ComponentOne

DragDropManager for WPF

ComponentOne, a division of GrapeCity

201 South Highland Avenue, Third Floor

Pittsburgh, PA 15206 • USA

Internet: info@ComponentOne.com

Web site: http://www.componentone.com

Sales

E-mail: sales@componentone.com

Telephone: 1.800.858.2739 or 1.412.681.4343 (Pittsburgh, PA USA Office)

This manual was produced using ComponentOne Doc-To-Help™.

Table of Contents

ComponentOne DragDropManager for WPF Overview

Effortlessly add drag-and-drop operations anywhere with **ComponentOne DragDropManager™ for WPF**. The **C1DragDropManager** class provides a visually appealing drag-and-drop UI with custom drag sourcing and drop targeting.

For a list of the latest features added to **ComponentOne Studio for WPF**, visit What's New in Studio for WPF.

Getting Started

Get started with the following topics:

Help with ComponentOne Studio for WPF

Getting Started

For information on installing **ComponentOne Studio for WPF**, licensing, technical support, namespaces and creating a project with the control, please visit Getting Started with Studio for WPF.

What's New

For a list of the latest features added to **ComponentOne Studio for WPF**, visit What's New in Studio for WPF.

DragDropManager for WPF Key Features

ComponentOne DragDropManager for WPF includes several key features, such as:

- **Control Drag-and-drop Behavior**

  C1DragDropManager has an extensive set of methods and events that allow you to control the entire drag-and-drop process. Just register some elements as drag sources and some as drop targets, and then handle the **DragDrop** event to move or copy the elements to their new location.

- **Customize Drag Markers**

  C1DragDropManager exposes properties that allow you to customize the appearance of the drag markers for both the drag source and drop target. This makes the entire operation more user-friendly and visually appealing.

- **Scrolling Support**

  Dragging objects near the edges of a scrollable target causes it to scroll automatically, allowing end-users to drop elements exactly where they want in a single operation.

DragDropManager for WPF Quick Start

The following quick start guide is intended to get you up and running with **ComponentOne DragDropManager for WPF**. You'll create a new project in Visual Studio, add a Grid with child elements to the page, and then implement the **C1DragDropManager** to allow users to drag elements from one grid cell to another.

Step 1 of 3: Creating a WPF Application

In this step you'll begin in Visual Studio to create a WPF application which will use **ComponentOne DragDropManager for WPF** to manage user interactions.

To set up and add controls to your application, complete the following steps:

1. In Visual Studio, select **File | New | Project**.
2. In the **New Project** dialog box, select a language in the left pane, and in the templates list select **WPF Application**. Enter a **Name** for your project, for example "QuickStart", and click **OK**. The **New WPF Application** dialog box will appear.
3. Click **OK** to accept default settings, close the **New WPF Application** dialog box, and create your project. The **MainPage.xaml** file should open.
4. Right-click the project in the Solution Explorer window and select **Add Reference**.
5. In the **Add Reference** dialog box, locate and select the **C1.WPF.dll** assembly and click **OK** to add a reference to your project.
6. In the XAML window of the project, place the cursor between the `<Grid>` and `</Grid>` tags and click once.
7. Add the following XAML markup between the `<Grid>` and `</Grid>` tags in the **MainPage.xaml** file:

```
<Grid x:Name="ddGrid" Background="Lavender" ShowGridLines="True"
Width="400" Height="300" >
    <Grid.RowDefinitions>
```

```
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
        <ColumnDefinition/>
        <ColumnDefinition/>
        <ColumnDefinition/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <TextBlock Text="Drag Me" FontSize="14" Grid.Row="1"
Grid.Column="2" />
    <TextBlock Text="Or Me" FontSize="14" Grid.Row="3" Grid.Column="4"
/>
    <Rectangle Fill="Red" Grid.Row="2" Grid.Column="0"/>
    <Rectangle Fill="Blue" Grid.Row="0" Grid.Column="4"/>
</Grid>
```

This markup creates a grid with row definitions in the grid to keep **TextBlock** and **Rectangle** controls in separate areas.

What You've Accomplished

You've successfully created and set up a WPF application and added controls to the page. In the next step you'll add code to add functionality to your application.

Step 2 of 3: Adding Code to the Application

In the last step you set up a WPF application, but you have not added drag-and-drop functionally to the application. In this step you'll continue by adding code to add functionality to the application.

Complete the following steps:

1. Navigate to the Solution Explorer, right-click **MainPage.xaml** file, and select **View Code** to switch to Code view.
2. In Code view, add the following import statements to the top of the page:
   - Visual Basic
     ```
     Imports C1.WPF
     ```
   - C#
     ```
     using C1.WPF;
     ```
3. Add code to the **MainPage.xaml.cs** (or **.vb**) file in the page constructor so it looks similar to the following:
   - Visual Basic
     ```
     Public Sub New()
         InitializeComponent()
         ' Initialize the C1DragDropManager
         Dim dd As New C1DragDropManager()
         dd.RegisterDropTarget(_ddGrid, True)
         For Each e As UIElement In _ddGrid.Children
             dd.RegisterDragSource(e, DragDropEffect.Move,
     ModifierKeys.None)
     ```

```
    Next
    AddHandler dd.DragDrop, AddressOf dd_DragDrop
End Sub
```

- C#

```
public MainPage()
{
  InitializeComponent();
  // Initialize the C1DragDropManager
  C1DragDropManager dd = new C1DragDropManager();
  dd.RegisterDropTarget(_ddGrid, true);
  foreach (UIElement e in _ddGrid.Children)
  {
    dd.RegisterDragSource(e, DragDropEffect.Move, ModifierKeys.None);
  }
  dd.DragDrop += dd_DragDrop;
}
```

The code initiates a new instance of a **C1DragDropManager** and then calls the **RegisterDropTarget** method to indicate that the grid should act as a drop target by default. It then calls the **RegisterDragSource** method to indicate that users should be allowed to drag the elements in the grid and finally, the code attaches an event handler to the **DragDrop** event so the application can receive a notification and move the element being dragged into its new position

4. Add the following event handler to the **MainPage.xaml.cs** (or **.vb**) file, below all the other methods in the **MainPage** class:

- Visual Basic

```
Private Sub dd_DragDrop(source As Object, e As DragDropEventArgs)
    ' Get mouse position
    Dim pMouse As Point = e.GetPosition(_ddGrid)

    ' Translate into grid row/col coordinates
    Dim row As Integer, col As Integer
    Dim pGrid As New Point(0, 0)
    For row = 0 To _ddGrid.RowDefinitions.Count - 1
        pGrid.Y += _ddGrid.RowDefinitions(row).ActualHeight
        If pGrid.Y > pMouse.Y Then
        Exit For
        End If
    Next
    For col = 0 To _ddGrid.ColumnDefinitions.Count - 1
        pGrid.X += _ddGrid.ColumnDefinitions(col).ActualWidth
        If pGrid.X > pMouse.X Then
        Exit For
        End If
    Next

    ' Move the element to the new position
    e.DragSource.SetValue(Grid.RowProperty, row)
    e.DragSource.SetValue(Grid.ColumnProperty, col)
End Sub
```

- C#

```
private void dd_DragDrop(object source, DragDropEventArgs e)
{
  // Get mouse position
  Point pMouse = e.GetPosition(_ddGrid);
  // Translate into grid row/col coordinates
```

```
  int row, col;
  Point pGrid = new Point(0, 0);
  for (row = 0; row < _ddGrid.RowDefinitions.Count; row++)
    {
      pGrid.Y += _ddGrid.RowDefinitions[row].ActualHeight;
      if (pGrid.Y > pMouse.Y)
      break;
    }
  for (col = 0; col < _ddGrid.ColumnDefinitions.Count; col++)
  {
    pGrid.X += _ddGrid.ColumnDefinitions[col].ActualWidth;
    if (pGrid.X > pMouse.X)
    break;
  }
  // Move the element to the new position
  e.DragSource.SetValue(Grid.RowProperty, row);
  e.DragSource.SetValue(Grid.ColumnProperty, col);
}
```

The event handler starts by converting the mouse coordinates into row/column values. Then it uses the **SetValue** method to update the **Grid.RowProperty** and **Grid.ColumnProperty** values on the element that was dragged. Similar logic could be used to drag elements within other types of panel or list-type controls, or from one panel to another.

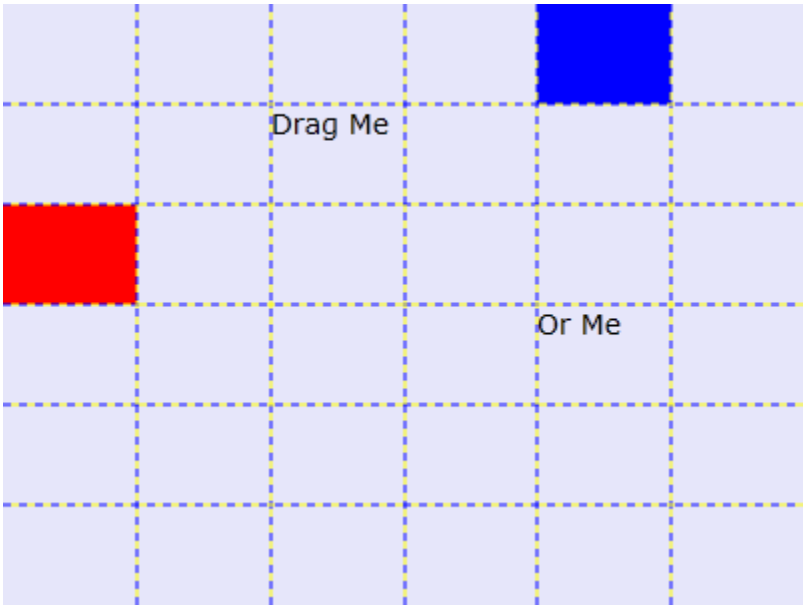
What You've Accomplished

In this step you added code to add functionality to your application. In the next step you'll run your application and observe some of the run-time interactions possible with **ComponentOne DragDropManager for WPF**.

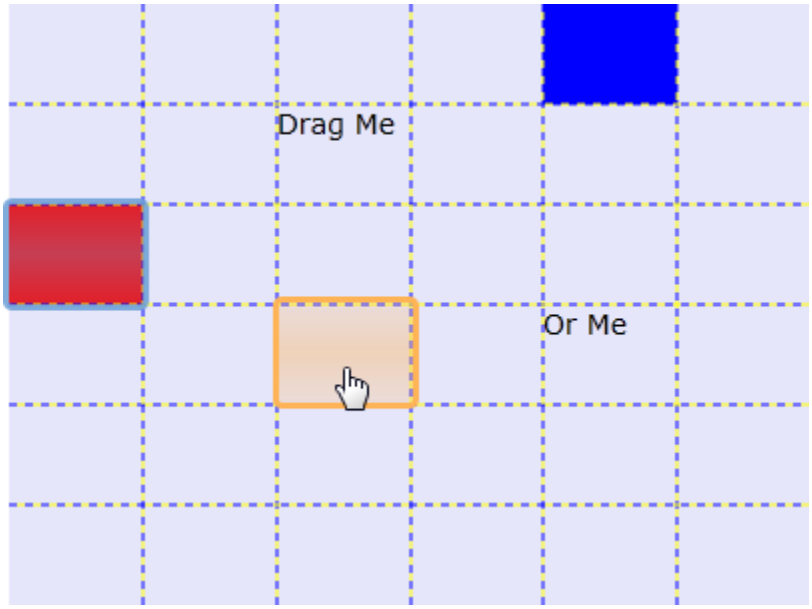
Step 3 of 3: Running the Application

Now that you've created a WPF application and, set up the application, and added code to add functionality to the application, the only thing left to do is run your application. To observe your application's run-time interactions, complete the following steps:

1. Choose **Debug | Start Debugging** from the menu to run your application. The application will appear similar to the following image:



2. Click the red **Rectangle** and drag it to another square in the grid. Notice that as the drag process is in action an extra border appears around the item you are dragging and a transparent rectangle is moved with the mouse to indicate where the item will be dropped:



3. Click another item, such as a **TextBlock**, and move it to a new location.

What You've Accomplished

Congratulations, you've completed the **DragDropManager for WPF** quick start! You've created a simple application that uses **DragDropManager for WPF** to move items in a grid.

To learn more about the features and functionality of **ComponentOne DragDropManager for WPF**, see the Working with DragDropManager for WPF (page 8) topic.

Working with DragDropManager for WPF

ComponentOne DragDropManager™ for WPF provides a simple way to add drag-and-drop interactions to your WPF application. The C1DragDropManager is a class that provides drag-and-drop services on behalf of other elements. It is not a control and has no visual representation at design time.

The C1DragDropManager class follows the patterns found in the WinForms drag-and-drop implementation. It provides a DoDragDrop method that is called to initiate the drag-and-drop operations, and fires events that allow you to customize the whole process (DragStart, DragEnter, DragOver, DragLeave, DragDrop).

The event parameters provide information on what item is being dragged (e.DragSource) and where it is about to be dropped (e.DropTarget). The event handlers are responsible for checking this information and setting the value of the e.Effect parameter to indicate whether the operation is valid or not. The handler for the DragDrop event is responsible for actually moving or copying the source element to the target location.

The C1DragDropManager class also provides two helper methods that simplify the whole process. The methods are called RegisterDragSource and RegisterDropTarget. They provide a simple generic implementation that handles most common drag-and-drop scenarios with very little code. Once you have registered elements as sources and targets with these methods, the C1DragDropManager will monitor the mouse to automatically initiate and manage the drag-and-drop operation. In this case, all you have to do is handle the DragDrop event to perform the move or copy operation.

Basic Properties

ComponentOne DragDropManager for WPF includes several properties that allow you to set the functionality of the C1DragDropManager control. Some of the more important properties are listed below.

The following properties let you customize the C1DragDropManager control:

Property	Description
AutoScroll	Gets or sets whether the **C1DragDropManager** should automatically scroll the **ScrollViewer** that contains the drop target.
AutoScrollDelay	Gets or sets the number of milliseconds between auto scroll steps.
AutoScrollEdge	Gets or sets the distance between the mouse and the edges of a drag target element that triggers the auto scroll process.
AutoScrollStep	Gets or sets the number of pixels to scroll in each auto scroll step.
Canvas	Gets a reference to the Canvas being used to show the drag-and-drop process.
DragThreshold	Gets or sets the distance in pixels that the mouse must move before a drag operation starts.
SourceMarker	Gets a reference to the Border used to highlight the drag source.
TargetMarker	Gets the Border used to indicate the drop location.

Basic Methods

ComponentOne DragDropManager for WPF includes several methods that allow you to customize the control. Some of the more important methods are listed below.

The following methods let you customize the C1DragDropManager control:

Method	Description
ClearSources	Removes all the registered sources.
ClearTargets	Removes all the registered targets.
DoDragDrop	Initiates a drag drop operation using a **UIElement** as a source, supporting a specified **DragDropEffect**.
RegisterDragSource	Registers a **UIElement** to act as a drag source.
RegisterDropTarget	Registers (or unregisters) an element as a drop target.

Basic Events

ComponentOne DragDropManager for WPF includes several events that allow you to set interaction and customize the control. Some of the more important events are listed below.

The following events let you customize the C1DragDropManager control:

Event	Description
DragAutoScroll	Fires after the **C1DragDropManager** automatically scrolls a **ScrollViewer** in order to keep the drop location within view.
DragDrop	Fires at the end of a drag drop process, when the user releases the mouse button over a registered drop target.
DragEnter	Fires during a drag drop process, when the cursor enters a registered drop target.
DragLeave	Fires during a drag drop process, when the cursor leaves a registered drop target.
DragOver	Fires during a drag drop process, when the cursor moves over a registered drop target.
DragStart	Fires when a drag drop process starts.

DragDropManager for WPF Samples

ComponentOne DragDropManager for WPF includes C# examples as part of the **ControlExplorer** sample. By default samples are installed in the **Documents** or **My Documents** folder in the **ComponentOne Samples\Studio for WPF\General\CS\ControlExplorer** folder.

The following examples are included within the **ControlExplorer** sample:

Sample	Description
DragDropManager	This sample demonstrates how the **C1DragDropManager** control can be used to create a simple checkers game.
DragDropManager/List	The sample shows how to use **C1DragDropManager** to drag and drop items between

Box	list boxes.