Extended TreeView

A WPF Custom control

Summary

This document introduces a new custom control and describes its features. Code sample is provided to demonstrate typical usage.

Table of Contents

Introduction

The Extended TreeView custom control is a customized control for the WPF (Windows Presentation Framework) environment[1]. It aims at providing the same features as the standard TreeView control in general, as well as new features such as Drag & Drop. There are some major differences in the expected type of objects this control can display, but the How to section can help you configure the custom control to obtains a behavior similar to the standard control.

The control is provided as a signed 64-bits assembly at www.numbatsoft.com. An unsigned (and untested) 32-bits version is also available.

A demo application can be downloaded that demonstrates the control behavior when changing its properties.

Feature list

Themes

The control supports the Vista and Windows 7 themes (aero, luna, etc.). These themes are selected automatically during a standard resource selection step and no action is necessary to activate them.

Standard properties

The custom control is a class inheriting from MultiSelector, and through it, from ItemsControl. Therefore, all properties that come from these WPF classes are available in the custom control, with one notable exception.

For instance, you can use the Background property, the AlternationCount property and so on.

1- For more information about WPF, look at the Microsoft .NET Framework documentation. The control has been designed for .NET version 4.5.

ItemsSource

This property is the exception and must not be changed, or an exception will be thrown. Use the Content custom property instead (see below). This change comes from the fact that all items in the tree are defined relatively to one single root item.

Custom Control Properties

Content

Contrary to the standard TreeView control, the custom control does not expect a list of items but the root item of a tree. The Content property is where you put the reference to this root item, and any change to this property refreshes the entire tree view.

There are two ways to define the tree. The first way is to use ExtendedTreeView as the control and make all objects in the tree inherit from IExtendedTreeNode. This interface includes properties that are used to navigate the tree and you have to implement them (you can check the demo application source code for an example of how to do that).

The second way is to use ExtendedTreeViewBase as the control parent, and define your own control inheriting from it. In that case, you can use any kind of object in the tree, but you have to implement the features that navigate through the tree yourself. Take a look at the ExtendedTreeViewGeneric class for an example of how to do that.

Basically, you must provide either the data or the features that enable the custom control to navigate through the tree of objects.

Selection Mode

This property is similar to the selection mode you can find in a ListBox and should behave identically.

IsRootAlwaysExpanded

If set to true, the root item is considered always expanded and the expand/collapse button for the root is removed.

IsItemExpandedAtStart

If true, any item added to the tree (directly or indirectly by adding its parent) starts expanded. Otherwise, the item starts collapsed.

AllowDragDrop

Turn this property to true to enable drag & drop of items in the tree. Setting this property to true automatically turns the standard property AllowDrop to true as well. Therefore your tree view might start receiving notifications of attempts to drop objects from other places.

UseDefaultCursors

The custom control allows the application to override cursors used during drag & drop operations. If this property is set to true, these application cursors are ignored and the default cursors defined in the .NET framework are used. Note that the default value for this property is false, but also that the cursors are copied initially from the default system cursors, effectively behaving like if this property

was set to true.

CursorForbidden

If UseDefaultCursors is set to false (the default), the application can override this property with a custom cursor that will be displayed when the mouse is over a location where dropping is forbidden.

CursorMove

If UseDefaultCursors is set to false (the default), the application can override this property with a custom cursor that will be displayed when the mouse is over a location where dropping items will move them.

CursorCopy

If UseDefaultCursors is set to false (the default), the application can override this property with a custom cursor that will be displayed when the mouse is over a location where dropping items will copy them. To be able to copy items, all of them must inherit from ICloneable, or this feature will be disabled.

ExpandButtonWidth

Set this property to the width of the area for the expand/collapse button you want. This is normally defined through styling. The height of that area is defined by the design of the button.

ExpandButtonStyle

Set this property to change the style of the expand/collapse button, for instance to redesign its template or add events. To add or subtract setters to the default style, locate the resource with key "{x:Type ToggleButton}" (the key is not a string, but a Type object) and create a new style based on that one.

IndentationWidth

Set this property to the width you want to be added on the left of children relative to their parent. This is normally defined through styling.

HasContextMenuOpen

This is a read-only property that indicates if the control has detected that a context menu is currently open. It is used mainly by the ExtendedTreeViewItemBase class to paint the correct selection brush, but is otherwise freely usable by anyone.

Custom Control Events

In addition to usual events such as Loaded, Selected, SelectionChanged, the custom control defines new event described below.

PreviewCollectionModified

This event is triggered before a collection of children is going to be changed, either because of a user's drag & drop action, or the application programmatically adding or removing items in the tree. The CollectionModified event is triggered once the change has taken place.

Any handler for this event receives a TreeViewCollectionModifiedEventArgs object that gives the type of operation (Insert or Remove) and the number of items currently visible in the tree. This number of items must not be confused with the number of items in the tree. For instance, if the root item is collapsed, this object will always report only 1 visible item.

CollectionModified

See PreviewCollectionModified.

DragStarting

This event is the first notification of a drag & drop operation taking place. This is initiated by an action from the user.

All notifications of drag & drop operations are sent with a DragDropEventArgs object that contains the following information:

- The parent item of objects begin dragged. This cannot be null since the root itself cannot be dragged (there would be no valid destination for it).
- A boolean indicating if the drag & drop operation can result in items being copied. For this to happen, items must all inherit from the ICloneable interface (and the user may in fact decide later to just move them, so this is only an information about a restriction).
- The root item, which is also the value of the Content property.
- A structured list of items being dragged. Each item in that list also has its children dragged.
- A flat list of items being dragged. Each item in that list also has its children somewhere else in the list.

The drag operation can be canceled by any handler of the event.

DropCheck

This event is sent to the application before items are dropped, to allow the entire operation to be canceled. See also DragStarting.

In addition to other information, this event reports the object on which items are being dropped, to let the application selectively decide to allow it or not.

DropCompleted

This event is sent to the application after items are dropped (and this operation therefore cannot be canceled). See also DragStarting.

In addition to other information, this event reports the object on which items have been dropped, and if it was a copy operation, provides a list of the cloned items.

Default styles

The used styles are usually decided by the style hierarchy, in this order:

- Theme,
- Currently defined style with the type key,
- Explicit style defined by the ExpandButtonStyle and ItemContainerStyle properties.

To make things easier for developers and avoid copy/paste of setters, the ExtendedTreeViewBase type defines two static style objects, called DefaultStyle and DefaultItemContainerStyle, that contain the default style for an ExtendedTreeViewBase and ExtendedTreeViewItemBase object respectively.

It is therefore easier to rely on these default styles and the BasedOn style property when mixing tree view styles.

Control interface

In addition to properties and events, the custom control provides methods to check selected items, and to expand or collapse items. This section describes the public methods of the control.

GetVisibleItems

Returns a list of objects in the tree that are visible. This list always includes the root, and is sorted (the root which is also the top item is first, and the bottom item is last).

ScrollIntoView

This method brings an object into view, scrolling through the tree if necessary. The object must be visible or nothing happens.

IsExpanded

Tells if an object is expanded. It may not be visible if one of its parent is collapsed.

Expand

Expand the specified object. It may not change the tree aspect if one of its parent is collapsed.

Collapse

Collapse the specified object. It may not change the tree aspect if one of its parent is collapsed.

IsSelected

Tells if an object is selected by the user. Only visible objects are selectable, therefore an object can be in fact selected even if the method returns false, in the case of one of its parent is selected and collapsed.

IsItemVisible

Tells if an item is visible (all its parents are expanded).

Navigation with the keyboard

The custom control supports usual keys for navigating through a vertical list of items, and selecting some of them. It also supports the following keys:

- Right, on an expandable item, will expand it.
- Left, on a collapsible item will collapse it.

How to

Style a container

Add a style to your application's resources with a key and target type set to ExtendedTreeItemBase.

For instance:

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                    xmlns:ctrl="CustomControlsLibrary">
    <Style x:Key="{x:Type ctrl:ExtendedTreeViewItemBase}" TargetType="{x:Type
ctrl:ExtendedTreeViewItemBase}">
        <Setter Property="Background" Value="Green"/>
    </Style>
</ResourceDictionary>
```

Alternatively, set the ItemContainerStyle property of the control:

```
<Window x:Class="MyNamespace.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:ctrl="CustomControlsLibrary">
    <Window.Resources>
        <Style x:Key="MyStyle" TargetType="{x:Type ctrl:ExtendedTreeViewItemBase}"
 BasedOn="{x:Static ctrl:ExtendedTreeView.DefaultItemContainerStyle}">
            <Setter Property="Background" Value="Green"/>
        </Style>
    </Window.Resources>
    <Grid>
        <ctrl:ExtendedTreeView ItemContainerStyle="{StaticResource MyStyle}" />
    </Grid>
</Window>
```

Style the expand/collapse button

Similarly to the container, you can define a global style for ToggleButton, or use the ExpandButtonStyle property of the custom control:

```
<Window x:Class="MyNamespace.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:ctrl="CustomControlsLibrary">
    <Window.Resources>
        <Style x:Key="MyStyle" TargetType="{x:Type ToggleButton}">
            <Setter Property="Background" Value="Green"/>
        </Style>
    </Window.Resources>
    <Grid>
        <ctrl:ExtendedTreeView ExpandButtonStyle="{StaticResource MyStyle}"/>
    </Grid>
</Window>
```

Style any item

Use a DataTemplate in the application resources that decribes how your item should be displayed. Note that you can use objects from different classes in the tree, and therefore can use a separate

style for each:

```
<DataTemplate DataType="{x:Type app:MyTreeItemClass}">
    <StackPanel Orientation="Horizontal">
        <Image Source="pack://application:,,,/SomeIcon.png" VerticalAlignment="Center"/>
        <TextBlock Text="{Binding MyText, FallbackValue={x:Null}}"
VerticalAlignment="Center"/>
    </StackPanel>
</DataTemplate>
```

Hide the root

You can simply use a special class for the root object and define a style with invisible content for it.