WPF Interview Assignment: Tree View

# **Introduction**

This assignment is simple but not trivial and will take some time to complete. Completed assignment should be presented as working WPF application displaying a long list of searchable and selectable hierarchical entries.

Category1

* Item1
* Item2

Category2

* Item3
* Item26
* Item8

Category3

* Item23

Load

Filter

# Application

# **Specification**

**Application panel is a small window with three controls:**

1. tree view,
2. Load button
3. Input text box for filter string.

User can exit application by closing main window using standard windows frame button.

**Tree view is a two-level tree.**

1. Top level items are named as **CategoryXXX**, where XXX is sequential number starting from 1 (there shouldn’t be two top-level items with same number).
2. Initial number of Categories is 25
3. Categories are arranged according to the increasing category number (natural order of date generation, no need for sorting)
4. Each category has between 1 to 10 second level items named as **ItemYY**, where YY is number from 1 to 30; - like Item5, Item28, Item30. Numbers should not be continuous or even increasing – they indeed form a random sequence. All Items within same category should have different numbers (items belonging to different categories may have same number).
5. If the view cannot accommodate all items – the slider bar is added on its right edge.

**Application Startup**

Upon application start, the tree is empty

**Load Button**

Upon hitting the Load button, number of categories is increased by 25 and complete data structure is rebuilt fresh. For example, second load will result in 50 categories.

To simulate real world scenario, tree structure computation will execute asynchronously to the main thread. If new tree structure contains item that was previously selected in the UI, after load operation is complete the same item need to be selected. Items are the same if there name are the same. If such item does not exist in a new structure no item is selected in UI.

**Filter Input Box**

Upon clicking into the input box user can type any free string

Upon clicking outside input box – it becomes unresponsive to typing

As user types, tree view should refresh (immediately or after slight delay) to show only reduced set of categories and items filtered by these rules:

1. Entered text is used for ***substring case insensitive*** ***match*** against title of the categories and items. For example:
   1. input texts “Ca”, “go”, “ego” matches all categories
   2. Input texts “e”, “Te” matches all items and all categories
   3. Input text “RY2” matches Category2, Category23, Category238, and so on
   4. Input text “tEm3” matches Item3, Item30
2. If input text matches some of the categories:
   1. all branches respecting to those categories are displayed and fully opened showing all their items
   2. The Category name is displayed in GREEN
   3. All items are displayed in BLACK
3. If input text matches some of the items (say input “em2”):
   1. Each category containing matching item is shown on reduced tree
   2. Among second level items only those that match the typed string are shown (say Item2, Item23)
   3. Other items that doesn’t match the string (say Item7, Item14, Item30) are not shown
   4. Matching items are shown in GREEN
   5. Hosting category is shown in BLACK
4. If input string matches both – categories and items within same category (say string “e”):
   1. Both rules are applied in parallel
   2. Categories that match the string shown in GREEN, others in BLACK
   3. Items that match the string are shown in GREEN, others in BLACK

**Selection on the tree control**

1. Clicking anywhere within tree control causes selection of the item lined up with the cursor position
2. Selected item should be highlighted
3. By default, WPF tree view control highlights ***only*** text (content) of a selected item. In this application, selected category or item **should be highlighted from left to right rim of the tree** view box as shown on illustration). This is called ***full line highlight***.

# **Solution Notes**

* **UX**

Demonstrate best practices in resource and theme/styles organization

Bonus is given for elegant styling of the application beyond standard WPF Luna theme

* **MVVM pattern**

Demonstrate that you can structure application using MVVM pattern and put minimal (if at all) logic in a code behind. Use your preferred framework (e.g. [MVVM Light](http://www.mvvmlight.net/), Caliburn, etc) or use support classes provided in a blank project (*ObservableBase* and *DelegateCommand*).

* **Full line highlight**

There are several ways to implement full line highlight. Advantage is given to a more reusable/structured way that works with arbitrary hierarchy.

* **Filtering large data sets**

Filtering large data sets can be perceived by user as slow. You are not required to optimize perceived performance. However, bonus is given if this problem is solved too.