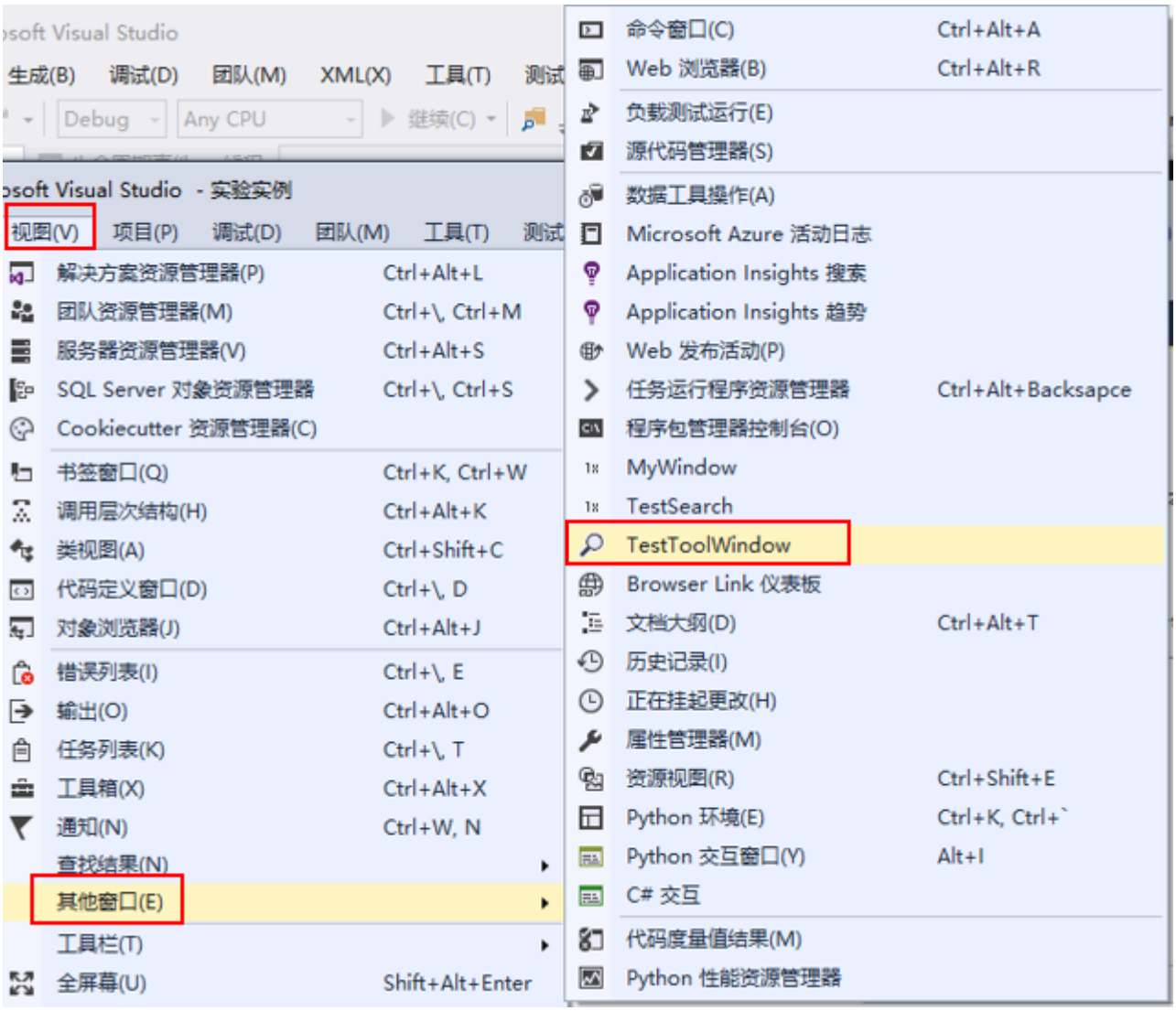
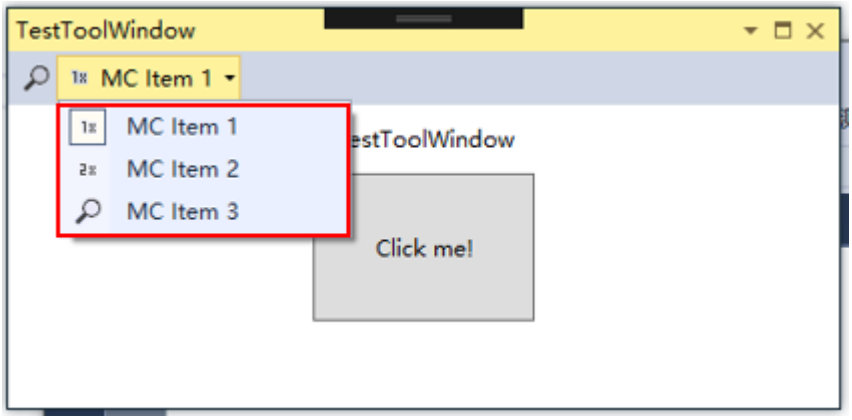
将菜单控制器添加到工具栏

实验结果：

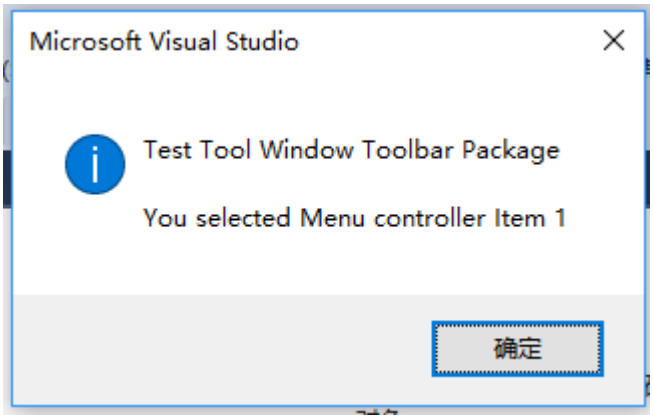


菜单控制器是拆分控件。 菜单控制器的左侧显示最近使用的命令，单击命令可以运行。 右侧的菜单控制器是一个向下的小三角，单击这个小三角时，可以打开其他命令的列表。如下图：它的列表中有三个MC Item， 当您单击列表中的某一命令时，这个命令运行，并且它取代了菜单控制器左侧命令。 菜单控制器始终会显示一个列表中的最近使用的命令的命令按钮。

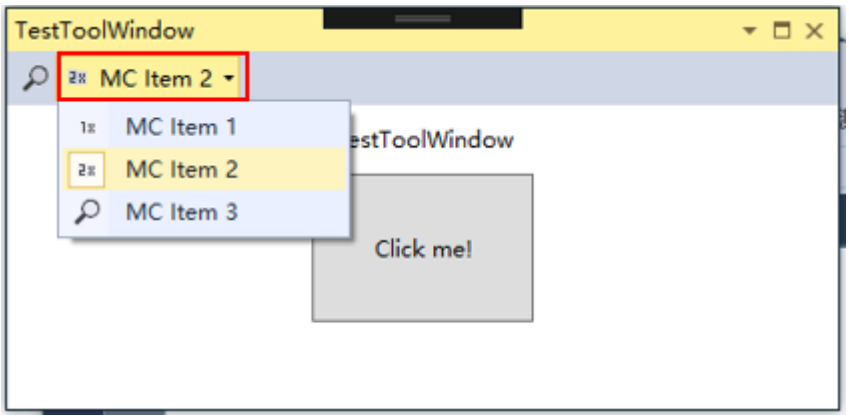
菜单控制器可以出现在菜单上，但它们最常用于在工具栏上。



由上图可以看到三个MC Item项，其中第一个已选中. 单击**MC Item 1**。会弹出下列对话框



当点击MC Item 2时，菜单控制器左侧会出现MC Item 2，取代了MC Item 1.如下图

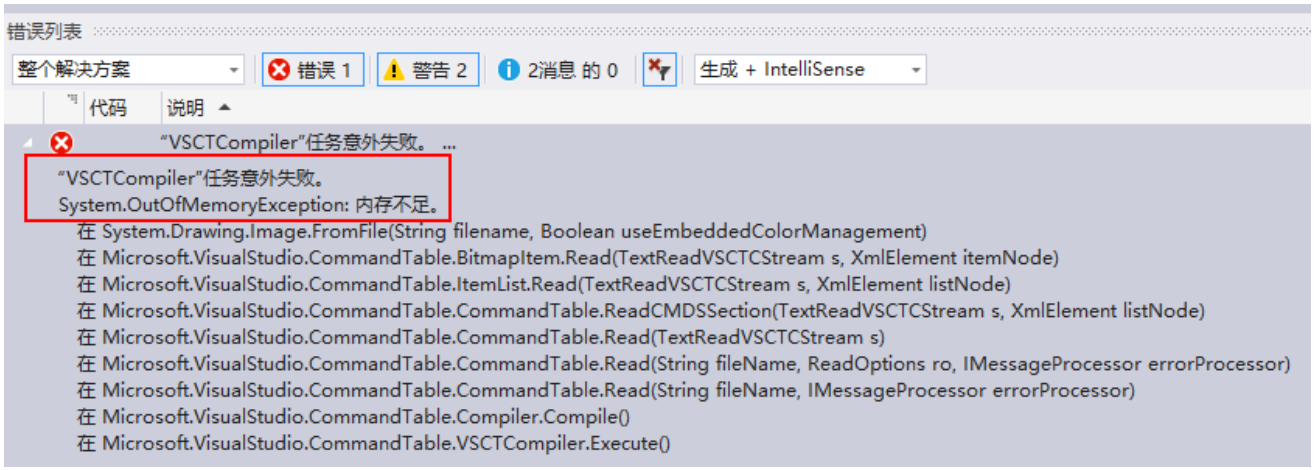


实现过程：

从上面图中可以看出此菜单控制器列表中有三个MC Item，那么这三个项是怎么关联起来的呢，首先还是定义一个菜单控制器组，然后把这三个项添加到这个组中，这样这三个项就和这个菜单控制器关联起来了，然后把这个菜单控制器添加到工具栏组中，接着把这个工具栏组和工具栏关联起来，这样就把这个菜单控制器就添加到了工具栏。

代码链接：https://msdn.microsoft.com/zh-cn/library/bb165748.aspx

实验过程中遇到的问题：



在网上找到的解决方法：http://www.datazx.cn/visualcsharp/20161111492.html

找到方法了之后，然后去试验：打开项目所在的文件夹，进入Resources目录下可以看到下面三个文件，发现TestToolWindowCommand.png无法正常显示，模板提供的默认的png有问题

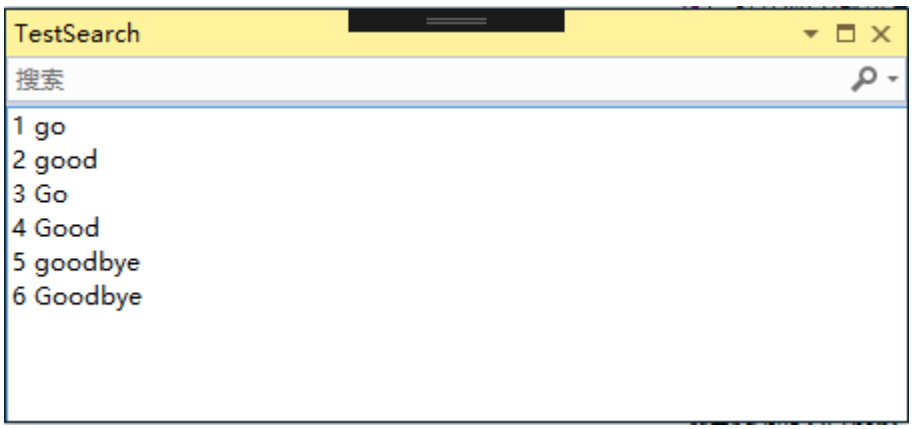
在TWTestCommandPackage.vsct文件中可以看到在标签下引用了这两个.png文件，这两个图片里面都是六个图标，于是我把TestToolWindowCommand.png删了，重新复制了TWTestCommand.png，并改名为TestToolWindowCommand.png，然后重新生成代码并调试，问题解决了。

```
<Bitmaps>
  <Bitmap guid="guidImages" href="Resources\TWTestCommand.png" usedList="bmpPic1, bmpPic2, bmpPicSearch, bmpPicX, bmpPicArrows, bmpPicStrikethrough" />
  <Bitmap guid="guidImages1" href="Resources\TestToolWindowCommand.png" usedList="bmpPic1, bmpPic2, bmpPicSearch, bmpPicX, bmpPicArrows, bmpPicStrikethrough" />
</Bitmaps>
```

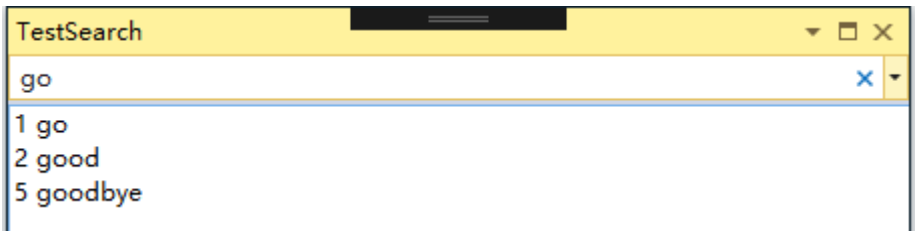
添加搜索框到工具栏

实验结果如下：

在工具窗口的顶部，出现了搜索水印和放大镜图标。



在搜索窗口中输入一些文本，然后单击Enter。出现正确的结果。



搜索框右面有个小三角，当点击小三角的时候，出现下拉列表，下拉列表中有你的搜索历史，还有一个区分大小写的复选框，当你点击了这个复选框之后，搜索的时候就会匹配大小写的搜索

实现过程：

创建 VSPackage 项目。

添加一个只读的TextBox到UserControl工具窗口中。

向工具窗口添加一个搜索框

添加搜索实现。

启用即时搜索和显示一个进度栏。

添加区分大小写选项。

首先创建一个工具窗口，然后添加一个只读的TextBox到UserControl工具窗口中。

在`TestToolWindowSearch`项目中，打开 TestSearchControl.xaml 文件。用下面的内容替换现有`<StackPanel>`块

```
<StackPanel Orientation="Vertical">
    <TextBox Name="resultsTextBox" Height="800.0"  Width="800.0"  IsReadOnly="True">
    </TextBox>
</StackPanel>
```

在TestSearchControl类中，添加下面的代码。此代码将添加一个公共TextBox属性名为SearchResultsTextBox和一个名为SearchContent的公共字符串属性。 在构造函数中，SearchResultsTextBox 被设置为文本框中，同时 SearchContent 初始化到一组新的换行分隔的字符串集。 文本框中的内容也将初始化为字符串集。

```
public partial class TestSearchControl : UserControl
{
    public TextBox SearchResultsTextBox { get; set; }
    public string SearchContent { get; set; }

    public TestSearchControl()
    {
        InitializeComponent();

        this.SearchResultsTextBox = resultsTextBox;
        this.SearchContent = BuildContent();

        this.SearchResultsTextBox.Text = this.SearchContent;
    }

    private string BuildContent()
    {
        StringBuilder sb = new StringBuilder();
        sb.AppendLine("1 go");
        sb.AppendLine("2 good");
        sb.AppendLine("3 Go");
        sb.AppendLine("4 Good");
        sb.AppendLine("5 goodbye");
        sb.AppendLine("6 Goodbye");

        return sb.ToString();
    }
}
```

向工具窗口添加一个搜索框

在 TestSearch.cs 文件中，添加以下代码到TestSearch类。 该代码重写SearchEnabled属性，以便 get 访问器返回 true。若要启用搜索，必须重写SearchEnabled属性。 ToolWindowPane类实现IVsWindowSearch ，并提供一个默认实现，不能通过搜索。

```
public override bool SearchEnabled
{
    get { return true; }
}
```

添加搜索实现

重写CreateSearch方法来创建一个搜索任务。重写ClearSearch方法以恢复文本框的状态。当用户取消搜索任务，当用户设置或取消了选项或过滤器时，就会调用此方法。在UI线程上调用CreateSearch和ClearSearch。因此，您不需要通过ThreadHelper访问文本框。创建一个名为TestSearchTask的类，它继承了VsSearchTask，它提供了IVsSearchTask的默认实现。在TestSearchTask中，构造函数设置引用工具窗口的私有字段。为了提供搜索方法，重写OnStartSearch和OnStopSearch方法。OnStartSearch方法是实现搜索过程的地方。这个过程包括执行搜索，在文本框中显示搜索结果，并调用该方法的基类实现来报告搜索已经完成。

自定义搜索行为

通过更改搜索设置，您可以对搜索控件的显示方式以及搜索的执行方式进行各种更改。 例如，可以更改水印 （在搜索框中显示的默认文本）、 所需的最低和最大宽度的搜索控件，以及是否显示一个进度栏。您还可以更改搜索结果开始出现的点(按需或即时搜索)，以及是否显示最近搜索的术语列表。您可以在SearchSettingsDataSource类中找到设置的完整列表。

在 TestSearch.cs 文件中，添加以下代码TestSearch类。 此代码启用即时搜索而不是按需搜索 （用户无需单击输入的含义）。 该代码重写ProvideSearchSettings中的方法TestSearch类，该类是需要更改默认设置。

```
public override void ProvideSearchSettings(IVsUIDataSource pSearchSettings)
{
    Utilities.SetValue(pSearchSettings,
        SearchSettingsDataSource.SearchStartTypeProperty.Name,
        (uint)VSSEARCHSTARTTYPE.SST_INSTANT);
}
```

生成并调试程序，每次你在搜索框中输入一个字符，搜索结果就会出现。

在`ProvideSearchSettings`方法中，添加以下行，可以显示一个进度栏。

```
public override void ProvideSearchSettings(IVsUIDataSource pSearchSettings)
{
    Utilities.SetValue(pSearchSettings,
        SearchSettingsDataSource.SearchStartTypeProperty.Name,
         (uint)VSSEARCHSTARTTYPE.SST_INSTANT);
    Utilities.SetValue(pSearchSettings,
        SearchSettingsDataSource.SearchProgressTypeProperty.Name,
         (uint)VSSEARCHPROGRESSTYPE.SPT_DETERMINATE);
}
```

若要显示的进度条，必须报告进度。 若要报告进度，请取消注释下面的代码`OnStartSearch`方法`TestSearchTask`类：

```
SearchCallback.ReportProgress(this, progress++, (uint)contentArr.GetLength(0));
```

缓慢足够的处理进度条是否可见，请取消注释中的以下行将`OnStartSearch`方法`TestSearchTask`类：

```
System.Threading.Thread.Sleep(100);
```

通过重新生成解决方案并启动到 debugb 测试新的设置。

进度栏出现在搜索窗口 （作为一条蓝线下方的搜索文本框） 每次执行搜索。

若要使用户能够改进其搜索

您可以允许用户以如通过选项来优化其搜索**区分大小写**或**全字匹配**。 选项可以是布尔值，它显示为复选框或显示为按钮的命令。 对于本演练，你将创建一个布尔值的选项。

在 TestSearch.cs 文件中，添加以下代码到`TestSearch`类。 该代码重写`SearchOptionsEnum`方法，它允许搜索实现来检测到给定的选项是开还是关。 中的代码`SearchOptionsEnum`中添加一个选项以匹配用例与[IVsEnumWindowSearchOptions](枚举器。 区分大小写的选项都还可作为`MatchCaseOption`属性。

```
private IVsEnumWindowSearchOptions m_optionsEnum;
public override IVsEnumWindowSearchOptions SearchOptionsEnum
{
    get
    {
        if (m_optionsEnum == null)
        {
            List<IVsWindowSearchOption> list = new List<IVsWindowSearchOption>();

            list.Add(this.MatchCaseOption);

            m_optionsEnum = new WindowSearchOptionEnumerator(list) as IVsEnumWindowSearchOptions;
        }
        return m_optionsEnum;
    }
}

private WindowSearchBooleanOption m_matchCaseOption;
public WindowSearchBooleanOption MatchCaseOption
{
    get
    {
        if (m_matchCaseOption == null)
        {
            m_matchCaseOption = new WindowSearchBooleanOption("Match case", "Match case", false);
        }
        return m_matchCaseOption;
    }
}
```

在`TestSearchTask`类中，取消注释中的 matchCase 行`OnStartSearch`方法：

```
private IVsEnumWindowSearchOptions m_optionsEnum;
public override IVsEnumWindowSearchOptions SearchOptionsEnum
{
    get
    {
        if (m_optionsEnum == null)
        {
            List<IVsWindowSearchOption> list = new List<IVsWindowSearchOption>();

            list.Add(this.MatchCaseOption);

            m_optionsEnum = new WindowSearchOptionEnumerator(list) as IVsEnumWindowSearchOptions;
        }
        return m_optionsEnum;
    }
}

private WindowSearchBooleanOption m_matchCaseOption;
public WindowSearchBooleanOption MatchCaseOption
{
    get
     {
        if (m_matchCaseOption == null)
        {
            m_matchCaseOption = new WindowSearchBooleanOption("Match case", "Match case", false);
        }
        return m_matchCaseOption;
    }
}
```

测试选项：

生成项目并启动调试。 将显示的实验实例。

在工具窗口中，选择文本框右侧的向下箭头。

区分大小写显示复选框。

选择区分大小写复选框，然后再执行某些搜索。

您可以访问 Visual Studio 中的任何工具窗口。 本演练演示如何将工具窗口有关的信息集成到一个新选项页和上的新设置属性页上，以及如何将写入到任务列表和输出windows。

先决条件

启动 Visual Studio 2015 中，您并不安装 Visual Studio SDK 从下载中心获得。 它将包括作为 Visual Studio 安装程序中的可选功能。 您还可以在以后安装 VS SDK。 有关详细信息，请参阅安装 Visual Studio SDK。

使用一个工具窗口创建扩展

1. 创建一个名为项目**TodoList**使用 VSIX 模板，并将添加一个名为的自定义工具窗口项模板**TodoWindow**。

> 说明
>
> 有关使用一个工具窗口创建扩展的详细信息，请参阅使用一个工具窗口创建扩展。

设置工具窗口

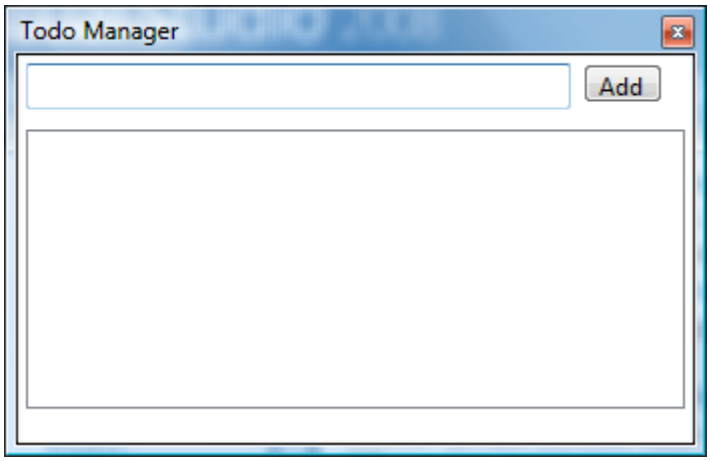
添加在其中可键入一个新的 ToDo 项目，以添加新项到列表中，一个按钮和列表框中显示的项在列表中的文本框。

1. 在 TodoWindow.xaml，从 UserControl 中删除按钮、 文本框中，和 StackPanel 控件。

   > 说明
   >
   > 此时不会删除**button1_Click**事件处理程序，则将在稍后的步骤中重用。

2. 从**所有 WPF 控件**部分**工具箱**，拖动**画布**到网格控件。

3. 拖动**TextBox**、**按钮**，和一个**ListBox**到画布上。 排列元素，使文本框和按钮在同一级别，以及 ListBox 填充其余部分的下方，如下面的图片中所示的窗口。

4. 在 XAML 窗格中，找到按钮，并将其内容的属性设置为**添加**。 通过添加重新连接到按钮控件按钮事件处理程序`Click="button1_Click"`属性。 画布块应如下所示：

   XML

```
<Canvas HorizontalAlignment="Left" Width="306">
    <TextBox x:Name="textBox" HorizontalAlignment="Left" Height="23" Margin="10,10,0,0"
TextWrapping="Wrap" Text="TextBox" VerticalAlignment="Top" Width="208"/>
        <Button x:Name="button" Content="Add" HorizontalAlignment="Left" Margin="236,13,0,0"
VerticalAlignment="Top" Width="48" Click="button1_Click"/>
        <ListBox x:Name="listBox" HorizontalAlignment="Left" Height="222" Margin="10,56,0,0"
VerticalAlignment="Top" Width="274"/>
</Canvas>
```

自定义构造函数

1. 在 TodoWindowControl.xaml.cs 文件中，添加以下 using 语句：

   C#

```
using System;
```

2. 添加对 TodoWindow 的公共引用并让 TodoWindowControl 构造函数采用 TodoWindow 参数。 该代码应如下所示：

   C#

   ```
   public TodoWindow parent;

   public TodoWindowControl(TodoWindow window)
   {
       InitializeComponent();
       parent = window;
   }
   ```

3. TodoWindow.cs，在将更改 TodoWindowControl 构造函数，以便包括 TodoWindow 参数。 该代码应如下所示：

   C#

   ```
   public TodoWindow() : base(null)
   {
       this.Caption = "TodoWindow";
       this.BitmapResourceID = 301;
       this.BitmapIndex = 1;

        this.Content = new TodoWindowControl(this);
   }
   ```

创建选项页

您可以提供中的页**选项**对话框中，以便用户可以更改工具窗口中的设置。 创建选项页需要两个类，用于描述的选项和 TodoListPackage.cs 或 TodoListPackage.vb 文件中的条目。

1. 添加一个名为类`ToolsOptions.cs`。 使 ToolsOptions 类继承自DialogPage。

   C#

   ```
   class ToolsOptions : DialogPage
   {
   }
   ```

2. 添加以下 using 语句：

   C#

```
using Microsoft.VisualStudio.Shell;
```

3. 在本演练中选项页提供了名为 DaysAhead 只能有一个选项。 添加一个名为的私有字段**daysAhead**和一个名为属性**DaysAhead**到 ToolsOptions 类：

   C#

   ```
   private double daysAhead;

   public double DaysAhead
   {
       get { return daysAhead; }
       set { daysAhead = value; }
   }
   ```

现在，您必须使该项目注意该选项页。

向用户提供选项页

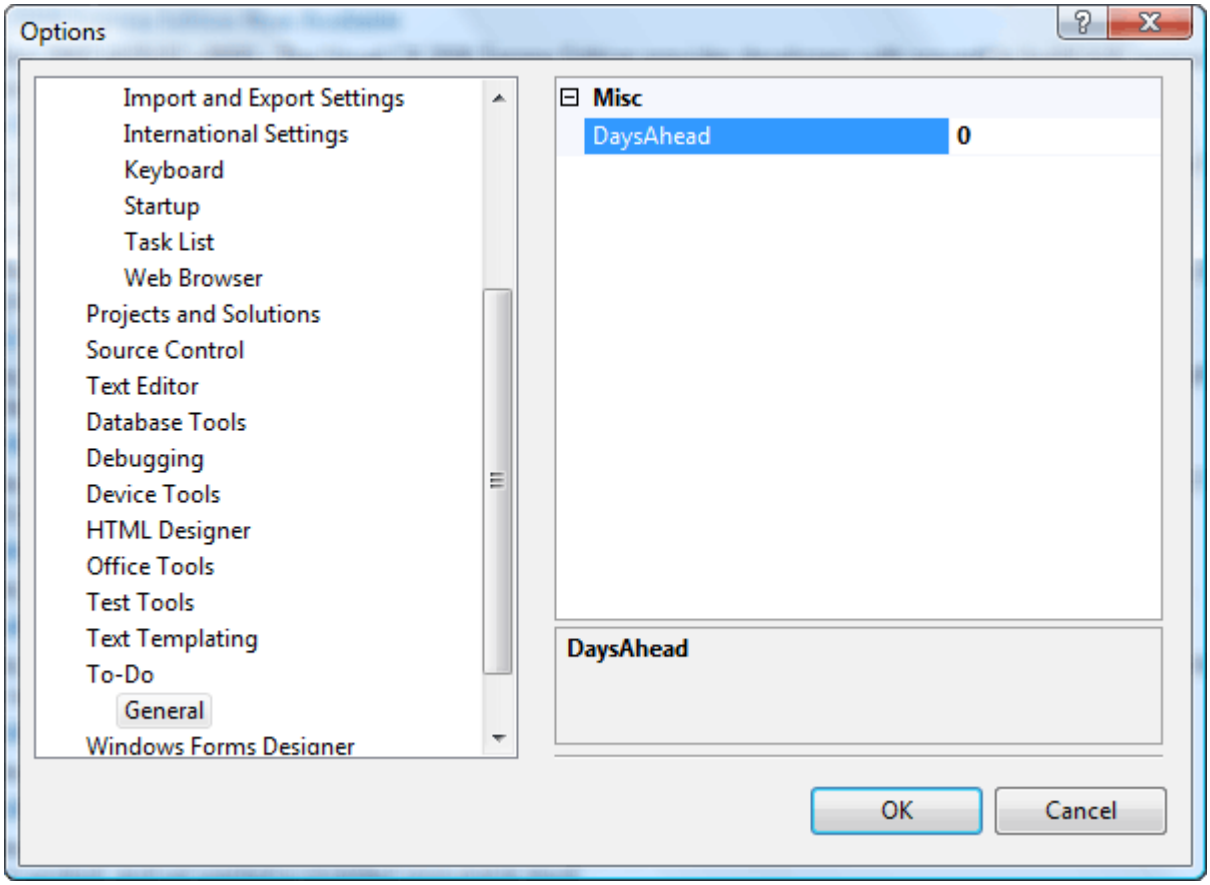
1. 在 TodoWindowPackage.cs，添加ProvideOptionPageAttribute到 TodoWindowPackage 类：

   C#

   ```
   [ProvideOptionPage(typeof(ToolsOptions), "ToDo", "General", 101, 106, true)]
   ```

2. ProvideOptionPage 构造函数的第一个参数是类 ToolsOptions，以前创建的类型。 第二个参数，"ToDo"是中类别的名称**选项**对话框。 第三个参数，"常规"是的子类别的名称**选项**选项页将在其中可用的对话框。 接下来两个参数是字符串; 尝试添加的资源 Id第一种是该类别的名称，第二个是子类别的名称。 最后一个参数确定是否可以通过使用自动化访问此页。

   当用户打开选项页上时，它应类似于下图。



请注意该类别**ToDo**和 subcategory常规。

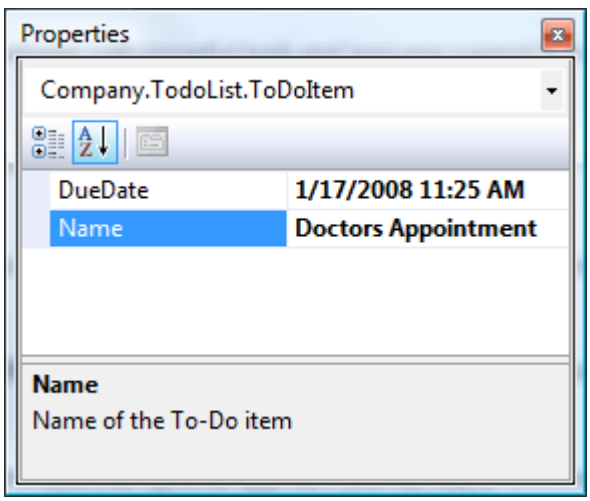
使数据可用于属性窗口

通过创建一个名为将有关的各个项的信息存储在 ToDo 列表中的 TodoItem 类可以使任务列表信息可用。

1. 添加一个名为类`TodoItem.cs`。

   向用户提供的工具窗口时，将由 TodoItems 表示列表框中的项。 当用户选择其中一项在列表框中，**属性**窗口将显示与项有关的信息。

   若要使数据中可用**属性**窗口中，您将数据转换为具有两个特殊属性的公共属性`Description`和`Category`。`Description`是在底部显示的文本**属性**窗口。 `Category`确定该属性应出现时**属性**窗口显示在**按分类顺序**视图。 如下图中**属性**窗口处于**按分类顺序**视图中，**名称**中的属性**ToDo 字段**选择类别后，和的说明**名称**属性显示在窗口的底部。



2. 添加以下 using 语句 TodoItem.cs 文件。

C#

```
using System.ComponentModel;
using System.Windows.Forms;
using Microsoft.VisualStudio.Shell.Interop;
```

3. 添加 `public` 到类声明的访问修饰符。

C#

```
public class TodoItem
{
}
```

添加两个属性名称和 DueDate。 我们将更高版本执行的 UpdateList() 和 CheckForErrors()。

C#

```
public class TodoItem
{
    private TodoWindowControl parent;
    private string name;
    [Description("Name of the ToDo item")]
    [Category("ToDo Fields")]
    public string Name
    {
        get { return name; }
        set
        {
            name = value;
            parent.UpdateList(this);
        }
    }

    private DateTime dueDate;
    [Description("Due date of the ToDo item")]
    [Category("ToDo Fields")]
    public DateTime DueDate
    {
        get { return dueDate; }
        set
        {
            dueDate = value;
            parent.UpdateList(this);
            parent.CheckForErrors();
        }
    }
}
```

4. 添加到用户控件的专用引用。 添加的构造函数的用户控制和此 ToDo 项的名称。 若要查找 daysAhead 值，它获取选项页属性。

   C#

   ```
   private TodoWindowControl parent;

   public TodoItem(TodoWindowControl control, string itemName)
   {
       parent = control;
       name = itemName;
       dueDate = DateTime.Now;

       double daysAhead = 0;
       IVsPackage package = parent.parent.Package as IVsPackage;
       if (package != null)
       {
           object obj;
           package.GetAutomationObject("ToDo.General", out obj);

           ToolsOptions options = obj as ToolsOptions;
           if (options != null)
           {
               daysAhead = options.DaysAhead;
           }
       }

       dueDate = dueDate.AddDays(daysAhead);
   }
   ```

5. 因为实例 `TodoItem` 类将存储在列表框和列表框将调用 `ToString` 函数，还必须重载 `ToString` 函数。 在构造函数之后和类的末尾之前，请将以下代码添加到 TodoItem.cs。

   C#

   ```
   public override string ToString()
   {
       return name + " Due: " + dueDate.ToShortDateString();
   }
   ```

6. 在 TodoWindowControl.xaml.cs，添加存根 （stub） 方法的 TodoWindowControl 类 `CheckForError` 和 `UpdateList` 方法。 将它们放在 ProcessDialogChar 之后和之前的文件的末尾。

   C#

```
public void CheckForErrors()
{
}
public void UpdateList(TodoItem item)
{
}
```

`CheckForError` 方法将调用父对象中具有相同名称的方法，该方法将检查是否已发生的任何错误，并正确处理这些。 `UpdateList` 方法将更新的列表框中的父控件; 当调用该方法`Name`和`DueDate`中此类更改的属性。 它们将更高版本实现。

将集成到属性窗口

现在，编写代码，用于管理将绑定到 ListBox**属性**窗口。

必须更改该按钮单击处理程序来读取文本框、 创建 TodoItem，并将其添加到列表框。

1. 替换现有`button1_Click`函数包含用于创建新的 TodoItem 并将其添加到列表框的代码。 它将调用 TrackSelection()，将在以后定义。

   C#

   ```
   private void button1_Click(object sender, RoutedEventArgs e)
   {
       if (textBox.Text.Length > 0)
       {
           var item = new TodoItem(this, textBox.Text);
           listBox.Items.Add(item);
           TrackSelection();
           CheckForErrors();
       }
   }
   ```

2. 在设计视图选择列表框控件。 在**属性**窗口中，单击**事件处理程序**按钮并找到 SelectionChanged 事件。 在文本框中填充**listBox_SelectionChanged**。 执行此操作将添加 SelectionChanged 处理程序存根，并将其分配给该事件。

3. 实现 TrackSelection() 方法。 由于您将需要获取SVsUIShellSTrackSelection服务，需要进行GetService TodoWindowControl 可访问。 将以下方法添加到 TodoWindow 类：

   ```
   internal object GetVsService(Type service)
   {
       return GetService(service);
   }
   ```

4. 添加以下 using 语句 TodoWindowControl.xaml.cs:

   C#

```
using System.Runtime.InteropServices;
using Microsoft.VisualStudio.Shell.Interop;
using Microsoft.VisualStudio;
using Microsoft.VisualStudio.Shell;
```

5. 填充 SelectionChanged 处理程序中，如下所示：

```
private void listBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    TrackSelection();
}
```

6. 现在，填写 TrackSelection 函数，它将提供与集成**属性**窗口。 在用户将某项添加到列表框，或单击列表框中的项目时，调用此函数。 它将 ListBox 的内容添加到 SelectionContainer 并将传递到 SelectionContainer**属性**窗口的OnSelectChange事件处理程序。 TrackSelection 服务跟踪的用户界面 (UI) 中选定的对象，并显示其属性

   C#

```
private SelectionContainer mySelContainer;
private System.Collections.ArrayList mySelItems;
private IVsWindowFrame frame = null;

private void TrackSelection()
{
    if (frame == null)
    {
        var shell = parent.GetVsService(typeof(SVsUIShell)) as IVsUIShell;
        if (shell != null)
        {
            var guidPropertyBrowser = new
            Guid(ToolWindowGuids.PropertyBrowser);
            shell.FindToolWindow((uint)__VSFINDTOOLWIN.FTW_fForceCreate,
            ref guidPropertyBrowser, out frame);
        }
    }
    if (frame != null)
        {
            frame.Show();
        }
    if (mySelContainer == null)
    {
        mySelContainer = new SelectionContainer();
    }

    mySelItems = new System.Collections.ArrayList();

    var selected = listBox.SelectedItem as TodoItem;
    if (selected != null)
    {
        mySelItems.Add(selected);
    }

    mySelContainer.SelectedObjects = mySelItems;

    ITrackSelection track = parent.GetVsService(typeof(STrackSelection))
                            as ITrackSelection;
    if (track != null)
    {
        track.OnSelectChange(mySelContainer);
    }
}
```

现在，您有一个类，**属性**窗口可以使用，您可以将集成**属性**与工具窗口的窗口。 当用户单击工具窗口中，在列表框中的项**属性**应相应地更新窗口。 同样，当用户更改中的 ToDo 项**属性**窗口中，应更新关联的项。

7. 现在，在 TodoWindowControl.xaml.cs 添加 UpdateList 函数代码的其余部分。 它应删除并重新从列表框中添加已修改的 TodoItem。

   C#

```
public void UpdateList(TodoItem item)
{
    var index = listBox.SelectedIndex;
    listBox.Items.RemoveAt(index);
    listBox.Items.Insert(index, item);
    listBox.SelectedItem = index;
}
```

8. 测试您的代码。 生成项目并启动调试。 应显示的实验实例。
9. 打开**工具 / 选项**页。 您应看到在左窗格中的 ToDo 类别。 按字母顺序列出了类别，因此在 Ts 下查找。
10. 在 Todo 选项页中，您应看到 DaysAhead 属性设置为**0**。 将其更改为**2**。
11. 在视图上 / 其他窗口菜单打开**TodoWindow**。 类型**EndDate**在文本框中单击**添加**。
12. 在列表框中，您应该看到两个天晚于今天的日期。

将文本添加到输出窗口和任务列表项

有关**任务列表**，您创建新的对象的类型为 Task，，然后添加到该任务对象**任务列表**通过调用其 Add 方法。 要写入到**输出**窗口中，调用其 GetPane 方法以获取窗格对象，，，然后调用窗格中对象的 OutputString 方法。

1. 在 TodoWindowControl.xaml.cs 中，`button1_Click`方法中，添加代码以获取**常规**窗格**输出**窗口中 （这是默认值），并向其中写入。 该方法应如下所示：

   C#

```
private void button1_Click(object sender, EventArgs e)
{
    if (textBox.Text.Length > 0)
    {
        var item = new TodoItem(this, textBox.Text);
        listBox.Items.Add(item);

        var outputWindow = parent.GetVsService(
            typeof(SVsOutputWindow)) as IVsOutputWindow;
        IVsOutputWindowPane pane;
        Guid guidGeneralPane = VSConstants.GUID_OutWindowGeneralPane;
        outputWindow.GetPane(ref guidGeneralPane, out pane);
        if (pane != null)
        {
             pane.OutputString(string.Format(
                "To Do item created: {0}\r\n",
             item.ToString()));
    }
        TrackSelection();
        CheckForErrors();
    }
}
```

2. 若要将项添加到任务列表中，你需要将嵌套的类添加到 TodoWindowControl 类。 嵌套的类需要派生自 TaskProvider。 将以下代码添加到 TodoWindowControl 类的末尾。

   C#

   ```
   [Guid("72de1eAD-a00c-4f57-bff7-57edb162d0be")]
   public class TodoWindowTaskProvider : TaskProvider
   {
       public TodoWindowTaskProvider(IServiceProvider sp)
           : base(sp)
       {
       }
   }
   ```

3. 接下来将对 TodoTaskProvider 的私有引用和 CreateProvider() 方法添加到 TodoWindowControl 类。 该代码应如下所示：

   C#

   ```
   private TodoWindowTaskProvider taskProvider;
   private void CreateProvider()
   {
       if (taskProvider == null)
       {
           taskProvider = new TodoWindowTaskProvider(parent);
           taskProvider.ProviderName = "To Do";
       }
   }
   ```

4. 将 ClearError()，清除任务列表，并 ReportError()，将条目添加到任务列表中，添加到 TodoWindowControl 类。

   C#

```
private void ClearError()
{
    CreateProvider();
    taskProvider.Tasks.Clear();
}
private void ReportError(string p)
{
    CreateProvider();
    var errorTask = new Task();
    errorTask.CanDelete = false;
    errorTask.Category = TaskCategory.Comments;
    errorTask.Text = p;

    taskProvider.Tasks.Add(errorTask);

    taskProvider.Show();

    var taskList = parent.GetVsService(typeof(SVsTaskList))
        as IVsTaskList2;
    if (taskList == null)
    {
        return;
    }

    var guidProvider = typeof(TodoWindowTaskProvider).GUID;
     taskList.SetActiveProvider(ref guidProvider);
}
```

5. 现在，如下所示实现 CheckForErrors 方法。

   C#

```
public void CheckForErrors()
{
    foreach (TodoItem item in listBox.Items)
    {
        if (item.DueDate < DateTime.Now)
        {
            ReportError("To Do Item is out of date: "
                + item.ToString());
        }
    }
}
```

尝试一下

1. 生成项目并启动调试。 将显示的实验实例。
2. 打开 TodoWindow (**视图 / 其他窗口 / TodoWindow**)。
3. 在文本框中键入内容，然后单击**添加**。

到期日期 2 天后今天添加到列表框中。 未出现任何错误，与**任务列表(视图 / 任务列表)** 应没有任何条目。

4. 现在将设置更改启用**工具 / 选项 / ToDo**来自页**2**回**0**。
5. 键入内容中的其他**TodoWindow** ，然后单击**添加**再次。 这会触发错误以及将项记入**任务列表**。

   添加项时，初始日期设置为现在再加上 2 天。
6. 在**视图**菜单上，单击**输出**若要打开**输出**窗口。

   请注意，每次添加项，一条消息将显示在**任务列表**窗格。
7. 单击其中一个列表框中的项。

   属性窗口将显示两个项的属性。
8. 更改其中一个属性，然后按 enter 键。

   在列表框中更新项。