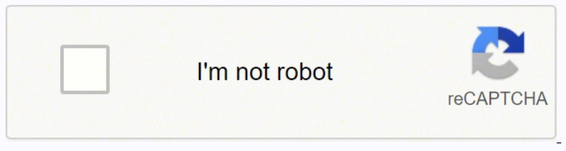

Open

Wpf hierarchicaldatatemplate expanded

Wpf treeview hierarchicaldatatemplate expanded event. Wpf treeview hierarchicaldatatemplate expanded. C# wpf hierarchicaldatatemplate expanded.

Here's the code: Select next Toggle expansion using System; using System.Collections.Generic; using System.Windows; using System.Collections.ObjectModel; using System.ComponentModel; using System.Windows.Controls; namespace WpfTutorialSamples.TreeView_control { public partial class TreeViewSelectionExpansionSample : Window { public TreeViewSelectionExpansionSample() { InitializeComponent(); List persons = new List(); Person person1 = new Person() { Name = "John Doe", Age = 42 }; Person person2 = new Person() { Name = "Jane Doe", Age = 39 }; Person child1 = new Person() { Name = "Sammy Doe", Age = 13 }; person1.Children.Add(child1); person2.Children.Add(child1); person2.Children.Add(new Person() { Name = "Jenny Moe", Age = 17 }); Person person3 = new Person() { Name = "Becky Toe", Age = 25 }; persons.Add(person1); persons.Add(person2); persons.Add(person3); person2.IsExpanded = true; person2.IsSelected = true; trvPersons.ItemsSource = persons; } private void btnSelectNext_Click(object sender, RoutedEventArgs e) { if(trvPersons.SelectedItem != null) { var list = (trvPersons.ItemsSource as List); int curIndex = list.IndexOf(trvPersons.SelectedItem as Person); if(curIndex >= 0) curIndex++; if(curIndex >= list.Count) curIndex = 0; if(curIndex >= 0) list[curIndex].IsSelected = true; } } private void btnToggleExpansion_Click(object sender, RoutedEventArgs e) { if(trvPersons.SelectedItem != null) (trvPersons.SelectedItem as Person).IsExpanded = !(trvPersons.SelectedItem as Person).IsExpanded; } } public class Person : TreeViewItemBase { public Person() { this.Children = new ObservableCollection(); } public string Name { get; set; } public int Age { get; set; } public ObservableCollection Children { get; set; } } public class TreeViewItemBase : INotifyPropertyChanged { private bool isSelected; public bool IsSelected { get { return this.isSelected; } set { if(value != this.isSelected) { this.isSelected = value; NotifyPropertyChanged("IsSelected"); } } } private bool isExpanded; public bool IsExpanded { get { return this.isExpanded; } set { if(value != this.isExpanded) { this.isExpanded = value; NotifyPropertyChanged("IsExpanded"); } } } public event PropertyChangedEventHandler PropertyChanged; public void NotifyPropertyChanged(string propName) { if(this.PropertyChanged != null) this.PropertyChanged(this, new PropertyChangedEventArgs(propName)); } } }I'm sorry for the rather large amount of code in one place. As always with programming, it's all about using the right tool for the job at hand. If you haven't read the chapters on styling yet, you might not completely understand that part, but it's simply a matter of tying together the properties on our own custom class with the IsSelected and IsExpanded properties on the TreeViewItems, which is done with Style setters. There are many solutions to tackle this problem with, and while this should do the trick, you might be able to find a solution that fits your needs better. This article has been fully translated into the following languages: Chinese French German Portuguese Spanish Vietnamese Is your preferred language not on the list? In praxis this means that you can't select or expand/collapse a given node from code-behind.Lots of solutions exists to handle this, ranging from "hacks" where you use the item generators of the TreeView to get the underlying TreeViewItem, where you can control the IsExpanded and IsSelected properties, to much more advanced MVVM-inspired implementations. You should be aware that the TreeViewItemBase class implements the INotifyPropertyChanged interface and uses it to notify of changes to these two essential properties - without this, selection/expansion changes won't be reflected in the UI. For this example, I've chosen the base class method, because it allows me to very easily get the same functionality for my other objects. Click here to help us translate this article into your language! Allow me to explain what happens in the example.XAML partI have defined a couple of buttons to be placed in the bottom of the dialog, to use the two new properties. Then we have the TreeView, for which I have defined an ItemTemplate (as demonstrated in a previous chapter) as well as an ItemContainerStyle. In this article I would like to show you a solution that lies somewhere in the middle, making it easy to implement and use, while still not being a complete hack.A TreeView selection/expansion solutionThe basic principle is to implement two extra properties on your data class: IsExpanded and IsSelected. This is what causes the second person (Jane Doe) to be initially selected and expanded, as shown on the screenshot. If this is not feasible for your solution, you could create an interface for it and then implement this instead, to establish a common ground. You can learn more about them elsewhere in this tutorial.Code-behind partIn the code-behind, I have defined a Person class, with a couple of properties, which inherits our extra properties from the TreeViewItemBase class. This works really well, but it does leave you with one problem: Because each tree node is now represented by your custom class, for instance FamilyMember as we saw in the previous article, you no longer have direct control over TreeView node specific functionality like selection and expansion state. These two properties are then hooked up to the TreeView, using a couple of styles targeting the TreeViewItem, inside of the ItemContainerStyle for the TreeView.You could easily implement these two properties on all of your objects, but it's much easier to inherit them from a base object. The TreeView control: In the previous couple of TreeView articles, we used data binding to display custom objects in a WPF TreeView. In a real world solution, it would obviously be spread out over multiple files instead and the data for the tree would likely come from an actual data source, instead of being generated on the fly. We also use these two properties on the Person objects (inherited from the TreeViewItemBase class) in the event handlers for the two test buttons (please bear in mind that, to keep the code as small and simple as possible, the selection button only works for the top level items).SummaryBy creating and implementing a base class for the objects that you wish to use and manipulate within a TreeView, and using the gained properties in the ItemContainerStyle, you make it a lot easier to work with selections and expansion states. The concept of notification changes are explained in the Data binding chapters.In the main Window class I simply create a range of persons, while adding children to some of them. I add the persons to a list, which I assign as the ItemsSource of the TreeView, which, with a bit of help from the defined template, renders them the way they are shown on the screenshot.The most interesting part happens when I set the IsExpanded and IsSelected properties on the person2 object.