TouchToolkit for WinForms
Trademarks

The ComponentOne product name is a trademark and ComponentOne is a registered trademark of GrapeCity, Inc. All other trademarks used herein are the properties of their respective owners.

Warranty

ComponentOne warrants that the media on which the software is delivered is free from defects in material and workmanship, assuming normal use, for a period of 90 days from the date of purchase. If a defect occurs during this time, you may return the defective media to ComponentOne, along with a dated proof of purchase, and ComponentOne will replace it at no charge. After 90 days, you can obtain a replacement for the defective media by sending it and a check for $2.5 (to cover postage and handling) to ComponentOne.

Except for the express warranty of the original media on which the software is delivered is set forth here, ComponentOne makes no other warranties, express or implied. Every attempt has been made to ensure that the information contained in this manual is correct as of the time it was written. ComponentOne is not responsible for any errors or omissions. ComponentOne’s liability is limited to the amount you paid for the product. ComponentOne is not liable for any special, consequential, or other damages for any reason.

Copying and Distribution

While you are welcome to make backup copies of the software for your own use and protection, you are not permitted to make copies for the use of anyone else. We put a lot of time and effort into creating this product, and we appreciate your support in seeing that it is used by licensed users only.
# Table of Contents

- TouchToolKit for WinForms Overview
- Help with WinForms Edition
- Key Features
- System Requirements
- Key Components
- TouchToolKit For WinForms Sample
- TouchToolKit for WinForms Quick Start
- C1Zoom Overview
  - C1Zoom Appearance Properties
  - Zoom Appearance Zoom Preview
    - Alternative Content
  - C1Zoom Behavior Properties
    - C1Zoom DoubleTap
    - C1Zoom Behavior Snap Points
      - Adding Snap Points at Design Time
    - C1Zoom Mouse Wheel
  - C1Zoom Scrolling Appearance and Behavior
    - C1Zoom Pan Scrolling
    - C1Zoom Railed Scrolling
    - C1Zoom Scroll Indicator
      - Disabling Scroll Indicator
      - Displaying Scrollbars
    - C1Zoom Boundary Feedback
  - C1Zoom Elements
    - C1Zoom Pan Window Element
      - Showing a Panning Window
    - C1Zoom Closing Pan Window
      - Verifying Whether or not PanWindow is Shown
  - C1Zoom Control Bar Element
    - Adding Control Bar Element
    - Control Bar Position
    - Control Bar Opacity
    - Control Bar Transparency
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detecting Pen</td>
<td>61-63</td>
</tr>
<tr>
<td>Detecting Tap and Hold</td>
<td>63</td>
</tr>
<tr>
<td>Disabling PenFlicks</td>
<td>64</td>
</tr>
<tr>
<td>Detecting Maximum Touch Count</td>
<td>64-66</td>
</tr>
<tr>
<td>C1TouchEventProviderLimitations</td>
<td>66</td>
</tr>
<tr>
<td>C1Magnify Overview</td>
<td>67</td>
</tr>
<tr>
<td>Enabling Magnify Window</td>
<td>67-68</td>
</tr>
<tr>
<td>Hiding Mouse Cursor Inside the Magnifier Window</td>
<td>68</td>
</tr>
<tr>
<td>Changing ZoomFactor for Magnifier Window</td>
<td>68-69</td>
</tr>
<tr>
<td>Magnifier Duration</td>
<td>69</td>
</tr>
<tr>
<td>Magnifier Scrolling</td>
<td>69</td>
</tr>
<tr>
<td>Magnifier Appearance Properties</td>
<td>69-70</td>
</tr>
<tr>
<td>Magnifier Shape</td>
<td>70</td>
</tr>
<tr>
<td>Magnifier Size</td>
<td>70</td>
</tr>
<tr>
<td>Magnifier Border Width</td>
<td>70-71</td>
</tr>
<tr>
<td>Magnifier Border Background</td>
<td>71</td>
</tr>
<tr>
<td>Magnify Events</td>
<td>71</td>
</tr>
<tr>
<td>Using a Different Magnifier Window in Different Controls</td>
<td>72-73</td>
</tr>
<tr>
<td>Posting a Message when the Magnifier is Closed</td>
<td>73-74</td>
</tr>
<tr>
<td>Posting a Message when Magnifier Window Moves</td>
<td>74</td>
</tr>
<tr>
<td>Magnifier Limitations</td>
<td>75</td>
</tr>
<tr>
<td>C1MultiScaleImage Overview</td>
<td>76</td>
</tr>
<tr>
<td>Adding Image for each ZoomFactor</td>
<td>76</td>
</tr>
<tr>
<td>Removing All Images</td>
<td>76-77</td>
</tr>
<tr>
<td>Zoom Policies</td>
<td>78</td>
</tr>
<tr>
<td>Zoom Policies for Standard Controls</td>
<td>78-79</td>
</tr>
<tr>
<td>ComboBox ZoomPolicy</td>
<td>79</td>
</tr>
<tr>
<td>DataGridViewZoomPolicy</td>
<td>79</td>
</tr>
<tr>
<td>FlowLayoutPanelZoomPolicy</td>
<td>79-80</td>
</tr>
<tr>
<td>ListBoxZoomPolicy</td>
<td>80</td>
</tr>
<tr>
<td>ListView Zoom Policy</td>
<td>80-81</td>
</tr>
<tr>
<td>MonthCalendarZoomPolicy</td>
<td>81-82</td>
</tr>
<tr>
<td>PictureBoxZoomPolicy</td>
<td>82-83</td>
</tr>
<tr>
<td>PropertyGridZoomPolicy</td>
<td>83-84</td>
</tr>
<tr>
<td>SplitContainerZoomPolicy</td>
<td>84</td>
</tr>
<tr>
<td>TableLayoutPanelZoomPolicy</td>
<td>84</td>
</tr>
</tbody>
</table>
Setting the ChartZoomPolicy for ComponentOne Studio
Setting the Chart3DZoomPolicy for ComponentOne Studio
Setting the C1MainMenuZoomPolicy for ComponentOne Studio
Setting the C1DockingTabZoomPolicy for ComponentOne Studio
Setting the C1NavBarZoomPolicy for ComponentOne Studio
Setting the C1OutBarZoomPolicy for ComponentOne Studio
Setting the C1FlexGridZoomPolicy for ComponentOne Studio
Setting the C1CalendarZoomPolicy for ComponentOne Studio
Setting the C1ScheduleZoomPolicy for ComponentOne Studio
Setting the C1PrintPreviewControlZoomPolicy for ComponentOne Studio
Setting the C1RibbonZoomPolicy for ComponentOne Studio
Setting the C1TrueDBGridZoomPolicy for ComponentOne Studio
How to Use VisualBasicPowerPacksZoomPolicy
How to Use WPFElementHostZoomPolicy
TileControlZoomPolicy
C1SplitContainerZoomPolicy
TabControlZoomPolicy
C1GanttViewZoomPolicy
TouchToolkit for WinForms Overview

Turn your Windows Forms applications into touch robust applications with TouchToolkit for Windows Forms. The suite includes 7 components to enhance the touch operations. They include the following:

- C1ApplicationZoom
- C1Magnify
- C1MultiScaleImage
- C1TouchEventProvider
- C1Zoom
- C1ZoomPanel
- C1ZoomCommandsProvider

Once you add the C1Zoom component to the Form, end-users can zoom the Form's content by 2 finger gesture, called pinch-out or stretch in mobile devices, to make a comfortable size for touch. This enables users to continue their work on the optimized Form using the mouse/keyboard while using TouchToolkit. As a result you can continue supporting both Mouse users and Touch screen users without trade-offs.

Help with WinForms Edition

Getting Started

For information on installing ComponentOne Studio WinForms Edition, licensing, technical support, namespaces and creating a project with the control, please visit Getting Started with WinForms Edition.
Key Features

The TouchToolkit for WinForms product includes the following key features.

- **Enable Touch Support in any WinForms Application**—no matter what controls you are using. TouchToolkit for WinForms lets you touch-enable any WinForms application, even if you are using standard Microsoft or other 3rd-party controls.

- **Add Gesture Support** to your applications. With touch devices so prevalent these days, standard touch and multitouch gestures are second nature to most technology adopters. Let users seamlessly transition to using touch in your WinForms applications with support for 17 event types and 9 different gestures. Improve touch experience by adding a C1Zoom component to the Form to increase the size of small buttons, labels, and any unfriendly UI elements. You can zoom-in/zoom-out of the Form by multi-touch after you add a C1Zoom component to the Form. The original Form layout does not need to be rewritten to support touch. You can save end-user learning cost and document update cost by keeping current workflow on the Windows Forms Application.

- **Zoom, Magnification, and Pan Support** make your apps truly interactive. TouchToolkit for WinForms includes 5 controls to make adding zoom and magnification a breeze. Give your users the freedom to zoom with touch.

- **The Touch Friendly Control Bar** makes navigation simple. The included control bar lets you implement zooming in, zooming out, scrolling, and more as touch-enabled command buttons to give your users a fully touch-centric experience.
## System Requirements

Minimum System requirements for full touch support in TouchToolkit include the following:

<table>
<thead>
<tr>
<th>Operating Systems</th>
<th>Touch enabled Windows 8 or later</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDE (Visual Studio)</td>
<td>Visual Studio 2008 or later</td>
</tr>
</tbody>
</table>
## Key Components

*TouchToolKit for Windows Forms* contains the following components.

<table>
<thead>
<tr>
<th>Control Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>C1Zoom</strong></td>
<td>Provides touch zoom ability to the Form. For more information see C1Zoom Overview.</td>
</tr>
<tr>
<td><strong>C1ZoomPanel</strong></td>
<td>Makes it possible to zoom the controls in a container. For more information see C1ZoomPanel Overview.</td>
</tr>
<tr>
<td><strong>C1ApplicationZoom</strong></td>
<td>Make all Forms in the application support gesture zoom. For more information see C1ApplicationZoom Overview.</td>
</tr>
<tr>
<td><strong>C1ZoomCommandsProvider</strong></td>
<td>Makes it possible to implement a ControlBar for zooming by using a UserControl.</td>
</tr>
<tr>
<td><strong>C1Magnify</strong></td>
<td>Provides magnifier ability to the control.</td>
</tr>
<tr>
<td><strong>C1MultiScaleImage</strong></td>
<td>Enables users to open a multi-resolution image, which can display different images for different zoom factors of the control.</td>
</tr>
<tr>
<td><strong>C1TouchEventProvider</strong></td>
<td>Allows you to implement Touch, Pen, and Gesture events in Windows Forms Controls.</td>
</tr>
</tbody>
</table>
# TouchToolkit For WinForms Sample

Please be advised that this ComponentOne software tool is accompanied by various sample projects and/or demos which may make use of other development tools included with ComponentOne Studio.

Please refer to the pre-installed product samples through the following path: **Documents\ComponentOne Samples\WinForms**  
The following tables provide a short description of each sample.

The following pages within the ControlExplorer sample installed with TouchToolkit for WinForms detail the C1Zoom, C1ZoomPanel, C1ApplicationZoom, C1Magnify, C1MultiScaleImage, and C1TouchEventProvider’s functionality:

## Visual Basic Samples

<table>
<thead>
<tr>
<th>Sample</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>C1Zoom Basic Zoom and Scroll</strong></td>
<td>This page demonstrates how to use C1Zoom’s features and its following properties: ZoomPreviewMode, BoundaryFeedbackMode, AllowDoubleTapZoom, ScrollIndicatorMode, IsHorizontalRailEnabled, IsVerticalRailEnabled, and KeepAspectRatio. Select the settings for these properties on the page and click the Show Form button to see how these settings effect the zooming.</td>
</tr>
<tr>
<td><strong>C1Zoom ControlBar</strong></td>
<td>Demonstrates C1ControlBar’s built-in actions and customizable actions.</td>
</tr>
<tr>
<td><strong>C1Zoom Show Control</strong></td>
<td>Demonstrates the behavior of the AutoShowControl property when it is enabled.</td>
</tr>
<tr>
<td><strong>C1Zoom Pan Window</strong></td>
<td>Shows how to use different methods to show the Pan Window.</td>
</tr>
<tr>
<td><strong>C1Zoom Zoom Policy</strong></td>
<td>Demonstrates how each zoom policy effects the control when zooming.</td>
</tr>
<tr>
<td><strong>C1Zoom Misc</strong></td>
<td>This page showcases a few of C1Zoom’s appearance and layout properties. The following properties are used in this sample to illustrate the various appearance and layout settings for C1Zoom: Backcolor, BackgroundImage, BackgroundImageLayout FullScreenMode, InnerPanelLayoutMode, and KeepAspectRatio.</td>
</tr>
<tr>
<td><strong>C1ZoomPanel Basic (Zoom and Scroll)</strong></td>
<td>This page demonstrates how to use C1ZoomPanel’s features and its following properties: ZoomPreviewMode, BoundaryFeedbackMode, AllowDoubleTapZoom, ScrollIndicatorMode, IsHorizontalRailEnabled, IsVerticalRailEnabled, and KeepAspectRatio. Select the settings for these properties on the page and click the Show Form button to see how these settings effect the zooming. When you zoom in by touch the percentage of the ZoomFactor appears in the Event logs.</td>
</tr>
<tr>
<td><strong>C1ApplicationZoom</strong></td>
<td>This page demonstrates how to use C1ZoomApplication’s features and its following properties: ZoomPreviewMode, BoundaryFeedbackMode, AllowDoubleTapZoom, ScrollIndicatorMode, IsHorizontalRailEnabled, IsVerticalRailEnabled, and KeepAspectRatio. Select the settings for these properties on the page and click the Show Form button to see how these settings effect the zooming.</td>
</tr>
<tr>
<td><strong>C1Magnify</strong></td>
<td>This page demonstrates how to use C1Magnifier’s behavior and appearance properties. Select the settings for these properties on the</td>
</tr>
</tbody>
</table>
page and tap and hold your finger inside the textbox area to make the magnifier popup.

C1MultiScaleImage
This page illustrates the difference between a PictureBox and C1MultiScaleImage. To see the difference, zoom in the PictureBox and C1MultiScaleImage by touch.

Touch Events List
This sample page showcases the three types of user interaction events: pointer, tap, and manipulation.

ContextualUI
Shows how to provide a touch-optimized contextual UI if its opened by touch.

AutoRepeat
Demonstrates how to use C1TouchEventProvider's EnablePressAndHold feature.

ImageViewer
Demonstrates how to use C1TouchEventProvider to make a picture viewer where you can zoom, pan, and rotate the picture.

Drawing Board for Pen
Shows how to use the pen or your finger to draw on the screen.

C# Samples

<table>
<thead>
<tr>
<th>Sample</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1Zoom Basic Zoom and Scroll</td>
<td>This page demonstrates how to use C1Zoom's features and its following properties: ZoomPreviewMode, BoundaryFeedbackMode, AllowDoubleTapZoom, ScrollIndicatorMode, IsHorizontalRailEnabled, IsVerticalRailEnabled, and KeepAspectRatio. Select the settings for these properties on the page and click the Show Form button to see how these settings effect the zooming.</td>
</tr>
<tr>
<td>C1Zoom ControlBar</td>
<td>Demonstrates C1ControlBar's built-in actions and customizable actions.</td>
</tr>
<tr>
<td>C1Zoom Show Control</td>
<td>Demonstrates the behavior of the AutoShowControl property when it is enabled.</td>
</tr>
<tr>
<td>C1Zoom Pan Window</td>
<td>Shows how to use different methods to show the Pan Window.</td>
</tr>
<tr>
<td>C1Zoom Zoom Policy</td>
<td>Demonstrates how each zoom policy effects the control when zooming.</td>
</tr>
<tr>
<td>C1Zoom Misc</td>
<td>This page showcases a few of C1Zoom's appearance and layout properties. The following properties are used in this sample to illustrate the various appearance and layout settings for C1Zoom: Backcolor, BackgroundImage, BackgroundImageLayout, FullScreenMode, InnerPanelLayoutMode, and KeepAspectRatio.</td>
</tr>
<tr>
<td>C1ZoomPanel Basic (Zoom and Scroll)</td>
<td>This page demonstrates how to use C1ZoomPanel's features and its following properties: ZoomPreviewMode, BoundaryFeedbackMode, AllowDoubleTapZoom, ScrollIndicatorMode, IsHorizontalRailEnabled, IsVerticalRailEnabled, and KeepAspectRatio. Select the settings for these properties on the page and click the Show Form button to see how these settings effect the zooming. When you zoom in by touch the percentage of the ZoomFactor appears in the Event logs.</td>
</tr>
<tr>
<td>C1ApplicationZoom</td>
<td>This page demonstrates how to use C1ZoomApplication's features and its following properties: ZoomPreviewMode, BoundaryFeedbackMode,</td>
</tr>
</tbody>
</table>
AllowDoubleTapZoom, ScrollIndicatorMode, IsHorizontalAlignment, IsVerticalRailEnabled, and KeepAspectRatio. Select the settings for these properties on the page and click the Show Form button to see how these settings affect the zooming.

| **C1Magnify** | This page demonstrates how to use C1Magnifier’s behavior and appearance properties. Select the settings for these properties on the page and tap and hold your finger inside the textbox area to make the magnifier popup. |
| **C1MultiScaleImage** | This page illustrates the difference between a PictureBox and C1MultiScaleImage. To see the difference, zoom in the PictureBox and C1MultiScaleImage by touch. |
| **Touch Events List** | This sample page showcases the three types of user interaction events: pointer, tap, and manipulation. |
| **ContextualUI** | Shows how to provide a touch-optimized contextual UI if its opened by touch. |
| **AutoRepeat** | Demonstrates how to use C1TouchEventProvider’s EnablePressAndHold feature. |
| **ImageViewer** | Demonstrates how to use C1TouchEventProvider to make a picture viewer where you can zoom, pan, and rotate the picture. |
| **Drawing Board for Pen** | Shows how to use the pen or your finger to draw on the screen. |
TouchToolKit for WinForms Quick Start

The following quick start guide is intended to get you up and running with TouchToolKit for Windows Forms. In this quick start you’ll create a Windows Forms application that allows you to zoom the Form’s content by using the C1Zoom component.

1. Open the ClickOnceDemo sample from TouchToolKit for WinForms. Please refer to the pre-installed product samples through the following path:
   Documents\ComponentOne Samples\WinForms
   If you do not have that installed open Visual Studio, and create a new Windows Forms Application project.
2. Double-click the C1Zoom component from the Toolbox to add it to the Form’s component tray. If there is no C1Zoom component in the Toolbox, see “Adding component to the project” page.
3. Run the project.
4. Touch the screen by 2 fingers, and make a stretch motion with your thumb. Then, the Form is zoomed based on the space of thumbs.
5. Stop the motion and release finger from the screen when you get best scale for touch. This gesture is called “Pinch-out” or "Stretch".

6. To scroll the Form by touch, put your finger on the screen and keep contact as you move it around. This gesture is called “Pan”.

---

Copyright © 2017 GrapeCity, inc. All rights reserved.
7. Go back to original scale by a pinching motion with your thumb.
C1Zoom Overview

The **C1Zoom** component allows users to zoom controls of the Form by multi-touch gesture, called Pinch-out or Stretch. You can keep your existing design on the form, and users can zoom it on demand.

You can zoom the Form when the C1Zoom component is added to the Form. This function is enabled if the **C1Zoom.AllowPinchZoom** property is **True**.

C1Zoom Appearance Properties

C1Zoom includes a variety of appearance properties to visually enhance and customize the control. The control’s preview style can easily be customized through the using C1Zoom’s appearance properties. These properties can be set at design time through the Properties window or programmatically.

The following section introduces some of the common appearance behavior properties used for the C1Zoom control.

C1Zoom includes the following appearance properties:

- **C1Zoom.AlternativeContentSettings**
- **C1Zoom.BackColor**
- **C1Zoom.BackgroundImage**
- **C1Zoom.BackgroundImageLayout**
- **C1Zoom.ZoomPreviewMode**

Zoom Preview

A preview image is available for touch zoom. Users can select a preview mode before they use the zoomfactor to operate the Form by touch.

If you would like to disable the preview set the **C1Zoom.ZoomPreviewMode** property to **NoPreview**. Also, you can use a message instead of the preview when the **C1Zoom.ZoomPreviewMode** property is set to **AlternativeContent**.

To programatically set the **C1Zoom.ZoomPreviewMode** property to **AlternativeContent**, use the following code:

```csharp
using C1.Win.TouchToolKit;

c1Zoom1.ZoomPreviewMode = ZoomPreviewMode.AlternativeContent;
```

```vbnet
Imports C1.Win.TouchToolKit
C1Zoom1.ZoomPreviewMode = ZoomPreviewMode.AlternativeContent
```

Alternative Content

You can show a text message to notify “the Form is zooming” to users when the **C1Zoom.ZoomPreviewMode** property is set to **AlternativeContent**. Once the **C1Zoom.ZoomPreviewMode** property is set to **AlternativeContent** you can then modify the content’s Font, ForeColor and Text. If you use “{Percentage}” in the **C1ZoomAlternativeContent.Format** property, the string is replaced to the current zoom factor percentage.

To programatically set the format to the “{ZoomFactor}” string for the **C1Zoom.AlternativeContentSettings** property, use the following code:
using C1.Win.TouchToolKit;

c1Zoom1.ZoomPreviewMode = ZoomPreviewMode.AlternativeContent;
c1Zoom1.AlternativeContentSettings.Font = new Font("Arial", 20.0f);
c1Zoom1.AlternativeContentSettings.ForeColor = Color.Blue;
c1Zoom1.AlternativeContentSettings.Format = "Zooming x{ZoomFactor}";

The following image illustrates the modified content’s Font, ForeColor, and Format settings for the AlternativeContent:
C1Zoom Behavior Properties

C1Zoom includes a variety of behavior properties to visually enhance and customize the control. These properties can be set at design time through the Properties window or programmatically.

The following section introduces some of the common behavior properties used for the C1Zoom control.

C1Zoom includes the following behavior properties:

- C1Zoom.AllowDoubleTapZoom
- C1Zoom.AllowMouseWheelScroll
- C1Zoom.MouseWheelZoom
- C1Zoom.AllowPanScroll
- C1Zoom.AllowPinchZoom
- C1Zoom.AllowResizeByZoom
- C1Zoom.AllowZoomByResize
- C1Zoom.BoundaryFeedbackMode
- C1Zoom.IsHorizontalRailedEnabled
- C1Zoom.IsVerticalRailEnabled
- C1Zoom.ZoomPolicies
- C1Zoom.ZoomSnapDistance
- C1Zoom.ZoomSnapPoint

C1Zoom DoubleTap
You can allow users to double-tap by quickly tapping the screen twice to zoom the Form. To disable the function, set the `C1Zoom.AllowDoubleTapZoom` property to `False`.

<table>
<thead>
<tr>
<th>Visual Basic</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1Zoom1.AllowDoubleTapZoom = False</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C#</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1Zoom1.AllowDoubleTapZoom = false;</td>
</tr>
</tbody>
</table>

### C1Zoom Behavior Snap Points

A rough result can occur when you set the zoom factor by touch to values such as 197% or 202% rather than a number like 200%. The SnapPoints feature helps the zoom factor stop at a good number like 200%. SnapPoints has a group of stop values for the zoom factor, and it has the minimum distance that can cause the current zoom factor snap to a zoom factor in the group.

The `C1Zoom.ZoomSnapPoints` can be set like the following in code:

<table>
<thead>
<tr>
<th>C#</th>
</tr>
</thead>
<tbody>
<tr>
<td>using C1.Win.C1TouchToolKit;</td>
</tr>
<tr>
<td>private void Form1_Load(object sender, EventArgs e)</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>c1Zoom1.ZoomPreviewMode = ZoomPreviewMode.AlternativeContent;</td>
</tr>
<tr>
<td>c1Zoom1.MaxZoomFactor = 4.0f;</td>
</tr>
<tr>
<td>c1Zoom1.ZoomSnapDistance = 0.05f;</td>
</tr>
<tr>
<td>c1Zoom1.ZoomSnapPoints.Add(2.0f);</td>
</tr>
<tr>
<td>c1Zoom1.ZoomSnapPoints.Add(3.0f);</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>private void c1Zoom1_ZoomFactorChanged(object sender, EventArgs e)</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>this.Text = string.Format(&quot;{0:P}&quot;, c1Zoom1.ZoomFactor);</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Visual Basic</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1Zoom1.ZoomPreviewMode = ZoomPreviewMode.AlternativeContent;</td>
</tr>
<tr>
<td>c1Zoom1.MaxZoomFactor = 4.0f;</td>
</tr>
<tr>
<td>c1Zoom1.ZoomSnapDistance = 0.05f;</td>
</tr>
<tr>
<td>c1Zoom1.ZoomSnapPoints.Add(2.0f);</td>
</tr>
<tr>
<td>c1Zoom1.ZoomSnapPoints.Add(3.0f);</td>
</tr>
<tr>
<td>this.Text = string.Format(&quot;{0:P}&quot;, c1Zoom1.ZoomFactor);</td>
</tr>
</tbody>
</table>
Imports C1.Win.C1TouchToolKit

Private Sub Form1_Load(sender As System.Object, e As System.EventArgs) Handles MyBase.Load
    C1Zoom1.ZoomPreviewMode = ZoomPreviewMode.AlternativeContent

    C1Zoom1.MaxZoomFactor = 4.0F
    C1Zoom1.ZoomSnapDistance = 0.05F
    C1Zoom1.ZoomSnapPoints.Add(2.0F)
    C1Zoom1.ZoomSnapPoints.Add(3.0F)
End Sub

Private Sub C1Zoom1_ZoomFactorChanged(sender As System.Object, e As System.EventArgs) Handles C1Zoom1.ZoomFactorChanged
    Me.Text = String.Format("{0:P}", C1Zoom1.ZoomFactor)
End Sub

Adding Snap Points at Design Time

To add the snap points to the ZoomSnapPoints property at design time, complete the following:

1. Set the ZoomPreviewMode property to AlternativeContent.
2. Set the MaxZoomFactor property to 400%.
3. Set the ZoomSnapDistance to 5%.
4. Click on the ellipsis button next to the ZoomSnapPoints property.
5. Click on the ellipsis button next to the ZoomSnapPoints property.
6. Click Add and set the value to 2.0.
7. Click Add and set the second member’s value to 3.0.
C1Zoom Mouse Wheel

You can zoom the Form by Ctrl + Mouse Wheel scroll when the active control does not handle the MouseWheel message. To enable this behavior, use the following code:

<table>
<thead>
<tr>
<th>Visual Basic</th>
<th>C#</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1Zoom1.AllowMouseWheelZoom = True</td>
<td>c1Zoom1.AllowMouseWheelZoom = true;</td>
</tr>
</tbody>
</table>

**Note:**
- If you set the C1Zoom.AllowPinchZoom property to False, the Form still can zoom, because the Ctrl + MouseWheel message is valid if C1Zoom.AllowMouseWheelZoom = True.
- If you add a C1ZoomPanel to the Form. Make sure the focus is in the C1ZoomPanel control. Pinch outside the C1ZoomPanel, the C1ZoomPanel will zoom if the C1ZoomPanel.AllowMouseWheelZoom = True.

C1Zoom Scrolling Appearance and Behavior

The C1Zoom control includes the following scroll behaviors such as Pan and Railed scrolling. The following topics will provide more details about the Pan and Railed scrolling as well as determining whether or not to display scroll indicators or scrollbars.

C1Zoom Pan Scrolling

When you zoom the Form by touch, you will see partial controls of the Form. In this, you can touch the Form then slide with keeping touch to scroll the Form to see other controls. This is called “Pan” operation.

You can disable this operation by setting the C1Zoom.AllowPanScroll property to False.

<table>
<thead>
<tr>
<th>Visual Basic</th>
<th>Title Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1Zoom1.AllowPanScroll = False</td>
<td>c1Zoom1.AllowPanScroll = false;</td>
</tr>
</tbody>
</table>

C1Zoom Railed Scrolling

When you pan the horizontal or vertical direction first, it will lock the scroll direction so you can only scroll in horizontal or vertical direction. This is called “Railed” scrolling. You can freely touch and move your finger on the screen, so the Railed Scrolling is helpful to make a better scroll result.

You can disable the Railed Scrolling by setting the C1Zoom.IsHorizontalRailEnabled property and C1Zoom.IsVerticalRailEnabled property to False, like the following code:

<table>
<thead>
<tr>
<th>Visual Basic</th>
</tr>
</thead>
</table>
C1Zoom1.IsHorizontalRailEnabled = False
C1Zoom1.IsVerticalRailEnabled = False

C#
c1Zoom1.IsHorizontalRailEnabled = false;
c1Zoom1.IsVerticalRailEnabled = false;

### C1Zoom Scroll Indicator

The Horizontal/Vertical bars appear when you scroll by touch. The Scroll Indicator is indicated by a red triangle like shown below:

![C1Zoom Scroll Indicator](image)

#### Disabling Scroll Indicator

To disable the Scroll Indicator, set the `C1Zoom.ScrollIndicatorMode` property to `None` like the following code:

**Visual Basic**
```vbnet
C1Zoom1.ScrollIndicatorMode = ScrollIndicatorMode.None
```

**C#**
```csharp
c1Zoom1.ScrollIndicatorMode = ScrollIndicatorMode.None;
```

#### Displaying Scrollbars

To show Scrollbars instead of the Scroll Indicator, set the `C1ZoomScrollIndicatorMode` property to `ScrollBar`.

**Visual Basic**
```vbnet
C1Zoom1ScrollIndicatorMode = ScrollIndicatorMode.ScrollBar
```

**C#**
```csharp
c1Zoom1ScrollIndicatorMode = ScrollIndicatorMode.ScrollBars;
```
C1Zoom Boundary FeedBack

You can specify how to show visual feedback when the scroll reaches the end point by using the `C1Zoom.BoundaryFeedbackMode` property.

If the `C1Zoom.BoundaryFeedbackMode` property is `Split` when the scrolling reaches the end point, the content area will continue to scroll a little, and then bound back. This is the default behavior.

If the `C1Zoom.BoundaryFeedbackMode` property is `Standard` when the scrolling reaches the end point, the content area will continue to scroll a little, and then bound back.

Use the following code to programatically set the `C1Zoom.BoundaryFeedbackMode` property to `Standard`:

<table>
<thead>
<tr>
<th>Language</th>
<th>Code</th>
</tr>
</thead>
</table>

C1Zoom Auto Show Control

The control can auto show when it has focus or the touch keyboard popups if the `C1Zoom.AutoShowControl` property is True.

Use the following code to programatically set the `C1Zoom.AutoShowControl` property to True:

<table>
<thead>
<tr>
<th>Language</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visual Basic</td>
<td>C1Zoom1.AutoShowControl = True</td>
</tr>
<tr>
<td>C#</td>
<td>c1Zoom1.AutoShowControl = true;</td>
</tr>
</tbody>
</table>

To control auto show for individual controls, you can use the `C1Zoom.ControlAutoShowing` event. You can get the displayed area of the control by using the `C1Zoom.GetDisplayRectangle` method.

C1Zoom Elements

The following topics provide information on the Pan Window and ControlBar elements in the C1Zoom control.

C1Zoom Pan Window Element

In the Pan Window, you can see the current displayed area by a red rectangle with a preview image of the entire Form in a popup window. You can allow users to use the Pan Window to know which part of the Form is zoomed.
Showing A Panning Window

To show Pan Window, use the **C1Zoom.ShowPanWindow** method. The method can set the initial position and initial size of the PanWindow by using the overload method.

**Visual Basic**

C1Zoom1.ShowPanWindow()

**C#**

c1Zoom1.ShowPanWindow();

C1Zoom Closing Pan Window

To close the Pan Window, use the **C1Zoom.ClosePanWindow** method.

**Visual Basic**

C1Zoom1.ClosePanWindow()

**C#**

c1Zoom1.ClosePanWindow();

Verifying Whether or not PanWindow is Shown

**Verifying whether or not the PanWindow is Shown**

To verify whether or not the PanWindow is shown, use the **C1Zoom.IsPanWindowShown** property.
**C1Zoom Control Bar Element**

The **ControlBar** can visualize the zoom capability of the C1Zoom component on the form. The **C1ZoomCommandProvider** and C1Zoom control need to be used in order to create a functioning controlbar. In order to create a control bar to use with the C1Zoom control you need to do the following:

1. Add a user control and add buttons that will represent the control bar.
2. Add a C1ZoomCommandProvider control to the user control.
3. Add a C1Zoom control to your form and a control that you wish to zoom in on.

The following image represents a control bar created from buttons each button performs some type of zoom behavior when you click on it such as ScrollUp, ScrollLeft, ScrollRight, etc. You can implement **Zoom In/Out, Scrolling, Switch FullScreen** and other commands as ControlBar buttons when the **C1ZoomCommandProvider** is added to the user control form.

The following images illustrates a custom control bar created out of buttons with different zoom commands:
Adding Control Bar Element

The following topic shows how to add the ControlBar to your project:

1. Add a UserControl to the project. In the Solution Explorer, right-click the solution name and select Add | UserControl. Name the user control (MyControlBar1).
2. Click Add in the Add New Item dialog box.
3. Add two Standard Buttons, Button1 and Button2 to MyControlBar1.
4. Add the C1ZoomCommandProvider from the Toolbox. If it does not appear in the Toolbox, right-click the Toolbox and select Add Tab. Name the tab TouchToolkit and right-click the area in the tab and select Choose Items. Browse to the location of the C1.Win.C1TouchToolkit.4.dll and click Open. The C1ZoomCommandProvider1 item will appear in the component tray.
5. Click the smart-tag anchor of Button1, then select ZoomIn from the list of C1ZoomCommands. Note: The C1.Win.C1TouchToolkit.4.Design should be referenced to your project for the smart-tag anchor to appear when you click on the control.
6. Repeat same step for Button2, and select ZoomOut from the list.
7. Set Button1’s Text to Zoom In and Button2’s Text to Zoom Out.
8. Build the project.
9. In design-time, add a C1Zoom to the Form. (C1Zoom1).
10. Select C1Zoom1 from the Properties Window and click on the ellipsis button next to the ControlBars property in the Properties Window.

The ControlBar Collection Editor appears:

11. Click Add in the ControlBar Collection Editor window.
12. Click on the dropdown arrow next to the Content property and select WindowsForms1.MyControlBar1.
13. Click OK to save and close the ControlBar Collection Editor.
14. Add a Standard control such as a Button to Form1.
15. Run the project, and click on the **Zoom In** button to zoom in on the **Button** control and click on the **Zoom Out** button to zoom out on the button control.

Notice the **button1** control will appear larger each time you click on the Zoom In button and then gradually zoom out each time you click on the Zoom Out button.

**Control Bar Position**

You can use the `ControlBar.PositionMode` property to set the position mode of the ControlBar.

When the `ControlBar.PositionMode` property value is set to **Dock**, the Control bar will dock in the parent form. When the parent form’s size is changed, the control bar will adjust the position.

If you want to make a space between the Form’s border and the ControlBar, you can use the `ControlBar.Margin` property to make the space. The following table shows which property takes effect when the **DockPosition** is changed:
When the `ControlBar.PositionMode` property value is **Floating**, you can change the ControlBar’s position by code or dragging it in the UI.

When the `ControlBar.PositionMode` property value is **Fixed**, you can change the ControlBar’s position by code, but you can't change it by dragging it in the UI.

**Control Bar Opacity**

You can set the opacity level of the ControlBar window by using the `ControlBar.Opacity` property.

**Control Bar Transparency**

You can set the color that will represent transparent areas of the ControlBar(UserControl) by using the `ControlBar.DefaultTransparencyKey` property. The property is enabled if the `ControlBar.TransparentBackground` property is set to True. In the transparent mode, users cannot operate the controls on the Form behind through the transparency.

**C1Zoom Layout**
In this section you can see how to use the following layout properties for the C1Zoom control:

- C1Zoom.AutoShowControl
- C1Zoom.FullScreenMode
- C1Zoom.InnerPanelLayoutMode
- C1Zoom.KeepAspectRatio
- C1Zoom.MaxZoomFactor
- C1Zoom.ScrollIndicatorMode
- C1Zoom.ZoomFactor

**Centering Content On Form**

In some cases, controls on the form move to the TopLeft direction when you zoom the Form because the control’s Anchor property value is TopLeft.

To keep the control’s position in the center of the form, set the C1Zoom.InnerPanelLayoutMode property to MiddleCenter, like the following:

<table>
<thead>
<tr>
<th>Visual Basic</th>
<th>C#</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1Zoom1.InnerPanelLayoutMode = InnerPanelLayoutMode.MiddleCenter</td>
<td>c1Zoom1.InnerPanelLayoutMode = InnerPanelLayoutMode.MiddleCenter;</td>
</tr>
</tbody>
</table>

**Setting Background Content**

You will see a whitespace when the Form is maximized, but its zoomfactor is still 100%. You can set the Background color or Background image for the whitespace.

To set the Background color, you can use the C1Zoom.BackColor property like the following:

[C#]

```csharp
C1Zoom1.BackColor = Color.DarkGray;
```

[Visual Basic]

```vbnet
C1Zoom1.BackColor = Color.DarkGray
```

To set the Background image, you can use the C1Zoom.BackgroundImage property and the C1Zoom.BackgroundImageLayout property like the following:

[C#]

```csharp
C1Zoom1.BackgroundImage = new Bitmap("Background.png");
C1Zoom1.BackgroundImageLayout = ImageLayout.Tile;
```

Copyright © 2017 GrapeCity, inc. All rights reserved.
Resizing the Form

You can keep the aspect ratio of the form’s size by using the C1Zoom.KeepAspectRatio property. If the property value is true, the Form's width and height change together when resized.

To programatically set the C1Zoom.KeepAspectRatio to true, use the following code:

[C#]

c1Zoom1.KeepAspectRatio = true;

[Visual Basic]

C1Zoom1.KeepAspectRatio = True

To set the minimum size or maximum size of the form, use the Form.MinimumSize property or Form.MaximumSize property.

Specifying Full Screen Modes

You can switch to full screen view on your Form. In Fullscreen mode, you can block unexpected user actions for the Title bar or Windows Taskbar.

The following behaviors exist when setting the different values of the FullScreenMode enumeration:

- When the C1Zoom.FullScreenMode property is FullScreen, the Form switches to full screen.
- When the C1Zoom.FullScreenMode property is FullScreenWithTaskbar, the Form switches to full screen with the Windows Taskbar.
- When the C1Zoom.FullScreenMode property is None, the Form switches to normal view.

Use the following code to switch your form to full screen view:
[C#]

c1Zoom1.FullScreenMode = C1.Win.TouchToolKit.FullScreenMode.FullScreen;

[Visual Basic]

C1Zoom1.FullScreenMode = C1.Win.TouchToolKit.FullScreenMode.FullScreen

---

Note: MDI Child Form does not support Fullscreen mode.

---

**Setting Zoom Factor in Code**

You can get or set the form’s ZoomFactor by using the `C1Zoom.ZoomFactor` property. The following code sets the ZoomFactor to 200% when the button is clicked.

[C#]

```csharp
private void button1_Click(object sender, EventArgs e)
{
    c1Zoom1.ZoomFactor = 2.0f;
}
```

[Visual Basic]

```vbnet
Private Sub Button1_Click(sender As System.Object, e As System.EventArgs) Handles Button1.Click
    C1Zoom1.ZoomFactor = 2.0F
End Sub
```

The following image illustrates the **ZoomFactor** set to **200%** when the button is clicked.
The following image illustrates the ZoomFactor set to 100% when the button is clicked.

**Setting Max Value of ZoomFactor**

You can set a maximum value for the ZoomFactor by using the C1Zoom.MaxZoomFactor property. The minimum fixed value for the ZoomFactor is 100%.

[C#]

```csharp
C1Zoom1.MaxZoomFactor = 4.0f;
```

[Visual Basic]

```vbnet
C1Zoom1.MaxZoomFactor = 4.0F
```
C1Zoom Control Events and Examples

You can use the following events to customize zooming.

- **ControlAutoShowing** - Occurs before C1Zoom starts showing the target control.
- **ControlBoundsZooming** - Occurs before C1Zoom starts zooming the target control's bounds.
- **ControlFontZooming** - Occurs before C1Zoom starts zooming the target control's font.
- **ManipulationStarting** - Occurs when the manipulation processor is first created.
- **ZoomFactorChanged** - Occurs when the ZoomFactor changed.

Detecting Auto Showing

You can use the **C1Zoom.ControlAutoShowing** event to determine the cancelation of the control's Auto Showing based on the cause of the event.

For example, you can cancel the auto scroll of the Form when the control is partially displayed.

### Visual Basic

```vbnet
Imports C1.Win.TouchToolKit

Private Sub C1Zoom1_ControlAutoShowing(sender As System.Object, e As ControlAutoShowingEventArgs) Handles C1Zoom1.ControlAutoShowing
    ' Do not auto scroll if a control is displayed
    If C1Zoom1.GetDisplayRectangle(e.Showinc1ontrol).Size = e.Showinc1ontrol.Size Then
        e.Cancel = True
    End If
    ' Do not auto scroll if a control is partial displayed
    If C1Zoom1.GetDisplayRectangle(e.Showinc1ontrol).IsEmpty Then
        e.Cancel = True
    End If
    ' Do not auto scroll if a control's most part is shown
    If C1Zoom1.GetDisplayRectangle(e.Showinc1ontrol).Size.Height > 30 Then
        e.Cancel = True
    End If
End Sub
```

### C#

```csharp
using C1.Win.TouchToolKit;

private void C1Zoom1_ControlAutoShowing(object sender, ControlAutoShowingEventArgs e)
{
    // Do not auto scroll if a control is displayed
    if (C1Zoom1.GetDisplayRectangle(e.Showinc1ontrol).Size == e.Showinc1ontrol.Size)
    {
```
Detecting Zooming

You can detect control bounds zooming by C1Zoom.ControlBoundsZooming event, and you can detect font size zooming by C1Zoom.ControlFontZooming event.

Normally, ZoomPolicy is better way to share custom zoom behavior in multiple projects.

In C1Zoom.ControlBoundsZooming event, you can cancel built-in control bounds zooming and you can implement your custom code for a control.

[C#]

```csharp
private void c1Zoom1_ControlBoundsZooming(object sender, ControlBoundsZoomingEventArgs e)
{
    if (e.TargetControl == button1)
    {
        e.Handled = true;
    }
}
```

[Visual Basic]

```vbnet
Imports C1.Win.TouchToolKit
```

{ 
    e.Cancel = true;
} 

// Do not auto scroll if a control is partial displayed
if (!C1Zoom1.GetDisplayRectangle(this.button1).IsEmpty)
{
    e.Cancel = true;
}

// Do not auto scroll if a control's most part is shown
if (C1Zoom1.GetDisplayRectangle(this.button1).Size.Height > 30)
{
    e.Cancel = true;
}
Private Sub C1Zoom1_ControlBoundsZooming(sender As System.Object, e As ControlBoundsZoomingEventArgs)
Handles C1Zoom1.ControlBoundsZooming
    If e.TargetControl Is Button1 Then
        e.Handled = True
    End If
End Sub

In C1Zoom.ControlFontZoming event, you can cancel built-in Font zooming and you can implement your custom code for a control. In the event, you need change default Font of target control because Control.Font property inherits parent control’s Font setting if the Control.Font property is default. Also, C1Zoom.InnerPanel object need to be excluded.

[C#]

using C1.Win.TouchToolKit;

private void Form1_Load(object sender, EventArgs e)
{
    // There are 3 buttons on the Form, and change default font of a button
    button1.Font = new Font("MS UI Gothic", 9.0f);
}

private void c1Zoom1_ControlFontZooming(object sender, ControlFontZoomingEventArgs e)
{
    if (e.TargetControl == button1)
    {
        e.Handled = true;
    }
}

[Visual Basic]

Imports C1.Win.TouchToolKit

Private Sub Form1_Load(sender As System.Object, e As System.EventArgs) Handles MyBase.Load
    ' There are 3 buttons on the Form, and change default font of a button
    Button1.Font = New Font("MS UI Gothic", 9.0F)
End Sub
Private Sub C1Zoom1_ControlFontZooming(sender As System.Object, e As ControlFontZoomingEventArgs) Handles C1Zoom1.ControlFontZooming
    If e.TargetControl Is Button1 Then
        e.Handled = True
    End If
End Sub

### Zoom Detecting Manipulation String

C1Zoom.ManipulationStarting event occurs when the manipulation processor is first created. It means you can handle zoom or pan gesture before C1Zoom process it. For example, if you want to give priority to scrolling ListBox than scrolling the Form, use the following code.

[C#]

```csharp
using C1.Win.TouchToolKit;

private void c1Zoom1_ManipulationStarting(object sender, ZoomManipulationStartingEventArgs e)
{
    if (e.TargetControl is ListBox)
    {
        e.Mode = ZoomManipulationModes.Zoom;
    }
    else
    {
        e.Mode = ZoomManipulationModes.All;
    }
}
```

[Visual Basic]

```vbnet
Imports C1.Win.TouchToolKit

Private Sub C1Zoom1_ManipulationStarting(sender As System.Object, e As ZoomManipulationStartingEventArgs) Handles C1Zoom1.ManipulationStarting
    If TypeOf e.TargetControl Is TextBox AndAlso DirectCast(e.TargetControl, TextBox).Text.Length > 0 Then
        e.Mode = ZoomManipulationModes.Zoom
    Else
        e.Mode = ZoomManipulationModes.All
    End If
End Sub
```
Zoom Factor Changed

C1Zoom.ZoomFactorChanged event occurs when C1Zoom.ZoomFactor changed. For example, you can show current ZoomFactor on the Titlebar of the Form.

[C#]

```csharp
private void c1Zoom1_ZoomFactorChanged(object sender, EventArgs e)
{
    this.Text = string.Format("{0:P}", c1Zoom1.ZoomFactor);
}
```

[Visual Basic]

```vbnet
Private Sub C1Zoom1_ZoomFactorChanged(sender As System.Object, e As System.EventArgs) Handles C1Zoom1.ZoomFactorChanged
    Me.Text = String.Format("{0:P}", C1Zoom1.ZoomFactor)
End Sub
```

To see how the **ZoomFactorChanged** event works you can look at the **ZoomFactorFeatures** demo in the **TouchToolkit for WinForms** explorer sample.

When you select the **ZoomFactorFeatures** demo, the default ZoomFactor appears as 100%:
Now, select the area on the form that you wish to zoom by pinching two fingers together and then gradually expanding your fingers apart. The following image shows the form in the preview mode before the items zoom in:

Now, select the area on the form that you wish to zoom by pinching two fingers together and then gradually expanding your fingers apart. The following image shows the form in the preview mode before the items zoom in:

Release your fingers and the form appears like the following:
The log on the sample shows the changed value of the **ZoomFactor** property:
C1Zoom Limitations

The following limitations exist for the C1Zoom control:

- Form.AutoScroll property is invalid. Set Form.AutoScroll=False.
- Form.Capture property is invalid. Use C1Zoom.InnerPanel.Capture instead.
- Form.Controls property is invalid. Use C1Zoom.Controls or C1Zoom.InnerPanel.Controls instead.
- C1Zoom and C1Resize of PlusPak can’t be used in same Form.
- C1Zoom and C1StylePlus of PlusPak can’t be used in same Form.
- C1Zoom and C1SizerLight of Winforms Edition can’t be used in same Form.
- C1Zoom and C1Sizer of Winforms Edition can’t be used in same Form.
- C1Zoom and C1ZoomPanel can’t be used in same Form.
- Can’t zoom ContextMenu and ToolTip.
- Can’t show zoom preview for ActiveX control.
- Can’t show zoom preview for WebBrowser control. A gray cross and the document title is displayed instead.

Run Time Compatibility with C1Zoom

You can provide touch zoom functionality for the controls of the form without modifying C1Zoom’s settings at design time, but you will need to modify some of the Form’s functions. At runtime, the C1Zoom component moves all controls on the Form to its own inner Panel.

Zoom Run Time Properties

Use the following properties when using zoom functionality at runtime:

- C1Zoom.InnerPanelCapture property – This property works instead of using the Form.Capture property
- C1Zoom.Controls or C1Zoom.InnerPanel Controls – This property works instead of using the Form.Controls property
C1ZoomCommandProvider Overview

The C1ZoomCommandProvider is a component that can be added to a user control's component tray. The following section describes the C1ZoomCommandProvider's built-in actions and custom actions.

To add a C1ZoomCommandProvider to your user control complete the following:

1. Add a user control to your Visual Studio project.
2. Add the C1ZoomCommandProvider from the Toolbox. If it does not appear in the Toolbox, right-click the Toolbox and select Add Tab. Name the tab TouchToolKit and right-click the area in the tab and select Choose Items. Browse to the location of the C1.Win.C1TouchToolkit.4.dll and click Open.

   The C1ZoomCommandProvider1 item will appear in the component tray.

C1ZoomCommandProvider Built-In Commands

Built-in actions

Any standard control you add to the user control can be associated with C1ZoomCommandProvider's built-in commands.

Adding a built-in command to any standard control

To add a C1ZoomCommandProvider's built-in command to any standard control, complete the following:

1. Add a UserControl to the project. In the Solution Explorer, right-click the solution name and select Add|User Control. Name the user control (MyUserControl).
2. Click Add in the Add New Item dialog box.
3. Add a control such as a Button to MyUserControl.
4. Add the C1ZoomCommandProvider from the Toolbox. If it does not appear in the Toolbox, right-click the Toolbox and select Add Tab. Name the tab TouchToolKit and right-click the area in the tab and select Choose Items. Browse to the location of the C1.Win.C1TouchToolkit.4.dll and click Open. The C1ZoomCommandProvider1 item will appear in the component tray.
5. Add a control such as a Button, Button1.
6. Add the C1ZoomCommandProvider from the Toolbox. If it does not appear in the Toolbox, right-click the Toolbox and select Add Tab. Name the tab TouchToolKit and right-click the area in the tab and select Choose Items. Browse to the location of the C1.Win.C1TouchToolkit.4.dll and click Open. The C1ZoomCommandProvider1 item will appear in the component tray.
7. Click the smart-tag anchor of Button1.

The following built-in Zoom Commands appear:
The following table lists and describes the built-in **Zoom Commands**:

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Does not do any commands, default value.</td>
</tr>
<tr>
<td>ZoomIn</td>
<td>ZoomIn 10% when executing this command. If current factor reaches the max factor value, then do nothing.</td>
</tr>
<tr>
<td>ZoomOut</td>
<td>ZoomOut 10% when executing this command. If the current factor reaches the max factor value then it does nothing.</td>
</tr>
<tr>
<td>ResetZoom</td>
<td>Changes zoom factor to 100%.</td>
</tr>
<tr>
<td>ScrollLeft</td>
<td>Scrolls Left 1/3 of the form content width. If the form has scrolled to the end point, then it does nothing.</td>
</tr>
<tr>
<td>ScrollRight</td>
<td>Scrolls Right 1/3 of form content width. If the form has scrolled to the end point, then it does nothing.</td>
</tr>
<tr>
<td>ScrollUp</td>
<td>Scrolls Up 1/3 of form content height. If the form has scrolled to the end point, then it does nothing.</td>
</tr>
<tr>
<td>ScrollDown</td>
<td>Scrolls Down 1/3 of form content height. If the form has scrolled to the end point, then it does nothing.</td>
</tr>
<tr>
<td>SwitchFullScreen</td>
<td>Changes the form to full screen. If the form is already in full screen, then it remains in full screen.</td>
</tr>
<tr>
<td>ShowOrHidePreviewWindow</td>
<td>Shows the pan window. If the pan window is shown, close pan window.</td>
</tr>
<tr>
<td>SelectNextControl</td>
<td>Moves active control to next.</td>
</tr>
<tr>
<td>SelectPreviousControl</td>
<td>Moves active control to previous.</td>
</tr>
</tbody>
</table>
Custom action

You can access the Form which hosts the ControlBar by using the `C1ZoomCommandProvider.OwnerForm` property. Use the following code to close the Form.

```csharp
C1ZoomCommandProvider1.OwnerForm.Close();
```

You can access the `C1Zoom` component which is in the Form by using the `C1ZoomCommandProvider.OwnerC1Zoom` property. Use the following code to scroll to the right-down direction.

```csharp
private void button1_Click(object sender, EventArgs e)
{
    Point currentLocation = C1ZoomCommandProvider1.OwnerC1Zoom.ScrollLocation;
    currentLocation.X -= 100;
    currentLocation.Y -= 100;
    C1ZoomCommandProvider1.OwnerC1Zoom.ScrollLocation = currentLocation;
}
```
C1ZoomPanel Overview

You can add zoom ability to part of the Form by the C1ZoomPanel. It is useful to exclude some part of the Form, MenuStrip, ToolStrip or other controls. The C1ZoomPanel is similar to a Panel control, except it allows users to zoom controls in the Panel.

In C1ZoomPanel, the following items are not available: ControlBar, Pan Window, Full-screen mode and Keep Aspect Ratio. The C1ZoomPanel, C1Zoom and C1ApplicationZoom controls can’t be used together.

Adding C1ZoomPanel to WinForms Explorer Sample

To use the C1ZoomPanel to make both TreeView and ListView zoomable in an Explorer Form, complete the following:

1. Open the Visual Basic WinForms ControlExplorer sample located in Documents\ComponentOne Samples\WinForms\ControlExplorer.
2. Select Add New Item from the Project menu in Visual Studio.
3. Click on the Project menu and select WindowsApplication 1 Properties.
4. Select Explorer1 from the Start-up form dropdown menu.
5. Add **C1ZoomPanel** from the Toolbox to **Explorer1**.

6. From the View menu click **View| Other windows| Document Outline** to show the **Document Outline** window.

7. Select C1ZoomPanel1 on the Document Outline window, then move it to a place between ToolstripContainer.ContentPanel and SplitContainer.
8. Drag SplitContainer to move on the C1ZoomPanel1.

9. Select C1ZoomPanel1 on the Document Outline window, then click C1ZoomPanel's smart tag to open C1ZoomPanel’s Tasks menu and select Dock to Parent.
10. Run the project. Make sure both **TreeView** and **ListView** can be zoomed by touch.

The following features are also available in the C1Zoom component:

- Zooming
- Scrolling
- Zooming Events

### Adding Controls at Run Time

To add some controls to the **C1ZoomPanel** at runtime, use **C1ZoomPanel.BeginAddControls** and **C1ZoomPanel.EndAddControls** methods. You also have to use the **C1ZoomPanel.InnerControls** property instead of the **C1ZoomPanel.Controls** property.

1. Add C1ZoomPanel to the Form. (C1ZoomPanel1)
2. Add a Button control to the C1ZoomPanel. (Button1)
3. Use the following code for Button1.Click event.

```csharp
private void button1_Click(object sender, EventArgs e)
{
    c1ZoomPanel1.BeginAddControls();

    Button button2 = new Button();
    c1ZoomPanel1.InnerControls.Add(button2);
    button2.Text = "Button2";
    button2.Visible = true;
    button2.Left = button1.Left;
    button2.Top = button1.Top + button1.Height + 10;

    c1ZoomPanel1.EndAddControls();
}
```
Visual Basic

Private Sub Button1_Click(sender As System.Object, e As System.EventArgs) Handles Button1.Click
    C1ZoomPanel1.BeginAddControls()

    Dim button2 As New Button()
    C1ZoomPanel1.InnerControls.Add(button2)
    button2.Text = "Button2"
    button2.Visible = True
    button2.Left = Button1.Left
    button2.Top = Button1.Top + Button1.Height + 10

    C1ZoomPanel1.EndAddControls()
End Sub

4. Run the project, Zoom-up C1ZoomPanel content and then click Button1.
5. Confirm Button2 is added in the expected position and size.

ZoomPanel Limitations

The following limitation exists for the C1ZoomPanel control:
- Same limitations with C1Zoom.
C1ApplicationZoom Overview

The **C1ApplicationZoom** component allows users to zoom all controls of all Forms in the project. Creating and using a **C1ApplicationZoom** is very similar to using a regular **C1Zoom** component. In **C1Zoom**, you need to add a **C1Zoom** component to each Form. In **C1ApplicationZoom**, you need to add a **C1ApplicationZoom** component only to the start-up Form because **C1ApplicationZoom** automatically attaches **C1Zoom** components to each Form which is accessible from the **Application.OpenForms** static property. If there is a **C1Zoom** or **C1ZoomPanel** on a Form, **C1ApplicationZoom** skips the Form so you can continue use these controls to implement custom zoom settings for the specific Form.

- **C1ApplicationZoom** component does not support PanWindow.
- The start-up form is a Form which is selected in the Start-up Form dropdown list in the project properties window in a Visual Basic project. In a Visual C# project, the Start-up Form is a Form of a parameter of the Program.cs in the Application.Run method.

C1ApplicationZoom Supported Forms

C1ApplicationZoom can zoom all Forms which can be referenced from **Application.OpenForms** static property. **MessageBox**, **InputBox** and other common dialogs are not supported.

Getting the Associated Zoom

Getting the Associated C1Zoom

You can get the associated C1Zoom by a specific form programmatically by using the **C1ApplicationZoom.GetC1Zoom** static method. It always works even if the current application has a **C1ApplicationZoom** instance or not.

[C#]

```csharp
using C1.Win.TouchToolKit;

private void button1_Click(object sender, EventArgs e)
{
    Form2 subForm = new Form2();
    C1Zoom c1Zoom = C1ApplicationZoom.GetC1Zoom(subForm);
    if (c1Zoom == null)
    {
        MessageBox.Show("There is no C1Zoom.");
    }
    else
    {
```

Copyright © 2017 GrapeCity, inc. All rights reserved.
MessageBox.Show("There is a C1Zoom.");
c1Zoom.AboutBox();
}
subForm.Show();
}

[Visual Basic]
Imports C1.Win.TouchToolKit

Private Sub Button1_Click(sender As System.Object, e As System.EventArgs) Handles Button1.Click
    Dim subForm As New Form2()
    Dim c1Zoom As C1Zoom = C1ApplicationZoom.GetC1Zoom(subForm)
    If c1Zoom Is Nothing Then
        MessageBox.Show("There is no C1Zoom.")
    Else
        MessageBox.Show("There is a C1Zoom.")
        c1Zoom.AboutBox()
    End If
    subForm.Show()
End Sub

Excluding a Form

You can cancel attaching of C1ApplicationZoom by using C1ApplicationZoom.Attaching event. Following code excludes Form2 when the project has Form1 and Form2, and Form1 has C1ApplicationZoom.

[C#]
private void c1Application_Attaching(object sender, C1ZoomAttachedEventArgs e)
{
    if(e.Form is Form2)
    {
        e.Cancel = true; // Form2 is excluded
    }
}
Setting Max Zoom Factor

Setting the Maximum ZoomFactor of a Specific Form

The following code sets the maximum ZoomFactor of a specific Form.

[C#]

```csharp
private void c1Application_Attaching(object sender, C1ZoomAttachedEventArgs e)
{
    if(e.Form is Form2)
    {
        e.C1Zoom.MaxZoomFactor = 3.0f;
    }
}
```

[Visual Basic]

```vbnet
Imports C1.Win.TouchToolBar

Private Sub C1ApplicationZoom1_C1ZoomAttaching(sender As Object, e As C1ZoomAttachingEventArgs) Handles C1ApplicationZoom1.C1ZoomAttaching
    If TypeOf e.Form Is Form2 Then
        e.C1Zoom.MaxZoomFactor = 3.0F
    End If
End Sub
```
C1ApplicationZoom Limitations

C1ZoomPanel Limitations

The following limitation exists for the C1ZoomPanel control:
- Same limitations with C1Zoom.
C1TouchEventProvider Overview

C1TouchEventProvider enables you to detect Touch and Pen(Gesture), Manipulation, and Pointer events. In a Button control, both Mouse Click and Tap raises the Button.Click event. In the C1TouchEventProvider, the Control.Click event does not occur by Tap, and the C1TouchEventProvider.Tapped event occurs instead. Therefore, you can use different events to invoke different actions.

If you are looking for C1Zoom, C1ZoomPanel and C1ApplicationZoom events, see Zooming events.

TouchEventProvider Events

The following table lists and describes the events of the C1TouchEventProvider.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DoubleTapped</td>
<td>Occurs when an otherwise unhandled DoubleTap interaction occurs over the hit test area of this element.</td>
</tr>
<tr>
<td>Holding</td>
<td>Occurs when an otherwise unhandled Hold interaction occurs over the hit test area of the control.</td>
</tr>
<tr>
<td>ManipulationCompleted</td>
<td>Occurs when a manipulation on the control is complete.</td>
</tr>
<tr>
<td>ManipulationDelta</td>
<td>Occurs when the input device changes position during a manipulation.</td>
</tr>
<tr>
<td>ManipulationInertiaStarting</td>
<td>Occurs when the input device loses contact with the UIElement object during a manipulation and inertia begins.</td>
</tr>
<tr>
<td>ManipulationStarting</td>
<td>Occurs when the manipulation processor is first created.</td>
</tr>
<tr>
<td>ManipulationStarted</td>
<td>Occurs when an input device begins a manipulation on the control.</td>
</tr>
<tr>
<td>PointerCanceled</td>
<td>Occurs when a pointer that made contact abnormally loses contact.</td>
</tr>
<tr>
<td>PointerCaptureLost</td>
<td>Occurs when pointer capture previously held by this element moves to another control or elsewhere.</td>
</tr>
<tr>
<td>PointerEntered</td>
<td>Occurs when a pointer enters the hit test area of the control.</td>
</tr>
<tr>
<td>PointerExited</td>
<td>Occurs when a pointer leaves the hit test area of the control.</td>
</tr>
<tr>
<td>PointerMoved</td>
<td>Occurs when a pointer moves while the pointer remains within the hit test area of the control.</td>
</tr>
<tr>
<td>PointerPressed</td>
<td>Occurs when the pointer device initiates a Press action within the control.</td>
</tr>
<tr>
<td>Event</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PointerReleased</td>
<td>Occurs when the pointer device that previously initiated a Press action is released, while within the control. Note that the end of a Press action is not guaranteed to fire a PointerReleased event; other events may fire instead. For more info, see Remarks.</td>
</tr>
<tr>
<td>RightTapped</td>
<td>Occurs when a right-tap input stimulus happens while the pointer is over the element.</td>
</tr>
<tr>
<td>Tapped</td>
<td>Occurs when an otherwise unhandled Tap interaction occurs over the hit test area of the control.</td>
</tr>
</tbody>
</table>

**Routing Policy**

All events in the **C1TouchEventProvider** are Routed Events. That means you can attach a **C1TouchEventProvider** to a container and you can use an event for children of the container.

To try the **C1TouchEventProvider** event add three buttons on a panel and attach the **C1TouchEventProvider** event to the Panel using the following code below.

[C#]

```csharp
using C1.Win.TouchToolKit;

private void Form1_Load(object sender, EventArgs e)
{
    c1TouchEventProvider1.SetEnableTouchEvents(panel1, true);
    c1TouchEventProvider1.Tapped += new EventHandler<TappedEventArgs>(c1TouchEventProvider1_Tapped);
}

private void c1TouchEventProvider1_Tapped(object sender, TappedEventArgs e)
{
    if (e.TargetControl != null)
        Console.WriteLine(e.TargetControl.Name);
}
```

[Visual Basic]

```csharp
Imports C1.Win.TouchToolKit

Private Sub Form1_Load(sender As Object, e As EventArgs)
    c1TouchEventProvider1.SetEnableTouchEvents(panel1, True)
    c1TouchEventProvider1.Tapped += New EventHandler(Of TappedEventArgs)(c1TouchEventProvider1_Tapped)
End Sub

Private Sub c1TouchEventProvider1_Tapped(sender As Object, e As TappedEventArgs)
    If e.TargetControl IsNot Nothing Then
        Console.WriteLine(e.TargetControl.Name)
    End If
End Sub
```
The C1TouchEventProvider.Tapped event occurs when you tap a button on the Panel.

**Routed Direction**

The routed direction is from child control to parent control. For example, Button1 is a child of Panel1. Both Button1 and Panel1 are attached to the C1TouchEventProvider. Button1 first receives the TouchEvents, then Panel1 receives the TouchEvents.

[C#]

```csharp
using C1.Win.TouchToolKit;

private void Form1_Load(object sender, EventArgs e)
{
    c1TouchEventProvider1.SetEnableTouchEvents(button1, true);
    c1TouchEventProvider1.SetEnableTouchEvents(panel1, true);
    c1TouchEventProvider1.Tapped += new EventHandler<TappedEventArgs>(c1TouchEventProvider1_Tapped);
}

private void c1TouchEventProvider1_Tapped(object sender, TappedEventArgs e)
{
    if (e.TargetControl != null)
        Console.WriteLine(”{0},{1}”, DateTime.Now.ToString(), e.TargetControl.Name);
}
```

[Visual Basic]

```vbnet
Imports C1.Win.TouchToolKit
```

Copyright © 2017 GrapeCity, inc. All rights reserved.
Private Sub Form1_Load(sender As System.Object, e As System.EventArgs) Handles MyBase.Load
    C1TouchEventProvider1.SetEnableTouchEvents(Button1, True)
    C1TouchEventProvider1.SetEnableTouchEvents(Panel1, True)
End Sub

Private Sub C1TouchEventProvider1_Tapped(sender As System.Object, e As TappedEventArgs) Handles C1TouchEventProvider1.Tapped
    If Not e.TargetControl Is Nothing Then
        Console.WriteLine("{0},{1}, {2}", DateTime.Now.ToString(), e.TargetControl.Name, e.OriginalSource.Name)
    End If
End Sub

**Routed Event Information**

All event args of **C1TouchEventProvider** are inherited from the **TouchRoutedEventArgs** class. You can get the TargetControl and OriginalSource from the TouchRoutedEventArgs. The TargetControl is the control for current event. The OriginalSource is the first control which received this event.

For example, Button1 is a child of Panel1. Both Button1 and Panel1 are attached to the C1TouchEventProvider. When Button1 receives the TouchEvent, the TargetControl is "Button1" and the OriginalSource is "Button1". When Panel1 receives the TouchEvent, TargetControl is "Panel1" and OriginalSource is "Button1".

[C#]

```csharp
using C1.Win.TouchToolKit;

private void Form1_Load(object sender, EventArgs e)
{
    this.c1TouchEventProvider1.SetEnableTouchEvents(button1, true);
    this.c1TouchEventProvider1.SetEnableTouchEvents(panel1, true);
    this.c1TouchEventProvider1.Tapped += new EventHandler<TappedEventArgs>(c1TouchEventProvider1_Tapped);
}

private void c1TouchEventProvider1_Tapped(object sender, TappedEventArgs e)
{
    if (e.TargetControl != null)
        Console.WriteLine("(0),(1),(2)", DateTime.Now.ToString(), e.TargetControl.Name, e.OriginalSource.Name);
}
```
Imports C1.Win.TouchToolKit

Private Sub Form1_Load(sender As System.Object, e As System.EventArgs) Handles MyBase.Load
    C1TouchEventProvider1.SetEnableTouchEvents(Button1, True)
    C1TouchEventProvider1.SetEnableTouchEvents(Panl1, True)
End Sub

Private Sub C1TouchEventProvider1_Tapped(sender As System.Object, e As TappedEventArgs) Handles C1TouchEventProvider1.Tapped
    If Not e.TargetControl Is Nothing Then
        Console.WriteLine("{0},{1},{2}", DateTime.Now.ToString(), e.TargetControl.Name, e.OriginalSource.Name)
    End If
End Sub

Handle the Event

If the Child control is handed a specific TouchEvent and you do not want the parent control to get this event you can set the TouchRoutedEventArgs.Handled property to True to block its routed event.

[C#]

using C1.Win.TouchToolKit;

private void Form1_Load(object sender, EventArgs e)
{
    this.c1TouchEventProvider1.SetEnableTouchEvents(button1, true);
    this.c1TouchEventProvider1.SetEnableTouchEvents(panel1, true);
    this.c1TouchEventProvider1.Tapped += new EventHandler<TappedEventArgs>(c1TouchEventProvider1_Tapped);
}

private void c1TouchEventProvider1_Tapped(object sender, TappedEventArgs e)
{
    if (e.TargetControl != null)
    {
        Console.WriteLine("(0),(1),(2)", DateTime.Now.ToString(), e.TargetControl.Name, e.OriginalSource.Name);
        e.Handled = true;
    }
}
Detecting Touch

You can detect Tap, Double-Tap and Right-Tap by using the C1TouchEventProvider. To detect Hold instead of Right-Tap, see Detecting Tap and Hold (Long-Tap).

The following steps indicates how to detect Click by Mouse or Tap by Touch.

1. Add a Button control to the Form. (Button1).
2. Add a C1TouchEventProvider to the Form. (C1TouchEventProvider1).
3. Select the Button1, Set "EnableTouchEvents of C1TouchEventProvider1" property to True in the Property Window.
4. Select the Button1, Add the following code to its Click event.
5. Select the C1TouchEventProvider1, Add the following code to its Tapped event.

[C#]

```csharp
using C1.Win.TouchToolKit;

private void Form1_Load(object sender, EventArgs e)
{
    c1TouchEventProvider1.SetEnableTouchEvents(this.button1, true);
    c1TouchEventProvider1.Tapped += c1TouchEventProvider1_Tapped;
    button1.Click += button1_Click;
}
```
private void C1TouchEventProvider1_Tapped(object sender, TappedEventArgs e) {
    if (e.TargetControl == button1) {
        MessageBox.Show("Pressed by Touch");
    }
}

private void button1_Click(object sender, EventArgs e) {
    MessageBox.Show("Pressed by Mouse");
}

[Visual Basic]
Imports C1.Win.TouchToolKit

Private Sub Form1_Load(sender As System.Object, e As System.EventArgs) Handles MyBase.Load
    C1TouchEventProvider1.SetEnableTouchEvents(Button1, True)
End Sub

Private Sub Button1_Click(sender As System.Object, e As System.EventArgs) Handles Button1.Click
    If e.TargetControl = Button1 Then
        MessageBox.Show("Pressed by Mouse")
    End If
End Sub

Private Sub C1TouchEventProvider1_Tapped(sender As System.Object, e As C1.Win.TouchToolKit.TappedEventArgs) Handles C1TouchEventProvider1.Tapped
    MessageBox.Show("Pressed by Touch")
End Sub

6. Run the project, and Click or Tap Button1.

**Detecting Gesture**
The following steps indicate how to detect gestures, rotation, and scale and transition.

1. **Add a Label control to the Form (Label1)** and set the following properties:
   - AutoSize = False
   - BackColor = Color.Cyan
   - Size = Size(300, 300)

2. **Add a C1TouchEventProvider to the Form. (C1TouchEventProvider1)**

3. **Select Label1** and set the "EnableTouchEvents of C1TouchEventProvider1" property to **True** in the Property Window.

4. **Select the C1TouchEventProvider1**, and add the following code to its **ManipulationDelta** event.

[C#]

```csharp
using C1.Win.TouchToolKit;

private void Form1_Load(object sender, System.EventArgs e)
{
    label1.AutoSize = false;
    label1.BackColor = Color.Cyan;
    label1.Size = new Size(300, 300);
    c1TouchEventProvider1.SetEnableTouchEvents(label1, true);
    c1TouchEventProvider1.ManipulationDelta += c1TouchEventProvider1_ManipulationDelta;
}

private void c1TouchEventProvider1_ManipulationDelta(object sender, ManipulationDeltaEventArgs e)
{
    if (e.TargetControl == this.label1)
    {
        label1.Text = string.Format("Scale:{0}\nRotation:{1}\nTranslation:{2}", e.Delta.Scale, e.Delta.Rotation, e.Delta.Translation);
    }
}
```

[Visual Basic]

```vbnet
Imports C1.Win.TouchToolKit

Private Sub Form1_Load(sender As System.Object, e As System.EventArgs) Handles MyBase.Load
    Label1.AutoSize = False
    Label1.BackColor = Color.Cyan
    Label1.Size = New Size(300, 300)
    c1TouchEventProvider1.SetEnableTouchEvents(Label1, True)
    AddHandler c1TouchEventProvider1.ManipulationDelta, Sub(sender, e) Handles c1TouchEventProvider1.ManipulationDelta
        If e.TargetControl Is This.Label1 Then
        End If
    End Sub
```

Copyright © 2017 GrapeCity, inc. All rights reserved.
Label1.Size = New Size(300, 300)
C1TouchEventProvider1.SetEnableTouchEvents(Label1, True)
End Sub

Private Sub C1TouchEventProvider1_ManipulationDelta(sender As System.Object, e As ManipulationDeltaEventArgs)
Handles C1TouchEventProvider1.ManipulationDelta
    If e.TargetControl Is Label1 Then
    End If
End Sub

5. Run the project, and operate Rotation or Scale multi-touch actions on the Label1.

You can use C1TouchEventProvider.ManipulationMode property to detect specific gestures. To detect only rotation gesture, use the following code.

[C#]


[Visual Basic]


To detect multiple gesture, use the following code.

[C#]

using C1.Win.TouchToolKit;

c1TouchEventProvider1.ManipulationMode = ManipulationModes.Rotate | ManipulationModes.Scale;

[Visual Basic]

Imports C1.Win.TouchToolKit

C1TouchEventProvider1.ManipulationMode = ManipulationModes.Rotate Or ManipulationModes.Scale
Detecting Pen

The `PointerEventArgs.DeviceType` property indicates the type of device used in the event handler. The property returns `Touch` for finger and `Pen` for Pen device.

1. Ensure your device supports the Pen device.
2. Create a new Windows Forms Application Project.
3. Add a Standard `Button` control and a `C1TouchEventProvider` control to the Form.
4. Use the following code:

[C#]

```csharp
using C1.Win.TouchToolKit;

private void Form1_Load(object sender, EventArgs e)
{
    c1TouchEventProvider1.SetEnableTouchEvents(button1, true);
    c1TouchEventProvider1.Tapped += c1TouchEventProvider1_Tapped;
    button1.Click += button1_Click;
}

private void c1TouchEventProvider1_Tapped(object sender, TappedEventArgs e)
{
    if (e.TargetControl == this.button1)
    {
        if (e.PointerDeviceType == PointerDeviceType_touchType)
        {
            MessageBox.Show("Pressed by Touch");
        }
        else if (e.PointerDeviceType == PointerDeviceType_Pen)
        {
            MessageBox.Show("Pressed by Pen");
        }
    }
}

private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Pressed by Mouse");
}
```

Copyright © 2017 GrapeCity, inc. All rights reserved.
Detecting Tap and Hold

The Tap and Hold touch action is converted to the Right-Click action on Windows. You can catch this event by using the `C1TouchEventProvider.RightTapped` event. When the `C1TouchEventProvider.EnablePressAndHold` property is `False`, the `RightTapped` event does not occur and the `Tapped` event occurs repeatedly during the tap and hold action. For example, you can use this to implement the custom SpinButton for touch by using a couple of buttons.

[C#]

```csharp
C1TouchEventProvider1.EnablePressAndHold = false;
```

[Visual Basic]

```vbnet
C1TouchEventProvider1.EnablePressAndHold = False
```

Copyright © 2017 GrapeCity, inc. All rights reserved.
Disabling Pen Flicks

Pen Flicks may be available to quickly navigate and perform shortcuts on your device with a flick of your pen. The following screenshot indicates the Pen Flicks option of the Control Panel when the Pen Flicks is available on the device.

To disable the Pen Flicks, set the `C1TouchEventProvider.EnablePenFlicks` property to False.

[C#]

```csharp
C1TouchEventProvider1.EnablePenFlicks = false;
```

[Visual Basic]

```vbnet
C1TouchEventProvider1.EnablePenFlicks = False
```

Detecting Maximum Touch Count

In order to get the maximum touch count capability on the system, you can use the `C1TouchEventProvider.MaximumTouches` static property. You can detect Windows 8 and Windows Server 2012 by
using the `IsWindows8` function and then you can show a message to the users whether or not the multi-touch is supported like in the following code:

[C#]
```
using C1.Win.TouchToolKit;

// Returns true if the OS is Windows 8, Windows Server 2012 or newer. Otherwise false.
private static bool IsWindows8()
{
    return false;
}

private void button1_Click(object sender, EventArgs e)
{
    if (C1TouchEventProvider.MaximunTouches < 2 || IsWindows8() == false)
    {
        MessageBox.Show("This is Unsupported environment.");
    }
    else
    {
        MessageBox.Show("This is Supported environment.");
    }
}
```

[Visual Basic]
```
Imports C1.Win.TouchToolKit

' Returns true if the OS is Windows 8, Windows Server 2012 or newer. Otherwise false.
Private Shared Function IsWindows8()
    Return False
End Function

Private Sub Button1_Click(sender As System.Object, e As System.EventArgs) Handles Button1.Click
    If C1TouchEventProvider.MaximunTouches < 2 AndAlso IsWindows8() = False Then
```
C1TouchEventProviderLimitations

The following limitation exists for the C1TouchEventProvider control:

- Can't get touch event of DropDown window.
- Can't get touch event of ElementHost control.
C1Magnify Overview

With the C1Magnify component you can view a popup Magnifier Window over any UIElement by tapping the UIElement you wish to magnify. In Windows Desktop, the text in the TextBox is hidden by your finger when you are selecting the text by touch. C1Magnify allows you to look at the selected text via the Magnifier Window.

The Magnifier Window is displayed, by default, when you tap and hold your finger on the control which enables C1Magnify.

The following table illustrates how the Magnifier window frame appears when its C1Magnify.ShowMouseCursor property is set to True and/or False:

<table>
<thead>
<tr>
<th>False (Default)</th>
<th>True</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Magnifier Window False" /></td>
<td><img src="image2" alt="Magnifier Window True" /></td>
</tr>
</tbody>
</table>

Enabling Magnify Window
Enable the Magnifier Window

You can enable the Magnifier Window by the following steps:

1. Add a TextBox control to the Form. (TextBox1)
2. Add a C1Magnify component to the Form. (C1Magnify)
3. Select TextBox1 and set the "EnableMagnifier of C1Magnify1" property to True in the Property Window.
4. Run the Project.
5. Tap TextBox1 for a few seconds, then you will see the Magnifier Window appear above your finger.

Hiding Mouse Cursor Inside the Magnifier Window

Hiding the Mouse Cursor Inside the Magnifier Window

You can hide the Mouse Cursor inside the Magnifier Window by setting the C1Magnify.ShowMouseCursor property to False.

[C#]

C1Magnify1.ShowMouseCursor = false;

[Visual Basic]

C1Magnify1.ShowMouseCursor = False

Changing ZoomFactor for Magnifier Window

Magnifier ZoomFactor

You can change ZoomFactor of the Magnifier Window by using the C1Magnify.ZoomFactor property.

[C#]

C1Magnify1.ZoomFactor = 4.0f;

[Visual Basic]

C1Magnify1.ZoomFactor = 4.0F

The following table illustrates how the Magnifier window frame appears when its C1Magnify.ZoomFactor property is...
You can specify how long it will take for the magnifier popup window to appear after you press your finger on a UIElement by setting an integer value for the C1Magnify.PopupDelay property. The value is measured in seconds.

You can use the \texttt{C1Magnify.IsHorizontalRailEnabled} or \texttt{IsVerticalRailEnabled} property to fix the moving direction of the Magnifier window.

You can change the appearance of the Magnifier Window by using the \texttt{BorderBackgroundMode}, \texttt{BorderWidth}, \texttt{MagnifierShape} and \texttt{Size} property. The following code creates a rectangular shaped Magnifier Window:

\begin{verbatim}
Visual Basic
C1Magnify1.BorderWidth = 3
C1Magnify1.Shape = C1.Win.TouchToolKit.MagnifierShape.Rectangle
C1Magnify1.Size = New Size(300, 100)
C#
C1Magnify1.BorderWidth = 3;
\end{verbatim}
C1Magnify1.Shape = C1.Win.TouchToolKit.MagnifierShape.Rectangle;
C1Magnify1.Size = new Size(300, 100);

**Magnifier Shape**

You can set Magnifier window frame by using the **C1Magnify.Shape** property. The following table illustrates how the Magnifier window frame appears when its **C1Magnify.Shape** property is set to Circle(Default), Rectangle, and RoundedRectangle:

<table>
<thead>
<tr>
<th>Circle(Default)</th>
<th>Rectangle</th>
<th>RoundedRectangle</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="Image" alt="Circle" /></td>
<td><img src="Image" alt="Rectangle" /></td>
<td><img src="Image" alt=" RoundedRectangle" /></td>
</tr>
</tbody>
</table>

**Magnifier Size**

**Magnifier Size**

You can set the size of the Magnifier window by using the C1Magnify.Size property. The following table illustrates the effect of the difference values set for the C1Magnify.Size property:

<table>
<thead>
<tr>
<th>100x100 px(Default)</th>
<th>150x150 px</th>
<th>150x100 px</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="Image" alt="100x100 px" /></td>
<td><img src="Image" alt="150x150 px" /></td>
<td><img src="Image" alt="150x100 px" /></td>
</tr>
</tbody>
</table>

**Magnifier Border Width**
**Magnifier Border width**

You can set the border width of the Magnifier window by using the `C1Magnify.BorderWidth` property. The following table illustrates the effect of the different values set for the `C1Magnify.BorderWidth` property:

<table>
<thead>
<tr>
<th>Border Width</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 px (Default)</td>
<td>![Image of 5 px border]</td>
</tr>
<tr>
<td>10 px</td>
<td>![Image of 10 px border]</td>
</tr>
<tr>
<td>1 px</td>
<td>![Image of 1 px border]</td>
</tr>
</tbody>
</table>

**Magnifier Border Background**

You can set background image for the border of the Magnifier window by using the `C1Magnify.BorderBackground` property.

<table>
<thead>
<tr>
<th>Background</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Light</td>
<td>![Image of light background]</td>
</tr>
<tr>
<td>Dark</td>
<td>![Image of dark background]</td>
</tr>
<tr>
<td>Custom</td>
<td>![Image of custom background]</td>
</tr>
</tbody>
</table>

**Magnify Events**

In this section you will learn how to use the following C1Magnify events:

- `C1Magnify.Move`
- `C1Magnify.MagnifierShowing`
- `C1Magnify.Close`
Using a Different Magnifier Window in Different Controls

You can use the `C1Magnify.MagnifierShowing` event to change the Magnifier Window before the window is displayed. In the event, you can use different settings in different controls. Also, you can cancel showing the Magnifier Window when the `e.Cancel` property is set to `True`.

The following code shows a rectangular Magnifier window for a TextBox. It does not show the Magnifier window for `NumericUpDown`, but it shows a circular Magnifier window for other controls.

**Visual Basic**

```vbnet
Imports C1.Win.TouchToolKit

Private Sub Form1_Load(sender As System.Object, e As System.EventArgs) Handles MyBase.Load
    C1Magnify1.SetEnableMagnifier(TextBox1, True)
    C1Magnify1.SetEnableMagnifier(NumericUpDown1, True)
End Sub

    If e.TargetControl Is TextBox1 Then
        Dim magnifier As C1Magnify = DirectCast(sender, C1Magnify)
        magnifier.Shape = MagnifierShape.Rectangle
        magnifier.Size = New Size(200, 30)
    ElseIf e.TargetControl Is NumericUpDown1 Then
        e.Cancel = True
    Else
        Dim magnifier As C1Magnify = DirectCast(sender, C1Magnify)
        magnifier.Shape = MagnifierShape.Circle
        magnifier.Size = New Size(100, 100)
    End If
End Sub
```

**C#**

```csharp
using C1.Win.TouchToolKit;

private void Form1_Load(object sender, EventArgs e)
{
    C1Magnify1.SetEnableMagnifier(textBox1, true);
    C1Magnify1.SetEnableMagnifier(numericUpDown1, true);
}
```
private void C1Magnify1_MagnifierShowing(object sender, MagnifierEventArgs e)
{
    if (e.TargetControl == textBox1)
    {
        C1Magnify magnifier = sender as C1Magnify;
        magnifier.Shape = MagnifierShape.Rectangle;
        magnifier.Size = new Size(200, 30);
    }
    else if (e.TargetControl == numericUpDown1)
    {
        e.Cancel = true;
    }
    else
    {
        C1Magnify magnifier = sender as C1Magnify;
        magnifier.Shape = MagnifierShape.Circle;
        magnifier.Size = new Size(100, 100);
    }
}

Posting a Message when the Magnifier is Closed

Posting Message when the Magnifier Window is Closed

You can use the C1Magnify.MessageWhenClose property to post a message to the attached control when the Magnifier Window is closed. The Default value is None.

- None
- Right Click
- Left Click

The C1Magnify.Close event is available to implement a custom action.

The following code shows the ContextMenu of the TextBox when the Magnifier Window is closed.

Visual Basic

Imports C1.Win.TouchToolBar

Private Sub Form1_Load(sender As System.Object, e As System.EventArgs) Handles MyBase.Load
    TextBox1.Text = "Hello"
    C1Magnify1.SetEnableMagnifier(TextBox1, True)
    C1Magnify1.MessageWhenClose = MessageWhenClose.RightClick
Posting a Message when Magnifier Window Moves

Posting Message when the Magnifier Window Moves

You can use the C1Magnify.MessageWhenMove property to post a message to the attached control when the Magnifier Window is moving. The Default value is Mouse Move.

- None
- Mouse Move
- Mouse Left Button Down

The C1Magnify.Move event is available to implement custom action.

In the default value, you can select text when the Magnifier Window moves on the TextBox. Use the following code to move the caret on the textbox in the same case.

**Visual Basic**

```vbnet
Imports C1.Win.TouchToolKit

Private Sub Form1_Load(sender As System.Object, e As System.EventArgs) Handles MyBase.Load
    C1Magnify1.SetEnableMagnifier(TextBox1, True)
    C1Magnify1.MessageWhenMove = MessageWhenMove.LeftDown
End Sub
```

**C#**

```csharp
using C1.Win.TouchToolKit;

private void Form1_Load(object sender, EventArgs e)
{
    textBox1.Text = "Hello";
    C1Magnify1.SetEnableMagnifier(textBox1, true);
    C1Magnify1.MessageWhenMove = MessageWhenClose.RightClick;
}
```
Magnifier Limitations

C1Magnify Limitations

The following limitation exists for the C1Magnify control:

- Can't popup the Magnifier window for DropDown window.
C1MultiScaleImage Overview

Display higher resolution images by using the C1MultiScaleImage control. C1MultiScaleImage control can show different images for different zoom factors. For example, the control can show normal size images for 100% zoom and the control can show higher resolution images if the control is zoomed to 200%.

Adding Image for each ZoomFactor

To add a C1MultiScaleImage to the Form from the Toolbox, use the following code:

[C#]
```csharp
c1MultiScaleImage1.Image = image1; // Add an image for 100%
c1MultiScaleImage1.Images.Add(new ImageItem(image2, 2.0f)); // Add an image for 200%
c1MultiScaleImage1.Images.Add(new ImageItem(image3, 3.0f)); // Add an image for 300%
```

[Visual Basic]
```vbnet
C1MultiScaleImage1.Image = image1 ' Add an image for 100%
C1MultiScaleImage1.Images.Add(New ImageItem(image2, 2.0F)) ' Add an image for 200%
C1MultiScaleImage1.Images.Add(New ImageItem(image3, 3.0F)) ' Add an image for 300%
```

Removing All Images

To remove all images from a C1MultiScaleImage, use the following code:

[C#]
```csharp
c1MultiScaleImage1.Image = null;
c1MultiScaleImage1.Images.Clear();
```
C1MultiScaleImage1.Image = Nothing
C1MultiScaleImage1.Images.Clear()
Zoom Policies

If a control has a property named “ZoomFactor”, C1Zoom or C1ZoomPanel will get the control’s zoomfactor automatically and set C1Zoom or C1ZoomPanel’s zoomfactor the same as the control’s zoomfactor.

**TouchToolkit for Windows Forms** provides built-in ZoomPolicy items and ZoomPolicy templates to support standard controls and major 3rd party controls.

ZoomPolicy templates are provided as source code of C# and Visual Basic. You can modify a copy of the template to implement your custom ZoomPolicy.

To uninstall the ZoomPolicy templates from Visual Studio, delete the following zip files manually located in the C:\Program Files (x86)\Microsoft Visual Studio 11.0\Common7\IDE\ItemTemplates\CSharp\1033 folder and then run the deven.exe //installvstemplates.

- C1Chart3DZoomPolicy.zip
- C1ChartZoomPolicy.zip
- C1CommandZoomPolicy.zip
- C1FlexGridZoomPolicy.zip
- C1ReportZoomPolicy.zip
- C1ScheduleZoomPolicy.zip
- C1TrueDBGridZoomPolicy.zip
- VisualBasicPowerPacksZoomPolicy.zip
- WPF_ElementHostZoomPolicy.zip

Zoom Policies for Standard Controls

**TouchToolkit for Windows Forms** has built-in Zoom Policies to get the best zoom result for standard controls.

**To Edit Zoom Policies:**

1. Add C1Zoom, C1ZoomPanel, or C1ApplicationZoom onto the form.
2. Click Smart Tag anchor of C1Zoom, C1ZoomPanel or C1ApplicationZoom component in design time and select Edit Zoom Policies…

The following built-in zoom policies appear:
ComboBox ZoomPolicy

ComboBox will select all text when ComboBox.Font and ComboBox.Size change. The ComboBoxZoomPolicy is used to disable this default behavior.

DataGridViewZoomPolicy

TouchToolKit provides DataGridViewZoomPolicy to get a better zoom result for DataGridView. It can zoom the Size and Font for all Columns, Rows, or Cells.

If your C1DataGridView control has many rows and you try to change the default value for the rows or cells it may causes a performance problem. To avoid this, you can try to change DataGridViewZoomPolicy.ZoomRowHeight or DataGridViewZoomPolicy.ZoomCellStyleFont property value to False.

If you add new rows to the DataGridView control in code, use the C1Zoom.BeginAddControls and C1Zoom.EndAddControls method.

FlowLayoutPanelZoomPolicy

The FlowLayoutPanelZoomPolicy is used to make sure the controls’ layout is always correct when zooming in and zooming out. If you zoom in and then zoom out without using the FlowLayoutPanelZoomPolicy then the controls’ layout maybe incorrect. FlowLayoutPanelZoomPolicy can make sure that controls’ layout is always correct.

Initial zoomfactor is 100%, screenshot as shown below:
First zoom in the Form, then zoom out of the Form to 100%. The following table illustrates the difference between enabling and disabling the `FlowLayoutPanelZoomPolicy`.

<table>
<thead>
<tr>
<th>Without ZoomPolicy</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Please zoom in then zoom out.</td>
<td>Please zoom in then zoom out.</td>
</tr>
<tr>
<td>Button2</td>
<td>Button2</td>
</tr>
<tr>
<td>Button3</td>
<td>Button3</td>
</tr>
<tr>
<td>Button4</td>
<td>Button4</td>
</tr>
</tbody>
</table>

**ListBoxZoomPolicy**

ListBoxZoomPolicy ensures that the `ListBox.ColumnWidth` value changes when `ListBox.MultiColumn` is `True` and `ListBox.ColumnWidth` is bigger than `0`.

**ListView Zoom Policy**

The `ListViewZoomPolicy` changes the `ListView.Font` and `ListView.Columns[i].Width` to ensure the text is not clipped and it's viewable in the listview control while zooming.

Set zoomfactor to 120%, the following table illustrates difference between enabling and disabling the `ListZoomPolicy`.
The MonthCalendar.Font and MonthCalendar.Size properties always remain unchanged. The MonthCalendarZoomPolicy will move MonthCalendar to the center of the blank space.

Initial zoomfactor is 100%, as shown below:

<table>
<thead>
<tr>
<th>Without ZoomPolicy</th>
<th>With ZoomPolicy</th>
</tr>
</thead>
<tbody>
<tr>
<td>te... 1 2 3</td>
<td>Item Column</td>
</tr>
<tr>
<td>te... 4 5 6</td>
<td>Column 2</td>
</tr>
<tr>
<td>item3 7 8 9</td>
<td>Column 3</td>
</tr>
</tbody>
</table>

**MonthCalendarZoomPolicy**

The MonthCalendar.Font and MonthCalendar.Size properties always remain unchanged. The MonthCalendarZoomPolicy will move MonthCalendar to the center of the blank space.

Initial zoomfactor is 100%, as shown below:

**Note:** Blank Space’s BackColor is AliceBlue.

The following table illustrates the difference between enabling and disabling MonthCalendarZoomPolicy. The ZoomFactor is set to 120% for the enabled ZoomPolicy.

<table>
<thead>
<tr>
<th>Without ZoomPolicy</th>
<th>With ZoomPolicy</th>
</tr>
</thead>
</table>
**PictureBoxZoomPolicy**

*PictureBoxZoomPolicy* improves the zoom result when the *PictureBox.SizeMode* property isn’t set to Zoom and the *ZoomFactor* property is bigger than 1f.

Initial *ZoomFactor* is 100%, as shown below:

- **Note:** PictureBox's BackColor is AliceBlue.

To see how the built-in *PictureBoxZoomPolicy* works, complete the following:

1. Add a *C1Zoom* component.
2. Add a *PictureBox* control to your form.
3. Add in image in the *PictureBox* control.
4. Click on *C1Zoom*’s smart tag anchor and select **Edit Zoom Policies**. Notice the *PictureBoxZoomPolicy* Enabled
property is set to True by default.
5. Run the project and observe the PictureBox with the PictureBoxZoomPolicy.
6. Close the Form.
7. Set the PictureBoxZoomPolicy Enabled property to False.
8. Run the project and observe the PictureBox control without the PictureBoxZoomPolicy.

The following table illustrates the difference between enabling and disabling the PictureBoxZoomPolicy. Note that the ZoomFactor is set to 120% by default.

<table>
<thead>
<tr>
<th>Without ZoomPolicy</th>
<th>With ZoomPolicy</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="PictureBox" /></td>
<td><img src="image2.png" alt="PictureBox" /></td>
</tr>
</tbody>
</table>

**PropertyGridZoomPolicy**

In some cases the PropertyGrid's settings will not appear correct when zooming. The PropertyGridZoomPolicy gives you a better zoom result.

The following image illustrates the PropertyGrid with its starting ZoomFactor set to 100%.

![PropertyGrid](image3.png)

The following table illustrates the difference between the PropertyGrid without and with the ZoomPolicy when you zoom by touch:

<table>
<thead>
<tr>
<th>MaxZoomFactor</th>
<th>400%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies the maximum zoom factor of the form.</td>
<td></td>
</tr>
</tbody>
</table>
Without ZoomPolicy | With ZoomPolicy
--- | ---
![SplitContainer](image1)

**SplitContainerZoomPolicy**

When you use the zoom operation, the `SplitContainerZoomPolicy` changes the `SplitContainer.SplitLineWidth` and `SplitContainer.SplitterDistance` property values, so you can get the best zoom result.

**TableLayoutPanelZoomPolicy**

If the TableLayoutPanel's `ColumnStyle.SizeType` or `RowStyle.SizeType` property value is Absolute, the size of the TableLayoutPanel, ColumnStyle.Width and RowStyle.Height properties do not change. The `TableLayoutPanelZoomPolicy` will help you change the Column.Width and Row.Height in this case.

**ToolStripZoomPolicy**

By default, the ToolStrip's font can't zoom. When you use the `ToolStripZoomPolicy` it can help you get the best zoom result.

The following table illustrates the difference between enabling and disabling the `ToolStripZoomPolicy`.

<table>
<thead>
<tr>
<th>Without ZoomPolicy</th>
<th>With ZoomPolicy</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image2" alt="ToolStrip" /></td>
<td><img src="image3" alt="ToolStrip" /></td>
</tr>
</tbody>
</table>

**TreeViewZoomPolicy**

If the following TreeView properties are not the default value when you zoom the TreeView, the `TreeView.Size` or `TreeView.Font` property value will not change. The `TreeViewZoomPolicy` will help you change these value in this case.

- `TreeView.Indent`
Note: In order to change the TreeNode.NodeFont property value, TreeViewZoomPolicy will enum all nodes of the tree. If the tree has a lot of nodes, it may cause a performance issue. To avoid this problem, you can change the "Enabled" property of TreeViewZoomPolicy to False.

**ZoomPolicy**

TouchToolKit for WinForms provides a group of Zoom Policies to give special support for C1 controls so that TouchToolKit for WinForms and ComponentOne controls can work together well. In this section you will learn about the policy for each ComponentOne Studio control.

**C1ChartZoomPolicy**

C1ChartZoomPolicy provides ability to change special properties of C1Chart, such as the following.

- Font
- Size
- ChartLabel
- ChartArea
- Axis
- PlotArea

So that, you can get the best zoom result when zooming C1Chart by using the C1ChartZoomPolicy.

The following table illustrates the difference between using and not using the C1ChartZoomPolicy when the zoomfactor property is set to 120%:

<table>
<thead>
<tr>
<th>Without ZoomPolicy</th>
<th>With ZoomPolicy</th>
</tr>
</thead>
</table>

**Setting the ChartZoomPolicy for ComponentOne Studio**

1. Select Add New Item from the Visual Studio Project menu.
2. Select Visual C# Items from the left pane of the Add New Item window.
3. Next, select the C1ChartZoomPolicy from the middle pane.
4. Click the Add button.
5. Add the C1Chart control to your form.
6. Build the project.
7. Add the C1Zoom, C1ZoomPanel or C1ApplicationZoom component to your form at design time.
8. Click the smart tag anchor of the component, and click Edit ZoomPolicies....
9. Click the Add Custom Policy button in the ZoomPolicy Collection Editor window.
   The C1ChartZoomPolicy is added to the Members list.

![ZoomPolicy Collection Editor](image)

**Chart3DZoomPolicy**

The C1Chart3DZoomPolicy provides ability to support C1Chart3D's zooming behavior for optimal zooming.

The following image illustrates the Chart3D's initial zoomfactor at 100%:
The following table illustrates the difference between adding and not adding the `C1Chart3DZoomPolicy`.

In this example the zoomfactor is set to 160%.

<table>
<thead>
<tr>
<th>Without ZoomPolicy</th>
<th>With ZoomPolicy</th>
</tr>
</thead>
</table>

Setting the `Chart3DZoomPolicy` for ComponentOne Studio

1. Select **Add New Item** from the Visual Studio Project menu.
2. Select Visual C# Items from the left pane of the **Add New Item** window.
3. Next, select a `ZoomPolicy` you want to add from the middle pane.
4. Click the **Add** button.

5. Add the C1Chart3D control to your form.

6. Build the project.

7. Add the **C1Zoom**, **C1ZoomPanel** or **C1ApplicationZoom** component to your form at design time.

8. Click the smart tag anchor of the component, and click **Edit ZoomPolicies**.

9. Click the **Add Custom Policy** button in the **ZoomPolicy Collection Editor** window.

The C1Chart3DZoomPolicy is added to the Members list.
If you change Font or Size property's default value of C1MainMenu, you can get the best zoom result by using the C1MainMenuZoomPolicy.

In C1MainMenu, initial zoomfactor is 100%, as shown below.

```
File  Edit  Search

New
Open
Close
Save
Save all
Print...
Exit
```

Set the zoomfactor to 120%. The following table illustrates the difference between adding and not adding the C1MainMenuZoomPolicy.

<table>
<thead>
<tr>
<th>Without ZoomPolicy</th>
<th>With ZoomPolicy</th>
</tr>
</thead>
</table>

Copyright © 2017 GrapeCity, inc. All rights reserved.
Setting the C1MainMenuZoomPolicy for ComponentOne Studio

1. Select Add New Item from the Visual Studio Project menu.
2. Select Visual C# Items from the left pane of the Add New Item window.
3. Next, select the C1CommandZoomPolicy from the middle pane.
4. Click the Add button.
5. Add the C1MainMenu control to your form.
6. Build the project.
7. Add the C1Zoom, C1ZoomPanel or C1ApplicationZoom component to your form at design time.
8. Click the smart tag anchor of the component, and click Edit ZoomPolicies....
9. Click the Add Custom Policy button in the ZoomPolicy Collection Editor window.

The C1MainMenuZoomPolicy is added to the Members list.

C1DockingTabZoomPolicy

When you change Font or Size property’s default value of C1DockingTab, you can get the best zoom result by using C1DockingTabZoomPolicy.

Setting the C1DockingTabZoomPolicy for ComponentOne Studio

1. Select Add New Item from the Visual Studio Project menu.
2. Select Visual C# Items from the left pane of the Add New Item window.
3. Next, select the C1CommandZoomPolicy from the middle pane.
4. Click the Add button.
5. Add the **C1DockingTab** control to your form.

6. Build the project.

7. Add the **C1Zoom, C1ZoomPanel** or **C1ApplicationZoom** component to your form at design time.

8. Click the smart tag anchor of the component, and click **Edit ZoomPolicies...**

9. Click the **Add Custom Policy** button in the **ZoomPolicy Collection Editor** window.

The **C1DockingTabZoomPolicy** is added to the Members list.

## C1NavBarZoomPolicy

C1NavBarZoomPolicy provides ability to support C1NavBar's zooming behavior.

In C1NavBar, Initial zoomfactor is 100%, as shown below.

![Mail](Mail.png)

Set zoomfactor to 130%, the following table can explain the differences between adding and not adding the **C1NavBarZoomPolicy**.

<table>
<thead>
<tr>
<th>Without ZoomPolicy</th>
<th>With ZoomPolicy</th>
</tr>
</thead>
</table>
Setting the C1NavBarZoomPolicy for ComponentOne Studio

1. Select **Add New Item** from the Visual Studio Project menu.
2. Select **Visual C# Items** from the left pane of the **Add New Item** window.
3. Next, select the **C1CommandZoomPolicy** from the middle pane.
4. Click the **Add** button.
5. Add the **C1NavBar** control to your form.
6. Build the project.
7. Add the **C1Zoom, C1ZoomPanel** or **C1ApplicationZoom** component to the component tray design time.
8. Click the smart tag anchor of the component, and click **Edit ZoomPolicies**....
9. Click the **Add Custom Policy** button in the **ZoomPolicy Collection Editor** window.

The **C1NavBarZoomPolicy** is added to the Members list.

**C1OutBarZoomPolicy**

If you change Font or Size property’s default value of C1OutBar, you can get the best zoom result by using **C1OutBarZoomPolicy**.

In C1OutBar, Initial zoomfactor is 100%, screenshot is as below.
The following table illustrates the differences between adding and not adding the C1OutBarZoomPolicy. The ZoomFactor property is set to 130%.

<table>
<thead>
<tr>
<th>Without ZoomPolicy</th>
<th>With ZoomPolicy</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Without ZoomPolicy" /></td>
<td><img src="image2.png" alt="With ZoomPolicy" /></td>
</tr>
</tbody>
</table>

**Setting the C1OutBarZoomPolicy for ComponentOne Studio**

1. Select **Add New Item** from the Visual Studio Project menu.
2. Select **Visual C# Items** from the left pane of the **Add New Item** window.
3. Next, select the **C1CommandZoomPolicy** from the middle pane.
4. Click the **Add** button.
5. Add the **C1OutBar** control to your form.
6. Build the project.
7. Add the **C1Zoom**, **C1ZoomPanel** or **C1ApplicationZoom** component to the component tray design time.
8. Click the smart tag anchor of the component, and click Edit ZoomPolicies.

9. Click the Add Custom Policy button in the ZoomPolicy Collection Editor window.

The C1OutBarZoomPolicy is added to the Members list.

**C1FlexGridZoomPolicy**

If you change the values of the C1FlexGrid.Column.Width, C1FlexGrid.Row.Height, C1FlexGrid.Font, and C1FlexGrid.Size properties, adding the C1FlexGridZoomPolicy in your application can help you get a better zoom result.

Initial zoom factor is 100%, screenshot is as below.

The following table illustrates the difference between adding and not adding the C1FlexGridZoomPolicy. The ZoomFactor is set to 120%.
Setting the C1FlexGridZoomPolicy for ComponentOne Studio

1. Select Add New Item from the Visual Studio Project menu.
2. Select Visual C# Items from the left pane of the Add New Item window.
3. Next, select the C1FlexGridZoomPolicy from the middle pane
4. Click the Add button.
5. Add the C1FlexGrid control to your form.
6. Build the project.
7. Select the C1Zoom, C1ZoomPanel or C1ApplicationZoom component at design time.
8. Click the smart tag anchor of the component, and click Edit ZoomPolicies....
9. Click the Add Custom Policy button in the ZoomPolicy Collection Editor window.
The C1FlexGridZoomPolicy is added to the Members list.

C1CalendarZoomPolicy

If you set the C1Calendar.Theme.BaseFont, you can add the C1CalendarZoomPolicy to get the best zoom result when you zoom C1Calendar.

The Initial zoomfactor is 100% as shown below:

Set zoomfactor to 120%. The following table illustrates the difference between adding and not adding the C1CalendarZoomPolicy:

<table>
<thead>
<tr>
<th>Without ZoomPolicy</th>
<th>With ZoomPolicy</th>
</tr>
</thead>
</table>

Setting the C1CalendarZoomPolicy

1. Add the C1Calendar control to your form.
2. Select Add New Item from the Visual Studio Project menu.
3. Next, select the C1ScheduleZoomPolicy from the middle pane
4. Click the Add button.
5. Build the project.
6. Select the C1Zoom, C1ZoomPanel or C1ApplicationZoom component at design time.
7. Click the smart tag anchor of the component, and click Edit ZoomPolicies....
8. Click the Add Custom Policy button in the ZoomPolicy Collection Editor window.

The C1CalendarZoomPolicy is added to the Members list.

C1ScheduleZoomPolicy

When you set the C1Schedule.Theme.TimeRulerMinutesFont, you can get the best zoom result by using the C1ScheduleZoomPolicy.

C1Schedule’s initial zoomfactor is 100% as shown below:
The following table illustrates the ZoomFactor set at 120% for C1Schedule. The font size for the TimeRuler minutes font is set to 6 pixels. Notice the difference in the font when using and not using the \texttt{C1ScheduleZoomPolicy}.

<table>
<thead>
<tr>
<th>Without ZoomPolicy</th>
<th>With ZoomPolicy</th>
</tr>
</thead>
</table>

\begin{verbatim}

Setting the C1ScheduleZoomPolicy for ComponentOne Studio

1. Add the \texttt{C1Scheduler} control to your form.

\end{verbatim}
1. Select **Add New Item** from the Visual Studio Project menu.
2. Select **Visual C# Items** from the left pane of the **Add New Item** window.
3. Next, select the **C1ScheduleZoomPolicy** from the middle pane.
4. Click the **Add** button.
5. Build the project.
6. Select the **C1Zoom, C1ZoomPanel** or **C1ApplicationZoom** component at design time.
7. Click the smart tag anchor of the component, and click **Edit ZoomPolicies...**
8. Click the **Add Custom Policy** button in the **ZoomPolicy Collection Editor** window.

The **C1ScheduleZoomPolicy** is added to the Members list.

### C1PrintPreviewControlZoomPolicy

If you change the value of the **C1PrintPreviewControl.Font** property, add the **C1ReportZoomPolicy** to make sure that every element in the C1PrintPreviewControl zooms correctly.

The initial zoomfactor is 100%, as shown below:

![C1PrintPreviewControl Without ZoomPolicy and With ZoomPolicy](image)

To see the difference between using and not using the C1ReportZoomPolicy, complete the following:

1. Set the **ZoomFactor** property to 125%
2. Run the project to see the **C1PrintPreview** control zoomed using the **C1PrintPreviewControlZoomPolicy**.
3. At design time set the **C1PrintPreviewControlZoomPolicy** to disable and run the project.

The following table illustrates the ZoomFactor set at 125% for C1PrintPreview and illustrates the difference between adding and not adding the **C1PrintPreviewControlZoomPolicy**.

<table>
<thead>
<tr>
<th>Without ZoomPolicy</th>
<th>With ZoomPolicy</th>
</tr>
</thead>
</table>

Copyright © 2017 GrapeCity, inc. All rights reserved.
Setting the C1PrintPreviewControlZoomPolicy

1. Add the C1PrintPreviewControl control to your form.
2. Select Add New Item from the Visual Studio Project menu.
3. Select Visual C# Items from the left pane of the Add New Item window.
4. Next, select the C1ReportZoomPolicy from the middle pane.
5. Click the Add button.
6. Build the project.
7. Select the C1Zoom, C1ZoomPanel or C1ApplicationZoom component at design time.
8. Click the smart tag anchor of the component, and click Edit ZoomPolicies....
9. Click the Add Custom Policy button in the ZoomPolicy Collection Editor window.

The C1PrintPreviewControlZoomPolicy is added to the Members list.

C1RibbonZoomPolicy

C1RibbonZoomPolicy supports C1Ribbon and C1StatusBar’s zooming behavior.

When you zoom C1Ribbon, its Font can zoom, but its size can’t. As a result, C1Ribbon’s Font can’t be shown completely after zoom. TouchToolKit for Windows Forms provides the C1RibbonZoomPolicy to make sure that C1Ribbon’s Font and Size don’t change so they always display the original Font and Size when zooming.

Setting the C1RibbonZoomPolicy for ComponentOne Studio

1. Add the C1Ribbon control to your form.
2. Select Add New Item from the Visual Studio Project menu.
3. Select Visual C# Items from the left pane of the Add New Item window.
4. Next, select the C1RibbonZoomPolicy from the middle pane.
5. Click the Add button.
6. Build the project.
7. Select the C1Zoom, C1ZoomPanel or C1ApplicationZoom component at design time.
8. Click the smart tag anchor of the component, and click Edit ZoomPolicies...
9. Click the Add Custom Policy button in the ZoomPolicy Collection Editor window.

The C1RibbonZoomPolicy is added to the Members list.

### C1TrueDBGridZoomPolicy

C1TrueDBGridZoomPolicy changes C1TrueDBGrid.CellTipsWidth, C1TrueDBGrid.DefColWidth, C1TrueDBGrid.RecordSelectorWidth, or some other properties, so you can get the best zoom result.

Initial zoomfactor is 100%, part screenshot is as below.

<table>
<thead>
<tr>
<th>Country</th>
<th>City</th>
<th>CustomerID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argentina</td>
<td>Buenos Aires</td>
<td>RANCH</td>
</tr>
<tr>
<td>Austria</td>
<td>Salzburg</td>
<td>PICCO</td>
</tr>
<tr>
<td>Brazil</td>
<td>São Paulo</td>
<td>QUEEN</td>
</tr>
</tbody>
</table>

The following table illustrates the difference between adding and not adding the C1TrueDBGridZoomPolicy.

<table>
<thead>
<tr>
<th>Without ZoomPolicy</th>
<th>With ZoomPolicy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Print Preview</td>
<td>Print Preview</td>
</tr>
<tr>
<td>Export To...</td>
<td>Export To...</td>
</tr>
<tr>
<td>Argentina</td>
<td>Buenos Aires</td>
</tr>
<tr>
<td>Austria</td>
<td>Salzburg</td>
</tr>
<tr>
<td>Brazil</td>
<td>São Paulo</td>
</tr>
</tbody>
</table>

### Setting the C1TrueDBGridZoomPolicy for ComponentOne Studio

1. Add the C1TrueDBGrid control to your form.
2. Select Add New Item from the Visual Studio Project menu.
3. Select Visual C# Items from the left pane of the Add New Item window.
4. Next, select the C1TrueDBGridZoomPolicy from the middle pane.
5. Click the Add button.
6. Build the project.
7. Select the C1Zoom, C1ZoomPanel or C1ApplicationZoom component at design time.
8. Click the smart tag anchor of the component, and click Edit ZoomPolicies....
9. Click the Add Custom Policy button in the ZoomPolicy Collection Editor window.
   The C1TrueDBGridZoomPolicy is added to the Members list.

**VisualBasicPowerPacks**

Visual Basic Power Packs is a Visual Basic 6.0 compatibility feature provided by Microsoft. TouchToolKit for WinForms provides the VisualBasicPowerPacksZoomPolicy to support ShapeContainer’s zooming behavior.

The following images shows the initial zoomfactor set to 100%:

The following table illustrates the difference between adding and not adding the VisualBasicPowerPacksZoomPolicy. Note that the ZoomFactor is set to 130%.

<table>
<thead>
<tr>
<th>Without ZoomPolicy</th>
<th>With ZoomPolicy</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Without ZoomPolicy Image" /></td>
<td><img src="image2.png" alt="With ZoomPolicy Image" /></td>
</tr>
</tbody>
</table>

**How to Use VisualBasicPowerPacksZoomPolicy**
To add the VisualBasicPowerPacksZoomPolicy to your project, complete the following:

1. In Visual Studio, click **Project| Add New Item**.
2. In the left pane of the **Add New Item** dialog box, select **Visual C# Items**.
3. Next, select **VisualBasicPowerPacksZoomPolicy** from the middle pane.
4. Click the **Add** button.
5. Build the project.
6. Select C1Zoom, C1ZoomPanel or C1ApplicationZoom component in design time.
7. Click a Smart Tag anchor of the component, and Click **Edit ZoomPolicies...**
8. Click **Add Custom Policy** drop down button in the **ZoomPolicy Collection Editor** window, ShapeContainerZoomPolicy will be added.
WPF Interoperability

WPF Interoperability is a technology of using Windows Presentation Foundation (WPF) in a Windows Forms project. TouchToolKit for WinForms provides WPF_ElementHostZoomPolicy to support ElementHost’s zooming behavior.

How to Use WPF_ElementHostZoomPolicy

To use the zoom policy, complete the following:

1. In Visual Studio, click **Project| Add New Item**.
2. In the left pane of the **Add New Item** dialog box, select **Visual C# Items**.
3. Next, select **VisualBasicPowerPacksZoomPolicy** from the middle pane.
4. Click **Add** button.
5. Build the project.
6. Select **C1Zoom, C1ZoomPanel** or **C1ApplicationZoom** component in design time.
7. Click a Smart Tag anchor of the component, and click **Edit ZoomPolicies**.
8. Click **Add Custom Policy** drop down button in the **ZoomPolicy Collection Editor** window, **WPF_ElementHostZoomPolicy** will be added.
TileControlZoomPolicy

1. Select Add New Item from the Visual Studio Project menu.
2. Select Visual C# Items from the left pane of the Add New Item window.
3. Next, select the C1TileControlZoomPolicy from the middle pane.
4. Click the Add button.
5. Add the C1TileControl control to your form.
6. Build the project.
7. Add the C1Zoom, C1ZoomPanel or C1ApplicationZoom component to your form at design time.
8. Click the smart tag anchor of the component, and click Edit ZoomPolicies....
9. Click the Add Custom Policy button in the ZoomPolicy Collection Editor window.
10. The C1TileControlZoomPolicy is added to the Members list.

C1SplitContainerZoomPolicy

1. Select Add New Item from the Visual Studio Project menu.
2. Select Visual C# Items from the left pane of the Add New Item window.
3. Next, select the C1SplitContainerZoomPolicy from the middle pane.
4. Click the Add button.
5. Add the C1SplitContainer control to your form.
6. Build the project.
7. Add the C1Zoom, C1ZoomPanel or C1ApplicationZoom component to your form at design time.
8. Click the smart tag anchor of the component, and click Edit ZoomPolicies....
9. Click the Add Custom Policy button in the ZoomPolicy Collection Editor window.
10. The C1SplitContainerZoomPolicy is added to the Members list.

TabControlZoomPolicy
1. Select **Add New Item** from the Visual Studio Project menu.
2. Select **Visual C# Items** from the left pane of the **Add New Item** window.
3. Next, select the **C1TabControlZoomPolicy** from the middle pane.
4. Click the **Add** button.
5. Add the **C1TabControl** control to your form.
6. Build the project.
7. Add the **C1Zoom, C1ZoomPanel** or **C1ApplicationZoom** component to your form at design time.
8. Click the smart tag anchor of the component, and click **Edit ZoomPolicies....**
9. Click the **Add Custom Policy** button in the **ZoomPolicy Collection Editor** window.
10. The **C1TabControlZoomPolicy** is added to the Members list.

**C1GanttViewZoomPolicy**

1. Select **Add New Item** from the Visual Studio Project menu.
2. Select **Visual C# Items** from the left pane of the **Add New Item** window.
3. Next, select the **C1GanttViewZoomPolicy** from the middle pane.
4. Click the **Add** button.
5. Add the **C1GanttView** control to your form.
6. Build the project.
7. Add the **C1Zoom, C1ZoomPanel** or **C1ApplicationZoom** component to your form at design time.
8. Click the smart tag anchor of the component, and click **Edit ZoomPolicies....**
9. Click the **Add Custom Policy** button in the **ZoomPolicy Collection Editor** window.
10. The **C1GanttViewZoomPolicy** is added to the Members list.

**C1GaugeZoomPolicy**

1. Select **Add New Item** from the Visual Studio Project menu.
2. Select **Visual C# Items** from the left pane of the **Add New Item** window.
3. Next, select the **C1GaugeZoomPolicy** from the middle pane.
4. Click the **Add** button.
5. Add the **C1Gauge** control to your form.
6. Build the project.
7. Add the **C1Zoom, C1ZoomPanel** or **C1ApplicationZoom** component to your form at design time.
8. Click the smart tag anchor of the component, and click **Edit ZoomPolicies....**
9. Click the **Add Custom Policy** button in the **ZoomPolicy Collection Editor** window.
10. The **C1GaugeZoomPolicy** is added to the Members list.