ComponentOne

OrgChart for UWP

ComponentOne, a division of GrapeCity
201 South Highland Avenue, Third Floor
Pittsburgh, PA 15206 USA

Website: http://www.componentone.com
Sales: sales@componentone.com
Telephone: 1.800.858.2739 or 1.412.681.4343 (Pittsburgh, PA USA Office)

Trademarks

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

Table of Contents

OrgChart for UWP

Create hierarchical diagrams that show the structure and relationships of your data. **OrgChart for UWP** leverages the rich data binding mechanisms of the platform to provide a flexible, yet easy-to-use control.

Help with UWP Edition

Getting Started

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

Key Features

OrgChart for UWP allows you to create customized, rich applications. Make the most of **OrgChart for UWP** by taking advantage of the following key features:

- **Flexible Data Binding**

  The C1OrgChart control is an **ItemsControl**. Bind it to a single entity containing sub-items or an **IEnumerable** collection of items which each can contain sub-items.

- **Versatile Orientation and Flow**

  C1OrgChart can display items flowing horizontally or vertically in either direction. You can determine the flow of the chart by simply setting the C1OrgChart.Orientation and FlowDirection properties on the control.

- **Collapsible Nodes**

  Allow the user to hide an item's branches to create a more compact display. The C1OrgChart nodes have an C1OrgChart.IsCollapsed property that allows you to collapse or expand each node, similar to a **TreeView**.

- **Connector Line Customization**

  The C1OrgChart control exposes several properties that allow you to customize the lines used to connect nodes. These properties allow you to customize the brush, thickness and dash array used to create connector lines. You can even bind these properties to properties on the data item to customize the lines per relationship.

- **Child Spacing and Alignment Options**

  Customize the alignment and spacing of items in your **OrgChart** by simply setting a few properties. The control includes a C1OrgChart.ChildSpacing property that controls the separation between items (in pixels), as well as horizontal and vertical alignment properties which can give dramatic changes to the visualization.

- **Complex Hierarchical Displays**

  In addition to the flexibility provided by data templates and bindings, the C1OrgChart control also supports advanced binding scenarios to create complex hierarchical displays. Use different templates for certain nodes based upon properties of specific data items like conditional formatting. For example, you could use different templates to visually distinguish among directors, managers, and clerical staff in an employee organization chart.

OrgChart for UWP Quick Start

The following quick start guide is intended to get you up and running with **OrgChart for UWP**. In this quick start you'll create a simple project using a C1OrgChart control. You'll create a new Universal Windows application, add the C1OrgChart control to your application, and add data that will be displayed in the C1OrgChart control.

Step 1 of 3: Creating the C1OrgChart Application

In this step you'll create a Universal Windows application using **OrgChart for UWP**. C1OrgChart allows you to create hierarchical diagrams that show the structure and relationships of your data. To set up your project and add a C1OrgChart control to your application, complete the following steps:

1. In Visual Studio select **File | New | Project**.
2. Select Templates | Visual C# | Windows | Universal. From the templates list, select Blank App (Universal Windows).
3. Enter a **Name** and click **OK** to create your project. Open the XAML view of the MainPage.xaml file; in this quick start you'll add controls using XAML markup.
4. Navigate to the Toolbox and double-click the C1OrgChart icon. This will add the control to your application. This will add the reference and XAML namespace automatically.
5. Add the following to the <OrgChart:C1OrgChart> tag:

   - Name="c1OrgChart1"
   - ConnectorThickness="2"
   - ConnectorStroke="Black"
   - Orientation="Vertical"
   - ChildSpacing="20, 30"

   The XAML markup should resemble the following:

   Markup

   ```
   <OrgChart:C1OrgChart Name="c1OrgChart1" ConnectorThickness="2"
   ConnectorStroke="Black" Orientation="Vertical" ChildSpacing="20,30"
   VerticalAlignment="Center" HorizontalAlignment="Center" >
   ```

6. Place the following C1OrgChart.ItemTemplate between the <OrgChart:C1OrgChart> and </OrgChart:C1OrgChart> tags:

   Markup

   ```
   <OrgChart:C1OrgChart.ItemTemplate>
      <DataTemplate>
          <Border Background="#FF6AD400" Width="180" Height="90">
             <TextBlock Text="{Binding Name}" FontSize="20"
   HorizontalAlignment="Center" VerticalAlignment="Center" Foreground="White"/>
          </Border>
      </DataTemplate>
   </OrgChart:C1OrgChart.ItemTemplate>
   ```

You've successfully set up your application's user interface, but the C1OrgChart control currently contains no content. In the next step you'll add content to the C1OrgChart control.

Step 2 of 3: Adding Content to the C1Org Chart Control

In the previous step you created a Universal Windows application and added the C1OrgChart control to your project. In this step you'll add content to the C1OrgChart control.

To customize your project and add content to the C1OrgChart control in your application, complete the following steps:

1. In the Solution Explorer, right-click the MainPage.xaml file and select **View Code**. The code file will open.
2. Add the following imports statements to the top of the page:

Visual Basic

```
Imports C1.Xaml.OrgChart
```

C#

```
using C1.Xaml.OrgChart;
```

3. This code will add content to the application. Add the following code within the page constructor, directly below the **InitializeComponent()** method:

Visual Basic

```
' create hierarchy
Dim uwp As New Platform() With {.Name = "UWP"}
Dim winjs As New Platform() With {.Name = "HTML"}
Dim xaml As New Platform() With {.Name = "XAML"}
Dim dx As New Platform() With {.Name = "DirectX"}
uwp.Subplatforms = New List(Of Platform)()
uwp.Subplatforms.Add(winjs)
uwp.Subplatforms.Add(xaml)
uwp.Subplatforms.Add(dx)
winjs.Subplatforms = New List(Of Platform)()
winjs.Subplatforms.Add(New Platform() With {.Name = "JavaScript"})
xaml.Subplatforms = New List(Of Platform)()
xaml.Subplatforms.Add(New Platform() With {.Name = "C#"})
xaml.Subplatforms.Add(New Platform() With {.Name = "VB"})
dx.Subplatforms = New List(Of Platform)()
dx.Subplatforms.Add(New Platform() With {.Name = "C++"})
' set to orgchart
c1OrgChart1.Header = uwp
```

C#

```
// create hierarchy
            Platform uwp = new Platform() { Name = "Windows RT" };
            Platform winjs = new Platform() { Name = "HTML" };
            Platform xaml = new Platform() { Name = "XAML" };
            Platform dx = new Platform() { Name = "DirectX" };
            uwp.Subplatforms = new List<Platform>();
            uwp.Subplatforms.Add(winjs);
            uwp.Subplatforms.Add(xaml);
            uwp.Subplatforms.Add(dx);
            winjs.Subplatforms = new List<Platform>();
```

```
            winjs.Subplatforms.Add(new Platform() { Name = "JavaScript" });
            xaml.Subplatforms = new List<Platform>();
            xaml.Subplatforms.Add(new Platform() { Name = "C#" });
            xaml.Subplatforms.Add(new Platform() { Name = "VB" });
            dx.Subplatforms = new List<Platform>();
            dx.Subplatforms.Add(new Platform() { Name = "C++" });
            // set to orgchart
            c1OrgChart1.Header = uwp;
        }
    }
```

4. Add the following class below the page constructor:

Visual Basic

```
Public Class Platform
        Public Property Name() As String
        Public Property Subplatforms() As IList(Of Platform)
    End Class
```

C#

```
public class Platform
    {
        public string Name { get; set; }
        public IList<Platform> Subplatforms { get; set; }
    }
}
```

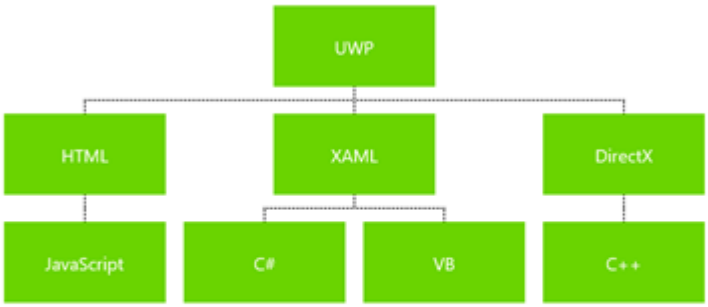
In this step you added content to the C1OrgChart control. In the next step you'll run your application.

Step 3 of 3: Running the C1OrgChart Application

Now that you've created a Universal Windows application and added content to the C1OrgChart control, the only thing left to do is run your application.

- From the **Debug** menu, select **Start Debugging** to view how your application will appear at run time.

The application will appear similar to the following:



Congratulations!

You've completed the **OrgChart for UWP** quick start, created a simple Universal Windows application and added and customized an **OrgChart for UWP** control.

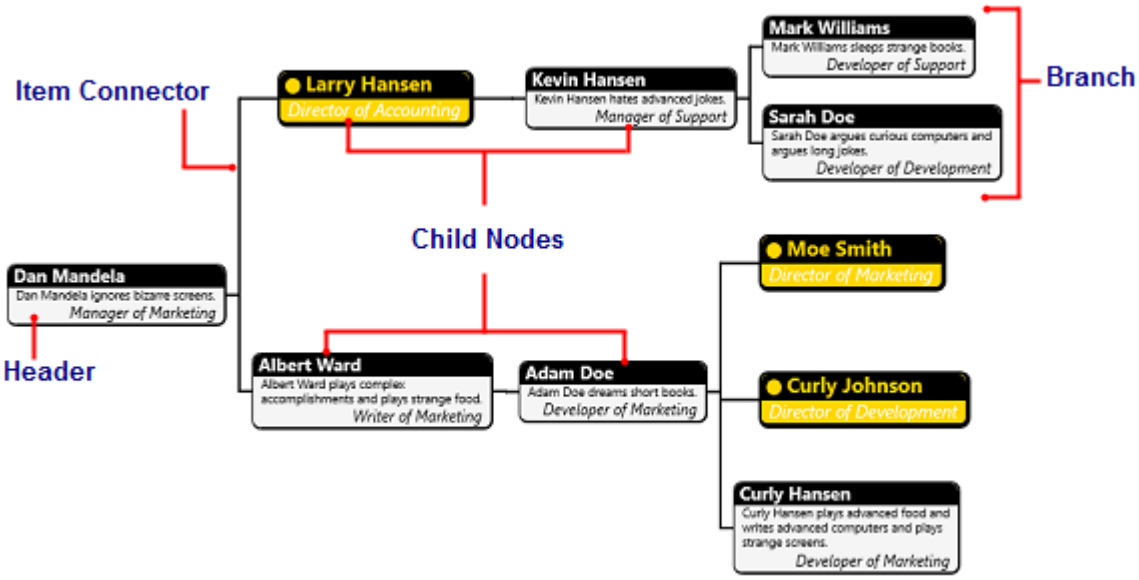
Working with OrgChart for UWP

Organizational charts (often called an organization chart, an org chart, or an organogram) are diagrams that show the structure of an organization and the relationships and relative ranks of its parts and positions/jobs. The term is also used for similar diagrams, for example ones showing the different elements of a field of knowledge or a group of languages.

The C1OrgChart control allows you to create organizational charts that show any type of hierarchical data. The control leverages the rich data binding mechanisms in UWP to provide a flexible, yet easy-to-use tool.

OrgChart Elements

The C1OrgChart control consists of several parts: the Header, the ChildNodes, and the **ItemConnector**. The image below identifies those parts. The image also identifies a Branch of the C1OrgChart:



C1OrgChart Core Properties

The C1OrgChart control is an **ItemsControl**. To use it, you will normally populate the control with the Header or ItemsSource properties and define the appearance of the items using the ItemTemplate property.

Use the Header property if you have a single data item that contains sub-items. Use the **ItemsSource** property if you have a collection of items with sub-items.

Either way, the data items must contain sub-items. In most cases, the sub-items are of the same type as the main items. For example, a **Person** class may contain properties about the person, and the **Subordinates** property may contains a list of employees who report to the parent **Person**:

C#

```
public class Person
    {
      ObservableCollection<Person> _list = new    ObservableCollection<Person>();
```

```
        #region ** object model

        public string Name { get; set; }
        public string Position { get; set; }
        public string Notes { get; set; }
        public IList<Person> Subordinates
        {
            get { return _list; }
        }
```

If you assign a **Person** object to the Header property, the C1OrgChart will automatically detect that the **Subordinates** property contains a collection of **Person** objects, and that is enough to establish the hierarchy of the data.

If your data class contains multiple collection properties, or if the collection is of a generic type (for example **IEnumerable**), then you should use the ChildItemsPath property to specify the name of the property that contains the child (subordinate) items.

If the items contain sub-items of different types, then you should use a HierarchicalDataTemplate to specify the items at each level. This is discussed later in this document.

The ItemTemplate property specifies how the C1OrgChart control should display the data items. This is a standard **DataTemplate**, which you can define in XAML as follows:

Markup

```
<Page.Resources>
    <local:PersonTemplateSelector x:Key="_personTplSelector">
      <local:PersonTemplateSelector.DirectorTemplate>
        <!-- data template for Directors -->
        <DataTemplate>
          <Border Background="Gold" BorderBrush="Black" BorderThickness="2 2 4 4"
CornerRadius="6" Margin="20" MaxWidth="200">
            <StackPanel Orientation="Vertical">
              <Border CornerRadius="6 6 0 0" Background="Black">
                <StackPanel Orientation="Horizontal">
                <Ellipse Width="12" Height="12" Fill="Gold" Margin="4" />
                <TextBlock Text="{Binding Name}" FontWeight="Bold" FontSize="16"
Foreground="Gold" />
                </StackPanel>
              </Border>
                <TextBlock Text="{Binding Position}" Padding="6 0" FontSize="14"
FontStyle="Italic" HorizontalAlignment="Right" />
            </StackPanel>
          </Border>
        </DataTemplate>
      </local:PersonTemplateSelector.DirectorTemplate>
    </local:PersonTemplateSelector>
 </Page.Resources>
```

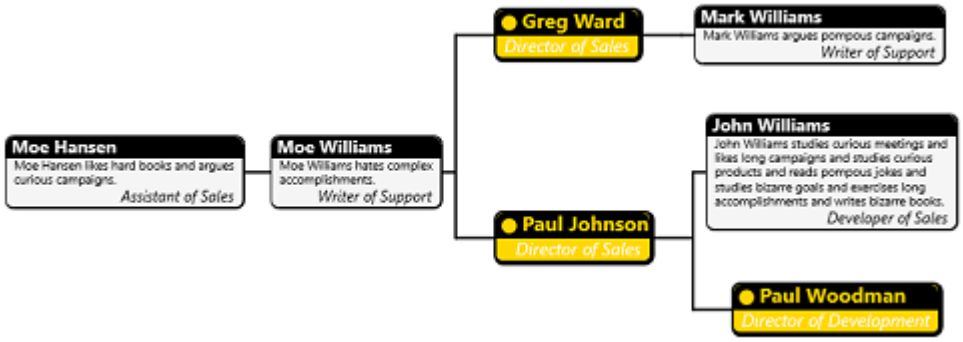
Once you have a **DataTemplate** defined as a resource, you can use it in a **C1OrgChart** control as follows:

Markup

```
<OrgChart:C1OrgChart Name="_orgChart" Grid.Row="1" ConnectorThickness="2"
ItemTemplateSelector="{StaticResource _personTplSelector}" Orientation="Horizontal">
```

```
</OrgChart:C1OrgChart>
```

To illustrate, the chart below was created with a slightly enhanced version of this template and some randomly-generated employees:



Using Bindings in C1OrgChart Properties

The **ItemTemplate** used in the examples above use bindings to show properties of the **Employee** class as visual elements. But you can also bind elements to properties of the C1OrgChart.

The most useful of these scenarios is binding a **CheckBox.IsChecked** property to the **C1OrgChart's** IsCollapsed property. This allows you to create collapsible **C1OrgCharts** that behave similarly to a **TreeView** control.

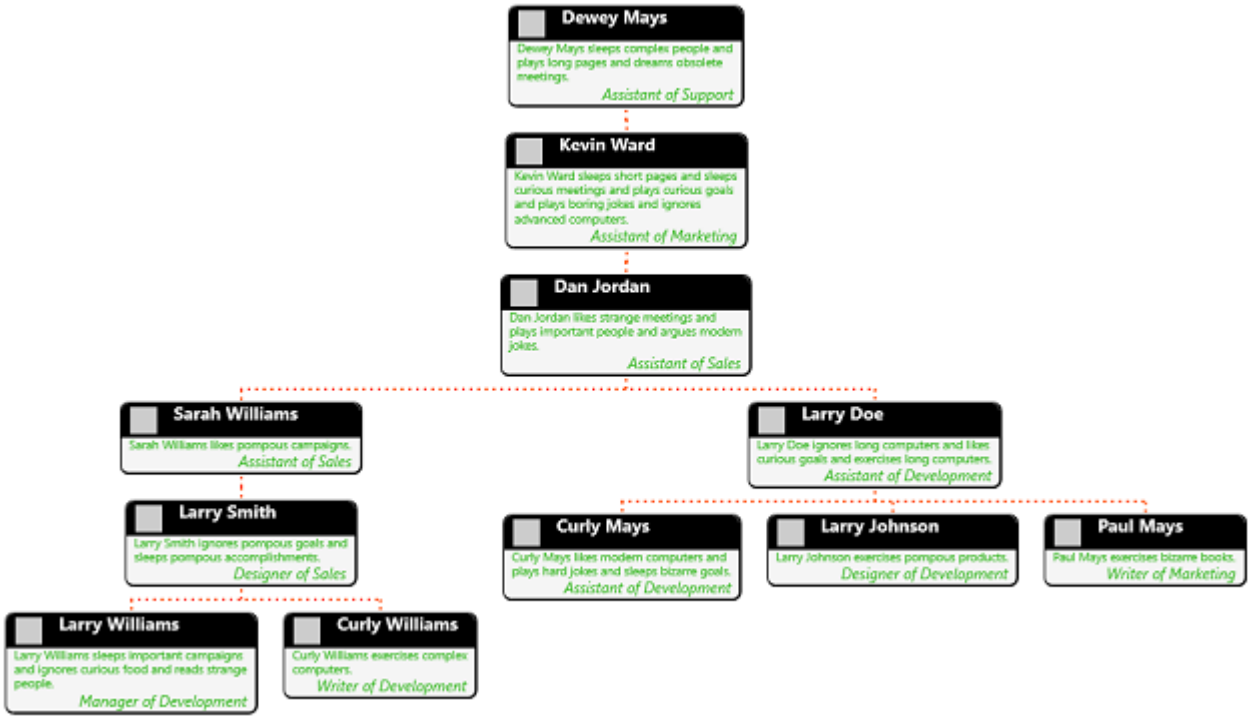
For example, here is a data template we assigned to the **C1OrgChart's** ItemTemplate property:

Markup

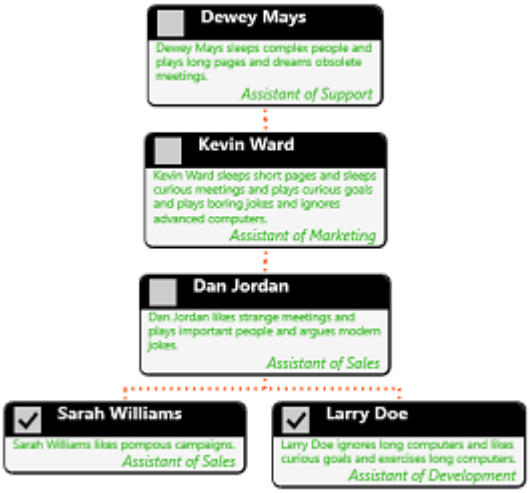
```
<orgchart:C1OrgChart.ItemTemplate>
   <DataTemplate>
   <!-- outer border -->
     <Border
       Background="WhiteSmoke" BorderBrush="Black"
       BorderThickness="1 1 2 2" CornerRadius="6"
       MaxWidth="200" >
        <StackPanel Orientation="Vertical" >
        <!-- item header -->
          <Border CornerRadius="6 6 0 0" Background="Black" >
            <StackPanel Orientation="Horizontal">
           <!-- bind CheckBox to containing C1OrgChart's IsCollapsed property -->
               <CheckBox Margin="4 0" Checked="CheckBox_Checked"
Unchecked="CheckBox_Unchecked"/>
               <!-- item header: person's Name -->
               <TextBlock Text="{Binding Name}" FontWeight="Bold" FontSize="14"
Foreground="WhiteSmoke" Padding="4 0 0 0" />
            </StackPanel>
          </Border>
          <!-- body: person's details -->
          <TextBlock Text="{Binding Notes}" Padding="6 0" FontSize="9.5"
TextWrapping="Wrap" />
          <TextBlock Text="{Binding Position}" Padding="6 0" FontSize="12"
FontStyle="Italic" HorizontalAlignment="Right" />
```

```
        </StackPanel>
      </Border>
    </DataTemplate>
  </orgchart:C1OrgChart.ItemTemplate>
```

The effect of this change is shown below:



Clicking the checkboxes on the Sarah Williams and Larry Doe nodes collapses the branches, resulting in this more compact display:



Advanced Binding Scenarios

In addition to the flexibility provided by data templates and bindings, the C1OrgChart control also supports advanced binding scenarios using two standard classes: DataTemplateSelector and HierarchicalDataTemplate.

DataTemplateSelector

The **DataTemplateSelector** class provides a way for you to select different templates based on the data object and the data-bound element. For example, you could use different templates to display directors, managers, and clerical staff.

To do this, you would create a custom class that inherits from **DataTemplateSelector** and override the **SelectTemplate** object. You would then create an instance of this class and assign it to the ItemTemplateSelector property of the C1OrgChart control.

The example below shows a simple **DataTemplateSelector** implementation:

C#

```
/// <summary>
/// Class used to select the templates for items being created.
/// </summary>
 public class PersonTemplateSelector : C1.Xaml.OrgChart.DataTemplateSelector
    {
      public override DataTemplate SelectTemplate(object item, DependencyObject
container)
        {
            var p = item as Person;


            if (p.Position.IndexOf("Director") > -1)
            {
                return DirectorTemplate;
            }
            else if (p.Position.IndexOf("Manager") > -1)
            {
                return ManagerTemplate;
            }
            else if (p.Position.IndexOf("Designer") > -1)
            {
                return DesignerTemplate;
            }
            else
            {
                return OtherTemplate;
            }
        }
```

Once the custom **DataTemplateSelector** is available, you can use it in XAML as usual:

Markup

```
<Page.Resources>
        <!-- TemplateSelector: picks _tplDirector or _tlpOther -->
        <local:PersonTemplateSelector x:Key="_personTplSelector">
```

```
            <local:PersonTemplateSelector.DirectorTemplate>
                <!-- data template for Directors -->
                <DataTemplate>
                    ...
                </DataTemplate>
            </local:PersonTemplateSelector.DirectorTemplate>
            <local:PersonTemplateSelector.ManagerTemplate>
                <!-- data template for managers -->
                <DataTemplate>
                    ...
                </DataTemplate>
            </local:PersonTemplateSelector.ManagerTemplate>
            <local:PersonTemplateSelector.DesignerTemplate>
                <!-- data template for designers -->
                <DataTemplate>
                    ...
                </DataTemplate>
            </local:PersonTemplateSelector.DesignerTemplate>
            <local:PersonTemplateSelector.OtherTemplate>
                <!-- data template for everyone else -->
                <DataTemplate>
                    ...
                </DataTemplate>
            </local:PersonTemplateSelector.OtherTemplate>
        </local:PersonTemplateSelector>
    </Page.Resources>
```

And bind to the Template in the <OrgChart:C1OrgChart> tag:

Markup

```
<OrgChart:C1OrgChart x:Name="c1OrgChart1"  Grid.Row="1" ItemTemplateSelector="
{StaticResource _personTplSelector}">
```

The image below shows the result:

ItemTemplateSelector

The **ItemTemplateSelector** is useful when the data items are of the same type, but you want to display certain items

differently based on the properties of specific data items.

HierarchicalDataTemplate class: This class allows you to bind the **C1OrgChart** control to items that contain items of different types. For example, you could create a chart that displays leagues, divisions within each league, and teams within each division.

To do this, you would create a **HierarchicalDataTemplate** for each of the classes with sub-items, and a regular data template for the last class in the hierarchy. In our example, you would create a **HierarchicalDataTemplate** for the leagues, another for the divisions, and finally a regular data template for the teams:

Markup

```
<Page.Resources>
        <!-- template for Team objects -->
        <DataTemplate x:Key="TeamTemplate" >
            <Border Background="LightBlue" Padding="4" >
                <TextBlock FontStyle="Italic" Text="{Binding Name}" />
            </Border>
        </DataTemplate>

        <!-- template for Division objects -->
        <Xaml:C1HierarchicalDataTemplate  x:Key="DivisionTemplate"
            ItemTemplate="{StaticResource TeamTemplate}" ItemsSource="{Binding
Teams}">
            <DataTemplate>
                <Border Background="Gold" >
                    <TextBlock Text="{Binding Name}" FontWeight="Bold"
HorizontalAlignment="Center" VerticalAlignment="Center" Padding="20" />
                </Border>
            </DataTemplate>
        </Xaml:C1HierarchicalDataTemplate >
        <!-- template for League objects -->
        <Xaml:C1HierarchicalDataTemplate  x:Key="LeagueTemplate"
            ItemTemplate="{StaticResource DivisionTemplate}" ItemsSource="{Binding
Divisions}">
            <DataTemplate>
                <Border Background="LightCoral" >
                    <TextBlock Text="{Binding Name}" FontWeight="Bold"
HorizontalAlignment="Center" VerticalAlignment="Center" Padding="40" />
                </Border>
            </DataTemplate>
        </Xaml:C1HierarchicalDataTemplate >
    </Page.Resources>
```

The top-level template is the **LeagueTemplate**. In addition to defining how **League** object should be displayed (as regular templates do), its **ItemsSource** property specifies that subordinate objects should be retrieved from the **League.Divisions** property. Finally, the **ItemTemplate** property specifies the template that should be used to display the subordinate objects.

In this case, the **ItemTemplate** is **DivisionTemplate**, another **HierarchicalDataTemplate** that specifies how **Division** objects should be displayed, that the **Division.Teams** property exposes subordinate objects, and that the subordinate objects should be displayed using the **TeamTemplate**, which is a regular (non-hierarchical) data template.

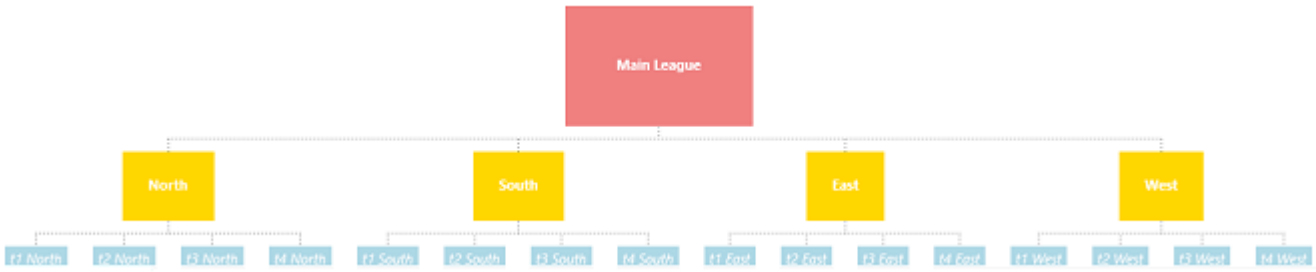
Notice the following important point:

- The **HierarchicalDataTemplate** class derives from the standard **DataTemplate** class and adds two properties: **ItemsSource** specifies the property that contains sub-items, and **ItemTemplate** specifies the template that should be used for the subordinate items.

Once the templates have been defined, using them is just a matter of setting the ItemTemplate property as usual:

Markup

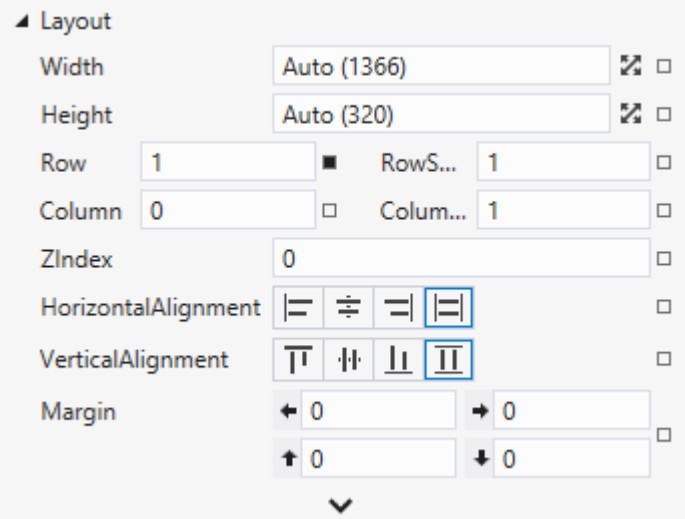
```
<OrgChart:C1OrgChart Name="_chart" ItemTemplate="{StaticResource LeagueTemplate}"
ConnectorDashArray="1 2" ConnectorStroke="Gray" HorizontalAlignment="Center"
VerticalAlignment="Center" />
```



Notice how the **C1OrgChart** uses the hierarchical data templates to navigate the hierarchy picking the right child collections and data templates.

Layout in a Panel

You can easily lay out the C1OrgChart and other controls in your application, using the attached layout properties. For example, you can lay out your control in a **Grid** panel with its **Row**, **ColumnSpan**, and **RowSpan** properties. For example, the C1OrgChart control includes the following **Layout** properties when located within a **Grid** panel:



You can change the sizing, alignment, and location of the C1OrgChart control within the **Grid** panel.

OrgChart for UWP Appearance Properties

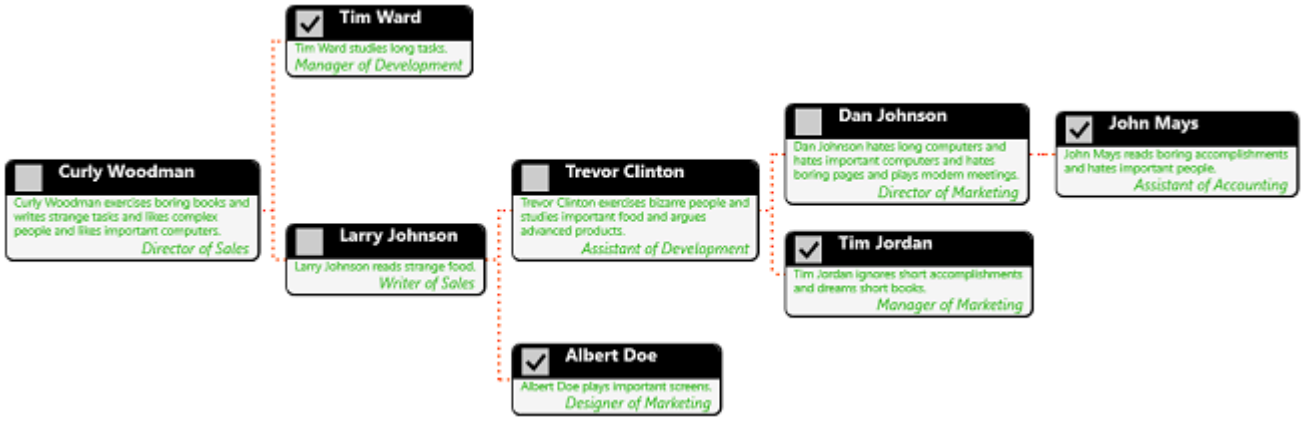
The ItemTemplate property gives you complete flexibility to specify the appearance of the nodes in the C1OrgChart. The C1OrgChart exposes several properties that allow you to customize the appearance of the chart itself.

Orientation

The Orientation property allows you to specify whether the chart should flow in the vertical or horizontal direction. By default a vertical **C1OrgChart** is displayed. Setting the **Orientation** property to **Horizontal** would have the following effect:

Markup

```
<OrgChart:C1OrgChart Name="_orgChart" Orientation="Horizontal" ItemTemplate="
{StaticResource EmployeeTemplate }" >
</OrgChart:C1OrgChart>
```

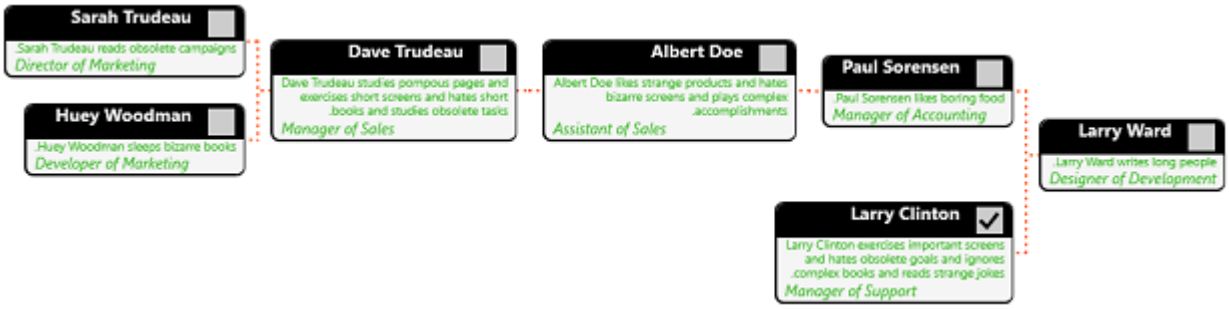


Flow Direction

The FlowDirection property allows you to specify whether the chart should flow right to left or left to right. By default, the C1OrgChart flows left to right. You can change the FlowDirection property to RightToLeft with the following markup:

Markup

```
<OrgChart:C1OrgChart Name="_orgChart" FlowDirection="RightToLeft" ItemTemplate="
{StaticResource EmployeeTemplate }" >
</OrgChart:C1OrgChart>
```



Child Spacing

ChildSpacing property: This property allows you to control the separation between items, in pixels. The default value is (20, 20), which spaces items by 20 pixels in the horizontal and vertical directions. For example, setting the ChildSpacing property to (20, 60) would have the following effect:

Markup

```
<OrgChart:C1OrgChart Name="_orgChart" Grid.Row="1" Orientation="Vertical"
ChildSpacing="20, 60" HorizontalAlignment="Center" VerticalAlignment="Center"
ConnectorStroke="OrangeRed" ConnectorThickness="2" ConnectorDashArray="1 2"
Foreground="#FF39B925" />
```

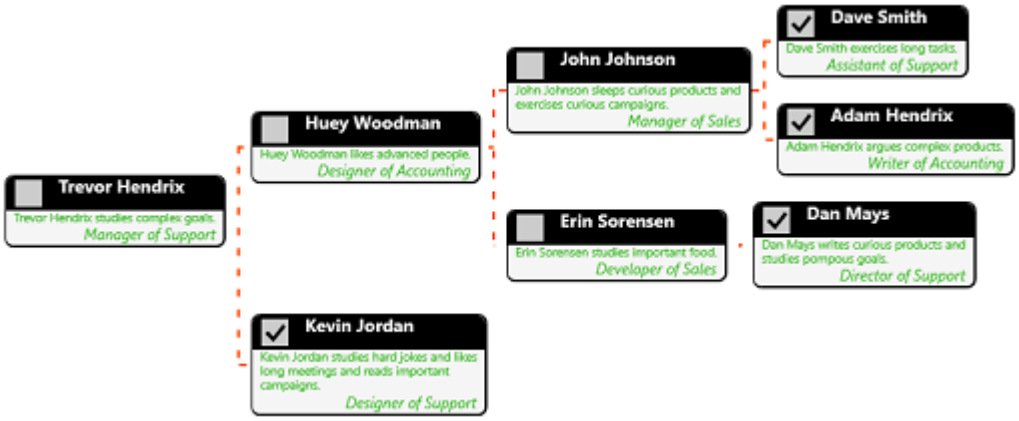
Connector

The C1OrgChart exposes several properties that allow you to customize the lines used to connect the nodes. These properties include: ConnectorStroke (specifies the Brush used to create the connectors), ConnectorThickness (the thickness of the lines), ConnectorDashArray (used to create dashed lines), and a few others. These properties are analogous to the ones in the Line element.
For example, if you wanted to connect the items using gray dotted lines you could use this XAML markup:

Markup

```
<OrgChart:C1OrgChart Name="_orgChart"
    ConnectorStroke="OrangeRed"
     ConnectorThickness="2"
     ConnectorDashArray="3 5"
     Foreground="#FF39B925" />
```

The ConnectorDashArray property uses a collection of double values that specify the widths of dashes and blank spaces, expressed in ConnectorThickness units.



Alignment

The HorizontalContentAlignment and VerticalContentAlignment properties allow you to customize the alignment of nodes within the chart. The default value is **Center** for both properties, so nodes are centered within the tree. The images below show the effect of the other possible settings:

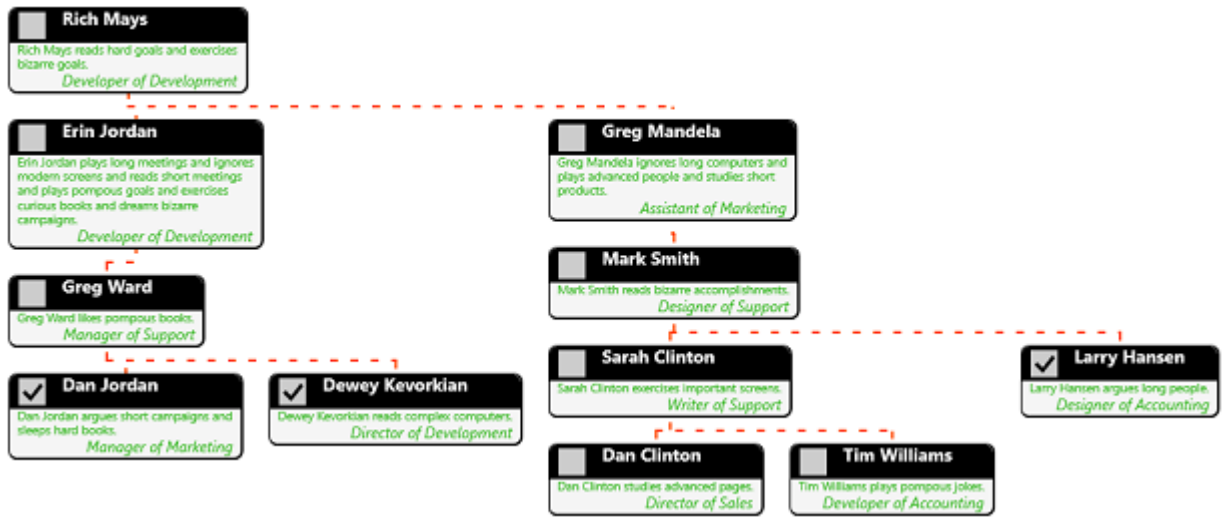
Left

If you set **HorizontalContentAlignment** to "Left":

Markup

```
<OrgChart:C1OrgChart
  Name="_orgChart"
  HorizontalContentAlignment="Left"
  ItemTemplate="{StaticResource EmployeeTemplate }" >
</OrgChart:C1OrgChart>
```

The **OrgChart** will appear as follows:



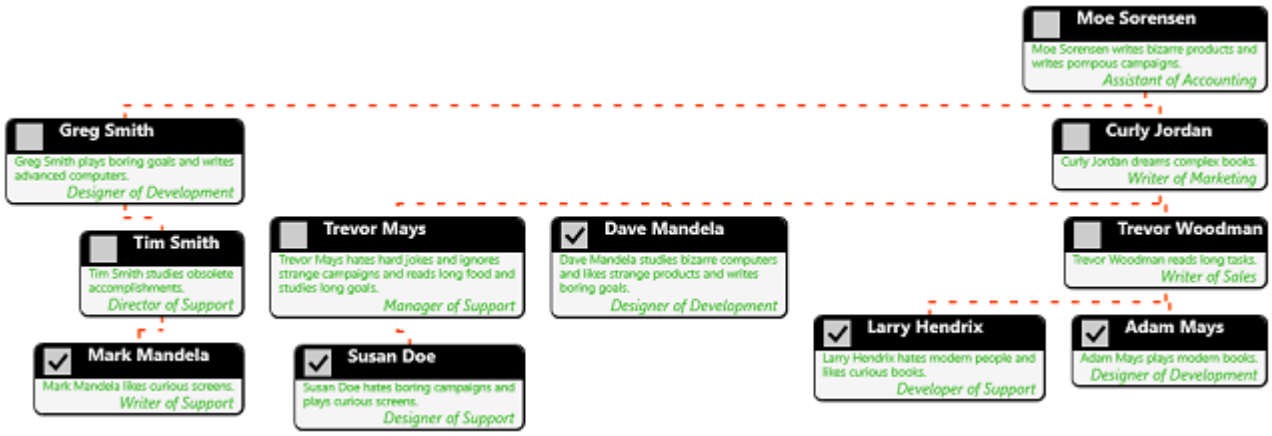
Right

If you set **HorizontalContentAlignment** to "Right":

Markup

```
<OrgChart:C1OrgChart
  Name="_orgChart"
  HorizontalContentAlignment="Right"
  ItemTemplate="{StaticResource EmployeeTemplate }" >
</OrgChart:C1OrgChart>
```

The **OrgChart** will appear as follows:



OrgChart for UWP Task- Based Help

The following task-based help topics assume that you are familiar with Visual Studio and know how to use the C1OrgChart control in general. If you are unfamiliar with the **OrgChart for UWP** product, please see the OrgChart for UWP Quick Start first.

Each topic in this section provides a solution for specific tasks using the **OrgChart for UWP** product. Most task-based help topics also assume that you have created a Universal Windows application and added a C1OrgChart control to the project – for information about creating the control, see Adding C1OrgChart to the Application.

Adding C1OrgChart to the Application

Complete the following steps to add a C1OrgChart control to your application:

1. In Visual Studio 2012 Select **File | New | Project**.
2. In the **New Project** dialog box, expand a language in the left pane, under the language select **Windows Store**, and in the templates list select **Blank App (XAML)**.
3. Enter a **Name** and click **OK** to create your project.
4. Navigate to the Toolbox and double-click the C1OrgChart icon. This will add the control to your application. This will add the reference and XAML namespace automatically.
5. Add the <OrgChart:C1OrgChart Name="C1OrgChart1" /> tag within the Grid tags on the page to add the C1OrgChart control to the application.
   The XAML will appear similar to the following:

   Markup

   ```
   <Grid Name="LayoutRoot" Background="White">
       <OrgChart:C1OrgChart Name="C1OrgChart1" />
   </Grid>
   ```

   This will add a C1OrgChart control named "C1OrgChart1" to the application. If you run the application now, you will see a blank page.

You've successfully set up your application's user interface, but if you run your application now you'll see that the C1OrgChart control currently contains no content.

> If the C1OrgChart control was installed to the Visual Studio Toolbox, simply double-clicking the icon or dragging the control onto a page will automatically perform the control-related steps above.

Changing C1OrgChart Orientation

C1OrgChart can be displayed with the chart flowing in either a horizontal and vertical direction. You can set the property either in XAML or in the Properties window.

In XAML

Insert Orientation=Vertical into the <OrgChart:C1OrgChart> opening tag. The XAML markup for the C1OrgChart control should resemble the following:

Markup

```
<OrgChart:C1OrgChart Name="_orgChart" Orientation="Horizontal">
```

In the Properties Window

Complete the following steps to change the C1OrgChart's Orientation property:

1. Locate the Orientation property in the Properties window.
2. Use the drop-down list to change the value to "**Vertical**"

Changing C1OrgChart Flow Direction

You can specify whether the chart shows from right to left or from left to right using the FlowDirection property.

In XAML

Locate the opening <OrgChart:C1OrgChart> tag and insert FlowDirection="RightToLeft" into the tag. The <OrgChart:C1OrgChart> markup should resemble the following:

Markup

```
<OrgChart:C1OrgChart Name="_orgChart" Orientation="Horizontal"
FlowDirection="RightToLeft">
```

In the Properties Window

1. Locate the **FlowDirection** property in the Properties window.
2. Use the drop-down list to change the value to **"RightToLeft"**.

Customizing the C1OrgChart Item Connector

You can use several properties to customize the lines used to connect the nodes of the C1OrgChart, including ConnectorStroke, ConnectorThickness, and ConnectorDashArray. These properties can be set in the XAML markup or in the Design View Properties window.

In XAML

To change the Item Connector color, insert ConnectorStroke="#FF970014" into the opening <OrgChart:C1OrgChart> tag.

To change the Item Connector thickness, insert ConnectorThickness="3" after the ConnectorStroke markup.

To customize the type of Item Connector used, insert ConnectorDashArray="1 1" after the ConnectorThickness markup. This will create a dashed connector.

The final XAML markup should resemble the following:

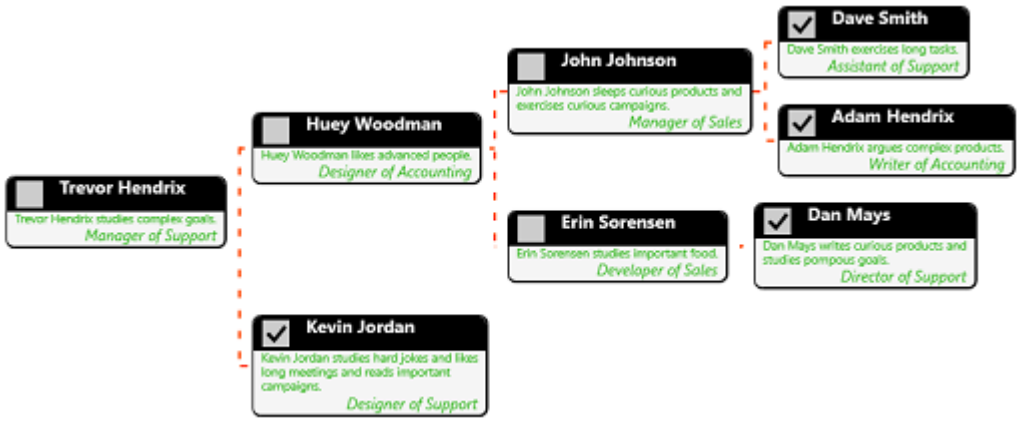
Markup

```
<OrgChart:C1OrgChart Name="_orgChart" Orientation="Horizontal"
ConnectorStroke="#FF970014" ConnectorThickness="2" ConnectorDashArray="1 1">
```

In the Properties Window

You can also customize the Item Connectors through the Properties Window in the Design View.

1. Locate the ConnectorStroke property and use the color picker to choose a new color for the Item Connector.
2. Locate the ConnectorThickness property and select a new thickness. For this Help, use "**3**".
3. Press F5 to run the application and note the changes you have made. The C1OrgChart control should appear as in the following image:



Expanding and Collapsing C1OrgChart Nodes

C1OrgChart allows you to create a collapsible C1OrgChart that behaves similarly to a TreeView control. Complete the following steps to expand and collapse C1OrgChart nodes:

1. Insert the following markup to create the C1OrgChart control and its control panel. The following XAML will add a ScrollViewer control in addition to the C1OrgChart control:

Markup

```
<!-- org chart -->
       <ScrollViewer Grid.Row="2" HorizontalScrollBarVisibility="Auto" VerticalScrollBarVisibility="Auto">
           <orgchart:C1OrgChart
   x:Name="_orgChart"
   Grid.Row="1"
   Orientation="Vertical"
   HorizontalAlignment="Center" VerticalAlignment="Center"
   ConnectorStroke="OrangeRed" ConnectorThickness="{Binding Path=Subordinates.Count}"
                    Foreground="#FF39B925" Xaml:C1NagScreen.Nag="True" >
               <!-- scale transform bound to slider -->
               <orgchart:C1OrgChart.RenderTransform>
                   <ScaleTransform
     ScaleX="{Binding Value, ElementName=_sliderZoom}"
     ScaleY="{Binding Value, ElementName=_sliderZoom}" />
               </orgchart:C1OrgChart.RenderTransform>
               <!-- template used to show tree nodes -->
               <orgchart:C1OrgChart.ItemTemplate>
                   <DataTemplate>
                       <!-- outer border -->
                       <Border
      Background="WhiteSmoke" BorderBrush="Black"
      BorderThickness="1 1 2 2" CornerRadius="6"
      MaxWidth="200" >
                           <StackPanel Orientation="Vertical" >
                               <!-- item header -->
                               <Border CornerRadius="6 6 0 0" Background="Black" >
                                   <StackPanel Orientation="Horizontal">
                                     <!-- bind CheckBox to containing C1OrgChart's IsCollapsed property -->
                                     <CheckBox Margin="4 0" Checked="CheckBox_Checked" Unchecked="CheckBox_Unchecked"/>
                                     <!-- item header: person's Name -->
                                     <TextBlock Text="{Binding Name}" FontWeight="Bold" FontSize="14"
                                      Foreground="WhiteSmoke" Padding="4 0 0 0" />
                                   </StackPanel>
                               </Border>
                               <!-- body: person's details -->
                               <TextBlock Text="{Binding Notes}" Padding="6 0" FontSize="9.5"
                                    TextWrapping="Wrap" />
                               <TextBlock Text="{Binding Position}" Padding="6 0" FontSize="12" FontStyle="Italic"
                                HorizontalAlignment="Right" />
                           </StackPanel>
                       </Border>
                   </DataTemplate>
               </orgchart:C1OrgChart.ItemTemplate>
           </orgchart:C1OrgChart>
       </ScrollViewer>
```

2. Add the following XAML markup below the <Grid> tag and before the <Scrollviewer> tag. This will add some Grid row definitions, a title for your application, and a control panel that contains the buttons and sliders. This way you can control the zoom, orientation, and data for your C1OrgChart:

Markup

```
<Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
            <RowDefinition />
        </Grid.RowDefinitions>
        <!-- sample title -->
        <StackPanel Orientation="Horizontal" >
            <TextBlock Text="C1OrgChart: Collapse/Expand Sample" FontSize="16" VerticalAlignment="Bottom"
             Foreground="Black" />
            <TextBlock x:Name="_tbTotal" VerticalAlignment="Bottom" />
        </StackPanel>
        <!-- control panel -->
        <StackPanel Orientation="Horizontal" VerticalAlignment="Top" Grid.Row="1">
            <Button Content="Refresh Data" Padding="8 0" Click="Button_Refresh_Click"
             Foreground="Black" BorderBrush="Black" />
            <TextBlock Text=" Zoom: " VerticalAlignment="Center" />
            <Slider x:Name="_sliderZoom" Minimum=".01" Maximum="1" Value="1" Width="100" SmallChange="0.01"
             StepFrequency="0.01" Height="44" HorizontalAlignment="Left" Foreground="#FFFF1B1B" Background="#29C91515"
             VerticalAlignment="Center" />
            <CheckBox Margin="20 0" Content="Horizontal" Click="CheckBox_Click" Background="#FF0Cffff" Foreground="Black" />
            <Button Content="Collapse to Level 3" Padding="8 0" Click="Button_Collapse_Click" Foreground="Black"
             BorderBrush="Black" />
        </StackPanel>
```

3. Right-click the page and select **View Code** from the list. Import the following namespace into the code file:

Visual Basic

```
Imports C1.Xaml.OrgChart
```

C#

```
using C1.Xaml.OrgChart
```

4. Insert the following code directly below the **InitializeComponent()** method:

Visual Basic

```
CreateData()
```

C#

```
CreateData();
```

5. Add the following code to handle the button click and checkbox click events, and to handle the C1OrgChart toggling:

Visual Basic

```
Private Sub CheckBox_Click(sender As Object, e As RoutedEventArgs)
        _orgChart.Orientation = If(DirectCast(sender, CheckBox).IsChecked.Value,
        Orientation.Horizontal, Orientation.Vertical)
    End Sub
    ' rebuild the chart using new random data
    Private Sub Button_Refresh_Click(sender As Object, e As RoutedEventArgs)
        CreateData()
    End Sub
    ' collapse the chart to level 3
    Private Sub Button_Collapse_Click(sender As Object, e As RoutedEventArgs)
        ToggleCollapseExpand(_orgChart, 0, 3)
    End Sub
    ' collapse the chart to a given level
    Private Sub ToggleCollapseExpand(node As C1OrgChart, level As Integer, maxLevel As Integer)
        If level >= maxLevel Then
            node.IsCollapsed = True
        Else
            node.IsCollapsed = False
            For Each subNode In node.ChildNodes
                ToggleCollapseExpand(subNode, level + 1, maxLevel)
            Next
        End If
    End Sub
```

C#

```
void CheckBox_Click(object sender, RoutedEventArgs e)
        {
            _orgChart.Orientation = ((CheckBox)sender).IsChecked.Value
                ? Orientation.Horizontal
                : Orientation.Vertical;
```

```
        }
        // rebuild the chart using new random data
        void Button_Refresh_Click(object sender, RoutedEventArgs e)
        {
            CreateData();
        }
        // collapse the chart to level 3
        void Button_Collapse_Click(object sender, RoutedEventArgs e)
        {
            ToggleCollapseExpand(_orgChart, 0, 3);
        }
        // collapse the chart to a given level
        void ToggleCollapseExpand(C1OrgChart node, int level, int maxLevel)
        {
            if (level >= maxLevel)
            {
                node.IsCollapsed = true;
            }
            else
            {
                node.IsCollapsed = false;
                foreach (var subNode in node.ChildNodes)
                {
                    ToggleCollapseExpand(subNode, level + 1, maxLevel);
                }
            }
        }
```

6. Add the following code to create new random data and handle the checkbox events that will allow the C1OrgChart to expand and collapse:

Visual Basic

```
' create some random data and assign it to the chart
    Private Sub CreateData()
        Dim p = Person.CreatePerson(15)
        _tbTotal.Text = String.Format(" ({0} items total)", p.TotalCount)
        _orgChart.Header = p
    End Sub
    ' add for supporting expand/collapse in OrgChart sample
    Private Sub CheckBox_Unchecked(sender As Object, e As RoutedEventArgs)
        CheckedChanged(sender)
    End Sub
    Private Sub CheckBox_Checked(sender As Object, e As RoutedEventArgs)
        CheckedChanged(sender)
    End Sub
    Private Sub CheckedChanged(sender As Object)
        Dim checkBox As CheckBox = TryCast(sender, CheckBox)
        Dim parent As FrameworkElement = TryCast(VisualTreeHelper.GetParent(checkBox), FrameworkElement)
        While parent IsNot Nothing AndAlso Not (TypeOf parent Is C1OrgChart)
            parent = TryCast(VisualTreeHelper.GetParent(parent), FrameworkElement)
        End While
        If parent IsNot Nothing Then
            Dim orgChart As C1OrgChart = TryCast(parent, C1OrgChart)
            If checkBox.IsChecked IsNot Nothing Then
                orgChart.IsCollapsed = checkBox.IsChecked.Value
            End If
        End If
    End Sub
```

C#

```
// create some random data and assign it to the chart
        void CreateData()
        {
            var p = Person.CreatePerson(15);
            _tbTotal.Text = string.Format(" ({0} items total)", p.TotalCount);
            _orgChart.Header = p;
        }
        // add for supporting expand/collapse in OrgChart sample
        private void CheckBox_Unchecked(object sender, RoutedEventArgs e)
        {
            CheckedChanged(sender);
        }
private void CheckBox_Checked(object sender, RoutedEventArgs e)
        {
            CheckedChanged(sender);
        }
        private void CheckedChanged(object sender)
        {
```

```
            CheckBox checkBox = sender as CheckBox;
            FrameworkElement parent = VisualTreeHelper.GetParent(checkBox) as FrameworkElement;
            while (parent != null && !(parent is C1OrgChart))
            {
                parent = VisualTreeHelper.GetParent(parent) as FrameworkElement;
            }
            if (parent != null)
            {
                C1OrgChart orgChart = parent as C1OrgChart;
                if (checkBox.IsChecked != null)
                {
                    orgChart.IsCollapsed = checkBox.IsChecked.Value;
                }
            }
        }
    }
}
```

7. Locate your application name in the **Solution Explorer**. Right-click on the name and select **Add | New Item** from the list. Select **Code File** from the template window and name the code file **Person.cs** or **Person.vb**.
8. Add the following namespaces to the Person code file:

Visual Basic

```
Imports System
Imports System.Collections
Imports System.Collections.Generic
Imports System.Collections.ObjectModel
```

C#

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Collections.ObjectModel;
```

9. Insert the following code below the namespaces to create the hierarchical data items which will be called to create the data in the C1OrgChart:

Visual Basic

```
Namespace Data
     ''' <summary>
     ''' Our hierarchical data item: A Person has Subordinates of type Person.
     ''' </summary>
     Public Class Person
         Private _list As New ObservableCollection(Of Person)()
#Region "** object model"
        Public Property Name() As String
            Get
                Return m_Name
            End Get
            Set(value As String)
                m_Name = Value
            End Set
        End Property
        Private m_Name As String
        Public Property Position() As String
            Get
                Return m_Position
            End Get
            Set(value As String)
                m_Position = Value
            End Set
        End Property
        Private m_Position As String
        Public Property Notes() As String
            Get
                Return m_Notes
            End Get
            Set(value As String)
                m_Notes = Value
            End Set
        End Property
        Private m_Notes As String
        Public ReadOnly Property Subordinates() As IList(Of Person)
            Get
                Return _list
```

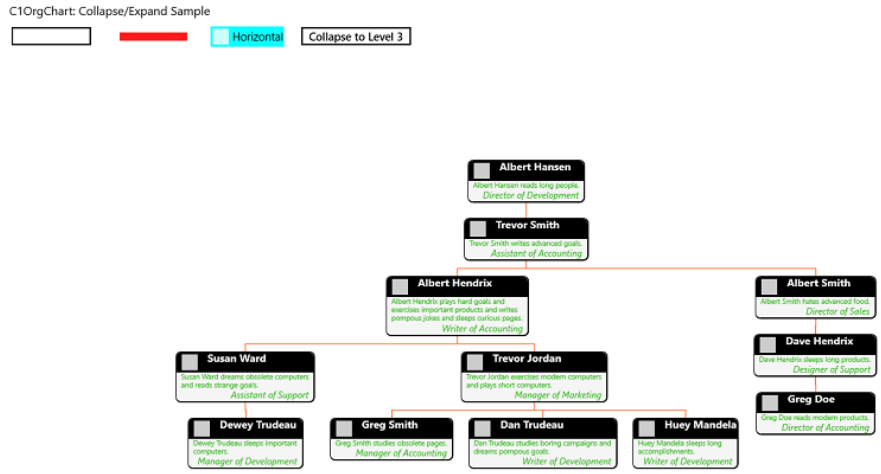
```
            End Get
        End Property
        Public ReadOnly Property TotalCount() As Integer
            Get
                Dim count = 1
                For Each p In Subordinates
                    count += p.TotalCount
                Next
                Return count
            End Get
        End Property
        Public Overrides Function ToString() As String
            Return String.Format("{0}:" & vbCr & vbLf & vbTab & "{1}", Name, Position)
        End Function
#End Region
#Region "** Person factory"
        Shared _rnd As New Random()
         Shared _positions As String() = "Director|Manager|Designer|Developer|Writer|Assistant".Split("|"c)
         Shared _areas As String() = "Development|Marketing|Sales|Support|Accounting".Split("|"c)
         Shared _first As String() =
"John|Paul|Dan|Dave|Rich|Mark|Greg|Erin|Susan|Sarah|Tim|Trevor|Kevin|Mark|Dewey|Huey|Larry|Moe|Curly|Adam|Albert".Split("|"c)
        Shared _last As String() =
"Smith|Doe|Williams|Sorensen|Hansen|Mandela|Johnson|Ward|Woodman|Jordan|Mays|Kevorkian|Trudeau|Hendrix|Clinton".Split("|"c)
         Shared _verb As String() =
"likes|reads|studies|hates|exercises|dreams|plays|writes|argues|sleeps|ignores".Split("|"c)
         Shared _adjective As String() =
"long|short|important|pompous|hard|complex|advanced|modern|boring|strange|curious|obsolete|bizarre".Split("|"c)
         Shared _noun As String() =
"products|tasks|goals|campaigns|books|computers|people|meetings|food|jokes|accomplishments|screens|pages".Split("|"c)
        Public Shared Function CreatePerson(level As Integer) As Person
            Dim p = CreatePerson()
            If level > 0 Then
                level -= 1
                For i As Integer = 0 To _rnd.[Next](1, 4) - 1
                    p.Subordinates.Add(CreatePerson(_rnd.[Next](level \ 2, level)))
                Next
            End If
            Return p
         End Function
         Public Shared Function CreatePerson() As Person
            Dim p = New Person()
            p.Position = String.Format("{0} of {1}", GetItem(_positions), GetItem(_areas))
            p.Name = String.Format("{0} {1}", GetItem(_first), GetItem(_last))
            p.Notes = String.Format("{0} {1} {2} {3}", p.Name, GetItem(_verb), GetItem(_adjective), GetItem(_noun))
            While _rnd.NextDouble() < 0.5
                p.Notes += String.Format(" and {0} {1} {2}", GetItem(_verb), GetItem(_adjective), GetItem(_noun))
            End While
           p.Notes += "."
            Return p
         End Function
         Private Shared Function GetItem(list As String()) As String
            Return list(_rnd.[Next](0, list.Length))
         End Function
#End Region
     End Class
 End Namespace
```

C#

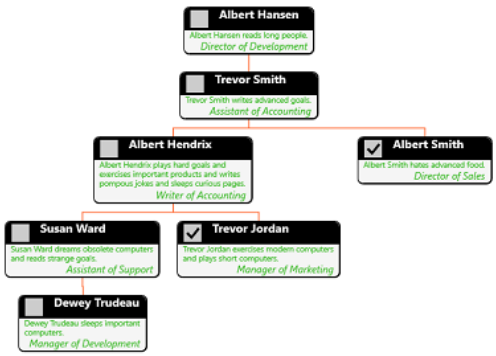
```
namespace CollapseExpand
 {
     /// <summary>
     /// Our hierarchical data item: A Person has Subordinates of type Person.
     /// </summary>
     public class Person
     {
         ObservableCollection<Person> _list = new ObservableCollection<Person>();
        #region ** object model
        public string Name { get; set; }
         public string Position { get; set; }
         public string Notes { get; set; }
         public IList<Person> Subordinates
         {
             get { return _list; }
         }
         public int TotalCount
         {
             get
```

```
            {
                var count = 1;
                foreach (var p in Subordinates)
                {
                    count += p.TotalCount;
                }
                return count;
            }
        }
        public override string ToString()
        {
            return string.Format("{0}:\r\n\t{1}", Name, Position);
        }
       #endregion
       #region ** Person factory
       static Random _rnd = new Random();
        static string[] _positions = "Director|Manager|Designer|Developer|Writer|Assistant".Split('|');
        static string[] _areas = "Development|Marketing|Sales|Support|Accounting".Split('|');
        static string[] _first =
"John|Paul|Dan|Dave|Rich|Mark|Greg|Erin|Susan|Sarah|Tim|Trevor|Kevin|Mark|Dewey|Huey|Larry|Moe|Curly|Adam|Albert".Split('|');
        static string[] _last =
"Smith|Doe|Williams|Sorensen|Hansen|Mandela|Johnson|Ward|Woodman|Jordan|Mays|Kevorkian|Trudeau|Hendrix|Clinton".Split('|');
        static string[] _verb = "likes|reads|studies|hates|exercises|dreams|plays|writes|argues|sleeps|ignores".Split('|');
        static string[] _adjective =
"long|short|important|pompous|hard|complex|advanced|modern|boring|strange|curious|obsolete|bizarre".Split('|');
        static string[] _noun =
"products|tasks|goals|campaigns|books|computers|people|meetings|food|jokes|accomplishments|screens|pages".Split('|');
       public static Person CreatePerson(int level)
        {
            var p = CreatePerson();
            if (level > 0)
            {
                level--;
                for (int i = 0; i < _rnd.Next(1, 4); i++)
                {
                    p.Subordinates.Add(CreatePerson(_rnd.Next(level / 2, level)));
                }
            }
            return p;
        }
        public static Person CreatePerson()
        {
            var p = new Person();
            p.Position = string.Format("{0} of {1}", GetItem(_positions), GetItem(_areas));
            p.Name = string.Format("{0} {1}", GetItem(_first), GetItem(_last));
            p.Notes = string.Format("{0} {1} {2} {3}", p.Name, GetItem(_verb), GetItem(_adjective), GetItem(_noun));
            while (_rnd.NextDouble() < .5)
            {
                p.Notes += string.Format(" and {0} {1} {2}", GetItem(_verb), GetItem(_adjective), GetItem(_noun));
            }
            p.Notes += ".";
            return p;
        }
        static string GetItem(string[] list)
       {
            return list[_rnd.Next(0, list.Length)];
        }
       #endregion
    }
}
```

10. Press F5 to run your application. The C1OrgChart should resemble the following image:



Click the CheckBox in the corner of one of the Main Item nodes. Note how the C1OrgChart collapses:



Using a Hierarchical Data Template

This topic will demonstrate advanced binding scenarios using the DataTemplateSelector and HierarchicalDataTemplate classes.

This topic assumes that you have created a Windows Store application and added the appropriate references to it.

1. Add the following XAML markup below the namespace declarations to create the Data Templates:

Markup

```
<Page.Resources>
        <!-- template for Team objects -->
        <DataTemplate x:Key="TeamTemplate" >
            <Border Background="LightBlue" Padding="4" >
                <TextBlock FontStyle="Italic" Text="{Binding Name}" />
            </Border>
        </DataTemplate>
        <!-- template for Division objects -->
        <Xaml:C1HierarchicalDataTemplate  x:Key="DivisionTemplate"
            ItemTemplate="{StaticResource TeamTemplate}" ItemsSource="{Binding
Teams}">
            <DataTemplate>
                <Border Background="Gold" >
                    <TextBlock Text="{Binding Name}" FontWeight="Bold"
HorizontalAlignment="Center" VerticalAlignment="Center" Padding="20" />
```

```
                </Border>
            </DataTemplate>
        </Xaml:C1HierarchicalDataTemplate >
        <!-- template for League objects -->
        <Xaml:C1HierarchicalDataTemplate  x:Key="LeagueTemplate"
            ItemTemplate="{StaticResource DivisionTemplate}" ItemsSource="{Binding
Divisions}">
            <DataTemplate>
            <Border Background="LightCoral" >
                <TextBlock Text="{Binding Name}" FontWeight="Bold"
HorizontalAlignment="Center" VerticalAlignment="Center" Padding="40" />
            </Border>
            </DataTemplate>
        </Xaml:C1HierarchicalDataTemplate >
    </Page.Resources>
```

2. Insert the XAML markup below to create your Grid layout, the C1OrgChart control, and the ScrollViewer control:

Markup

```
<Grid x:Name="LayoutRoot" Background="White">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="118*" />
            <RowDefinition Height="158*" />
        </Grid.RowDefinitions>
        <!-- sample title -->
        <StackPanel Orientation="Horizontal" >
            <TextBlock Text="C1OrgChart: HierarchicalDataTemplate" FontSize="16"
VerticalAlignment="Bottom" />
            <TextBlock x:Name="_tbTotal" VerticalAlignment="Bottom" />
        </StackPanel>
        <ScrollViewer Grid.Row="1" HorizontalScrollBarVisibility="Auto"
VerticalScrollBarVisibility="Auto" Padding="0" >
            <OrgChart:C1OrgChart
                Name="_chart" ItemTemplate="{StaticResource LeagueTemplate}"
                ConnectorDashArray="1 2" ConnectorStroke="Gray"
                HorizontalAlignment="Center" VerticalAlignment="Center" />
        </ScrollViewer>
        <Xaml:C1TreeView Name="_tree" Grid.Row="2" ItemTemplate="{StaticResource
LeagueTemplate}" />
    </Grid>
```

3. Locate a **C1TreeView** control in your ToolBox and add that to your application below the <ScrollViewer> </ScrollViewer> tags. Insert the following into the <Xaml:C1TreeView> tag:

Markup

```
Name="_tree" Grid.Row="2" ItemTemplate="{StaticResource LeagueTemplate}"
```

4. Switch to your code view by right-clicking your application and selecting View Code from the list.
5. Add the following namespace to the top of your page:

Visual Basic

```
Imports C1.Xaml.OrgChart
```

C#

```
using C1.Xaml.OrgChart;
```

6. Add the following code directly below the InitializeComponent() method:

Visual Basic

```
' create data object
        Dim league__1 = League.GetLeague()

        ' show it in C1OrgChart
        _chart.Header = league__1
        ' this has the same effect:
        '_chart.ItemsSource = new object[] { league };

        ' show it in TreeView
        _tree.ItemsSource = New Object() {league__1}
```

C#

```
// create data object
            var league = League.GetLeague();

            // show it in C1OrgChart
            _chart.Header = league;
            // this has the same effect:
            //_chart.ItemsSource = new object[] { league };
            // show it in TreeView
            _tree.ItemsSource = new object[] { league };
```

7. Insert the following code to create the teams, Leagues, and Divisions that will appear in the C1OrgChart and in the **C1TreeView** control:

Visual Basic

```
Public Class League
        Public Property Name() As String
            Get
                Return m_Name
            End Get
            Set(value As String)
                m_Name = Value
            End Set
        End Property
        Private m_Name As String
        Public Property Divisions() As List(Of Division)
            Get
                Return m_Divisions
            End Get
            Set(value As List(Of Division))
                m_Divisions = Value
            End Set
        End Property
```

```
        Private m_Divisions As List(Of Division)
        Public Shared Function GetLeague() As League
            Dim league = New League()
            league.Name = "Main League"
            league.Divisions = New List(Of Division)()
            For Each div In "North,South,East,West".Split(","c)
                Dim d = New Division()
                league.Divisions.Add(d)
                d.Name = div
                d.Teams = New List(Of Team)()
                For Each team In "t1,t2,t3,t4".Split(","c)
                    Dim t = New Team()
                    d.Teams.Add(t)
                    t.Name = String.Format("{0} {1}", team, div)
                Next
            Next
            Return league
        End Function
    End Class
```

C#

```
public class League
    {
        public string Name { get; set; }
        public List<Division> Divisions { get; set; }
        public static League GetLeague()
        {
            var league = new League();
            league.Name = "Main League";
            league.Divisions = new List<Division>();
            foreach (var div in "North,South,East,West".Split(','))
            {
                var d = new Division();
                league.Divisions.Add(d);
                d.Name = div;
                d.Teams = new List<Team>();
                foreach (var team in "t1,t2,t3,t4".Split(','))
                {
                    var t = new Team();
                    d.Teams.Add(t);
                    t.Name = string.Format("{0} {1}", team, div);
                }
            }
            return league;
        }
    }
```

8. Add the code below to create the Public Class that will Get and Set the values for the Teams and Divisions:

Visual Basic

```
Public Class Division
        Public Property Name() As String
            Get
```

```
                Return m_Name
            End Get
            Set(value As String)
                m_Name = Value
            End Set
        End Property
        Private m_Name As String
        Public Property Teams() As List(Of Team)
            Get
                Return m_Teams
            End Get
            Set(value As List(Of Team))
                m_Teams = Value
            End Set
        End Property
        Private m_Teams As List(Of Team)
    End Class
    Public Class Team
        Public Property Name() As String
            Get
                Return m_Name
            End Get
            Set(value As String)
                m_Name = Value
            End Set
        End Property
        Private m_Name As String
    End Class
```

C#

```
public class Division
    {
        public string Name { get; set; }
        public List<Team> Teams { get; set; }
    }
    public class Team
    {
        public string Name { get; set; }
    }
}
```

9. Run your application. Your application should resemble the following image: