Contents

시작(WPF)

2020-04-24 • 2 minutes to read • Edit Online

WPF(Windows Presentation Foundation)는 데스크톱 클라이언트 애플리케이션을 만드는 UI 프레임워크입니다. WPF 개발 플랫폼은 애플리케이션 모델, 리소스, 컨트롤, 그래픽, 레이아웃, 데이터 바인딩, 문서 및 보안을 포함하여 다양한 애플리케이션 개발 기능 집합을 지원합니다. .NET Framework의 하위 집합이므로 이전에 ASP.NET 또는 Windows Forms를 사용하여 .NET Framework로 애플리케이션을 빌드한 경우 프로그래밍 환경이 비슷합니다. WPF는 XAML(Extensible Application Markup Language)을 사용하여 애플리케이션 프로그래밍을 위한 선언적 모델을 제공합니다. 이 섹션의 항목에서는 WPF를 소개하고 WPF를 시작하는 데 유용한 정보를 제공합니다.

어디서 시작해야 합니까?

바로 시작	연습: 내 첫 WPF 데스크톱 애플리케이션
애플리케이션 UI를 디자인하려면 어떻게 해야 하나요?	Visual Studio에서 XAML 디자인
.NET을 처음 사용하세요?	.NET Framework의 개요 Visual C# 및 Visual Basic 시작
WPF에 대한 자세한 설명…	Visual Studio에서의 WPF 소개 XAML 개요(WPF) 컨트롤 데이터 바인딩 개요
Windows Forms 개발자인가요?	Windows Forms 컨트롤 및 해당 WPF 컨트롤 WPF 및 Windows Forms 상호 운용성

참고 항목

- 클래스 라이브러리
- 응용 프로그램 개발
- .NET Framework 개발자 센터

Visual Studio에서의 WPF 소개

2020-03-21 • 2 minutes to read • Edit Online

Visual Studio의 WPF(Windows 프레젠테이션 재단)는 개발자에게 Windows에서 업무 용 데스크톱 응용 프로그램을 빌드하기 위한 통합 프로그래밍 모델을 제공합니다.

Windows 프레젠테이션 기반을 사용하여 데스크톱 응용 프로그램 만들기

Visual Studio 및 Blend for Visual Studio에서 XAML 디자인

WPF 소개

.NET Framework의 WPF

Visual Studio 가져오기

WPF 버전 4.5의 새로운 기능

2020-03-21 • 20 minutes to read • Edit Online

이 항목에는 버전 4.5의 WPF(Windows Presentation Foundation) 새롭고 향상된 기능에 대한 정보가 포함되어 있습니다.

이 항목에는 다음과 같은 섹션이 포함되어 있습니다.

리본 컨트롤

WPF 4.5는 Ribbon 빠른 액세스 도구 모음, 응용 프로그램 메뉴 및 탭을 호스팅하는 컨트롤과 함께 제공됩니다. 자세한 내용은 리본 개요를 참조하세요.

그룹화된 큰 데이터 집합을 표시할 때의 성능 개선

화면에 표시되는 항목에 따라 많은 수의 데이터 항목에서 사용자 인터페이스(UI) 요소의 하위 집합이 생성될 때 UI 가상화가 발생합니다. 는 VirtualizingPanel 그룹화 IsVirtualizingWhenGrouping 된 데이터에 대 한 UI 가상화를 활성화 하는 연결 된 속성을 정의 합니다. 데이터 그룹화에 대한 자세한 내용은 방법: XAML에서 뷰를 사용하여 데이터 정렬 및 그룹화를 참조하세요. 그룹화된 데이터를 가상화하는 것에 IsVirtualizingWhenGrouping 대한 자세한 내용은 연결된 속성을 참조하십시오.

VirtualizingPanel의 새로운 기능

1. 연결된 속성을 VirtualizingPanel사용하여 VirtualizingStackPanel"의 "의"와 같은 ScrollUnit 부분 항목이 표시되는지 여부를 지정할 수 있습니다. 로 설정된 경우 ScrollUnit 완전히 표시되는 항목만 표시됩니다. VirtualizingPanel Item 로 설정된 경우 ScrollUnit VirtualizingPanel 부분적으로 표시되는 항목을 표시할 수 있습니다. Pixel

2. 연결된 속성을 사용하여 가상화할 때 VirtualizingPanel 뷰포트 앞과 후에 캐시 CacheLength 크기를 지정할 수 있습니다. 캐시는 항목이 가상화되지 않는 뷰포트 위 또는 아래 공간 양입니다. 캐시를 사용하면 뷰로 스크롤하는 동안 UI 요소가 생성되지 않도록 하여 성능을 향상시킬 수 있습니다. 캐시는 낮은 우선 순위에서 채워지므로 작업 중에는 애플리케이션이 응답하지 않게 됩니다. 속성에 VirtualizingPanel.CacheLengthUnit 의해 VirtualizingPanel.CacheLength사용되는 측정 단위가 결정됩니다.

정적 속성에 바인딩

데이터 바인딩 원본으로 정적 속성을 사용할 수 있습니다. 데이터 바인딩 엔진은 정적 이벤트가 발생할 경우 속성 값이 변경되는 것을 인식합니다. 예를 들어 `SomeClass` 클래스가 `MyProperty`라는 정적 속성을 정의하는 경우 `SomeClass`는 `MyProperty` 값이 변경될 때 발생하는 정적 이벤트를 정의할 수 있습니다. 정적 이벤트는 다음 서명 중 하나를 사용할 수 있습니다.

- `public static event EventHandler MyPropertyChanged;`
- `public static event EventHandler<PropertyChangedEventArgs> StaticPropertyChanged;`

첫 번째 경우 클래스는 Event 처리기에 전달 EventArgs 되는 *PropertyName* `Changed` 라는 정적 이벤트를 노출 합니다. 두 번째 경우 클래스는 이벤트 처리기에 `StaticPropertyChanged` 전달하는 PropertyChangedEventArgs 정적 이벤트를 노출합니다. 정적 속성을 구현하는 클래스에서 두 메서드 중 하나를 사용하여 속성-변경 알림이 발생하도록 선택할 수 있습니다.

UI가 아닌 스레드에서 컬렉션 액세스

WPF를 사용하면 컬렉션을 만들지 않은 스레드에서 데이터 컬렉션을 액세스하고 수정할 수 있습니다. 이를 통해 데이터베이스와 같은 외부 원본에서 데이터를 수신하는 데 백그라운드 스레드를 사용하고 UI 스레드에 데이터를 표시할 수 있습니다. 다른 스레드를 사용하여 콜렉션을 수정하면 사용자 인터페이스는 사용자 상호 작용에 응답성을 유지합니다.

동기적 및 비동기적으로 데이터 유효성 검사

이 INotifyDataErrorInfo 인터페이스를 사용하면 데이터 엔터티 클래스가 사용자 지정 유효성 검사 규칙을 구현하고 유효성 검사 결과를 비동기적으로 노출할 수 있습니다. 또한 이 인터페이스는 사용자 지정 오류 개체, 속성당 여러 오류, 속성 간 오류 및 엔터티 수준 오류도 지원합니다. 자세한 내용은 INotifyDataErrorInfo을 참조하세요.

데이터 바인딩 원본을 자동으로 업데이트

데이터 바인딩을 사용하여 데이터 원본을 업데이트하는 경우 Delay 속성을 사용하여 원본이 업데이트되기 전에 속성이 대상에서 변경된 후 전달할 시간을 지정할 수 있습니다. 예를 들어 데이터 개체의 Slider 속성에 Value 양방향 바인딩된 속성이 있는 a가 있고 UpdateSourceTrigger 속성이 PropertyChanged로 설정되었다고 가정합니다. 이 예제에서는 사용자가 Slider이동할 때 이동 하는 각 픽셀에 Slider 대 한 소스 업데이트 됩니다. 일반적으로 소스 개체는 슬라이더의 변경을 중지할 때만 Value 슬라이더 값이 필요합니다. 소스를 너무 자주 업데이트하지 않도록 Delay 하려면 엄지 손가락이 이동을 중지한 후 일정 시간이 경과할 때까지 소스를 업데이트하지 않도록 지정하는 데 사용합니다.

ICustomTypeProvider를 구현하는 형식에 바인딩

WPF는 사용자 지정 형식이라고도 하는 구현하는 ICustomTypeProvider개체에 대한 데이터 바인딩을 지원합니다. 다음과 같은 경우 사용자 지정 형식을 사용할 수 있습니다.

1. PropertyPath 데이터 바인딩에서. 예를 들어 Path a의 Binding 속성은 사용자 지정 형식의 속성을 참조할 수 있습니다.

2. DataType 속성의 값으로.

3. 에서 자동으로 생성된 열을 결정하는 형식입니다. DataGrid

바인딩 식에서 데이터 바인딩 정보 검색

경우에 따라 BindingExpression 바인딩의 Binding 소스 및 대상 개체에 대한 정보를 얻을 수 있습니다. 원본 또는 대상 개체나 연결된 속성을 얻도록 새로운 API가 추가되었습니다. BindingExpression이 API가 있는 경우 다음 API를 사용하여 대상 및 소스에 대한 정보를 가져옵니다.

바인딩의 다음 값을 찾으려면	다음 API 사용
대상 개체	BindingExpressionBase.Target
대상 속성	BindingExpressionBase.TargetProperty
소스 개체	BindingExpression.ResolvedSource
원본 속성	BindingExpression.ResolvedSourcePropertyName
에 BindingExpression 속하는지 여부BindingGroup	BindingExpressionBase.BindingGroup
의 소유자BindingGroup	Owner

유효한 DataContext 개체 확인

에 ItemsControl 있는 항목 DataContext 컨테이너의 연결이 끊어지는 경우가 있습니다. 항목 컨테이너는 ItemsControl에 항목을 표시하는 UI 요소입니다. is가 ItemsControl 컬렉션에 바인딩된 경우 각 항목에 대해 항목 컨테이너가 생성됩니다. 경우에 따라 항목 컨테이너는 시각적 트리에서 제거됩니다. 항목 컨테이너가 제거되는 두 가지 일반적인 경우는 기본 컬렉션에서 항목이 제거되고 ItemsControl에서 가상화가 활성화된 경우입니다. 이러한 경우 항목 DataContext 컨테이너의 속성은 정적 속성에 의해 반환 되는 sentinel 개체로 BindingOperations.DisconnectedSource 설정 됩니다. 항목 컨테이너에 DataContext DisconnectedSource 액세스하기 DataContext 전에 개체와 동일한지 확인해야 합니다.

데이터 값이 변경될 때 데이터의 위치 변경(라이브 셰이핑)

데이터의 컬렉션을 그룹화, 정렬 또는 필터링할 수 있습니다. WPF 4.5에서는 데이터가 수정되면 데이터를 다시 배열할 수 있습니다. 예를 들어, 애플리케이션에서 사용 하는 DataGrid 따라 주식 시장에 주식, 주식을 나열 하려면 정렬 합니다. 주식에서 실시간 정렬을 사용 하도록 설정 하는 경우 CollectionView에서 주식의 위치는 DataGrid 재고 값이 큰 이동 또는 보다 작은 다른 주식의 값입니다. 자세한 내용은 인터페이스를 ICollectionViewLiveShaping 참조하십시오.

이벤트에 대한 약한 참조 설정을 위한 지원 개선

이벤트 구독자가 추가 인터페이스 구현 없이 참여할 수 있으므로 약한 이벤트 패턴을 구현하기가 더 쉬워졌습니다. 또한 WeakEventManager 제네릭 클래스를 사용하면 특정 이벤트에 전용이 WeakEventManager 없는 경우 구독자가 약한 이벤트 패턴에 참여할 수 있습니다. 자세한 내용은 약한 이벤트 패턴을 참조하세요.

Dispatcher 클래스에 대한 새로운 메서드

Dispatcher 클래스는 동기 및 비동기 작업에 대해 새로운 메서드를 정의합니다. 동기 Invoke 메서드는 또는 Action Func<TResult> 매개 변수를 사용하는 오버로드를 정의합니다. 새 비동기 메서드는 InvokeAsync을 Action 호출 Func<TResult> 하거나 콜백 매개 변수로 DispatcherOperation DispatcherOperation<TResult>사용 하 고 또는 를 반환 합니다. 및 클래스는 속성을 정의합니다. Task DispatcherOperation DispatcherOperation<TResult> 호출할 InvokeAsync때 또는 연결된 `await` DispatcherOperation Task키워드를 사용할 수 있습니다. DispatcherOperation

또는 DispatcherOperation<TResult>에서 반환되는 DispatcherOperationWait Task 에 대해 동기적으로 기다려야 하는 경우 확장 메서드를 호출합니다. 작업이 Task.Wait 호출 스레드에서 큐에 대기 중인 경우 호출하면 교착 상태가 발생합니다. 비동기 작업을 수행하기 Task 위해 를 사용하는 것에 대한 자세한 내용은 작업 병렬 처리(작업 병렬 라이브러리)를참조하십시오.

이벤트에 대한 태그 확장

WPF 4.5에서는 이벤트에 대한 태그 확장을 지원합니다. WPF는 이벤트에 사용될 태그 확장을 정의하지 않지만 타사에서 이벤트에 사용할 수 있는 태그 확장을 만들 수 있습니다.

참고 항목

- .NET 프레임워크의 새로운 내용

자습서: Visual Studio 2019에서 첫 번째 WPF 응용 프로그램 만들기

2020-04-24 • 47 minutes to read • Edit Online

이 문서에서는 대부분의 WPF 응용 프로그램에 공통적인 요소인 확장 가능한 응용 프로그램 태그 언어(XAML) 태그, 코드 숨김, 응용 프로그램 정의, 컨트롤, 레이아웃, 데이터 바인딩 및 스타일과 같은 요소를 포함하는 WPF(Windows 프레젠테이션 Foundation) 데스크톱 응용 프로그램을 개발하는 방법을 보여 주었습니다. 응용 프로그램을 개발하려면 Visual Studio를 사용합니다.

이 자습서에서는 다음 작업 방법을 알아봅니다.

- WPF 프로젝트를 만듭니다.
- XAML을 사용하여 응용 프로그램의 사용자 인터페이스(UI)의 모양을 디자인합니다.
- 코드를 작성하여 응용 프로그램의 동작을 빌드합니다.
- 응용 프로그램을 관리하는 응용 프로그램 정의를 만듭니다.
- 컨트롤을 추가하고 레이아웃을 만들어 응용 프로그램 UI를 작성합니다.
- 응용 프로그램의 UI 전체에서 일관된 모양을 위한 스타일을 만듭니다.
- UI를 데이터에 바인딩하여 데이터에서 UI를 채우고 데이터와 UI를 동기화된 상태로 유지합니다.

자습서가 끝나면 사용자가 선택한 사용자에 대한 비용 보고서를 볼 수 있는 독립 실행형 Windows 응용 프로그램을 빌드했습니다. 응용 프로그램은 브라우저 스타일 창에서 호스팅되는 여러 WPF 페이지로 구성됩니다.

> **TIP**
> 이 자습서에서 사용되는 샘플 코드는 자습서 WPF 앱 샘플 코드에서시각적 기본 및 C#에 사용할 수 있습니다.
>
> 이 페이지 위에 있는 언어 선택기를 사용하여 샘플 코드의 코드 언어를 C#과 Visual Basic 간에 전환할 수 있습니다.

사전 요구 사항

- **.NET 데스크톱 개발** 워크로드가 설치된 Visual Studio 2019입니다.

  최신 버전의 Visual Studio 설치에 대한 자세한 내용은 Visual Studio 설치를참조하십시오.

응용 프로그램 프로젝트 만들기

첫 번째 단계는 응용 프로그램 정의, 두 페이지 및 이미지를 포함하는 응용 프로그램 인프라를 만드는 것입니다.

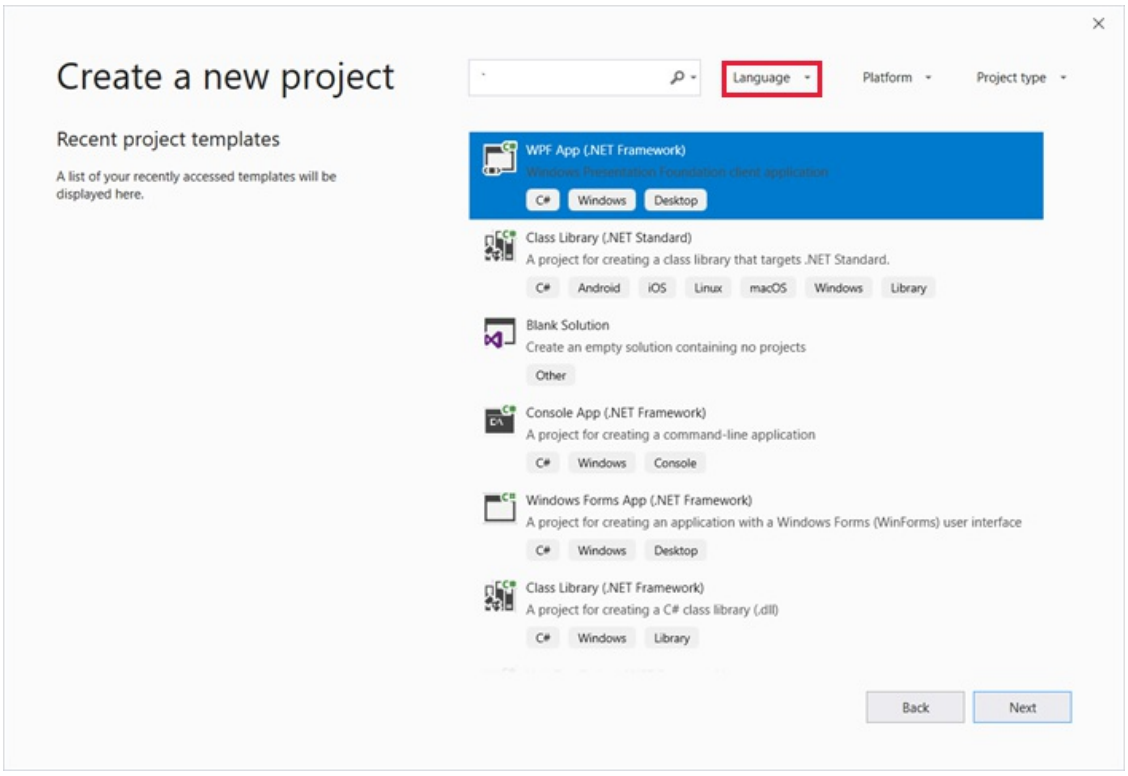
1. 시각적 기본 또는 시각적 C #라는 이름의 `ExpenseIt` 새 WPF 응용 프로그램 프로젝트를 만듭니다.

   a. 비주얼 스튜디오를 열고 **시작** 메뉴에서 **새 프로젝트 만들기**를 선택합니다.

      새 프로젝트 만들기 대화 상자가 열립니다.

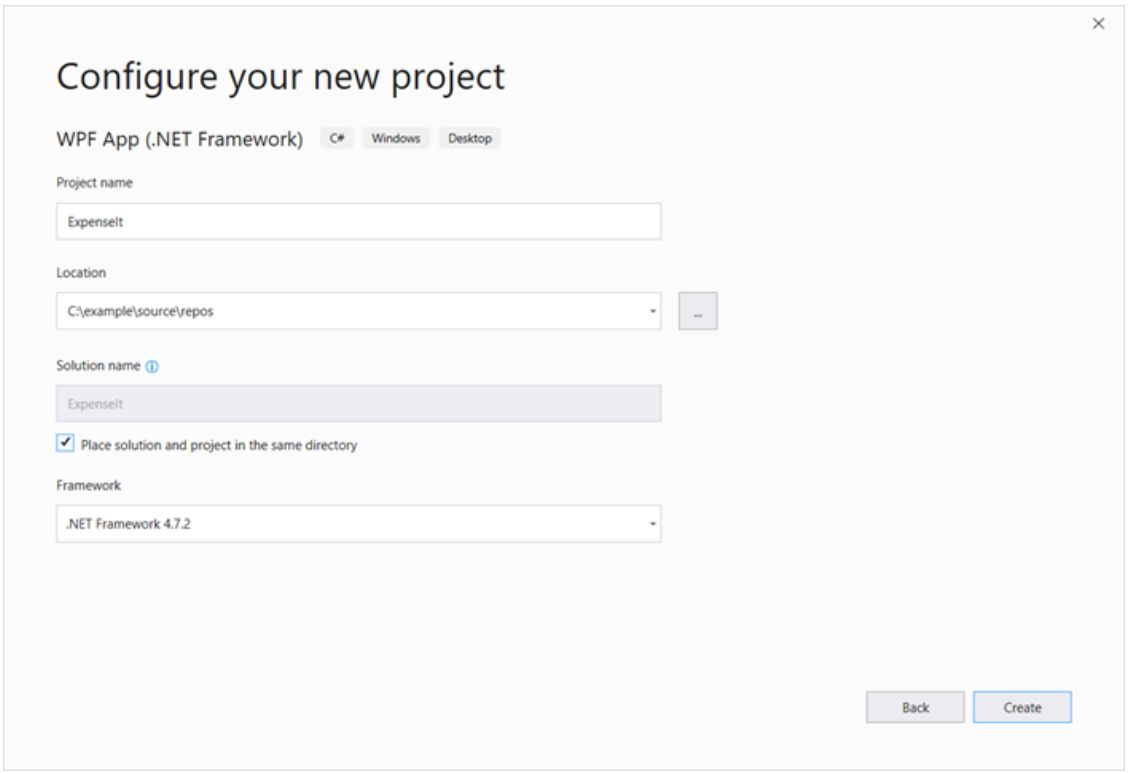
   b. **언어** 드롭다운에서 **C#** 또는 **시각적 기본**을 선택합니다.

   c. **WPF 앱(.NET 프레임워크)** 템플릿을 선택한 다음 을 **선택합니다**.



새 프로젝트 구성 대화 상자가 열립니다.

d. 프로젝트 이름을 `ExpenseIt` 입력한 다음 **만들기**를선택합니다.



Visual Studio에서 프로젝트를 만들고 **MainWindow.xaml**이라는 기본 응용 프로그램 창에 대 한 디자이너를 엽니다.

2. *응용 프로그램.xaml* (시각적 기본) 또는 *App.xaml* (C #)를 엽니 다.

이 XAML 파일은 WPF 응용 프로그램 및 모든 응용 프로그램 리소스를 정의합니다. 또한 이 파일을 사용하여 응용 프로그램이 시작될 때 자동으로 표시되는 이 경우 *MainWindow.xaml*을지정합니다.

XAML은 시각적 기본에서 다음과 같이 되어야 합니다.

```
<Application x:Class="Application"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    StartupUri="MainWindow.xaml">
    <Application.Resources>

    </Application.Resources>
</Application>
```

그리고 C #의 다음과 같습니다.

```
<Application x:Class="ExpenseIt.App"
     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
     StartupUri="MainWindow.xaml">
    <Application.Resources>

    </Application.Resources>
</Application>
```

3. *오픈 메인윈도우.xaml.*

   이 XAML 파일은 응용 프로그램의 기본 창이며 페이지에서 만든 콘텐츠를 표시합니다. 클래스는 Window 제목, 크기 또는 아이콘과 같은 창의 속성을 정의하고 닫거나 숨기는 등의 이벤트를 처리합니다.

4. 다음 Window XAML에 표시된 대로 요소를 NavigationWindow로 변경합니다.

   ```
   <NavigationWindow x:Class="ExpenseIt.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        ...
   </NavigationWindow>
   ```

   이 응용 프로그램은 사용자 입력에 따라 다른 콘텐츠로 이동합니다. 이것이 주 Window 를 로 변경해야 NavigationWindow하는 이유입니다. NavigationWindow의 Window모든 속성을 상속합니다. XAML 파일의 NavigationWindow 요소는 클래스의 인스턴스를 NavigationWindow 만듭니다. 자세한 내용은 탐색 개요를참조하십시오.

5. NavigationWindow 태그 사이에서 Grid 요소를 제거합니다.

6. 요소에 대한 XAML 코드의 다음 NavigationWindow 속성을 변경합니다.

   - Title 속성을 "로 `ExpenseIt` 설정합니다.
   - Height 속성을 350픽셀로 설정합니다.
   - Width 속성을 500픽셀로 설정합니다.

   XAML은 시각적 기본에 대해 다음과 같이 만들어야 합니다.

   ```
   <NavigationWindow x:Class="MainWindow"
       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       Title="ExpenseIt" Height="350" Width="500">

   </NavigationWindow>
   ```

   그리고 C #에 대한 다음처럼 :

```
<NavigationWindow x:Class="ExpenseIt.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="ExpenseIt" Height="350" Width="500">

</NavigationWindow>
```

7. *메인윈도우.xaml.vb* 또는 *MainWindow.xaml.cs.*

   이 파일은 *MainWindow.xaml*에서 선언 된 이벤트를 처리하는 코드를 포함하는 코드 숨은 파일입니다. 이 파일에는 XAML에 정의된 창의 부분 클래스가 포함되어 있습니다.

8. C#을 사용하는 경우 `MainWindow` 에서 파생되는 클래스를 변경합니다. NavigationWindow 시각적 기본에서 XAML에서 창을 변경할 때 자동으로 발생 합니다. 이제 C# 코드는 다음과 같이 표시됩니다.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace ExpenseIt
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : NavigationWindow
    {
        public MainWindow()
        {
            InitializeComponent();
        }
    }
}
```

응용 프로그램에 파일 추가

이 섹션에서는 애플리케이션에 두 페이지와 이미지를 추가합니다.

1. 프로젝트에 새 페이지를 추가하고 이름을 지정합니다. `ExpenseItHome.xaml`

   a. **솔루션 탐색기에서** 프로젝트 노드를 `ExpenseIt` 마우스 오른쪽 단추로 클릭하고**페이지 추가를** > 선택합니다.

   b. 새 **항목 추가** 대화 상자에서 **페이지(WPF)** 템플릿이 이미 선택되었습니다. 이름을 `ExpenseItHome` 입력한 다음 **에 추가를**선택합니다.

   이 페이지는 응용 프로그램을 시작할 때 표시되는 첫 번째 페이지입니다. 비용 보고서를 표시하기 위해 선택할 수 있는 사용자 목록이 표시됩니다.

2. 을 `ExpenseItHome.xaml` 엽니다.

3. Title "를 `ExpenseIt - Home` "로 설정합니다.

4. `DesignHeight` 350픽셀과 `DesignWidth` 500픽셀로 설정합니다.

   이제 XAML이 시각적 기본에 대해 다음과 같이 표시됩니다.

```
<Page x:Class="ExpenseItHome"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  mc:Ignorable="d"
  d:DesignHeight="350" d:DesignWidth="500"
  Title="ExpenseIt - Home">
    <Grid>

    </Grid>
</Page>
```

   그리고 C #에 대한 다음처럼 :

```
<Page x:Class="ExpenseIt.ExpenseItHome"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      mc:Ignorable="d"
      d:DesignHeight="350" d:DesignWidth="500"
    Title="ExpenseIt - Home">

    <Grid>

    </Grid>
</Page>
```

5. *오픈 메인윈도우.xaml.*

6. 요소에 속성을 추가하고 ""로 `ExpenseItHome.xaml` 설정합니다. Source NavigationWindow

   이 *`ExpenseItHome.xaml`* 설정은 응용 프로그램이 시작될 때 열리는 첫 번째 페이지로 설정됩니다.

   시각적 기본의 예제 XAML:

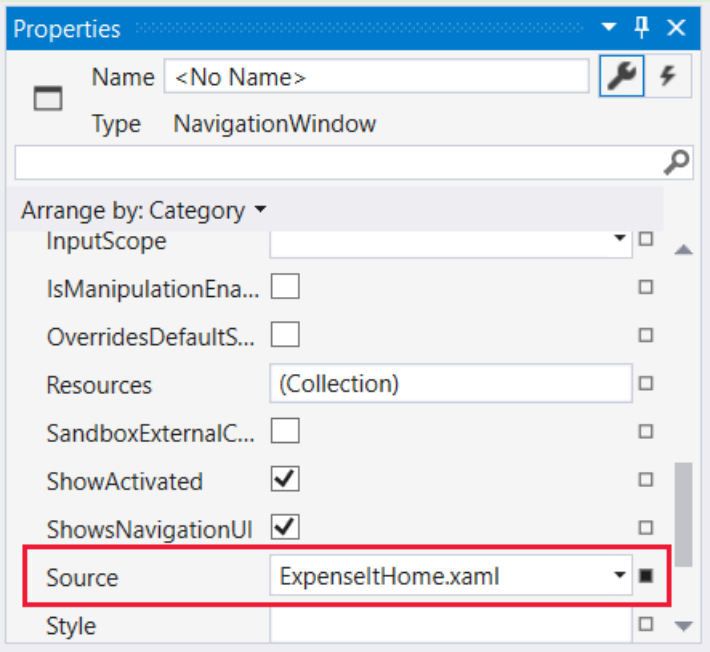
```
<NavigationWindow x:Class="MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="ExpenseIt" Height="350" Width="500" Source="ExpenseItHome.xaml">

</NavigationWindow>
```

   그리고 C #에서 :

```
<NavigationWindow x:Class="ExpenseIt.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="ExpenseIt" Height="350" Width="500" Source="ExpenseItHome.xaml">

</NavigationWindow>
```

TIP

속성 **창의 기타** 범주에서 **소스** 속성을 설정할 수도 있습니다.



7. 프로젝트에 다른 새 WPF 페이지를 추가하고 *ExpenseReportPage.xaml*의이름을 지정합니다.

   a. **솔루션 탐색기에서** 프로젝트 노드를 `ExpenseIt` 마우스 오른쪽 단추로 클릭하고**페이지 추가를** > 선택합니다.

   b. 새 **항목 추가** 대화 상자에서 **페이지(WPF)** 템플릿을 선택합니다. **비용 보고서 페이지**라는 이름을 입력한 다음 에 **추가를**선택합니다.

   이 페이지에는 `ExpenseItHome` 페이지에서 선택한 사람의 지출 보고서가 표시됩니다.

8. *ExpenseReportPage.xaml*을 엽니다.

9. Title "를 `ExpenseIt - View Expense` "로 설정합니다.

10. `DesignHeight` 350픽셀과 `DesignWidth` 500픽셀로 설정합니다.

    이제 지출 보고서Page.xaml은 시각적 기본에서 다음과 같이 표시됩니다.

```
<Page x:Class="ExpenseReportPage"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      mc:Ignorable="d"
      d:DesignHeight="350" d:DesignWidth="500"
      Title="ExpenseIt - View Expense">
    <Grid>

    </Grid>
</Page>
```

그리고 C #의 다음과 같습니다.

```
<Page x:Class="ExpenseIt.ExpenseReportPage"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      mc:Ignorable="d"
      d:DesignHeight="350" d:DesignWidth="500"
    Title="ExpenseIt - View Expense">

    <Grid>

    </Grid>
</Page>
```

11. 오픈 *ExpenseItHome.xaml.vb* 및 *비용 보고서Page.xaml.vb,* 또는 *ExpenseItHome.xaml.cs 및 ExpenseReportPage.xaml.cs*.

    새 페이지 파일을 만들 면 Visual Studio에서 *자동으로 코드 숨 파일이* 만들어집니다. 이러한 코드 숨김 파일은 사용자 입력에 응답하기 위한 논리를 처리합니다.

    코드는 `ExpenseItHome` 다음과 같아야 합니다.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace ExpenseIt
{
    /// <summary>
    /// Interaction logic for ExpenseItHome.xaml
    /// </summary>
    public partial class ExpenseItHome : Page
    {
        public ExpenseItHome()
        {
            InitializeComponent();
        }
    }
}
```

```
Class ExpenseItHome

End Class
```

그리고 **비용 보고서 페이지에** 대한 다음과 같은 :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace ExpenseIt
{
    /// <summary>
    /// Interaction logic for ExpenseReportPage.xaml
    /// </summary>
    public partial class ExpenseReportPage : Page
    {
        public ExpenseReportPage()
        {
            InitializeComponent();
        }
    }
}
```

```
Class ExpenseReportPage

End Class
```

12. 프로젝트에 *watermark.png라는* 이미지를 추가합니다. 사용자 고유의 이미지를 만들거나 샘플 코드에서 파일을 복사하거나 microsoft/WPF-샘플 GitHub 리포지토리에서 다운로드할 수 있습니다.

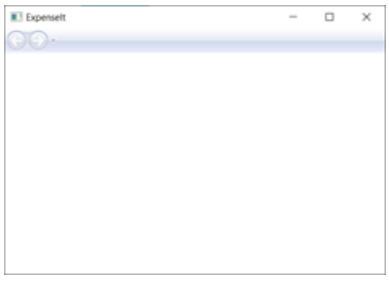
    a. 프로젝트 노드를 마우스 오른쪽 단추로 클릭하고**기존 항목 추가를** > 선택하거나 **Shift+Alt+A**를누릅니다.

    b. 기존 **항목 추가** 대화 상자에서 파일 필터를 **모든 파일** 또는 **이미지 파일로**설정하고 사용할 이미지 파일로 찾아이동한 다음 **추가를**선택합니다.

애플리케이션 빌드 및 실행

1. 응용 프로그램을 빌드하고 실행하려면 **F5를** 누르거나 **디버그** 메뉴에서 **디버깅 시작을** 선택합니다.

   다음 그림에서는 NavigationWindow 단추를 가진 응용 프로그램을 보여 주며 있습니다.



2. 응용 프로그램을 닫아 Visual Studio로 돌아갑니다.

레이아웃 만들기

레이아웃은 UI 요소를 배치하는 순서가 정해된 방법을 제공하며 UI의 크기를 조정할 때 해당 요소의 크기와 위치도

관리합니다. 일반적으로 다음 레이아웃 컨트롤 중 하나를 사용하여 레이아웃을 만듭니다.

- Canvas- 캔버스 영역을 기준으로 한 좌표를 사용하여 자식 요소를 명시적으로 배치할 수 있는 영역을 정의합니다.
- DockPanel- 서로 에 대해 가로 또는 세로로 자식 요소를 정렬 할 수있는 영역을 정의합니다.
- Grid- 열과 행으로 구성된 유연한 그리드 영역을 정의합니다.
- StackPanel- 자식 요소를 가로 또는 세로로 향할 수 있는 한 줄로 정렬합니다.
- VirtualizingStackPanel- 수평 또는 수직 방향의 한 줄에 콘텐츠를 정렬하고 가상화합니다.
- WrapPanel- 하위 요소를 왼쪽에서 오른쪽으로 순차적으로 배치하여 포함된 상자의 가장자리에 있는 다음 줄로 콘텐츠를 분리합니다. 후속 순서는 방향 속성의 값에 따라 위에서 아래로 또는 오른쪽에서 왼쪽으로 순차적으로 수행됩니다.

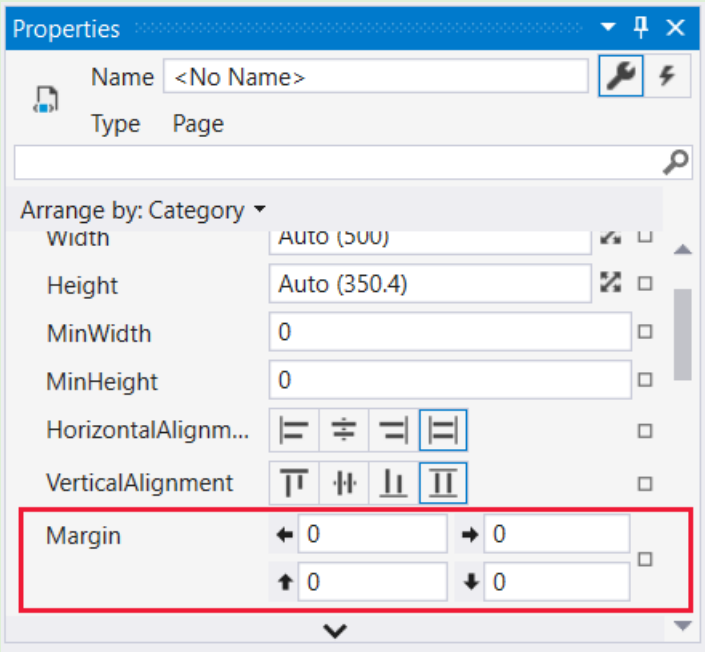
이러한 각 레이아웃 컨트롤은 자식 요소에 대한 특정 유형의 레이아웃을 지원합니다. `ExpenseIt` 페이지의 크기를 조정할 수 있으며 각 페이지에는 다른 요소와 함께 가로 및 세로로 정렬된 요소가 있습니다. 이 예제에서는 Grid 응용 프로그램의 레이아웃 요소로 사용됩니다.

> **TIP**
> 요소에 대한 Panel 자세한 내용은 패널 개요를참조하십시오. 레이아웃에 대한 자세한 내용은 레이아웃을 참조하십시오.

이 섹션에서는 Grid 에서 *`ExpenseItHome.xaml`* 열 및 행 정의를 추가하여 세 개의 행과 10픽셀 여백이 있는 단일 열 테이블을 만듭니다.

1. 에서 *`ExpenseItHome.xaml`* 요소의 Margin Grid 속성을 왼쪽, 위쪽, 오른쪽 및 아래쪽 여백에 해당하는 "10,0,10,10"으로 설정합니다.

```
<Grid Margin="10,0,10,10">
```

> **TIP**
> **레이아웃** 범주에서 **속성** 창에서 **여백** 값을 설정할 수도 있습니다.
>
> 
>

2. 태그 사이에 다음 XAML을 Grid 추가하여 행 및 열 정의를 만듭니다.

```
<Grid.ColumnDefinitions>
    <ColumnDefinition />
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
    <RowDefinition Height="Auto"/>
    <RowDefinition />
    <RowDefinition Height="Auto"/>
</Grid.RowDefinitions>
```

두 Height 행 중 의 Auto집합은 행의 내용에 따라 행의 크기가 조정됩니다. Height 기본값은 Star 크기 조정이며, 이는 행 높이가 사용 가능한 공간의 가중치 비율임을 의미합니다. 예를 들어 두 행에 Height 각각 "*"가 있는 경우 각각 사용 가능한 공간의 절반인 높이가 있습니다.

이제 Grid 다음 XAML이 포함되어야 합니다.

```
<Grid Margin="10,0,10,10">
    <Grid.ColumnDefinitions>
        <ColumnDefinition />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition />
        <RowDefinition Height="Auto"/>
    </Grid.RowDefinitions>
</Grid>
```

컨트롤 추가

이 섹션에서는 홈 페이지 UI를 업데이트하여 지출 보고서를 표시할 사람을 한 사람을 선택하는 사람 목록을 표시합니다. 컨트롤은 사용자가 애플리케이션과 상호 작용할 수 있게 하는 UI 개체입니다. 자세한 내용은 컨트롤을 참조하십시오.

이 UI를 만들려면 다음 요소를 추가합니다. `ExpenseItHome.xaml`

- A(사람 ListBox 목록)입니다.
- A(목록 Label 헤더용)입니다.
- A(목록에서 Button 선택한 사람의 지출 보고서를 보려면 클릭합니다).

각 컨트롤은 연결된 속성을 Grid 설정하여 Grid.Row 의 행에 배치됩니다. 연결된 속성에 대한 자세한 내용은 연결된 속성 개요를참조하십시오.

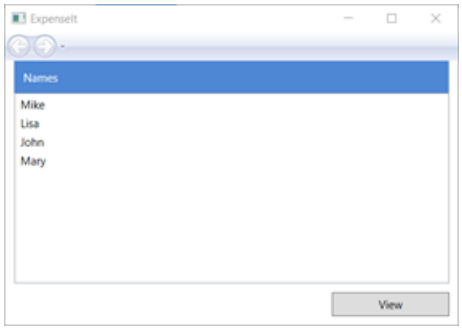
1. 에서 `ExpenseItHome.xaml` 태그 사이의 어딘가에 다음 XAML을 Grid 추가합니다.

```
<!-- People list -->
<Border Grid.Column="0" Grid.Row="0" Height="35" Padding="5" Background="#4E87D4">
    <Label VerticalAlignment="Center" Foreground="White">Names</Label>
</Border>
<ListBox Name="peopleListBox" Grid.Column="0" Grid.Row="1">
    <ListBoxItem>Mike</ListBoxItem>
    <ListBoxItem>Lisa</ListBoxItem>
    <ListBoxItem>John</ListBoxItem>
    <ListBoxItem>Mary</ListBoxItem>
</ListBox>

<!-- View report button -->
<Button Grid.Column="0" Grid.Row="2" Margin="0,10,0,0" Width="125" Height="25"
HorizontalAlignment="Right">View</Button>
```

> **TIP**
> **도구 상자** 창에서 디자인 창으로 드래그한 다음 **속성** 창에서 해당 속성을 설정하여 컨트롤을 만들 수도 있습니다.

2. 애플리케이션을 빌드 및 실행합니다.

   다음 그림에서는 만든 컨트롤을 보여 주며,



이미지 및 제목 추가

이 섹션에서는 이미지와 페이지 제목으로 홈 페이지 UI를 업데이트합니다.

1. 에서 `ExpenseItHome.xaml` 230픽셀로 ColumnDefinitions 고정된 Width 다른 열을 추가합니다.

```
<Grid.ColumnDefinitions>
    <ColumnDefinition Width="230" />
    <ColumnDefinition />
</Grid.ColumnDefinitions>
```

2. RowDefinitions에 다른 행을 추가하여 총 4개의 행을 추가합니다.

```
<Grid.RowDefinitions>
    <RowDefinition/>
    <RowDefinition Height="Auto"/>
    <RowDefinition />
    <RowDefinition Height="Auto"/>
</Grid.RowDefinitions>
```

3. 세 가지 컨트롤(테두리, ListBox 및 단추)에서 Grid.Column 속성을 1로 설정하여 컨트롤을 두 번째 열로 이동합니다.

4. 세 컨트롤(테두리, ListBox 및 Grid.Row Button) 및 테두리 요소에 대해 각 컨트롤의 값을 1씩 증가시켜 각 컨트롤을 행 아래로 이동합니다.

   이제 세 컨트롤에 대한 XAML은 다음과 같습니다.

```
  <Border Grid.Column="1" Grid.Row="1" Height="35" Padding="5" Background="#4E87D4">
      <Label VerticalAlignment="Center" Foreground="White">Names</Label>
  </Border>
  <ListBox Name="peopleListBox" Grid.Column="1" Grid.Row="2">
      <ListBoxItem>Mike</ListBoxItem>
      <ListBoxItem>Lisa</ListBoxItem>
      <ListBoxItem>John</ListBoxItem>
      <ListBoxItem>Mary</ListBoxItem>
  </ListBox>

  <!-- View report button -->
  <Button Grid.Column="1" Grid.Row="3" Margin="0,10,0,0" Width="125"
Height="25" HorizontalAlignment="Right">View</Button>
```

5. 다음 Panel.Background XAML을 태그와 `</Grid>` 태그 사이에 추가하여 다음 XAML을 `<Grid>` 추가하여 *filemark.png* 이미지 파일로 속성을 설정합니다.

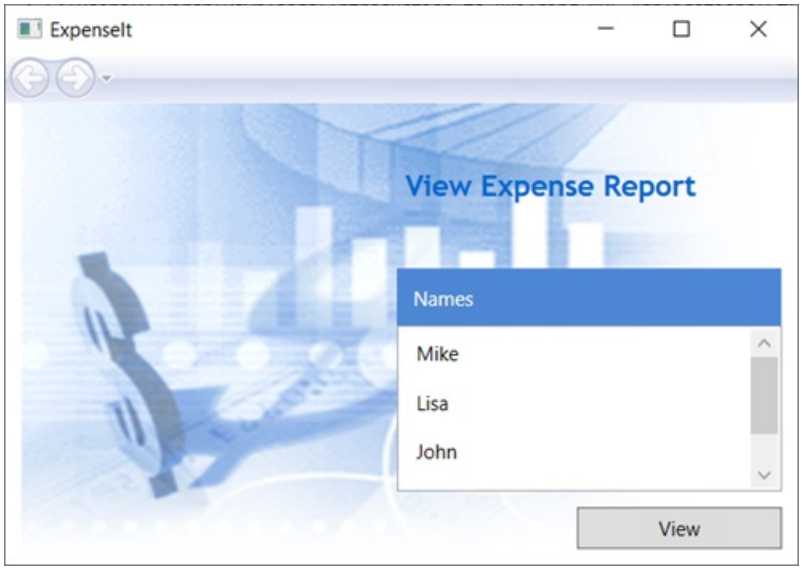
```
<Grid.Background>
    <ImageBrush ImageSource="watermark.png"/>
</Grid.Background>
```

6. 요소 Border 앞에 "비용 Label 보고서 보기"라는 내용과 함께 a를 추가합니다. 이 레이블은 페이지의 제목입니다.

```
<Label Grid.Column="1" VerticalAlignment="Center" FontFamily="Trebuchet MS"
        FontWeight="Bold" FontSize="18" Foreground="#0066cc">
    View Expense Report
</Label>
```

7. 애플리케이션을 빌드 및 실행합니다.

다음 그림에서는 방금 추가한 내용의 결과를 보여 주며,



이벤트를 처리하는 코드 추가

1. 에서 `ExpenseItHome.xaml` 요소에 Click 이벤트 처리기를 추가합니다. Button 자세한 내용은 방법: 간단한 이벤트 처리기 만들기를 참조하십시오.

```
  <!-- View report button -->
  <Button Grid.Column="1" Grid.Row="3" Margin="0,10,0,0" Width="125"
Height="25" HorizontalAlignment="Right" Click="Button_Click">View</Button>
```

2. 열기 `ExpenseItHome.xaml.vb` `ExpenseItHome.xaml.cs` 또는 .

3. `ExpenseItHome` 클래스에 다음 코드를 추가하여 단추 클릭 이벤트 처리기를 추가합니다. 이벤트 처리기가 **비용 보고서 페이지** 페이지를 엽니다.

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    // View Expense Report
    ExpenseReportPage expenseReportPage = new ExpenseReportPage();
    this.NavigationService.Navigate(expenseReportPage);
}
```

```
Private Sub Button_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
    ' View Expense Report
    Dim expenseReportPage As New ExpenseReportPage()
    Me.NavigationService.Navigate(expenseReportPage)

End Sub
```

비용 보고서 페이지에 대한 UI 만들기

비용 보고서Page.xaml `ExpenseItHome` 페이지에서 선택한 사람에 대 한 비용 보고서를 표시 합니다. 이 섹션에서는 **비용 보고서 페이지에** 대한 UI를 만듭니다. 또한 다양한 UI 요소에 배경 및 채우기 색상을 추가합니다.

1. *ExpenseReportPage.xaml*을 엽니다.

2. 태그 사이에 다음 XAML을 Grid 추가합니다.

```
<Grid.Background>
    <ImageBrush ImageSource="watermark.png" />
</Grid.Background>
<Grid.ColumnDefinitions>
    <ColumnDefinition Width="230" />
    <ColumnDefinition />
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
    <RowDefinition Height="Auto" />
    <RowDefinition />
</Grid.RowDefinitions>


<Label Grid.Column="1" VerticalAlignment="Center" FontFamily="Trebuchet MS"
FontWeight="Bold" FontSize="18" Foreground="#0066cc">
    Expense Report For:
</Label>
<Grid Margin="10" Grid.Column="1" Grid.Row="1">

    <Grid.ColumnDefinitions>
        <ColumnDefinition />
        <ColumnDefinition />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition />
    </Grid.RowDefinitions>

    <!-- Name -->
    <StackPanel Grid.Column="0" Grid.ColumnSpan="2" Grid.Row="0" Orientation="Horizontal">
        <Label Margin="0,0,0,5" FontWeight="Bold">Name:</Label>
        <Label Margin="0,0,0,5" FontWeight="Bold"></Label>
    </StackPanel>

    <!-- Department -->
    <StackPanel Grid.Column="0" Grid.ColumnSpan="2" Grid.Row="1" Orientation="Horizontal">
        <Label Margin="0,0,0,5" FontWeight="Bold">Department:</Label>
        <Label Margin="0,0,0,5" FontWeight="Bold"></Label>
    </StackPanel>

    <Grid Grid.Column="0" Grid.ColumnSpan="2" Grid.Row="2" VerticalAlignment="Top"
          HorizontalAlignment="Left">
        <!-- Expense type and Amount table -->
        <DataGrid  AutoGenerateColumns="False" RowHeaderWidth="0" >
            <DataGrid.ColumnHeaderStyle>
                <Style TargetType="{x:Type DataGridColumnHeader}">
                    <Setter Property="Height" Value="35" />
                    <Setter Property="Padding" Value="5" />
                    <Setter Property="Background" Value="#4E87D4" />
                    <Setter Property="Foreground" Value="White" />
                </Style>
            </DataGrid.ColumnHeaderStyle>
            <DataGrid.Columns>
                <DataGridTextColumn Header="ExpenseType" />
                <DataGridTextColumn Header="Amount"  />
            </DataGrid.Columns>
        </DataGrid>
    </Grid>
</Grid>
```

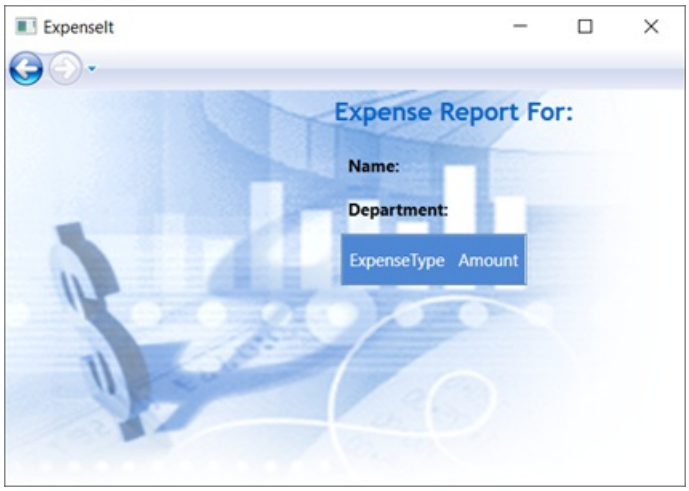
이 UI는 `ExpenseItHome.xaml` 에 표시된 보고서 데이터를 제외하면 와 DataGrid유사합니다.

3. 애플리케이션을 빌드 및 실행합니다.

4. 보기 단추를 **선택합니다**.

경비 보고서 페이지가 나타납니다. 또한 뒤로 탐색 버튼이 활성화되어 있습니다.

다음 그림에서는 *ExpenseReportPage.xaml*에추가된 UI 요소를 보여 주어집니다.



스타일 컨트롤

다양한 요소의 모양은 UI에서 동일한 형식의 모든 요소에 대해 동일합니다. UI는 스타일을 사용하여 여러 요소에서 모양을 재사용할 수 있도록 합니다. 스타일 재사용성은 XAML 생성 및 관리를 단순화하는 데 도움이 됩니다. 이 섹션에서는 이전 단계에서 정의된 요소별 특성을 스타일로 바꿉니다.

1. *응용 프로그램.xaml* 또는 *App.xaml*을엽니 다.

2. 태그 사이에 다음 XAML을 Application.Resources 추가합니다.

```
<!-- Header text style -->
<Style x:Key="headerTextStyle">
    <Setter Property="Label.VerticalAlignment" Value="Center"></Setter>
    <Setter Property="Label.FontFamily" Value="Trebuchet MS"></Setter>
    <Setter Property="Label.FontWeight" Value="Bold"></Setter>
    <Setter Property="Label.FontSize" Value="18"></Setter>
    <Setter Property="Label.Foreground" Value="#0066cc"></Setter>
</Style>

<!-- Label style -->
<Style x:Key="labelStyle" TargetType="{x:Type Label}">
    <Setter Property="VerticalAlignment" Value="Top" />
    <Setter Property="HorizontalAlignment" Value="Left" />
    <Setter Property="FontWeight" Value="Bold" />
    <Setter Property="Margin" Value="0,0,0,5" />
</Style>

<!-- DataGrid header style -->
<Style x:Key="columnHeaderStyle" TargetType="{x:Type DataGridColumnHeader}">
    <Setter Property="Height" Value="35" />
    <Setter Property="Padding" Value="5" />
    <Setter Property="Background" Value="#4E87D4" />
    <Setter Property="Foreground" Value="White" />
</Style>

<!-- List header style -->
<Style x:Key="listHeaderStyle" TargetType="{x:Type Border}">
    <Setter Property="Height" Value="35" />
    <Setter Property="Padding" Value="5" />
    <Setter Property="Background" Value="#4E87D4" />
</Style>

<!-- List header text style -->
<Style x:Key="listHeaderTextStyle" TargetType="{x:Type Label}">
    <Setter Property="Foreground" Value="White" />
    <Setter Property="VerticalAlignment" Value="Center" />
    <Setter Property="HorizontalAlignment" Value="Left" />
</Style>

<!-- Button style -->
<Style x:Key="buttonStyle" TargetType="{x:Type Button}">
    <Setter Property="Width" Value="125" />
    <Setter Property="Height" Value="25" />
    <Setter Property="Margin" Value="0,10,0,0" />
    <Setter Property="HorizontalAlignment" Value="Right" />
</Style>
```

이 XAML은 다음 스타일을 추가합니다.

- `headerTextStyle`: 페이지 제목 Label의 형식을 지정합니다.
- `labelStyle`: Label 컨트롤의 형식을 지정합니다.
- `columnHeaderStyle`: DataGridColumnHeader의 형식을 지정합니다.
- `listHeaderStyle`: 목록 헤더 Border 컨트롤의 형식을 지정합니다.
- `listHeaderTextStyle`: 목록 헤더의 Label서식을 지정하려면 .
- `buttonStyle`: 에 `ExpenseItHome.xaml` Button 서식을 지정하려면 .

스타일은 Application.Resources 속성 요소의 리소스 및 자식입니다. 이 위치에서 스타일은 애플리케이션의 모든 요소에 적용됩니다. .NET 앱에서 리소스를 사용하는 예는 응용 프로그램 리소스 사용을참조하십시오.

3. 에서 `ExpenseItHome.xaml` Grid 요소 간의 모든 것을 다음 XAML로 바꿉꿉입니다.

```
    <Grid.Background>
        <ImageBrush ImageSource="watermark.png"  />
    </Grid.Background>

    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="230" />
        <ColumnDefinition />
    </Grid.ColumnDefinitions>

    <Grid.RowDefinitions>
        <RowDefinition/>
        <RowDefinition Height="Auto"/>
        <RowDefinition />
        <RowDefinition Height="Auto"/>
    </Grid.RowDefinitions>

    <!-- People list -->

    <Label Grid.Column="1" Style="{StaticResource headerTextStyle}" >
        View Expense Report
    </Label>

    <Border Grid.Column="1" Grid.Row="1" Style="{StaticResource listHeaderStyle}">
        <Label Style="{StaticResource listHeaderTextStyle}">Names</Label>
    </Border>
    <ListBox Name="peopleListBox" Grid.Column="1" Grid.Row="2">
        <ListBoxItem>Mike</ListBoxItem>
        <ListBoxItem>Lisa</ListBoxItem>
        <ListBoxItem>John</ListBoxItem>
        <ListBoxItem>Mary</ListBoxItem>
    </ListBox>

    <!-- View report button -->
    <Button Grid.Column="1" Grid.Row="3" Click="Button_Click" Style="{StaticResource
buttonStyle}">View</Button>
```

스타일을 적용하면 각 컨트롤의 모양을 정의하는 VerticalAlignment 및 FontFamily 와 같은 속성이 제거되고 바뀝니다. 예를 들어"비용 `headerTextStyle` 보고서 보기"에 Label적용됩니다.

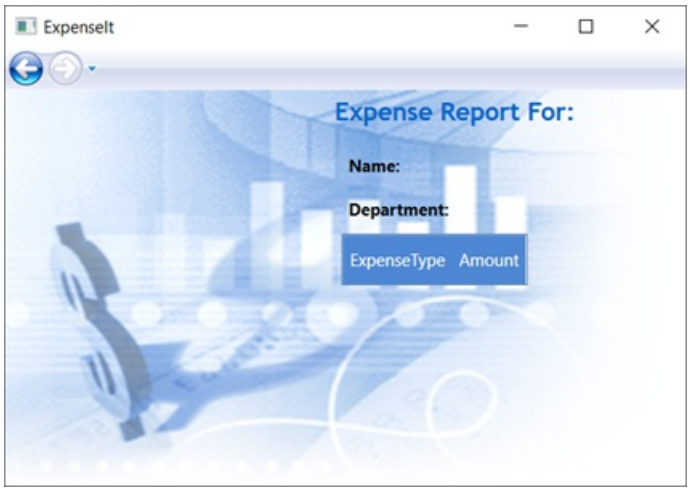
4. *ExpenseReportPage.xaml*을 엽니다.

5. Grid 요소 간의 모든 것을 다음 XAML로 바꿉꿉입니다.

```
<Grid.Background>
    <ImageBrush ImageSource="watermark.png" />
</Grid.Background>
<Grid.ColumnDefinitions>
    <ColumnDefinition Width="230" />
    <ColumnDefinition />
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
    <RowDefinition Height="Auto" />
    <RowDefinition />
</Grid.RowDefinitions>


<Label Grid.Column="1" Style="{StaticResource headerTextStyle}">
    Expense Report For:
</Label>
<Grid Margin="10" Grid.Column="1" Grid.Row="1">

    <Grid.ColumnDefinitions>
        <ColumnDefinition />
        <ColumnDefinition />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition />
    </Grid.RowDefinitions>

    <!-- Name -->
    <StackPanel Grid.Column="0" Grid.ColumnSpan="2" Grid.Row="0" Orientation="Horizontal">
        <Label Style="{StaticResource labelStyle}">Name:</Label>
        <Label Style="{StaticResource labelStyle}"></Label>
    </StackPanel>

    <!-- Department -->
    <StackPanel Grid.Column="0" Grid.ColumnSpan="2" Grid.Row="1"
Orientation="Horizontal">
        <Label Style="{StaticResource labelStyle}">Department:</Label>
        <Label Style="{StaticResource labelStyle}"></Label>
    </StackPanel>

    <Grid Grid.Column="0" Grid.ColumnSpan="2" Grid.Row="2" VerticalAlignment="Top"
          HorizontalAlignment="Left">
        <!-- Expense type and Amount table -->
        <DataGrid ColumnHeaderStyle="{StaticResource columnHeaderStyle}"
                  AutoGenerateColumns="False" RowHeaderWidth="0" >
            <DataGrid.Columns>
                <DataGridTextColumn Header="ExpenseType" />
                <DataGridTextColumn Header="Amount"  />
            </DataGrid.Columns>
        </DataGrid>
    </Grid>
</Grid>
```

이 XAML은 Label 및 Border 요소에 스타일을 추가합니다.

6. 애플리케이션을 빌드 및 실행합니다. 창 모양은 이전과 동일합니다.



7. 응용 프로그램을 닫아 Visual Studio로 돌아갑니다.

데이터를 컨트롤에 바인딩합니다.

이 섹션에서는 다양한 컨트롤에 바인딩된 XML 데이터를 만듭니다.

1. 에서 `ExpenseItHome.xaml` 열기 Grid 요소 후 다음 XAML을 추가하여 XmlDataProvider 각 사용자의 데이터를 포함하는 을 만듭니다.

```
<Grid.Resources>
    <!-- Expense Report Data -->
    <XmlDataProvider x:Key="ExpenseDataSource" XPath="Expenses">
        <x:XData>
            <Expenses xmlns="">
                <Person Name="Mike" Department="Legal">
                    <Expense ExpenseType="Lunch" ExpenseAmount="50" />
                    <Expense ExpenseType="Transportation" ExpenseAmount="50" />
                </Person>
                <Person Name="Lisa" Department="Marketing">
                    <Expense ExpenseType="Document printing"
          ExpenseAmount="50"/>
                    <Expense ExpenseType="Gift" ExpenseAmount="125" />
                </Person>
                <Person Name="John" Department="Engineering">
                    <Expense ExpenseType="Magazine subscription"
         ExpenseAmount="50"/>
                    <Expense ExpenseType="New machine" ExpenseAmount="600" />
                    <Expense ExpenseType="Software" ExpenseAmount="500" />
                </Person>
                <Person Name="Mary" Department="Finance">
                    <Expense ExpenseType="Dinner" ExpenseAmount="100" />
                </Person>
            </Expenses>
        </x:XData>
    </XmlDataProvider>
</Grid.Resources>
```

   데이터는 리소스로 Grid 만들어집니다. 일반적으로 이 데이터는 파일로 로드되지만 간단히 하기 위해 데이터가 인라인으로 추가됩니다.

2. `<Grid.Resources>` 요소 내에서 다음 `<xref:System.Windows.DataTemplate>` 요소를 추가하여 요소 다음에 데이터를 ListBox표시하는 방법을 정의합니다. `<XmlDataProvider>`

```
<Grid.Resources>
    <!-- Name item template -->
    <DataTemplate x:Key="nameItemTemplate">
        <Label Content="{Binding XPath=@Name}"/>
    </DataTemplate>
</Grid.Resources>
```

데이터 템플릿에 대한 자세한 내용은 데이터 템플릿 개요를참조하십시오.

3. 기존 ListBox XAML을 다음 XAML로 바꿉꿉입니다.

```
<ListBox Name="peopleListBox" Grid.Column="1" Grid.Row="2"
         ItemsSource="{Binding Source={StaticResource ExpenseDataSource}, XPath=Person}"
         ItemTemplate="{StaticResource nameItemTemplate}">
</ListBox>
```

이 XAML은 ItemsSource 의 속성을 ListBox 데이터 원본에 바인딩하고 데이터 ItemTemplate템플릿을 으로 적용합니다.

데이터를 컨트롤에 연결

다음으로 페이지에서 선택한 `ExpenseItHome` 이름을 검색하고 **ExpenseReportPage**의 생성자에게 전달하는 코드를 추가합니다. **ExpenseReportPage**는 *ExpenseReportPage.xaml*에 바인딩하는 컨트롤인 전달된 항목으로 데이터 컨텍스트를 설정합니다.

1. 비용 *보고서 열기페이지.xaml.vb* 또는 *ExpenseReportPage.xaml.cs*.

2. 선택한 사람의 비용 보고서 데이터를 전달할 수 있도록 개체를 사용하는 생성자를 추가합니다.

```
public partial class ExpenseReportPage : Page
{
    public ExpenseReportPage()
    {
        InitializeComponent();
    }

    // Custom constructor to pass expense report data
    public ExpenseReportPage(object data):this()
    {
        // Bind to expense report data.
        this.DataContext = data;
    }
}
```

```
Partial Public Class ExpenseReportPage
    Inherits Page
    Public Sub New()
        InitializeComponent()
    End Sub

    ' Custom constructor to pass expense report data
    Public Sub New(ByVal data As Object)
        Me.New()
        ' Bind to expense report data.
        Me.DataContext = data
    End Sub

End Class
```

3. 열기 *ExpenseItHome.xaml.vb* *ExpenseItHome.xaml.cs* 또는 .

4. Click 이벤트 처리기를 변경하여 선택한 사람의 비용 보고서 데이터를 전달하는 새 생성자를 호출합니다.

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    // View Expense Report
    ExpenseReportPage expenseReportPage = new ExpenseReportPage(this.peopleListBox.SelectedItem);
    this.NavigationService.Navigate(expenseReportPage);
}
```

```
Private Sub Button_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
    ' View Expense Report
    Dim expenseReportPage As New ExpenseReportPage(Me.peopleListBox.SelectedItem)
    Me.NavigationService.Navigate(expenseReportPage)

End Sub
```

데이터 템플릿으로 데이터 스타일

이 섹션에서는 데이터 템플릿을 사용하여 데이터 바인딩 된 목록의 각 항목에 대한 UI를 업데이트합니다.

1. *ExpenseReportPage.xaml*을 엽니다.

2. "이름" 및 "Department" Label 요소의 내용을 적절한 데이터 원본 속성에 바인딩합니다. 데이터 바인딩에 대한 자세한 내용은 데이터 바인딩 개요를참조하십시오.

```
<!-- Name -->
<StackPanel Grid.Column="0" Grid.ColumnSpan="2" Grid.Row="0" Orientation="Horizontal">
    <Label Style="{StaticResource labelStyle}">Name:</Label>
    <Label Style="{StaticResource labelStyle}" Content="{Binding XPath=@Name}"></Label>
</StackPanel>

<!-- Department -->
<StackPanel Grid.Column="0" Grid.ColumnSpan="2" Grid.Row="1" Orientation="Horizontal">
    <Label Style="{StaticResource labelStyle}">Department:</Label>
    <Label Style="{StaticResource labelStyle}" Content="{Binding XPath=@Department}"></Label>
</StackPanel>
```

3. 열기 Grid 요소 후 비용 보고서 데이터를 표시 하는 방법을 정의 하는 다음 데이터 템플릿을 추가 합니다.

```
<!--Templates to display expense report data-->
<Grid.Resources>
    <!-- Reason item template -->
    <DataTemplate x:Key="typeItemTemplate">
        <Label Content="{Binding XPath=@ExpenseType}"/>
    </DataTemplate>
    <!-- Amount item template -->
    <DataTemplate x:Key="amountItemTemplate">
        <Label Content="{Binding XPath=@ExpenseAmount}"/>
    </DataTemplate>
</Grid.Resources>
```

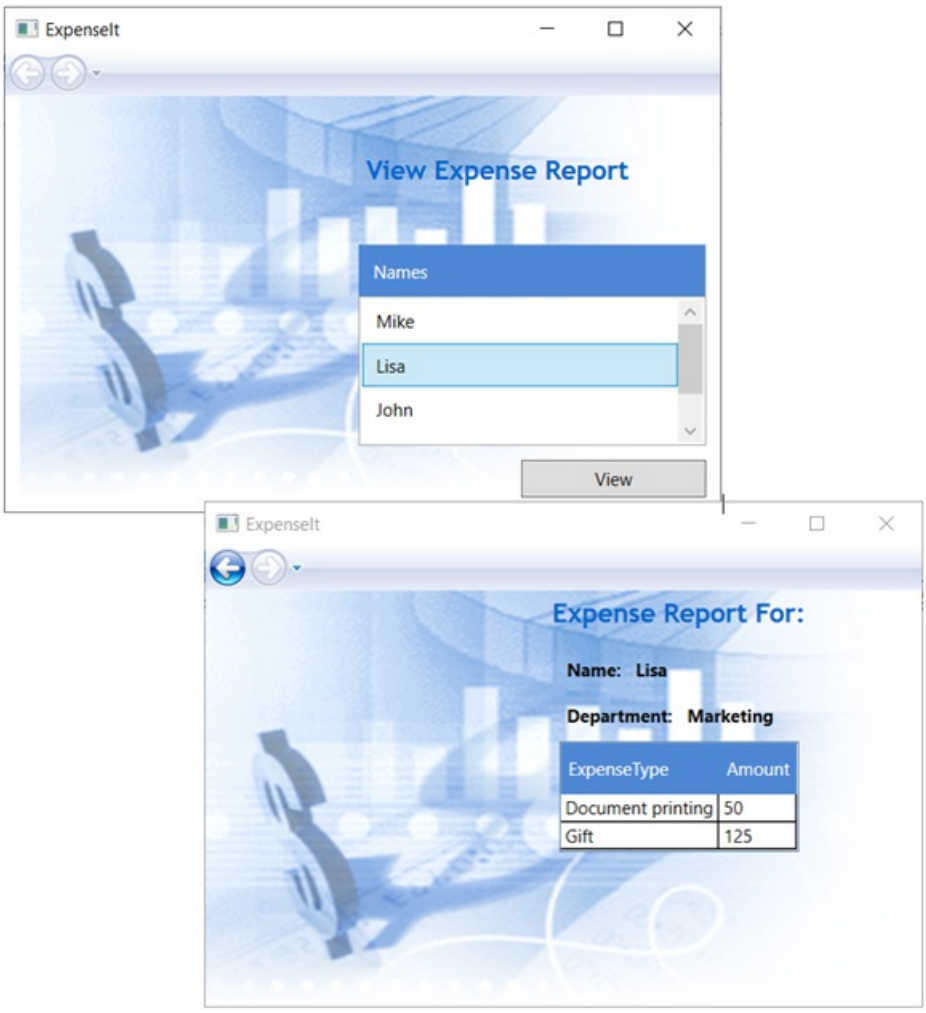
4. DataGridTextColumn 요소를 DataGrid 요소 아래에 있는 요소로 DataGridTemplateColumn 바꾸고 템플릿을 적용합니다.

```
<!-- Expense type and Amount table -->
<DataGrid ItemsSource="{Binding XPath=Expense}" ColumnHeaderStyle="{StaticResource columnHeaderStyle}"
AutoGenerateColumns="False" RowHeaderWidth="0" >

    <DataGrid.Columns>
        <DataGridTemplateColumn Header="ExpenseType" CellTemplate="{StaticResource typeItemTemplate}" />
        <DataGridTemplateColumn Header="Amount" CellTemplate="{StaticResource amountItemTemplate}" />
    </DataGrid.Columns>

</DataGrid>
```

5. 애플리케이션을 빌드 및 실행합니다.

6. 사람을 선택한 다음 **보기** 단추를 선택합니다.

다음 그림에서는 컨트롤, `ExpenseIt` 레이아웃, 스타일, 데이터 바인딩 및 데이터 템플릿이 적용된 응용 프로그램의 두 페이지를 보여 주었습니다.



NOTE

이 샘플에서는 WPF의 특정 기능을 보여 주며 보안, 지역화 및 접근성과 같은 모든 모범 사례를 따르지 않습니다. WPF 및 .NET 앱 개발 모범 사례에 대한 포괄적인 내용은 다음 항목을 참조하십시오.

- 액세스 가능성
- 보안
- WPF 전역화 및 지역화
- WPF 성능

다음 단계

이 연습에서는 WPF(Windows 프레젠테이션 파운데이션)를 사용하여 UI를 만드는 여러 가지 기술을 배웠습니다. 이제 데이터 바인딩 .NET 앱의 구성 요소를 기본적으로 이해해야 합니다. WPF 아키텍처 및 프로그래밍 모델에 대한 자세한 내용은 다음 항목을 참조하세요.

- WPF 아키텍처
- XAML 개요(WPF)
- 종속성 속성 개요
- 레이아웃

애플리케이션을 만드는 방법에 대한 자세한 내용은 다음 항목을 참조하세요.

- 애플리케이션 개발
- 컨트롤
- 데이터 바인딩 개요
- 그래픽 및 멀티미디어
- WPF의 문서

참고 항목

- 패널 개요
- 데이터 템플릿 개요
- WPF 응용 프로그램 빌드
- 스타일 및 템플릿

WPF 연습

2020-02-03 • 6 minutes to read • Edit Online

연습에서는 일반적인 시나리오에 대 한 단계별 지침을 제공 합니다. 제품 또는 특정 기능 영역 파악을 효율적으로 시작할 수 있습니다.

이 항목에는 Windows Presentation Foundation (WPF) 연습에 대 한 링크가 포함 되어 있습니다.

WPF 디자이너 연습

제목	DESCRIPTION
연습: WPF Designer를 사용 하 여 간단한 WPF 응용 프로그램 빌드	WPF Designer를 사용 하 여 간단한 WPF 응용 프로그램을 빌드하는 방법을 보여 줍니다.
연습: 동적 레이아웃 생성	Grid panel 컨트롤을 사용 하 여 동적 레이아웃을 만드는 방법을 보여 줍니다.
연습: WPF Designer를 사용 하 여 크기 조정 가능한 응용 프로그램 만들기	런타임에 사용자가 크기를 조정할 수 있는 창 레이아웃을 만드는 방법을 보여 줍니다.
연습: WPF Designer를 사용 하 여 데이터 바인딩 만들기	WPF 디자이너를 사용 하 여 데이터를 컨트롤에 연결 하는 데이터 바인딩을 만드는 방법을 보여 줍니다.
연습: DesignInstance를 사용 하 여 디자이너에서 데이터에 바인딩	WPF 디자이너를 사용 하 여 런타임에 할당 된 데이터 컨텍스트에 대해 디자인 타임에 데이터 바인딩을 만드는 방법을 보여 줍니다.

WPF 연습

제목	DESCRIPTION
연습: 내 첫 WPF 데스크톱 애플리케이션	컨트롤, 레이아웃 및 데이터 바인딩을 포함 하는 WPF의 여러 일반적인 기능을 사용 하 여 WPF 응용 프로그램을 만드는 방법을 보여 줍니다.
XAML을 사용하여 단추 만들기	애니메이션 단추를 만드는 방법을 보여 줍니다.
Microsoft Expression Blend를 사용하여 단추 만들기	Microsoft Expression Blend를 사용 하 여 사용자 지정 된 단추를 만드는 과정을 보여 줍니다.
연습: DataGrid 컨트롤에서 SQL Server 데이터베이스의 데이터 표시	SQL Server 데이터베이스에서 데이터를 검색 하 고 DataGrid 컨트롤에 해당 데이터를 표시 하는 방법을 보여 줍니다.

WPF의 마이그레이션 및 상호 운용성

제목	DESCRIPTION

제목	DESCRIPTION
연습: WPF에서 Windows Forms 컨트롤 호스팅	WPF 응용 프로그램에서 Windows Forms System.Windows.Forms.MaskedTextBox 컨트롤을 호스트 하는 방법을 보여 줍니다.
연습: WPF에서 Windows Forms 복합 컨트롤 호스팅	WPF 응용 프로그램에서 Windows Forms 데이터 엔트리 복합 컨트롤을 호스트 하는 방법을 보여 줍니다.
연습: Windows Forms에서 WPF 복합 컨트롤 호스팅	Windows Forms 응용 프로그램에서 WPF 데이터 엔트리 복합 컨트롤을 호스트 하는 방법을 보여 줍니다.
연습: WPF에서 Windows Forms 컨트롤 정렬	WPF 레이아웃 기능을 사용 하 여 혼합 응용 프로그램에서 Windows Forms 컨트롤을 정렬 하는 방법을 보여 줍니다.
연습: 혼합 애플리케이션에서 데이터 바인딩	Windows Forms 및 WPF 컨트롤을 모두 포함 하는 혼합 응용 프로그램에서 데이터 바인딩을 사용 하는 방법을 보여 줍니다.

관련 섹션

제목	DESCRIPTION
Visual Studio 연습	Visual Studio의 모든 프로그래밍 영역에 대 한 관련 연습 목록을 제공 합니다.

WPF 커뮤니티 사용자 의견

2020-02-12 • 15 minutes to read • Edit Online

Microsoft는 Windows Presentation Foundation (WPF)에 대해 배우고 논의 하 고 피드백을 제공할 수 있는 다양 한 커뮤니티 리소스를 제공 합니다. 이러한 리소스에는 포럼 및 Visual Studio 개발자 커뮤니티 사이트가 포함 됩니다. 각 커뮤니티 리소스는 서로 다른 혜택을 제공합니다. 이러한 혜택은 각각을 사용 하 여 광범위 하 고 Microsoft 커뮤니티에서 가장 적합 한 응답을 보장 하기 위한 모범 사례 모음에 설명 되어 있습니다.

> **NOTE**
> 각 페이지의 아래쪽에 있는 사용자 의견 섹션을 사용 하 여 제품 피드백을 보내세요. 이러한 링크는 문서에 대 한 사용자 의견에만 사용됩니다.

포럼

WPF 포럼 은 문제를 논의 하 고 해결 하기 위한 기본 커뮤니티 리소스입니다. 포럼에서는 다음을 포함한 포괄적인 지원 기능 집합을 제공하는 방식으로 쉽게 토론하고 문제를 해결하도록 합니다.

- 검색.
- 토론 추적
- 텍스트 및 코드의 다양한 서식 지정
- Visual Studio 통합
- MVP(Most Valued Professional) 및 커뮤니티 참여
- 게시물이 가장 빠른 시간 내에 응답되는지 확인하기 위한 모니터링.

WPF에 대 한 커뮤니티에 질문할 수 있는 다른 옵션은 Stack Overflow입니다.

포럼 모범 사례

다음 모범 사례를 사용 하면 가능한 가장 빠른 시간 내에 WPF 포럼에 게시 된 문제를 해결할 수 있습니다. 이러한 사례는 모든 포럼에 적용할 수 있습니다.

기존 게시물 검색

몇몇 문제는 이전에 다른 사용자가 경험할 만큼 광범위하게 발생합니다. 따라서 문제를 빠르게 해결할 수 있고 기존 토론에 입력을 추가할 수도 있습니다.

의미 있는 제목 사용

간결 하 고 의미 있는 제목은 게시물의 검색 가능성을 향상 시킵니다. 또한 다른 WPF 포럼 커뮤니티 멤버가 문제를 해결할 수 있는지 쉽게 확인할 수 있습니다.

적절 한 콘텐츠 포함

문제 및 문제를 해결 하는 방법을 설명 합니다. 가능한 경우 지원 코드 조각 또는 문제를 보여 주는 가장 간단한 가능한 샘플을 포함합니다. 이러한 모든 세부 정보를 포함하면 질문에 대 한 답변을 빠르게 받을 가능성이 커집니다.

Visual Studio 개발자 커뮤니티

문제를 해결하기 어렵거나 해결할 수 없는 경우도 있습니다. 이러한 상황은 기술상 버그, 특정 시나리오에 기술을 적용하기 어려움 또는 특정 시나리오에 대 한 지원 결여 때문에 발생합니다. 이 정보는 Microsoft에 중요 하며 Visual Studio 개발자 커뮤니티 사이트를 통해 제공할 수 있습니다.

WPF 제품 피드백 센터에 게시 된 항목은 WPF 팀의 내부 버그 데이터베이스로 라우팅됩니다. 따라서 WPF 기능 소유자에 게 피드백을 가져오는 가장 안정적인 방법입니다. 또한 제안 및 버그의 유효성을 검사 하 고 추적 하 여 WPF

팀에서 문제에 대 한 우선 순위를 지정할 수 있습니다.

개발자 커뮤니티 모범 사례

Visual Studio 개발자 커뮤니티에 게시 하는 경우에는 WPF 포럼에 게시 하는 것 처럼 기존 게시물을 검색 하 고, 의미 있는 제목 및 적절 한 콘텐츠를 제공 하는 것이 중요 한 모범 사례입니다. 적용해야 하는 추가 모범 사례는 다음과 같습니다.

기존 게시물 검색

몇몇 문제는 이전에 다른 사용자가 경험할 만큼 광범위하게 발생합니다. 따라서 문제를 신속 하 게 해결 하거나 기존 문제에 입력을 추가할 수 있습니다.

의미 있는 제목 사용

간결 하 고 의미 있는 제목을 통해 가장 적절 한 시간 내에 가장 적절 한 WPF 팀에 문제가 전송 될 가능성을 높일 수 있습니다. 이는 여러 상호 관련 기능을 포함 하는 WPF와 같은 기술에 특히 중요 합니다.

버그를 재현 하는 방법에 대해 설명 합니다.

버그에 대해 게시할 때 관련되어 있는 경우 다음을 포함해야 합니다.

- 버그에 대한 분명한 설명을 제공합니다.
- 코드 조각을 사용하여 버그 설명을 지원합니다.
- 버그 재현 방법을 보여 주는 단계 목록을 제공합니다.
- 버그를 재현하는 가장 작은 가능한 코드 샘플을 포함합니다.
- 버그가 일관되게 재현 가능한지 여부를 언급합니다.
- 관련 예외 정보를 포함합니다.

버그가 설치 또는 설정에 관련된 경우 관련 설치 로그 및 스냅샷(%temp% 폴더에 있는 "dd_" 접두사가 추가된 파일)을 첨부합니다.

컴파일 또는 빌드 문제인 경우 빌드 로그를 첨부합니다. 명령줄에서/v: 스위치를 사용 하거나 Visual Studio와 같은 IDE (통합 개발 환경)에서 적절 한 수준을 구성 하 여 다양 한 하거나의 로깅을 지원 하도록 MSBuild 시스템을 구성할 수 있습니다.

환경 정보 제공

게시물에 컨텍스트를 추가할 때 배경 정보가 도움이 될 수 있습니다. 특히 운영 체제 플랫폼, 프로세서 제품군 및 아키텍처 (예: "Windows 10 버전 1709, Intel (R) Xeon (R), x64")를 언급 합니다.

게시 중인 문제가 렌더링에 관련될 경우 그래픽 카드 및 드라이버 세부 정보도 포함해야 합니다(가능한 경우). WPF는 프레젠테이션 프레임 워크 이므로이 정보는 중요 합니다.

솔루션 또는 프로젝트 정보 제공

버그는 애플리케이션을 개발하고 빌드하는 데 사용되는 도구 및 빌드 중인 애플리케이션 형식에 관련될 수 있습니다. 따라서 다음을 지정하면 도움이 될 수 있습니다.

- 빌드 중인 애플리케이션 형식(예:
  - 응용 프로그램 (*.exe*) 또는 라이브러리 (*.dll*)
  - Extensible Application Markup Language (XAML) 브라우저 응용 프로그램 (XBAP)
  - 느슨한 XAML 응용 프로그램
  - 독립 실행형 설치 응용 프로그램
  - 독립 실행형 ClickOnce 배포 응용 프로그램
- 개발 도구(예:
  - MSBuild
  - 식 그래픽 디자이너
  - 식 대화형 디자이너
  - Visual Studio
- 솔루션 구성(예:

    - 솔루션
    - 단일 프로젝트
    - 여러 종속 프로젝트가 포함 된 솔루션
- 애플리케이션에 언어별 또는 언어 중립적인 리소스가 포함되는지 여부. 예를 들어 `UICulture`, `Application` 및 `Page` 형식에 `Resource` 프로젝트 속성 또는 지역화 가능한 메타데이터를 지정했나요?
- 중립적인 언어 설정을 AssemblyInfo.cs 또는 AssemblyInfo.vb 파일에서 사용했는지 여부.

시나리오 및 영향 정보 제공

버그 및 해당 영향을 매니페스트 하는 시나리오에 대 한 정보를 제공 합니다. 이 정보는 WPF 팀에서 문제를 수정 해야 하는 경우, 시기 및 방법을 결정 하는 경우 또는 적절 한 해결 방법을 대신 사용할 수 있는지 여부를 결정 하는 경우에 매우 중요 합니다.

일반적으로 충돌 및 데이터 손실 시나리오는 영향이 크기 때문에 우선 순위를 정하기가 가장 쉽습니다. 하지만 일부 버그는 흔하지 않은 시나리오에서만 나타납니다. 경우에 따라 이러한 시나리오가 주요 시나리오일 수도 있습니다. 시나리오 및 영향에 대 한 컨텍스트를 제공 하면 WPF 팀에서 적절 한 의사 결정을 내리는 데 도움이 됩니다.

참고 항목

- Visual Studio의 문제를 보고하는 방법