# Silverlight Tree View with Multiple Levels

There are many examples of the Silverlight Tree View that you will find on the web, however, most of them only show you how to go to two levels. What if you have more than two levels? This is where understanding exactly how the Hierarchical Data Templates works is vital. In this blog post, I am going to break down how these templates work so you can really understand what is going on underneath the hood. To start, let's look at the typical two-level Silverlight Tree View that has been hard coded with the values shown below:

```
<sdk:TreeView>
  <sdk:TreeViewItem Header="Managers">
    <TextBlock Text="Michael" />
    <TextBlock Text="Paul" />
  </sdk:TreeViewItem>
  <sdk:TreeViewItem Header="Supervisors">
    <TextBlock Text="John" />
    <TextBlock Text="Tim" />
    <TextBlock Text="David" />
  </sdk:TreeViewItem>
</sdk:TreeView>
```

Figure 1 shows you how this tree view looks when you run the Silverlight application.

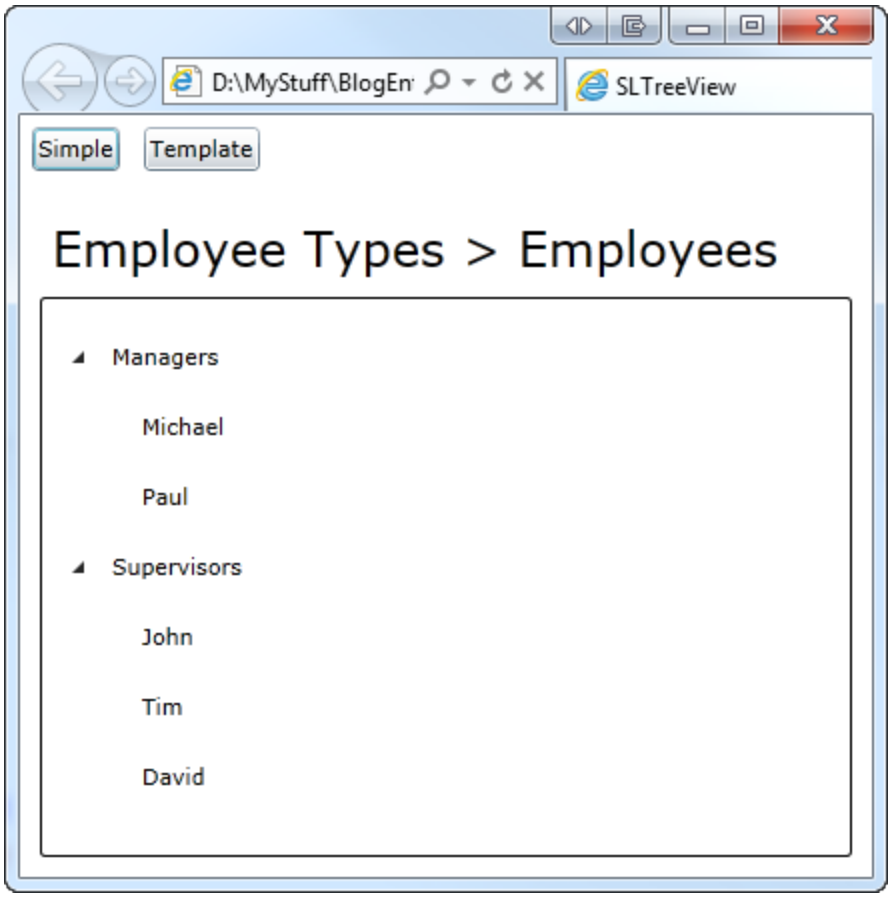


Figure 1: A hard-coded, two level Tree View.

Next, let's create three classes to mimic the hard-coded Tree View shown above. First, you need an Employee class and an EmployeeType class. The Employee class simply has one property called **Name**. The constructor is created to accept a "name" argument that you can use to set the Name property when you create an Employee object.

```
public class Employee
{
  public Employee(string name)
  {
    Name = name;
  }

  public string Name { get; set; }
}
```

Finally you create an EmployeeType class. This class has one property called EmpType and contains a generic List<> collection of Employee objects. The property that holds the collection is called **Employees**.

```
public class EmployeeType
{
  public EmployeeType(string empType)
  {
    EmpType = empType;
    Employees = new List<Employee>();
  }

  public string EmpType { get; set; }
  public List<Employee> Employees { get; set; }
}
```

Finally we have a collection class called EmployeeTypes created using the generic List<> class. It is in the constructor for this class where you will build the collection of EmployeeTypes and fill it with Employee objects:

```
public class EmployeeTypes : List<EmployeeType>
{
  public EmployeeTypes()
  {
    EmployeeType type;

    type = new EmployeeType("Manager");
    type.Employees.Add(new Employee("Michael"));
    type.Employees.Add(new Employee("Paul"));
    this.Add(type);

    type = new EmployeeType("Project Managers");
    type.Employees.Add(new Employee("Tim"));
    type.Employees.Add(new Employee("John"));
    type.Employees.Add(new Employee("David"));
    this.Add(type);
  }
}
```

You now have a data hierarchy in memory (Figure 2) which is what the Tree View control expects to receive as its data source.

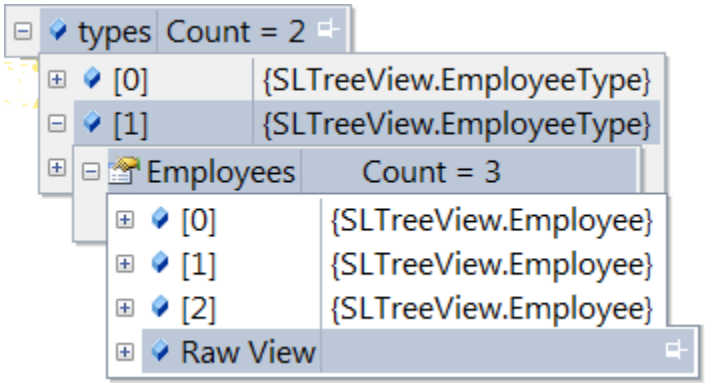


Figure 2: A hierachial data structure of Employee Types containing a collection of Employee objects.

To connect up this hierarchy of data to your Tree View you create an instance of the EmployeeTypes class in XAML as shown in line 13 of Figure 3. The key assigned to this object is “empTypes”. This key is used as the source of data to the entire Tree View by setting the ItemsSource property as shown in Figure 3, Callout #1.


Figure 3: You need to start from the bottom up when laying out your templates for a Tree View.

The ItemsSource property of the Tree View control is used as the data source in the Hierarchical Data Template with the key of **employeeTypeTemplate**. In this case there is only one Hierarchical Data Template, so any data you wish to display within that template comes from the collection of Employee Types. The TextBlock control in line 20 uses the EmpType property of the EmployeeType class. You specify the name of the Hierarchical Data Template to use in the ItemTemplate property of the Tree View (Callout #2).

For the second (and last) level of the Tree View control you use a normal <DataTemplate> with the name of **employeeTemplate** (line 14). The Hierarchical Data Template in lines 17-21 sets its ItemTemplate property to the key name of **employeeTemplate** (Line 19 connects to Line 14). The source of the data for the <DataTemplate> needs to be a property of the EmployeeTypes collection used in the Hierarchical Data Template. In this case that is the Employees property. In the Employees property there is a “Name” property of the Employee class that is used to display the employee name in the second level of the Tree View (Line 15).

What is important here is that your lowest level in your Tree View is expressed in a <DataTemplate> and should be listed first in your Resources section. The next level up in your Tree View should be a <HierarchicalDataTemplate> which has its ItemTemplate property set to the key name of the <DataTemplate> and the ItemsSource property set to the data you wish to display in the <DataTemplate>. The Tree View control should have its ItemsSource property set to the data you wish to display in the <HierarchicalDataTemplate> and its ItemTemplate property set to the key

name of the <HierarchicalDataTemplate> object. It is in this way that you get the Tree View to display all levels of your hierarchical data structure.

# Three Levels in a Tree View

Now let's expand upon this concept and use three levels in our Tree View (Figure 4). This Tree View shows that you now have EmployeeTypes at the top of the tree, followed by a small set of employees that themselves manage employees. This means that the EmployeeType class has a collection of Employee objects. Each Employee class has a collection of Employee objects as well.

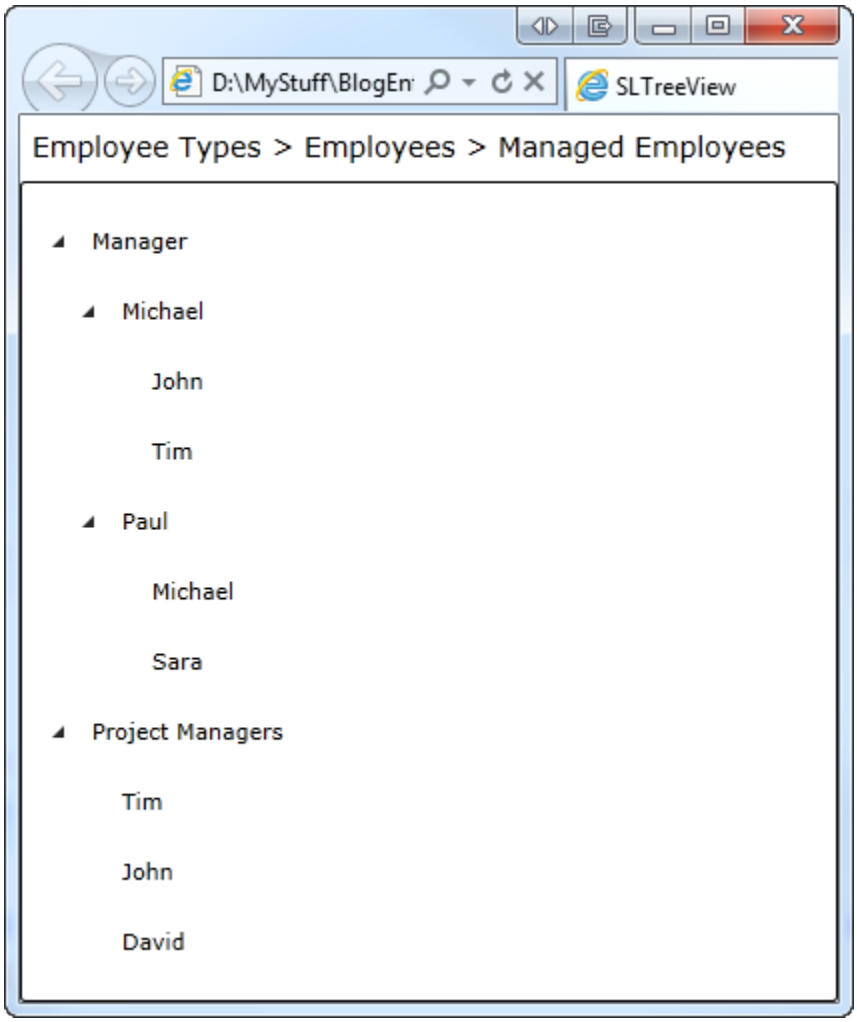


Figure 4: When using 3 levels in your TreeView you will have 2 Hierarchical Data Templates and 1 Data Template.

The EmployeeType class has not changed at all from our previous example. However, the Employee class now has one additional property as shown below:

```
public class Employee
{
  public Employee(string name)
  {
    Name = name;
    ManagedEmployees = new List<Employee>();
  }

  public string Name { get; set; }
  public List<Employee> ManagedEmployees { get; set; }
}
```

The next thing that changes in our code is the EmployeeTypes class. The constructor now needs additional code to create a list of managed employees. Below is the new code.

```
public class EmployeeTypes : List<EmployeeType>
{
  public EmployeeTypes()
  {
    EmployeeType type;
    Employee emp;
    Employee managed;

    type = new EmployeeType("Manager");
    emp = new Employee("Michael");
    managed = new Employee("John");
    emp.ManagedEmployees.Add(managed);
    managed = new Employee("Tim");
    emp.ManagedEmployees.Add(managed);
    type.Employees.Add(emp);

    emp = new Employee("Paul");
    managed = new Employee("Michael");
    emp.ManagedEmployees.Add(managed);
    managed = new Employee("Sara");
    emp.ManagedEmployees.Add(managed);
    type.Employees.Add(emp);
    this.Add(type);

    type = new EmployeeType("Project Managers");
    type.Employees.Add(new Employee("Tim"));
    type.Employees.Add(new Employee("John"));
    type.Employees.Add(new Employee("David"));
    this.Add(type);
  }
}
```

Now that you have all of the data built in your classes, you are now ready to hook up this three-level structure to your Tree View. Figure 5 shows the complete XAML needed to hook up your three-level Tree View. You can see in the XAML that there are now two Hierarchical Data Templates and one Data Template. Again you list the Data Template first since that is the lowest level in your Tree View. The next Hierarchical Data Template listed is the next level up from the lowest level, and finally you have a Hierarchical Data Template for the first level in your tree. You need to work your way from the bottom up when creating your Tree View hierarchy. XAML is processed from the top down, so if you attempt to reference a XAML key name that is below where you are referencing it from, you will get a runtime error.

```
<UserControl.Resources>
  <local:EmployeeTypes x:Key="empTypes" />
  <DataTemplate x:Key="managedEmployeeTemplate">
    <TextBlock Text="{Binding Path=Name}" />
  </DataTemplate>
  <sdk:HierarchicalDataTemplate x:Key="employeeTemplate"
                                ItemsSource="{Binding ManagedEmployees}"
                                ItemTemplate="{StaticResource managedEmployeeTemplate}">
    <TextBlock Text="{Binding Path=Name}" />
  </sdk:HierarchicalDataTemplate>
  <sdk:HierarchicalDataTemplate x:Key="employeeTypeTemplate"
                                ItemsSource="{Binding Employees}"
                                ItemTemplate="{StaticResource employeeTemplate}">
    <TextBlock Text="{Binding Path=EmpType}" />
  </sdk:HierarchicalDataTemplate>
</UserControl.Resources>
<Grid>
  <sdk:TreeView ItemTemplate="{StaticResource employeeTypeTemplate}"
                ItemsSource="{StaticResource empTypes}"
                SelectedItemChanged="tvEmps_SelectedItemChanged"
                Name="tvEmps" />
</Grid>
```

Figure 5: For three levels in a Tree View you will need two Hierarchical Data Templates and one Data Template.

Each Hierarchical Data Template uses the previous template as its ItemTemplate. The ItemsSource of each Hierarchical Data Template is used to feed the data to the previous template. This is probably the most confusing part about working with the Tree View control. You are expecting the content of the current Hierarchical Data Template to use the properties set in the ItemsSource property of that template. But you need to look to the template lower down in the XAML to see the source of the data as shown in Figure 6.

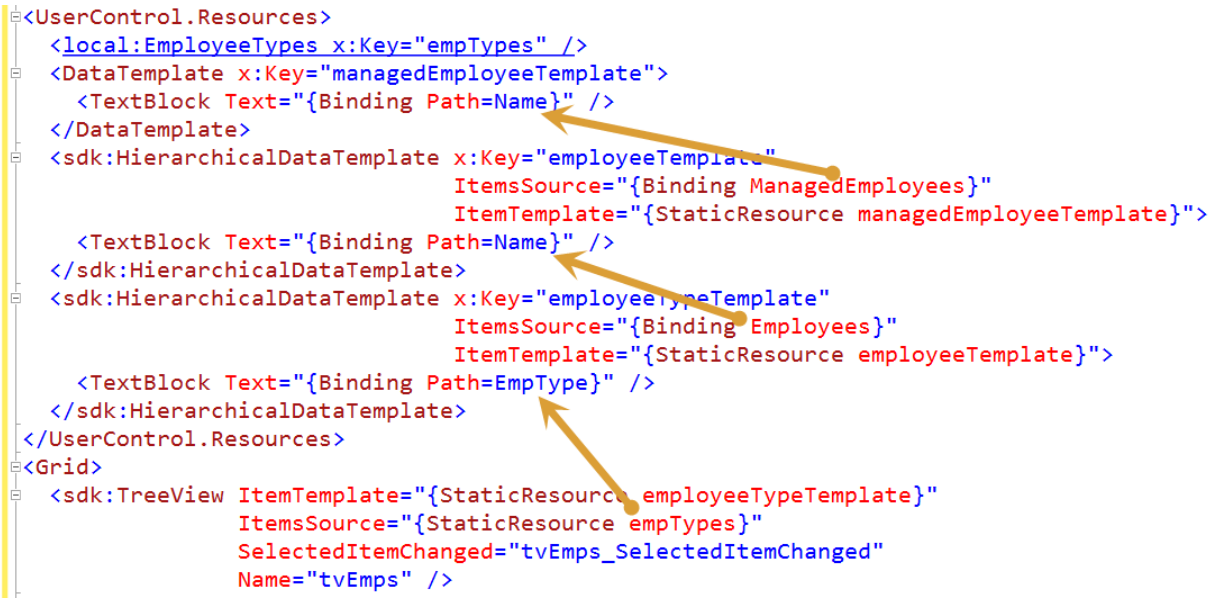


Figure 6: The properties you use within the Content of a template come from the ItemsSource of the next template in the resources section.

# Summary

Understanding how to put together your hierarchy in a Tree View is simple once you understand that you need to work from the bottom up. Start with the bottom node in your Tree View and determine what that will look like and where the data will come from. You then build the next Hierarchical Data Template to feed the data to the previous template you created. You keep doing this for each level in your Tree View until you get to the last level. The data for that last Hierarchical Data Template comes from the ItemsSource in the Tree View itself.

NOTE: You can download the sample code for this article by visiting my website at http://www.pdsa.com/downloads. Select “Tips & Tricks”, then select “Silverlight TreeView with Multiple Levels” from the drop down list.