You Can Push it Around in Your... ShoveBoxList

Mark Libbrecht

WIN 6.0 DOWNLOAD

Reading Alan Cooper's book *The Inmates are Running the Asylum* made Mark Libbrecht reflect on the use of the tree control. Alan says a lot of people have problems using the tree control—especially the drilling-down mechanism. Mark agrees, and since he writes programs for users and not for programmers, he thought there had to be a better way to manipulate those types of hierarchical data sets. In this article, Mark describes a new control he's developed to replace the tree control.

CONTROL No. 1 for storage and lookup of hierarchically structured data is without a doubt the tree control. In my opinion, its greatest quality is the possibility to keep a global overview by hiding the data that the user isn't interested in. Getting to the needed data is done by drilling down into the tree.

Personally, I very often use hierarchical structured data sets, especially bills of materials. Besides the actual items (offered, purchased, or billed items), these data sets also contain descriptive titles, structured in a hierarchical manner. Two specific characteristics of these data sets for our company's use are that they're relatively small (our offers generally contain between 10 and 100 rows) and that the user is interested in all of the nodes.

When I came across a kind of pitfall game, it inspired me to create a control in which items could be moved horizontally, thereby changing their hierarchical level. If you haven't seen this game, it consists of several flat plastic rectangular bars, one aligned above the next. Each bar has holes in it and a lever at both ends. The bars can be maneuvered in several positions, whereby in some positions the holes in one bar align with other holes that are made in the surface beneath the bars. The players have to place a small ball in the bars on the underlying surface and can shove the bars into a position so that the ball of the opponent falls into the hole in the surface.

I created a control consisting of boxes that I programmatically aligned into a kind of list. The boxes snap to an invisible grid, allowing the boxes to be moved horizontally (for the time being, vertical movement isn't allowed—that functionality is performed by cutting and pasting). The horizontal movement (shoving) changes the level of the element that's associated with the box. I used the ActiveX Slider control to mimic the elevator in a classical list or grid control. And behold, the ShoveBoxList was born! (See **Figure 1**.)

Use of the ShoveBoxList

The ShoveBoxList has the advantage of not needing to drill down, as is the case with the tree control. Creating a special control where the items are contained in separated objects offered the possibility to customize the formatting. The ShoveBoxList behaves a lot like a multi-column list or grid as far as vertical movement is concerned. Check boxes play the role of record marks and allow selection for deletion, cut and paste operations, and change of level.

The manipulation of the ShoveBoxList is quite straightforward. Several ways of manipulating the list are available, as shown in the following code. As with the keyboard, the Ctrl key in combination with the normal keys allows control movement in the following manner:

```
Ctrl +  → Decrease of the level (higher value)
Ctrl +  ← Increase of the level (lower value)
Ctrl +    ↑ To the previous shovebox
Ctrl +    ↓ To the next shovebox
Ctrl + PageUp Previous group of shoveboxes
Ctrl + PageDown Next group of shoveboxes
```

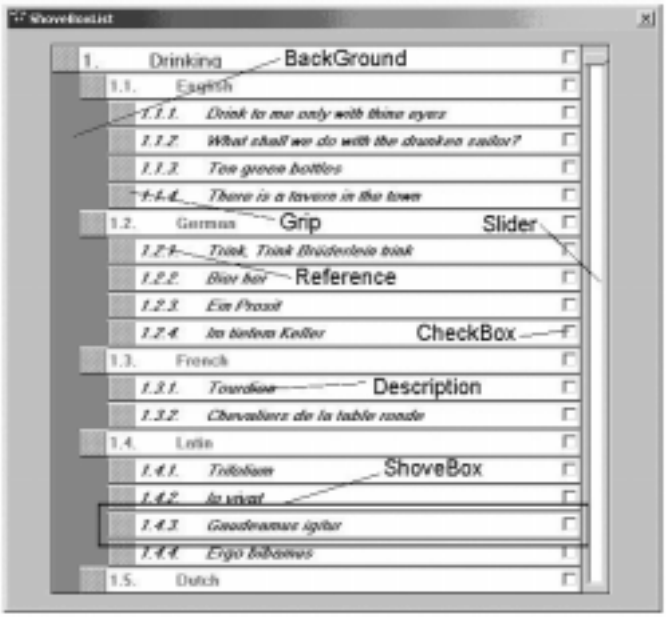

Figure 1. The appearance of the ShoveBoxList and its components.

Dragging or double-clicking (only decreasing the level) on the grip can also be used to change the level (horizontal shoving of the shoveboxes). The mouse is used in the same manner as in the case of a grid or multi-column including the mouse wheel, the latter being my preferred scrolling method. Right-clicking the mouse evokes the shortcut menu shown in **Figure 2**. This menu is subclassed from the VFP foundation class _menu. It permits item selection (check boxes), deletion, insertion, and cutting and pasting. Since I often forget whether inserting and putting page breaks in spreadsheets is done before or after the active row, I explicitly implemented a separate menu option. The option reset level changes the level of all items to the default level that's determined by the noffsetlevel property of the ShoveBoxList control.

The description and reference fields are edit boxes and text boxes, respectively, and can be simply filled in.

Classes

The different classes the ShoveBoxList control is composed of figure in the UML class diagram (class.html), which can be evoked by clicking the command button of the sample form Shoveboxlisttestform.scx. The actual control is an instance of the mlshblcontrol container class. It contains a background shape, a slider (the subclassed ActiveX Slider), a shortcut menu (a subclass of VFP's _shortcutmenu base class), and two datamanagers (based on the VFP custom class). One datamanager, mlshblformatdatamanger, manages the data that's needed to format the shoveboxes. The other, the mlshblrowdatamanager, manages the actual data of the shoveboxes.

At instantiation, the formatdatamanager adds a FormatData object that holds the individual Format objects: one for each level. In the same way, the rowdatamanager adds a RowData object that holds the Row objects, but it also adds an IndexData object that holds the Index objects and that's used to indirectly address the Row objects.

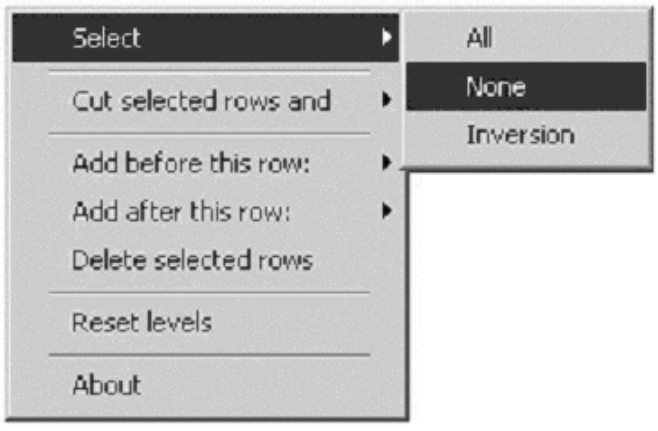

Figure 2. The shortcutmenu of the control invoked by right-clicking.

At instantiation of the ShoveBoxList control, the maximum number of shoveboxes that can be visualized at the same time is determined. This depends upon the height of the control and the shovebox row. This number of shoveboxes is added to the ShoveBoxList even if there aren't enough Row objects to fill all of the shoveboxes. In this case, these shoveboxes are made invisible. During scrolling, no additional shoveboxes are created. The existing ones get other values.

The shovebox (control class) is composed of a grip image, a reference object (text box), a description object (edit box), and a check box.

Data

The data and the datamanagers that are provided in the code use native VFP tables. The generic datamanagers are very basic. They're made for this example only and have to be adapted (or replaced) in order to hook in into the framework the developer is using.

The provided databasecontainer data1.dbc contains a stored procedure function called NewID. This function is called upon by the the sampledata table to determine the default value and realize an autoincrement for the keyfield nkey.

Formatting and special effects

Every time the level of an item is changed, the appearance (fontname, fontsize, forecolor, fontbold, fontitalic) of the reference and description are adapted according to values that correspond to the new level and that can be found in the levelformats.dbf file. For every level, this file contains the formats that correspond to that level. Once the control gets instantiated, the formatdatamanager creates a FormatData object that contains these values and stays in memory.

If the lsound property is set to .T., the shoving of the boxes produces a sound. The sound itself is determined by the value of the property vsound. If vsound is a numerical value, the messagebeeps function will be used with vsound as the parameter. If vsound is a character value and it points to a WAV file, that file will be played when the boxes are shoved. ▲

DOWNLOAD **04LIBBSC.ZIP at www.pinnaclepublishing.com/ft**

Mark Libbrecht is co-owner and CEO of a small electrical contracting company in As (written with only one *s* <g>), Flanders (the Dutch speaking part of Belgium—Europe). He has a university degree in electromechanical engineering from the Catholic University of Leuven Belgium (established 1425) and an MBA. Writing applications in Visual FoxPro has been a hobby of his. He enjoys creating applications for internal use. mark@libbrecht.be.