                        (reader.ReadLine()), Color)

                converter = System.ComponentModel.TypeDescriptor.GetConverter _
                    (GetType(Font))
                Me.MainButton.Font = _
                    CType(converter.ConvertFromString _
                        (reader.ReadLine()), Font)

            Catch ex As Exception
                Debug.WriteLine("Cannot read options " + _
                    ex.ToString())
            Finally
                reader.Close()
            End Try
        End If

    Catch ex As Exception
        Debug.WriteLine("Cannot read options " + ex.ToString())
    End Try
End Sub

' Writes the button options to the isolated storage.
Public Sub Write()
    Dim isoStore As System.IO.IsolatedStorage.IsolatedStorageFile = _
        System.IO.IsolatedStorage.IsolatedStorageFile. _
            GetUserStoreForDomain()

    Dim filename As String = "options.txt"
    Try
        ' Checks if the file exists, and if it does, tries to delete it.
        If (isoStore.GetFileNames(filename).GetLength(0) <> 0) Then
            isoStore.DeleteFile(filename)
        End If
    Catch ex As Exception
        Debug.WriteLine("Cannot delete file " + ex.ToString())
    End Try
End Sub
```

```

' Creates the options.txt file and writes the button options to it.
Dim writer As System.IO.StreamWriter
Try
    Dim isos As New System.IO.IsolatedStorage.IsolatedStorageFileStream _
        (filename, IO.FileMode.CreateNew, isoStore)

    writer = New System.IO.StreamWriter(isos)
    Dim converter As System.ComponentModel.TypeConverter

    converter = System.ComponentModel.TypeDescriptor.GetConverter _
        (GetType(Color))
    writer.WriteLine(converter.ConvertToString( _
        Me.MainButton.BackColor))
    writer.WriteLine(converter.ConvertToString( _
        Me.MainButton.ForeColor))

    converter = System.ComponentModel.TypeDescriptor.GetConverter _
        (GetType(Font))
    writer.WriteLine(converter.ConvertToString( _
        Me.MainButton.Font))

Catch ex As Exception
    Debug.WriteLine("Cannot write options " + ex.ToString())

Finally
    writer.Close()
End Try
End Sub

```

```

// Reads the button options from the isolated storage. Uses default values
// for the button if the options file does not exist.
public void Read()
{
    System.IO.IsolatedStorage.IsolatedStorageFile isoStore =
        System.IO.IsolatedStorage.IsolatedStorageFile.
        GetUserStoreForDomain();

    string filename = "options.txt";
    try
    {
        // Checks to see if the options.txt file exists.
        if (isoStore.GetFileNames(filename).GetLength(0) != 0)
        {
            // Opens the file because it exists.
            System.IO.IsolatedStorage.IsolatedStorageFileStream isos =
                new System.IO.IsolatedStorage.IsolatedStorageFileStream
                    (filename, System.IO.FileMode.Open, isoStore);
            System.IO.StreamReader reader = null;
            try
            {
                reader = new System.IO.StreamReader(isos);

                // Reads the values stored.
                TypeConverter converter ;
                converter = TypeDescriptor.GetConverter(typeof(Color));

                this.MainButton.BackColor =
                    (Color)(converter.ConvertFromString(reader.ReadLine()));
                this.MainButton.ForeColor =
                    (Color)(converter.ConvertFromString(reader.ReadLine()));

                converter = TypeDescriptor.GetConverter(typeof(Font));
                this.MainButton.Font =
                    (Font)(converter.ConvertFromString(reader.ReadLine()));
            }
            catch (Exception ex)
            {

```

```

        System.Diagnostics.Debug.WriteLine
            ("Cannot read options " + ex.ToString());
    }
    finally
    {
        reader.Close();
    }
}
}
catch (Exception ex)
{
    System.Diagnostics.Debug.WriteLine
        ("Cannot read options " + ex.ToString());
}
}

// Writes the button options to the isolated storage.
public void Write()
{
    System.IO.IsolatedStorage.IsolatedStorageFile isoStore =
        System.IO.IsolatedStorage.IsolatedStorageFile.
            GetUserStoreForDomain();

    string filename = "options.txt";
    try
    {
        // Checks if the file exists and, if it does, tries to delete it.
        if (isoStore.GetFileNames(filename).GetLength(0) != 0)
        {
            isoStore.DeleteFile(filename);
        }
    }
    catch (Exception ex)
    {
        System.Diagnostics.Debug.WriteLine
            ("Cannot delete file " + ex.ToString());
    }

    // Creates the options file and writes the button options to it.
    System.IO.StreamWriter writer = null;
    try
    {
        System.IO.IsolatedStorage.IsolatedStorageFileStream isos = new
            System.IO.IsolatedStorage.IsolatedStorageFileStream(filename,
                System.IO.FileMode.CreateNew, isoStore);

        writer = new System.IO.StreamWriter(isos);
        TypeConverter converter ;

        converter = TypeDescriptor.GetConverter(typeof(Color));
        writer.WriteLine(converter.ConvertToString(
            this.MainButton.BackColor));
        writer.WriteLine(converter.ConvertToString(
            this.MainButton.ForeColor));

        converter = TypeDescriptor.GetConverter(typeof(Font));
        writer.WriteLine(converter.ConvertToString(
            this.MainButton.Font));
    }
    catch (Exception ex)
    {
        System.Diagnostics.Debug.WriteLine
            ("Cannot write options " + ex.ToString());
    }
    finally
    {
        writer.Close();
    }
}

```

```
}
```

## 데이터베이스 액세스

데이터베이스에 액세스하는 데 필요한 권한은 데이터베이스 공급자에 따라 다릅니다. 그러나 적절한 권한으로 실행 중인 애플리케이션만 데이터 연결을 통해 데이터베이스에 액세스할 수 있습니다. 데이터베이스에 액세스하는 데 필요한 권한에 대한 자세한 내용은 [코드 액세스 보안 및 ADO.NET](#)을 참조하십시오.

부분 신뢰로 애플리케이션을 실행하려 하므로 데이터베이스에 직접 액세스할 수 없는 경우 데이터에 액세스하는 대체 방법으로 웹 서비스를 사용할 수 있습니다. 웹 서비스는 네트워크를 통해 프로그래밍 방식으로 액세스할 수 있는 소프트웨어입니다. 웹 서비스를 통해 애플리케이션은 코드 그룹 영역 간에 데이터를 공유할 수 있습니다. 기본적으로 로컬 인트라넷 및 인터넷 영역의 애플리케이션에는 동일한 서버에서 호스트된 웹 서비스를 호출할 수 있게 해주는 원본 사이트 액세스 권한이 부여됩니다. 자세한 내용은 [ASP.NET AJAX](#) 또는 [Windows 통신 재단의 웹 서비스](#)를 참조하십시오.

## 레지스트리 액세스

[RegistryPermission](#) 클래스는 운영 체제 레지스트리에 대한 액세스를 제어합니다. 기본적으로 로컬에서 실행 중인 애플리케이션만 레지스트리에 액세스할 수 있습니다. [RegistryPermission](#)은 레지스트리 액세스 시도 권한만 애플리케이션에 부여합니다. 운영 체제가 여전히 레지스트리에 대한 보안을 적용하므로 액세스가 성공한다는 보장은 없습니다.

부분 신뢰에서는 레지스트리에 액세스할 수 없으므로 다른 데이터 저장 방법을 찾아야 할 수도 있습니다. 애플리케이션 설정을 저장하는 경우 레지스트리 대신 격리된 스토리지를 사용합니다. 격리된 스토리지를 사용하여 다른 애플리케이션 관련 파일을 저장할 수도 있습니다. 또한 기본적으로 애플리케이션에 원본 사이트 액세스 권한이 부여되므로 서버 또는 원본 사이트에 대한 전역 애플리케이션 정보를 저장할 수 있습니다.

## 참고 항목

- [Windows Forms의 인쇄 추가 보안](#)
- [Windows Forms의 추가 보안 고려 사항](#)
- [Windows Forms의 보안 개요](#)
- [Windows Forms 보안](#)
- [Mage.exe\(매니페스트 생성 및 편집 도구\)](#)
- [MageUI.exe\(매니페스트 생성 및 편집 도구, 그래픽 클라이언트\)](#)

# Windows Forms의 인쇄 추가 보안

2020-02-03 • 3 minutes to read • [Edit Online](#)

Windows Forms 응용 프로그램에는 인쇄 기능이 포함 되는 경우가 많습니다. .NET Framework는 [PrintingPermission](#) 클래스를 사용 하여 인쇄 기능 및 연결 된 [PrintingPermissionLevel](#) 열거형 값에 대 한 액세스를 제어 하 여 액세스 수준을 표시 합니다. 기본적으로 인쇄는 로컬 인트라넷 및 인터넷 영역에서 사용 하도록 설정 됩니다. 그러나 액세스 수준은 두 영역에서 모두 제한 됩니다. 응용 프로그램이 인쇄 하거나, 사용자 조작이 필요 하거나, 응용 프로그램에 부여 된 사용 권한 값에 따라 인쇄 될 수 없는지 여부입니다. 기본적으로 로컬 인트라넷 영역에는 [DefaultPrinting](#) 액세스 권한이 부여 되 고 인트라넷 영역에는 [SafePrinting](#) 액세스 권한이 부여 됩니다.

다음 표에서는 각 인쇄 권한 수준에서 사용할 수 있는 기능을 보여 줍니다.

PRINTINGPERMISSIONLEVEL	DESCRIPTION
<a href="#">AllPrinting</a>	설치 된 모든 프린터에 대 한 모든 권한을 제공 합니다.
<a href="#">DefaultPrinting</a>	기본 프린터를 프로그래밍 방식으로 인쇄 하 고 제한 된 인쇄 대화 상자를 통해 보다 안전 하 게 인쇄할 수 있습니다. <a href="#">DefaultPrinting</a> 은 <a href="#">AllPrinting</a> 의 하위 집합입니다.
<a href="#">SafePrinting</a>	는 제한 된 대화 상자 에서만 인쇄 기능을 제공 합니다. <a href="#">SafePrinting</a> 은 <a href="#">DefaultPrinting</a> 의 하위 집합입니다.
<a href="#">NoPrinting</a>	프린터에 액세스할 수 없도록 합니다. <a href="#">NoPrinting</a> 은 <a href="#">SafePrinting</a> 의 하위 집합입니다.

## 참고 항목

- [Windows Forms의 파일 및 데이터 액세스 추가 보안](#)
- [Windows Forms의 추가 보안 고려 사항](#)
- [Windows Forms의 보안 개요](#)
- [Windows Forms 보안](#)

# Windows Forms의 추가 보안 고려 사항

2020-02-03 • 21 minutes to read • [Edit Online](#)

보안 설정을 .NET Framework 하면 로컬 컴퓨터의 부분 신뢰 환경에서 응용 프로그램이 다르게 실행 될 수 있습니다. .NET Framework는 파일 시스템, 네트워크 및 관리 되지 않는 Api와 같은 중요 한 로컬 리소스에 대 한 액세스를 제한 합니다. 보안 설정은 Microsoft Windows API 또는 보안 시스템에서 확인할 수 없는 기타 Api를 호출 하는 기능에 영향을 줍니다. 또한 파일, 데이터 액세스, 인쇄를 비롯한 애플리케이션의 다른 측면에도 영향을 줍니다. 부분 신뢰 환경에서의 파일 및 데이터 액세스에 대한 자세한 내용은 [Windows Forms의 파일 및 데이터 액세스 추가 보안](#)을 참조하세요. 부분 신뢰 환경에서의 인쇄에 대한 자세한 내용은 [Windows Forms의 인쇄 추가 보안](#)을 참조하세요.

다음 섹션에서는 클립보드를 사용 하여 작업 하고, 창 조작을 수행 하고, 부분 신뢰 환경에서 실행 되는 응용 프로그램에서 Windows API를 호출 하는 방법을 설명 합니다.

## 클립보드 액세스

합니다 [UIPermission](#) 클래스를 클립보드에 연결 된 액세스 제어 [UIPermissionClipboard](#) 열거형 값에 대 한 액세스 수준을 나타냅니다. 다음 표에서는 가능한 권한 수준을 보여 줍니다.

UIPERMISSIONCLIPBOARD 값	DESCRIPTION
<a href="#">AllClipboard</a>	클립보드 사용에 제한을 받지 않습니다.
<a href="#">OwnClipboard</a>	약간의 제한을 받으면서 클립보드를 사용할 수 있습니다. 복사 또는 잘라내기 명령 작업과 같이 클립보드에 데이터를 넣는 기능은 제한을 받지 않습니다. 텍스트 상자와 같이 붙여넣기를 허용하는 내장 컨트롤은 클립보드 데이터를 받아들일 수 있지만 사용자 정의 컨트롤은 프로그래밍 방식으로 클립보드에서 데이터를 읽을 수는 없습니다.
<a href="#">NoClipboard</a>	클립보드를 사용할 수 없습니다.

기본적으로 로컬 인트라넷 영역에는 [AllClipboard](#) 액세스 권한이 부여 되고 인터넷 영역에는 [OwnClipboard](#) 액세스 권한이 부여 됩니다. 이것은 애플리케이션에서 클립보드에 데이터를 복사할 수 있지만 프로그래밍 방식으로 클립보드에 데이터를 붙여넣거나 클립보드에서 데이터를 읽을 수는 없다는 것을 의미합니다. 이러한 제한으로 인해 신뢰 수준이 완전 신뢰가 아닌 프로그램은 다른 애플리케이션이 클립보드에 복사한 내용을 읽을 수 없습니다. 애플리케이션에 클립보드에 대한 모든 액세스가 필요하지만 해당 권한이 없을 경우에는 애플리케이션의 권한을 높여야 합니다. 권한 높이기에 대한 자세한 내용은 [일반 보안 정책 관리](#)를 참조하세요.

## 창 조작

또한 [UIPermission](#) 클래스는 창 조작 및 기타 UI 관련 작업을 수행할 수 있는 권한을 제어 하고, 연결 된 [UIPermissionWindow](#) 열거형 값은 액세스 수준을 나타냅니다. 다음 표에서는 가능한 권한 수준을 보여 줍니다.

기본적으로 로컬 인트라넷 영역에는 [AllWindows](#) 액세스 권한이 부여 되고 인터넷 영역에는 [SafeTopLevelWindows](#) 액세스 권한이 부여 됩니다. 이것은 인터넷 영역에서는 애플리케이션이 대부분의 창 작업 및 UI 작업을 수행할 수 있지만 창의 모양은 수정된다는 것을 의미합니다. 처음 실행한 경우 수정된 창에는 풍선 알림과 수정된 제목 표시줄 텍스트가 표시되며, 제목 표시줄에 닫기 단추가 필요합니다. 풍선 알림과 제목 표시줄은 애플리케이션 사용자에게 애플리케이션이 부분 신뢰 환경에서 실행되고 있음을 알려 줍니다.

UIPERMISSIONWINDOW 값	DESCRIPTION
AllWindows	사용자는 모든 창과 사용자 입력 이벤트를 제한 없이 사용할 수 있습니다.
SafeTopLevelWindows	사용자가 안전한 최상위 창과 안전한 하위 창만 그리기 작업에 사용할 수 있고, 이러한 최상위 창 및 하위 창 내의 사용자 인터페이스에는 사용자 입력 이벤트만 사용할 수 있습니다. 이러한 안전한 창에는 명확하게 레이블이 지정되며 최소 및 최대 크기 제한이 있습니다. 이러한 제한 사항은 모방 시스템 로그인 화면 또는 시스템 데스크톱과 같은 잠재적으로 유해한 스푸핑 공격을 방지 하고 부모 창, 포커스 관련 Api 및 ToolTip 컨트롤 사용에 대한 프로그래밍 방식의 액세스를 제한 합니다.
SafeSubWindows	사용자가 안전한 하위 창만 그리기 작업에 사용할 수 있고, 해당 하위 창 내의 사용자 인터페이스에는 사용자 입력 이벤트만 사용할 수 있습니다. 예를 들어, 브라우저 내에 표시되는 컨트롤은 안전한 하위 창입니다.
NoWindows	사용자가 창이나 사용자 인터페이스 이벤트를 사용할 수 없습니다. 사용자 인터페이스를 사용할 수 없습니다.

UIPermissionWindow 열거형으로 식별 되는 각 사용 권한 수준은 상위 수준 보다 더 작은 작업을 수행할 수 있습니다. 다음 표에서는 SafeTopLevelWindows 및 SafeSubWindows 값에 의해 제한 되는 작업을 표시 합니다. 각 멤버에 필요한 정확한 권한은 .NET Framework 클래스 라이브러리 설명서에서 해당 멤버의 항목을 참조하세요.

SafeTopLevelWindows 권한은 다음 표에 나열 된 작업을 제한 합니다.

구성 요소	제한된 작업
Application	- SafeTopLevelCaptionFormat 속성을 설정합니다.
Control	-Parent 속성을 가져옵니다. - Region 속성을 설정합니다. -FindForm, Focus, FromChildHandle 및 FromHandle, PreProcessMessage, ReflectMessage또는 SetTopLevel 메서드를 호출 합니다. -반환 된 컨트롤이 호출 하는 컨트롤의 자식이 아닌 경우 GetChildAtPoint 메서드를 호출 합니다. - 컨테이너 컨트롤 내에서 컨트롤 포커스를 수정합니다.
Cursor	- Clip 속성을 설정합니다. -Hide 메서드를 호출 합니다.
DataGrid	-ProcessTabKey 메서드를 호출 합니다.
Form	-ActiveForm 또는 MdiParent 속성을 가져옵니다. -ControlBox, ShowInTaskbar또는 TopMost 속성을 설정 합니다. -Opacity 속성을 50% 미만으로 설정 합니다. -WindowState 속성을 프로그래밍 방식으로 Minimized로 설정 합니다. -Activate 메서드를 호출 합니다. -None, FixedToolWindow및 SizableToolWindowFormBorderStyle 열거형 값을 사용 합니다.
NotifyIcon	-NotifyIcon 구성 요소를 사용 하는 것은 완전히 제한 됩니다.



[SafeSubWindows](#) 값은 [SafeTopLevelWindows](#) 값에 의해 배치 되는 제한 사항 외에도 다음 표에 나열 된 작업을 제한 합니다.

구성 요소	제한된 작업
<a href="#">CommonDialog</a>	- <a href="#">CommonDialog</a> 클래스에서 파생 된 대화 상자를 표시 합니다.
<a href="#">Control</a>	- <a href="#">CreateGraphics</a> 메서드를 호출 합니다. - <a href="#">Cursor</a> 속성을 설정합니다.
<a href="#">Cursor</a>	- <a href="#">Current</a> 속성을 설정합니다.
<a href="#">MessageBox</a>	- <a href="#">Show</a> 메서드를 호출 합니다.

#### 타사 컨트롤 호스팅

다른 종류의 창 조작은 양식에서 타사 컨트롤을 호스팅하는 경우에 발생할 수 있습니다. 타사 컨트롤은 사용자가 직접 개발 하고 컴파일하지 않은 사용자 지정 [UserControl](#)입니다. 호스팅 시나리오를 악용하기는 어렵지만 타사 컨트롤이 렌더링 화면을 확장하여 사용자 양식의 전체 영역을 가리는 것이 논리적으로 가능합니다. 이 컨트롤은 중요한 대화 상자를 모방하여 사용자 이름/암호 조합이나 은행 계좌 번호 등의 정보를 사용자에게 요청할 수 있습니다.

이러한 잠재적인 위험을 방지하려면 신뢰할 수 있는 공급업체의 타사 컨트롤만 사용해야 합니다. 확인할 수 없는 소스에서 다운로드한 타사 컨트롤을 사용할 경우에는 소스 코드를 검토하여 악용 가능성이 있는지 확인하는 것이 좋습니다. 소스에 악의적인 내용이 없는지 확인한 후 어셈블리를 직접 컴파일하여 소스와 어셈블리가 일치하는지 확인해야 합니다.

## Windows API 호출

응용 프로그램을 디자인할 때 Windows API에서 함수를 호출 해야 하는 경우 비관리 코드에 액세스 하게 됩니다. 이 경우 Windows API 호출 또는 값으로 작업할 때 창 또는 운영 체제에 대 한 코드 작업을 확인할 수 없습니다. [SecurityPermission](#) 클래스 및 [SecurityPermissionFlag](#) 열거형의 [UnmanagedCode](#) 값은 비관리 코드에 대 한 액세스를 제어 합니다. 응용 프로그램은 [UnmanagedCode](#) 권한이 부여 된 경우에만 비관리 코드에 액세스할 수 있습니다. 기본적으로 로컬에서 실행되고 있는 애플리케이션만 비관리 코드에 액세스할 수 있습니다.

일부 Windows Forms 멤버는 [UnmanagedCode](#) 권한이 필요한 관리 되지 않는 액세스를 제공 합니다. 다음 표에서는 권한이 필요한 [System.Windows.Forms](#) 네임 스페이스의 멤버를 나열 합니다. 멤버에 필요한 권한에 대한 자세한 내용은 .NET Framework 클래스 라이브러리 설명서를 참조하세요.

구성 요소	멤버
<a href="#">Application</a>	- <a href="#">AddMessageFilter</a> 메서드 - <a href="#">CurrentInputLanguage</a> 속성 - <a href="#">Exit</a> 메서드 - <a href="#">ExitThread</a> 메서드 - <a href="#">ThreadException</a> 이벤트
<a href="#">CommonDialog</a>	- <a href="#">HookProc</a> 메서드 - <a href="#">OwnerWndProc</a> 메서드 - <a href="#">Reset</a> 메서드 - <a href="#">RunDialog</a> 메서드

구성 요소	멤버
Control	<ul style="list-style-type: none"> <li>- <a href="#">CreateParams</a> 메서드</li> <li>- <a href="#">DefWndProc</a> 메서드</li> <li>- <a href="#">DestroyHandle</a> 메서드</li> <li>- <a href="#">WndProc</a> 메서드</li> </ul>
Help	<ul style="list-style-type: none"> <li>- <a href="#">ShowHelp</a> 메서드</li> <li>- <a href="#">ShowHelpIndex</a> 메서드</li> </ul>
NativeWindow	<ul style="list-style-type: none"> <li>- <a href="#">NativeWindow</a> 클래스</li> </ul>
Screen	<ul style="list-style-type: none"> <li>- <a href="#">FromHandle</a> 메서드</li> </ul>
SendKeys	<ul style="list-style-type: none"> <li>- <a href="#">Send</a> 메서드</li> <li>- <a href="#">SendWait</a> 메서드</li> </ul>

응용 프로그램에 비관리 코드를 호출할 수 있는 권한이 없는 경우 응용 프로그램에서 [UnmanagedCode](#) 권한을 요청 하거나 기능을 구현 하는 다른 방법을 고려해 야 합니다. 대부분의 경우 Windows Forms Windows API 함수에 대 한 관리 되는 대안을 제공 합니다. 이러한 대안이 없는 상황에서 비관리 코드에 액세스해야 할 경우에는 애플리케이션의 권한을 높여야 합니다.

비관리 코드를 호출할 수 있는 권한을 부여하면 애플리케이션이 거의 모든 작업을 수행할 수 있습니다. 따라서 비관리 코드를 호출할 수 있는 권한은 신뢰할 수 있는 소스의 애플리케이션에만 부여해야 합니다. 또는 애플리케이션에 따라 비관리 코드를 호출하는 기능을 옵션으로 지정하거나 완전 신뢰 환경에서만 이 기능을 사용하도록 할 수 있습니다. 위험한 권한에 대한 자세한 내용은 [위험한 권한 및 정책 관리](#)를 참조하세요. 권한 높이기에 대한 자세한 내용은 [일반 보안 정책 관리](#)를 참조하세요.

## 참고 항목

- [Windows Forms의 파일 및 데이터 액세스 추가 보안](#)
- [Windows Forms의 인쇄 추가 보안](#)
- [Windows Forms의 보안 개요](#)
- [Windows Forms 보안](#)
- [ClickOnce 애플리케이션 보안](#)

# Windows Forms에 대한 ClickOnce 배포

2020-02-03 • 6 minutes to read • [Edit Online](#)

다음 항목에서는 Windows Forms 응용 프로그램을 클라이언트 컴퓨터에 쉽게 배포 하는 데 사용 되는 기술인 ClickOnce에 대해 설명 합니다.

## 관련 섹션

### [ClickOnce 배포 전략 선택](#)

ClickOnce 응용 프로그램을 배포 하기 위한 몇 가지 옵션을 제공 합니다.

### [ClickOnce 업데이트 전략 선택](#)

ClickOnce 응용 프로그램을 업데이트 하기 위한 몇 가지 옵션을 제공 합니다.

### [ClickOnce 애플리케이션 보안](#)

ClickOnce 배포의 보안 영향에 대해 설명 합니다.

### [ClickOnce 배포 문제 해결](#)

ClickOnce 응용 프로그램을 배포할 때 발생할 수 있는 다양 한 문제를 설명 하 고 ClickOnce에서 생성할 수 있는 최 상위 오류 메시지를 문서화 합니다.

### [ClickOnce 및 애플리케이션 설정](#)

나중에 검색할 때 응용 프로그램 및 사용자 설정을 저장 하는 응용 프로그램 설정에서 ClickOnce 배포가 작동 하는 방식을 설명 합니다.

### [신뢰할 수 있는 애플리케이션 배포 개요](#)

클라이언트 컴퓨터에서 신뢰할 수 있는 응용 프로그램을 더 높은 수준의 사용 권한으로 실행할 수 있도록 하는 ClickOnce 기능을 설명 합니다.

### [ClickOnce 및 Authenticode](#)

신뢰할 수 있는 애플리케이션 배포에서 Authenticode 기술을 사용하는 방법을 설명합니다.

### [연습: ClickOnce 애플리케이션 수동 배포](#)

Visual Studio를 사용 하지 않고 명령줄 및 SDK 도구를 사용 하여 ClickOnce 응용 프로그램을 배포 하는 방법을 보여 줍니다.

### [방법: ClickOnce 애플리케이션의 클라이언트 컴퓨터에 신뢰할 수 있는 게시자 추가](#)

신뢰할 수 있는 애플리케이션 배포에 필요한 클라이언트 컴퓨터의 일회성 구성을 보여 줍니다.

### [방법: 배포 업데이트를 위한 대체 위치 지정](#)

SDK 도구를 사용 하여 다른 위치에서 응용 프로그램의 새 버전을 확인 하도록 ClickOnce 응용 프로그램을 구성 하는 방법을 보여 줍니다.

### [연습: ClickOnce 배포 API에서 요청 시 어셈블리 다운로드](#)

애플리케이션이 처음 로드하려고 시도할 때 API 호출을 사용하여 어셈블리를 검색하는 방법을 보여 줍니다.

### [방법: 온라인 ClickOnce 애플리케이션에서 쿼리 문자열 정보 검색](#)

ClickOnce 응용 프로그램 실행에 사용 되는 URL에서 매개 변수를 검색 하는 방법을 보여 줍니다.

### [ClickOnce 캐시 개요](#)

ClickOnce 응용 프로그램을 로컬 컴퓨터에 저장 하는 데 사용 되는 캐시를 설명 합니다.

### [ClickOnce 애플리케이션의 로컬 및 원격 데이터 액세스](#)

ClickOnce 응용 프로그램에서 로컬 데이터 파일 및 원격 데이터 소스에 액세스 하는 방법을 설명 합니다.

방법: [ClickOnce 애플리케이션에 데이터 파일 포함](#)

ClickOnce 데이터 디렉터리에서 사용할 수 있도록 파일을 표시 하는 방법을 보여 줍니다.

## 참고 항목

- [애플리케이션 설정 개요](#)
- [ClickOnce 애플리케이션 게시](#)
- [명령줄에서 ClickOnce 애플리케이션 빌드](#)
- [System.Deployment.Application을 사용하는 ClickOnce 애플리케이션 디버그](#)
- [ClickOnce를 사용하여 COM 구성 요소 배포](#)
- [방법: 게시 마법사를 사용하여 ClickOnce 애플리케이션 게시](#)

# .NET 코어 3.0에 대한 Windows 양식 컨트롤의 접근성 향상

2020-04-21 • 13 minutes to read • [Edit Online](#)

Windows Forms는 Windows Forms 고객을 더 잘 지원하기 위해 내게 필요한 옵션 기술과 함께 작동하는 방식을 지속적으로 개선하고 있습니다. 개선 사항에는 다음과 같은 변경 내용이 포함됩니다.

- 내레이터를 포함하여 내게 필요한 옵션 클라이언트 응용 프로그램과의 다양한 상호 작용 영역의 변경
- 액세스 가능한 계층 구조의 변경 내용(UI Automation 트리를 통한 탐색 개선).
- 키보드 탐색의 변경 내용입니다.

## IMPORTANT

.NET Framework 4.7.1에서 .NET Framework 4.8을 통해 변경한 액세스 가능 변경 사항은 .NET Core 3.0 이상에 포함되며 기본적으로 활성화됩니다. .NET Framework는 응용 프로그램이 새 내게 필요한 옵션 동작을 옵트아웃할 수 있도록 하는 호환성 스위치를 지원했습니다. 반면 .NET Core는 이러한 설정을 지원하지 않으며 응용 프로그램이 내게 필요한 옵션 동작을 옵트아웃할 수 없습니다.

.NET Core 3.0부터 Windows Forms 응용 프로그램은 추가 구성 없이 모든 새로운 내게 필요한 옵션 기능(.NET Framework 4.7.1 - 4.8에 도입)의 이점을 누릴 수 있습니다.

## 리스트박스 접근성 지원

컨트롤에는 다음 변경 [ListBox](#) 사항이 적용됩니다.

- 제어를 위한 `ListBox` UI 자동화 지원 지원.
- 항목에 추가하고 `ListBox` [ScrollItemPattern](#) 항목을 통해 접근성 이벤트 올리기 및 처리 및 내레이터 탐색을 향상시켜 접근성 지원을 개선했습니다(캡 잠금 탐색이 올바르게 작동하지 않으며 의도치 않게 컨트롤 외부로 탐색을 던지지 않습니다). `ListBox`

## 선택 목록상자 접근성 지원

컨트롤에는 다음 변경 [CheckedListBox](#) 사항이 적용됩니다.

- 항목에 `CheckedListBox` 대한 내게 필요한 옵션 속성에서 제공하는 수정된 경계입니다.
- 전반적인 `ListBox` `CheckedListBox` 및 접근성 개선: 속성 값 및 이벤트 모델 수정.

## 콤보박스 접근성 지원

컨트롤에는 다음 변경 [ComboBox](#) 사항이 적용됩니다.

- 함수가 재정의될 경우 안전하지 않을 수 있는 항목에서 해시 코드를 가져오는 대신 항목에 대한 암호를 생성할 수 있도록 항목의 접근성 개체를 가져오는 `ComboBox` 프로세스를 [GetHashCode](#) 업데이트했습니다.

## 데이터그리드뷰 접근성 지원

컨트롤에는 다음 변경 [DataGridView](#) 사항이 적용됩니다.

- 열, `DataGridView.Bounds` 행, 셀 및 해당 헤더에 대한 내게 필요한 옵션 속성에서 제공하는 수정, 경계 사각형 계산의 성능 향상. 모든 접근성 경계는 뷰포트와 함께 전체 컨트롤의 경계를 고려하여 올바르게 표시됩니다.

- 액세스 `Value.IsReadOnly` 가능한 클라이언트 응용 프로그램에 대해 제공하는 속성 값을 수정했습니다. 이제 속성에 `IsReadOnly` 셀에 대한 올바른 상태가 표시 됩니다.
- 첫 번째 `CellParsing` 셀 변경에 대한 이벤트 발생 문제를 수정했습니다. 셀 값은 제1 `DataGridView` 대조군 상호작용을 포함하는 어떠한 문제도 없이 변경될 수 있다.
- Windows `DataGridView` 고대비 테마를 사용할 때 배경 색 대비가 개선되었습니다. HC#1, HC#2 및 HC 검정 테마를 사용할 때 기본 백 색상을 변경했습니다. `DataGridView`

## 속성 그리드 내게 필요한 옵션 지원

컨트롤에는 다음 변경 `PropertyGrid` 사항이 적용됩니다.

- 그리드 `PropertyGrid.Bounds` 항목에 대한 내게 필요한 옵션 속성에서 제공하는 수정, 경계 사각형 계산의 성능 향상. 지금은 모든 접근성 경계가 뷰포트와 함께 전체 컨트롤의 경계를 고려하여 올바르게 표시됩니다.
- 컨트롤 형식 이름을 포함하지 않고 컨트롤 형식 이름에 대한 이중 발표를 피하기 위해 액세스 가능한 이름 및 하위 컨트롤 설명을 수정했습니다.

## 툴스트립 접근성 지원

컨트롤에는 다음 변경 `ToolStrip` 사항이 적용됩니다.

- `ToolStrip` `MenuStrip` 통한 탐색이 `StatusStrip` 향상되었습니다. `ToolStrip` 시프트 탭 위쪽 화살표를 누르면 메뉴 항목을 뒤로 반복하여 아래쪽 메뉴 요소로 이동합니다. `MenuStrip`
- 'MenuItem' 대신 '메뉴'의 하위 메뉴를 만들기 위해 하위 메뉴에 대한 접근성 탐색, 수정된 메뉴 접근 형 제어 유형이 개선되었습니다. `MenuStrip`

## PrintPreviewControl 및 인쇄미리보기대화언성 액세스 가능 지원

다음 변경 사항은 인쇄 컨트롤에 적용됩니다.

- 메뉴 항목을 통해 액세스 가능한 탐색(내레이터 탐색 포함)이 개선되었습니다.
- 향상된 고대비 테마 지원 및 컨트롤 요소를 더욱 대조적으로 만들었습니다.

## 스트링컬렉션편집기 내게 필요한 옵션 지원

이제 Windows Forms 디자이너는 향상된 접근성 지원을 통해 문자열 컬렉션 편집기를 사용합니다.

## 월별 캘린더 내게 필요한 옵션 지원(.NET 코어 3.1에서 사용 가능)

컨트롤에는 다음 변경 `MonthCalendar` 사항이 적용됩니다.

- 제어할 UI 자동화 `MonthCalendar` 서버 공급자를 추가하고 UI 자동화 그리드 패턴 및 테이블 패턴 공급자를 추가했습니다.
- Changed *table* accessible control type to *calendar* accessible control type for `MonthCalendar` except the case when the control has a preceding label control that defines `MonthCalendar` control accessible name, in this specific case accessible control type becomes *table*.
- 제어를 위해 선택한 `MonthCalendar` 날짜의 공지가 개선되었습니다.
- 화면 `MonthCalendar` 판독기 및 기타 내게 필요한 옵션 도구에 대한 향상된 제어 지원. 이 시점에서 사용자는 컨트롤 요소를 탐색하고 키보드 전용 입력을 사용하여 이러한 요소와 상호 작용할 수 있습니다. 내레이터를 사용하면 CAPS + 화살표 키를 사용하여 컨트롤 요소를 탐색하고 CAPS + Enter를 사용하여 요소 기본 작업을 호출합니다.
- 내레이터의 파란색 `MonthCalendar` 초점 사각형인 포커스 사각형이 있는 자식 요소에서 화살표 키 탐색이 개선되었습니다.
- 제공된 좌표로 자식 액세스 요소를 `MonthCalendar` 얻을 수 있도록 컨트롤 요소에 대한 적중 테스트 작업에 대한

## 도구 팁 내게 필요한 옵션(.NET 코어 3.1에서 사용 가능)

- NVDA 및 내레이터와 같은 화면 판독기 응용 프로그램에서 도구 설명 텍스트를 발표할 수 있는 기능이 추가되었습니다. 이제 화면 판독기 응용 프로그램은 도구 설명팁을 표시하도록 구성된 Windows Forms 컨트롤의 키보드 또는 마우스 도구 설명의 텍스트를 발표할 수 있습니다.

## DataGridView, 속성 그리드, 리스트 박스, 콤보 박스, 도구 스트립 및 기타 컨트롤에 대한 UI 자동화 지원

UI 자동화 지원은 런타임에 컨트롤에 사용할 수 있지만 디자인 타임에는 사용되지 않습니다. UI 자동화 개요는 [UI Automation 개요](#)를 참조하세요.

## 참고 항목

- [내게 필요한 옵션 모범 사례](#).

# 방법: Windows Forms에서 키 컬렉션 액세스

2020-02-03 • 2 minutes to read • [Edit Online](#)

- 키를 사용 하여 개별 컬렉션 항목에 액세스할 수 있습니다. 이 기능은 Windows Forms 응용 프로그램에서 일반적으로 사용 되는 많은 컬렉션 클래스에 추가 되었습니다. 다음 목록에서는 액세스할 수 있는 키가 있는 컬렉션을 포함 하는 몇 가지 컬렉션 클래스를 보여 줍니다.

- [ListView.ListViewItemCollection](#)
- [ListViewItem.ListViewSubItemCollection](#)
- [Control.ControlCollection](#)
- [TabControl.TabPageCollection](#)
- [TreeNodeCollection](#)

컬렉션의 항목과 연결 된 키는 일반적으로 항목의 이름입니다. 다음 절차에서는 컬렉션 클래스를 사용 하여 일반적인 작업을 수행 하는 방법을 보여 줍니다.

컨트롤 컬렉션에서 중첩 된 컨트롤을 찾고 포커스를 지정 하려면

- [Find](#) 및 [Focus](#) 메서드를 사용 하여 찾을 컨트롤의 이름을 지정 하고 포커스를 제공 합니다.

```
OrderForm OrderForm1 = new OrderForm();
OrderForm1.Show();
OrderForm1.Controls.Find("textBox1", true)[0].Focus();
```

```
Dim OrderForm1 As New OrderForm()
OrderForm1.Show()
OrderForm1.Controls.Find("textBox1", True)(0).Focus()
```

이미지 컬렉션의 이미지에 액세스 하려면

- [Item\[\]](#) 속성을 사용 하여 액세스 하려는 이미지의 이름을 지정 합니다.

```
this.BackgroundImage = imageList1.Images["logo.gif"];
```

```
Me.BackgroundImage = imageList1.Images("logo.gif")
```

탭 페이지를 선택 된 탭으로 설정 하려면

- [Item\[\]](#) 속성과 함께 [SelectedTab](#) 속성을 사용 하여 선택 된 탭으로 설정할 탭 페이지의 이름을 지정 합니다.

```
tabControl1.SelectedTab = tabControl1.TabPages["shippingOptions"];
```

```
tabControl1.SelectedTab = tabControl1.TabPages("shippingOptions")
```

## 참고 항목

- [Windows Forms 시작](#)



- 방법: Windows Forms ImageList 구성 요소를 사용하여 이미지 추가 또는 제거

# Windows Forms 애플리케이션 강화

2020-02-03 • 5 minutes to read • [Edit Online](#)

Windows Forms에는 사용자의 특정 요구에 맞게 Windows 기반 애플리케이션을 향상시키는 데 사용할 수 있는 다양한 기능이 포함되어 있습니다. 다음 항목에서는 이러한 기능과 사용 방법을 설명합니다.

## 섹션 내용

### [Windows Forms의 그래픽 및 그리기](#)

Windows Forms에서 그래픽 인터페이스를 사용하는 방법을 설명하고 보여 주는 항목의 링크를 포함합니다.

### [Windows Forms에 대한 애플리케이션 설정.](#)

애플리케이션 설정 기능을 사용하는 방법을 설명하고 보여 주는 항목의 링크를 포함합니다.

### [Windows Forms 인쇄 지원](#)

Windows Forms 애플리케이션에서 파일을 인쇄하는 방법을 설명하고 보여 주는 항목의 링크를 포함합니다.

### [끌어서 놓기 작업 및 클립보드 지원](#)

Windows Forms에서 끌어서 놓기 기능과 클립보드를 사용하는 방법을 설명하고 보여 주는 항목의 링크를 포함합니다.

### [Windows Forms 애플리케이션의 네트워킹](#)

Windows Forms에서 네트워킹을 사용하는 방법을 설명하고 보여 주는 항목의 링크를 포함합니다.

### [응용 프로그램 Windows Forms 전역화](#)

Windows Forms 애플리케이션을 전역화하는 방법을 보여 주는 항목의 링크를 포함합니다.

### [Windows Forms 및 관리되지 않는 애플리케이션](#)

Windows Forms 애플리케이션에서 COM 구성 요소에 액세스하는 방법을 설명하고 보여 주는 항목의 링크를 포함합니다.

### [시스템 정보 및 Windows Forms](#)

Windows Forms에서 시스템 정보를 사용하는 방법을 설명합니다.

### [Windows Forms의 전원 관리](#)

Windows Forms 애플리케이션에서 전원 사용을 관리하는 방법을 설명합니다.

### [Windows Forms 시각적 개체 상속](#)

기본 폼에서 상속하는 방법을 설명합니다.

### [MDI\(다중 문서 인터페이스\) 애플리케이션](#)

MDI(다중 문서 인터페이스) 애플리케이션을 만드는 방법을 설명합니다.

### [Windows Forms에 사용자 도움말 통합](#)

애플리케이션에 사용자 도움말을 통합하는 방법을 설명합니다.

### [Windows Forms 내게 필요한 옵션](#)

다양한 사용자가 애플리케이션을 사용할 수 있게 하는 방법을 설명합니다.

### [WPF 컨트롤 사용](#)

Windows Forms 기반 애플리케이션에서 WPF 컨트롤을 사용하는 방법을 설명합니다.

## 관련 섹션

## [Windows Forms 애플리케이션의 도움말 시스템](#)

Windows Forms 애플리케이션에서 사용자 도움말을 제공하는 방법을 설명하고 보여 주는 항목의 링크를 포함합니다.

## [Windows Forms 시작](#)

Windows Forms의 기본 기능을 사용하는 방법을 설명하는 항목의 링크를 포함합니다.