RadGanttView For Silverlight and WPF

This tutorial will introduce RadGanttView, part of the Telerik suite of XAML controls.

Setting Up The Project

To begin, open Visual Studio and click on the Telerik menu option. Under *Rad Controls For Silverlight* click on *Create New Telerik Project*. Name your project, accept Silverlight 5 and in the Project Configuration Wizard dialog check GanttView.

When you click ok, the necessary assembly is added to the References as shown in figure 1

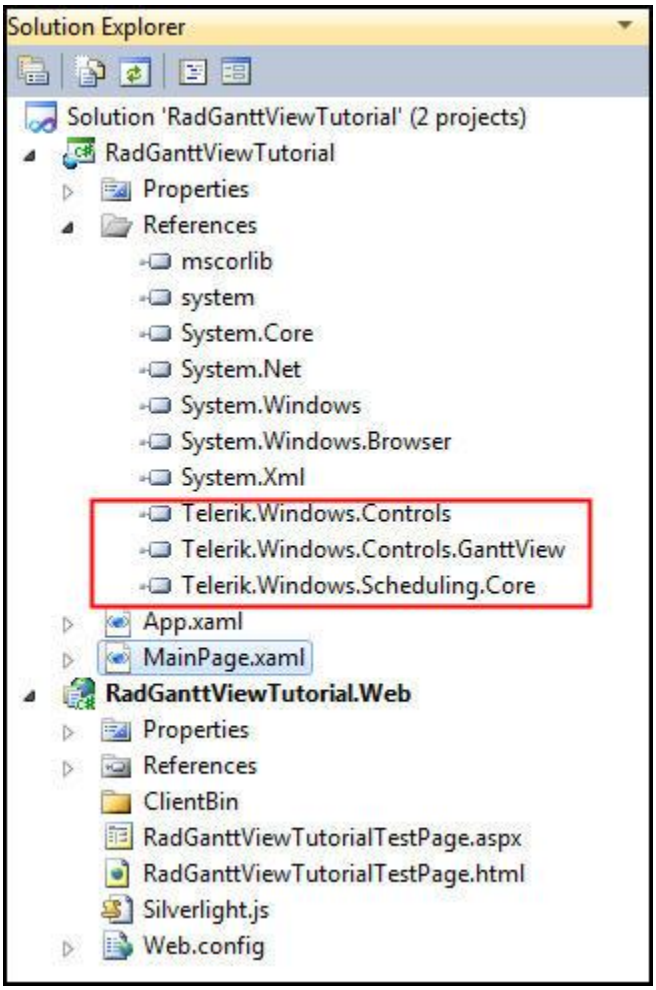

Figure 1

Your application will open to MainPage.xaml and, thanks to the Telerik Visual Studio extensions, the namespace t*elerik* will already have been created in the XAML heading.

```
<UserControl x:Class="RadBarCode.GettingStarted.MainPage"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:telerik="http://schemas.telerik.com/2008/xaml/presentation"
             mc:Ignorable="d" d:DesignWidth="640" d:DesignHeight="480">
```

Add a RadGanttView to the Layout grid in the XAML file,

```
        <telerik:RadGanttView Name="xGanttView">

        </telerik:RadGanttView>
```

Open the code-behind file, where we will do three things,

1. Define tasks to add to the GanttView
2. Set the Pixel Length to determine how wide the display of each day will be
3. Set the Visible Range

We set the visible range by providing an instance of VisibleRange passing in a starting date and an ending date,

```
var d = new DateTime(2012, 3, 1);
xGanttView.VisibleRange = new VisibleRange(d, d.AddMonths(2));
```

We set the PixelLength by passing in a TimeSpan, as shown in figure 2,

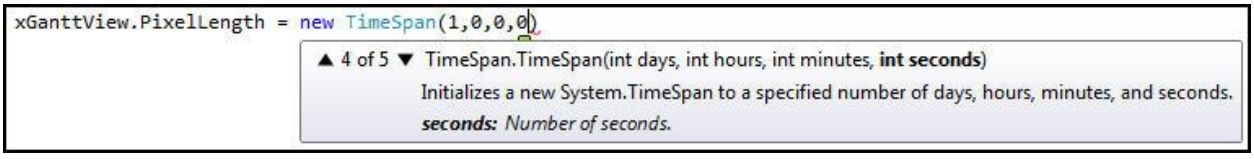

Figure 2

Finally, we need to add tasks (and sub-tasks) to the GanttView. We start by creating an observable collection of GanttTask objects,

```
var tasks = new ObservableCollection<GanttTask>();
```

Next, we'll create four "parent" tasks, each with a start, end and title,

```
for (int i = 0; i<4; i++)
{
    var gt = new GanttTask(
                  d.AddDays(14 * i),
                  d.AddDays(14 * i + 14),
                 "Title " + i.ToString());
```

For each task we'll create a set of child tasks based on the task's duration in days,

```
    for (int j = 0; j < gt.Duration.Days; j++)
    {
        var childGT = new GanttTask();
        childGT.Start = gt.Start.AddDays(j);
        childGT.End = childGT.Start.AddHours(23);
        childGT.Title = "Child " + i.ToString() + "/" + j.ToString();


Add each child to the children collection of the GanttTask

        gt.Children.Add(childGT);
    }

Add the GanttTask to the ObservableCollection

    tasks.Add(gt);
}

Finally, add the ObservableCollection as the TasksSource for the GanttView,

xGanttView.TasksSource = tasks;
```

When you run the application you'll see the tasks and subtasks, but because we set the PixelLength to 1 day, you can't see much detail. Change the PixelLength to 1 hour,

```
xGanttView.PixelLength = new TimeSpan(0, 1, 0, 0);
```

Run the application again and you should get a better view of the Gantt chart, complete with parent and child tasks and durations, as shown in figure 3,

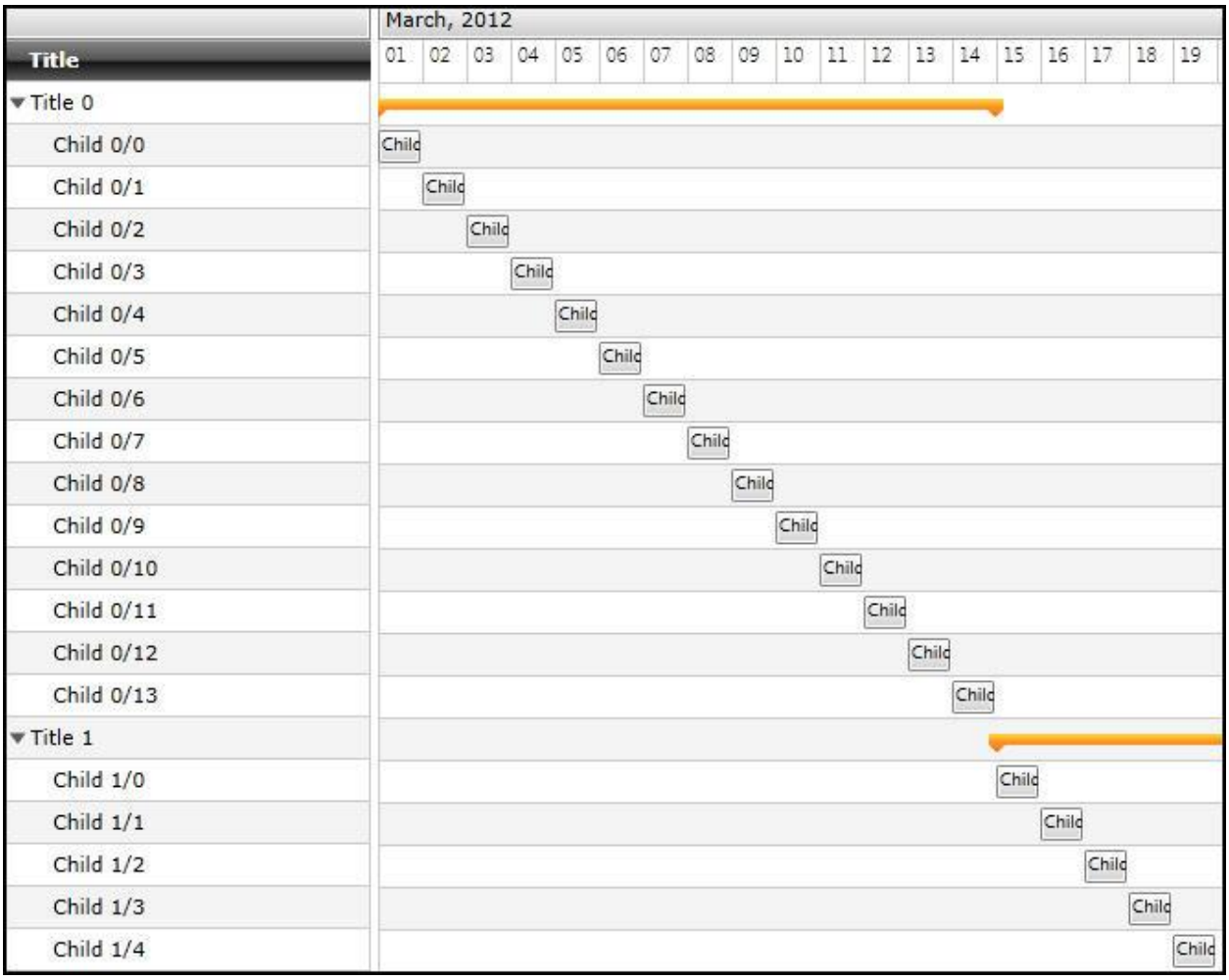

Figure 3

Relations

Let's establish a relationship among the various child tasks. Relationships add a link between tasks. To do so, first set the PixelLength to 15 minutes. Then add the following code immediately after the creation of ChildGt,

```
if ( j > 0 )
{
    GanttTask prevGt = gt.Children[j - 1] as GanttTask;
    prevGt.SetRelations(new List<Relation>() { new Relation() { Task = childGT } });
}
```

The result is that you can now get a better view of the relationships among the child tasks, as shown in figure 4,

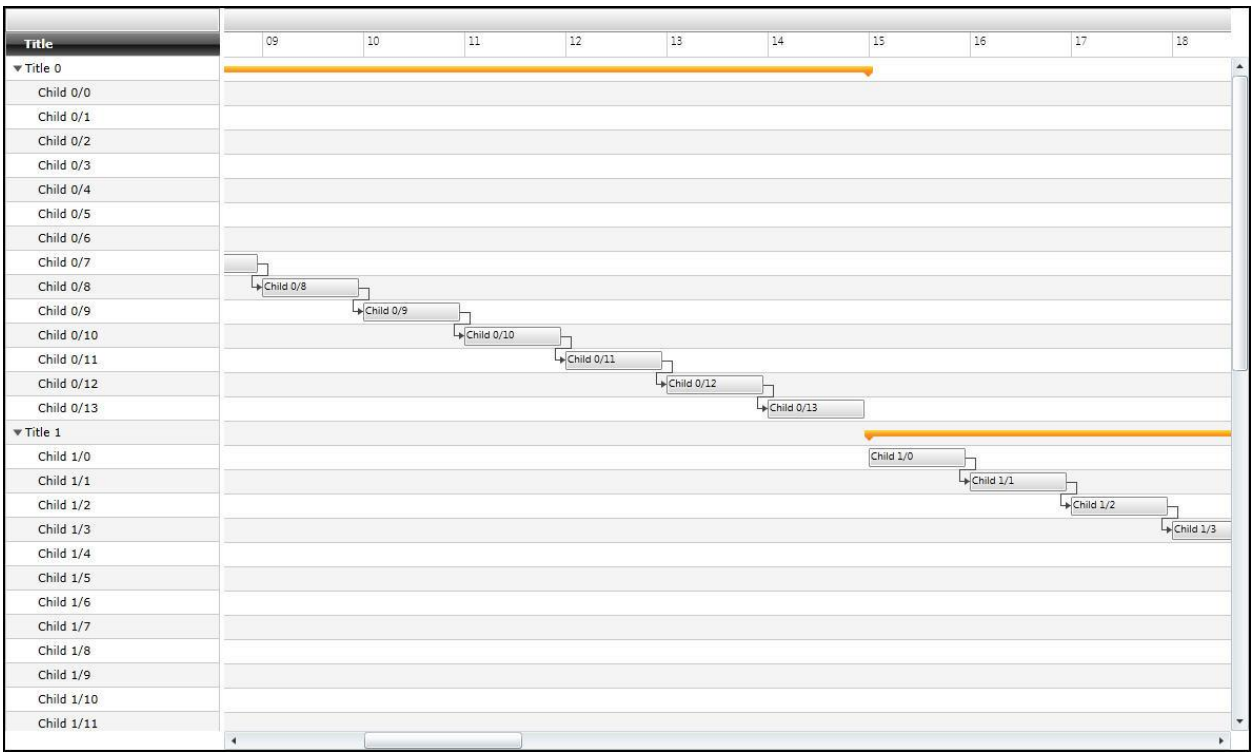

Figure 4

GanttViewHighlighting

To examine GanttViewHighlighting in depth, we'll start a new project, very similar to the previous except this time we'll also add Telerik.Windows.Controls.Input to the references.

Add two rows and two columns to the Layoutroot as shown here,

```
<Grid.RowDefinitions>
    <RowDefinition />
    <RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
    <ColumnDefinition />
    <ColumnDefinition />
</Grid.ColumnDefinitions>
<telerik:RadGanttView Name="xRadGanttView" Grid.ColumnSpan="2" />
```

Let's start by creating the ViewModel class. Name it GanttVM and be sure to set it to derive from ViewModelBase (a base for ViewModel classes provided by the Telerik library).

```
public class GanttVM : ViewModelBase
{
```

Declare an observable collection of GanttTask objects,

```
public ObservableCollection<GanttTask> Tasks { get; set; }
```

and initialize it in your constructor.

```
public GanttVM()
{
    Tasks = new ObservableCollection<GanttTask>();
    LoadTasks();
}
```

The LoadTasks method will be based on the code used in the previous example,

```
private void LoadTasks()
 {
     var d = new DateTime(2012, 3, 1);

     for (int i = 0; i < 4; i++)
     {
         var gt = new GanttTask(d.AddDays(14 * i), d.AddDays(14 * i + 14), "Title " +
i.ToString());

         for (int j = 0; j < gt.Duration.Days; j++)
         {
             var childGT = new GanttTask();
             childGT.Start = gt.Start.AddDays(j);
             childGT.End = childGT.Start.AddHours(23);
             childGT.Title = "Child " + i.ToString() + "/" + j.ToString();

             if (j > 0)
             {
                 GanttTask prevGt = gt.Children[j - 1] as GanttTask;
                 prevGt.SetRelations(new List<Relation>() { new Relation() { Task =
childGT } });
             }

             gt.Children.Add(childGT);

         }
         Tasks.Add(gt);
     }
 }
```

Declare a property of type VisibleRange,

```
public VisibleRange GanttRange { get; set; }
```

And initialize that property in the constructor,

```
GanttRange = new VisibleRange(new DateTime(2012, 3, 1), new DateTime(2012, 5, 1));
```

In this case we've initialized it to start on March 1, 2012 and to span two months.

Build the application just to make sure there are no errors or typos. Return to the Xaml file and add a local namespace,

```
xmlns:local="clr-namespace:GanttChartMVVM"
```

You'll use this to create a resource to hold the VM,

```
<UserControl.Resources>
    <local:GanttVM x:Key="xVM" />
</UserControl.Resources>
```

This lets us set the DataContext in the LayoutRoot,

```
<Grid x:Name="LayoutRoot" DataContext="{StaticResource xVM}">
```

We're now ready to update the bindings on the RadGanttView,

```
<telerik:RadGanttView Name="xGanttView" Grid.ColumnSpan="2"
                      TasksSource="{Binding Tasks}"
                      VisibleRange="{Binding GanttRange}"/>
```

You should now see the GanttChart in the designer. If not, try rebuilding the application.

Let's add a slider control to our application,

```
<telerik:RadSlider x:Name="xRadSlider"
                   Grid.Row="1"
                   Minimum="10000000"
                   Maximum="30000000000"
                   Value="18000000000"
                   Margin="5,10" />
```

The extremely large values are because we'll be converting these values to ticks for a timespan.

Create a new class TicksToTimespanConverter and add the following code,

```
public class TicksToTimeSpanConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, CultureInfo
culture)
    {
        return TimeSpan.FromTicks((long)(double)value);
    }

    public object ConvertBack(object value, Type targetType, object parameter,
CultureInfo culture)
```

```
    {
        throw new NotImplementedException();
    }
}
```

Build the application.

Add the converter to the resources in MainPage.xaml,

```
<UserControl.Resources>
    <local:GanttVM x:Key="xVM" />
    <local:TicksToTimeSpanConverter x:Key="xConverter" />
</UserControl.Resources>
```

We can now add the binding for the PixelLength to the RadGanttView in the XAML,

```
<telerik:RadGanttView Name="xGanttView" Grid.ColumnSpan="2"
  TasksSource="{Binding Tasks}"
  VisibleRange="{Binding GanttRange}"
  PixelLength="{
       Binding ElementName=xRadSlider,
       Path=Value,
       Converter={StaticResource xConverter}}"/>
```

Run the application. We now have a slider at the bottom which works to zoom in and out on our Gantt chart, as shown in figure 5

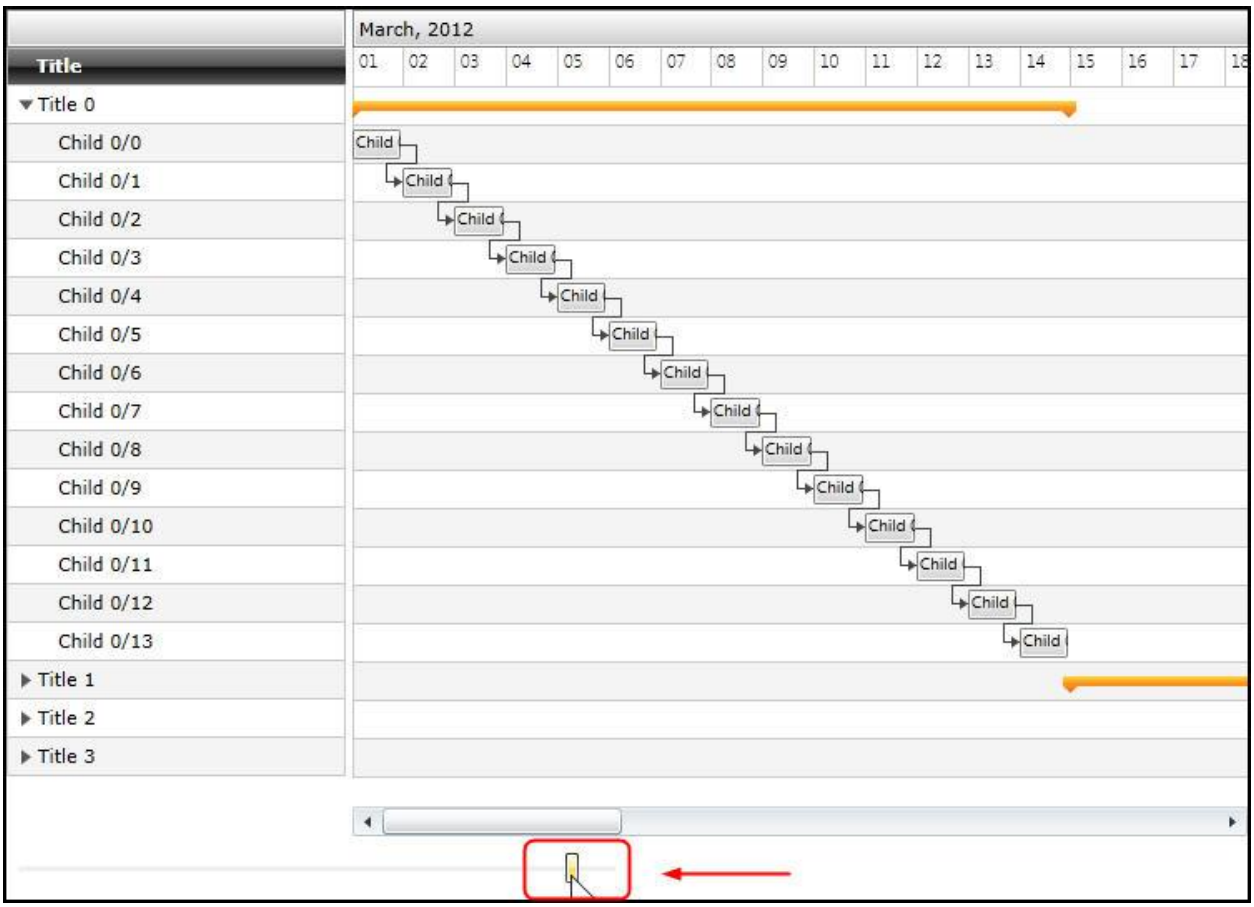

Figure 5

We're now going to add a toggle button which will turn highlighting on and off. We're using the RadTogglebutton and Radslider which are part of the suite of Telerik tools for Silverlight and WPF

```
<telerik:RadToggleButton Name="xRadToggleButton"
                         Grid.Row="1"
                         Grid.Column="1"
                         HorizontalAlignment="Center"
                         VerticalAlignment="Center"
                         Content="Highlight" />
```

We need a public property to bind the IsChecked attribute to. Open the ViewModel and add the following property,

```
private bool checkedValue;
 public bool CheckedValue
 {
     get { return checkedValue;  }
     set
     {
       if ( checkedValue != value)
         {
             checkedValue = value;
             OnPropertyChanged(() => this.CheckedValue);
```

```
        }
    }
}
```

You can now update the ToggleButton's IsChecked property,

```
<telerik:RadToggleButton Name="xRadToggleButton"
                         Grid.Row="1"
                         Grid.Column="1"
                         HorizontalAlignment="Center"
                         VerticalAlignment="Center"
                         Content="Highlight"
                         IsChecked="{Binding CheckedValue, Mode=TwoWay}"/
```

The toggle button doesn't yet affect the highlighting. First thing we need is another collection, just of the highlighted tasks,

```
public ObservableCollection<GanttTask> HighlightedTasks { get; set; }
```

Initialize that collection in the constructor,

```
HighlightedTasks = new ObservableCollection<GanttTask>();
```

We need a method to make the highlighting happen,

```
private void ToggleHighlight(bool? amIHighlighted)
{
    if (amIHighlighted == null)
        return;
```

If the argument is null, we just return. Otherwise, we clear the collection and if the argument is true, for this demo, we'll randomly add tasks and children to the list of highlighted tasks,

```
  HighlightedTasks.Clear();

  if (amIHighlighted == true)
  {
      var rnd = new Random();
      GanttTask parentGT = Tasks[rnd.Next(0, Tasks.Count)] as GanttTask;
      int randNumber = rnd.Next(2, parentGT.Children.Count - 2);
      for (int i = 2; i < randNumber; i++)
      {
          HighlightedTasks.Add(parentGT.Children[i] as GanttTask);
      }
  }

}
```

Returning to the property CheckedValue, after we set the value we'll call ToggleHighlight,

```
ToggleHighlight(CheckedValue);
```

Now that we have a collection of the tasks we want to highlight, return to the XAML and in the definition for the GanttView we add one more attribute,

```
<telerik:RadGanttView Name="xGanttView" Grid.ColumnSpan="2"
                      TasksSource="{Binding Tasks}"
                      VisibleRange="{Binding GanttRange}"
                      PixelLength="{Binding ElementName=xRadSlider, Path=Value,
                          Converter={StaticResource xConverter}}"
                      HighlightedItemsSource="{Binding HighlightedTasks}"/>
```

Run the application, click the Highlight button and scroll down until you find the highlighted task and children, as shown in figure 6,

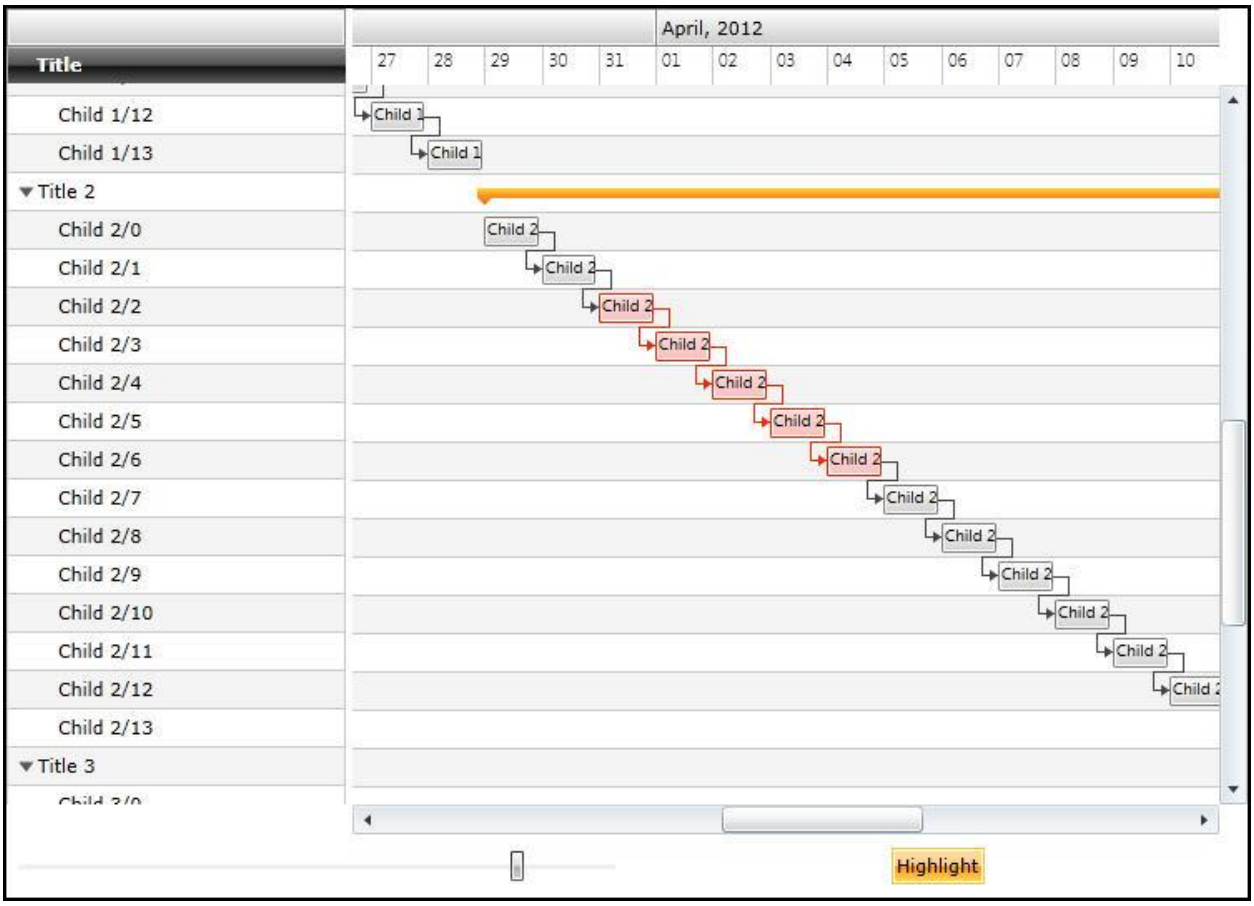

Figure 6

Clicking the toggle button a second time will make the highlighting disappear.

Custom GanttTask

Let's create a custom GanttTask so that we can add criteria to that task that does not exist in the default.

We'll create a new class called TeamGanttTask. The new class will inherit from the default GanttTask,

```
public class TeamGanttTask : GanttTask
```

We'll create the two constructors that our base class needs,

```
public TeamGanttTask() :
    base()
{}

public TeamGanttTask(DateTime startDate, DateTime endDate, string title) :
    base(startDate, endDate, title)
{}
```

Add a new property to the custom TeamGanttTask,

```
private int teamID;
public int TeamID
{
     get { return teamID; }
     set
     {
       if (teamID != value)
         {
             teamID = value;
             OnPropertyChanged(() => this.TeamID);
         }
     }
}
```

We'll use this new property to create custom business logic to control highlighting.

Add two new observable collections to the VM,

```
public ObservableCollection<TeamGanttTask> TeamTasks { get; set; }
public ObservableCollection<TeamGanttTask> HighlightedTeamTasks { get; set; }
```

Don't forget to modify the initialization in the constructor,

```
TeamTasks = new ObservableCollection<TeamGanttTask>();
HighlightedTeamTasks = new ObservableCollection<TeamGanttTask>();
```

We'll also modify LoadTasks to LoadTeamTasks,

```
//LoadTasks();
LoadTeamTasks();
```

Copy and paste LoadTasks to the new method, LoadTeamTasks and make the appropriate substitutions (e.g., TeamGanttTask for GanttTask).

Next, set the new property on the ChildGanttTask, childGT.TeamID,

```
Random rnd = new Random();
private void LoadTeamTasks()
{
    var d = new DateTime(2012, 3, 1);

    for (int i = 0; i < 4; i++)
    {
        var gt = new TeamGanttTask(d.AddDays(14 * i), d.AddDays(14 * i + 14),
            "Title " + i.ToString());

        for (int j = 0; j < gt.Duration.Days; j++)
        {
            var childGT = new TeamGanttTask();
            childGT.Start = gt.Start.AddDays(j);
            childGT.End = childGT.Start.AddHours(23);
            childGT.Title = "Child " + i.ToString() + "/" + j.ToString();
            childGT.TeamID = rnd.Next(0, 4);

            if (j > 0)
            {
                TeamGanttTask prevGt = gt.Children[j - 1] as TeamGanttTask;
                prevGt.SetRelations(new List<Relation>()
                   { new Relation() { Task = childGT } });
            }

            gt.Children.Add(childGT);

        }
        TeamTasks.Add(gt);
    }
}
```

Don’t forget to update the binding in MainPage.xaml,

```
<telerik:RadGanttView Name="xGanttView" Grid.ColumnSpan="2"
                      TasksSource="{Binding TeamTasks}"
                      VisibleRange="{Binding GanttRange}"
                      PixelLength="{Binding ElementName=xRadSlider, Path=Value,
                          Converter={StaticResource xConverter}}"
                      HighlightedItemsSource="{Binding HighlightedTeamTasks}"/>
```

Before we proceed we need a new property to hold the selectedTask,

```
private TeamGanttTask selectedTask;
 public TeamGanttTask SelectedTask
 {
     get { return selectedTask; }
     set
     {
       if ( selectedTask != value)
         {
             selectedTask = value;
```

```
            OnPropertyChanged(() => this.SelectedTask);
        }
    }
}
```

Returning to MainPage.xaml we'll use that new property for a new binding,

```
<telerik:RadGanttView Name="xGanttView" Grid.ColumnSpan="2"
                      TasksSource="{Binding TeamTasks}"
                      VisibleRange="{Binding GanttRange}"
                      PixelLength="{Binding ElementName=xRadSlider, Path=Value,
                          Converter={StaticResource xConverter}}"
                      HighlightedItemsSource="{Binding HighlightedTeamTasks}"
                      SelectedItem="{Binding SelectedTask, Mode=”TwoWay”}"/>
```

Finally, we just need to update our highlighting logic to work with our new TeamTasks. Copy and paste ToggleHighlight to ToggleTeamHighlight.

```
void ToggleTeamHighlight(bool? amIHightlighting)
{
    if (amIHightlighting == null || SelectedTask == null)
        return;
    HighlightedTeamTasks.Clear();

    if (amIHightlighting == true)
    {
        int counter = TeamTasks.Count;

        for (int i = 0; i < counter; i++)
        {
```

We iterate through the tasks and find the children where the teamID matches the selected team id,

```
TeamGanttTask tgt = TeamTasks[i];

var teamKids = tgt.Children.OfType<TeamGanttTask>().
       Where(x => x.TeamId == SelectedTask.TeamId);

foreach (var teamkid in teamKids)
{
    HighlightedTeamTasks.Add(teamkid);
}
```

Run the application and select a child. Toggle the Highlight button and all the children on the same team are highlighted as well.