Mergeable Cell Listbox API Documentation Version 1.1

Karen Atkocius keatk@verizon.net

Table of Contents

1. Introduction/Overview

In my past life often had to create tables to display/report/manage data. My tool of choice was a spreadsheet because it was both simple to do calculations, as well as lay out tables that was easy to read/understand.

One of the most useful tools for creating easily readable layouts was the ability to merge cells. This enabled data to be organized logically, clearly and to use space efficiently.

I missed that most when I tried to use the Xojo listbox for data presentation. This project is the result of that frustration. The main focus of these classes is to provide as comprehensive as possible support for merging cells across rows and columns, with a secondary objective to provide other commonly needed functionality not natively supported by the Xojo control.

This was accomplished by subclassing the standard Xojo listbox and is written entirely in Xojo. The documentation of the standard Xojo listbox is not repeated here. Only new or changed methods, properties and events the subclasses implement are included. Please refer to Xojo's documentation for standard listbox behavior.

I tried to design the extensions to be as consistent as possible with the Xojo listbox API (which is very flexible but not simple) as I could. However because a merged cell may cross both rows and columns one must take that into consideration when coding using the standard listbox APIs.

For example if a cell is merged across rows, in an event, just because listbox.Selected(row) is false does not mean a cell the row intersects is not selected! You would need to check using the new method SelectedCell(row,column) to be sure.

The demo's provided are not simple as they are examples of the type of real world UIs that merged cells make possible and make extensive use of both the standard and new listbox API's. However the documentation provided and that code should be enough to get started.

Besides merging and unmerging cells, additional convenience methods/functions are included. The most significant addition is support for more sophisticated cell text wrapping than you can do with g.DrawString using the wrapwidth parameter. Only whole lines that fit in the cell will be drawn, and an ellipsis added to the last line if all he text does not fit. You can also specify line spacing and vertical positioning within the cell.

A subtle new feature is that analogous to CellTextPaint, in CellBackgroundPaint g.forecolor is set to the expected background color before being passed to the event. As in TextPaint, if you change that and not return true, the color you set will be used for the background. In addition, regardless of what you return, that background color will be used by an algorithm to set the text color in CellTextPaint to either white or black to maximize readability on whatever background color you use.

For hierarchical listboxes you may optionally have lines drawn connecting the nodes . In addition there is support use the keyboard or mouse to optionally open a folder or all subfolders of a folder at once.

There are a number of other minor enhancements as well which are documented here.

All of the classes required are in ListBoxModule, with some being private to the module. This was done to "hide" classes only used internally.

There are 2 main public classes:

- ListBoxClass which contains all the enhanced listbox functionality EXCEPT that directly related to merging cells
- ListboxMergable which contains all the code/methods/properties related to merged cells and is a subclass of ListBoxClass

2. Version 1.1 Changes

2.1. New Functionality:

1) Getter and Setter: CellStyle(row As Integer, column As Integer) As CellStyle
   Allows setting/getting all style properties a of Cell (Border, Aligmment, Type , Bold etc) using an instance of the CellStyle class

2) CellDefinition(row As Integer, column As Integer) As CellDefinition
   Same as above except also includes tag , state and text. CellDefinition is a subclass of CellState

2.2. Updates to support new Xojo Listbox functionality and bug fixes:

1) For the ListboxMergeable subclass, the CellPaint events no longer have FullWidth and FullHeight parameters. The dimensions provided by graphics object passed in (g) are now correct for merged cells

2) The code to get the listbox highlight color was updated for Cocoa to get the correct color (previously Carbon declares were used)

3) A bug where after editing a cell the focus was not always returned to the listbox was fixed.

4) In Xojo 2014r3 new functionality was added to the standard hierarchical Xojo Listbox :

   - RowIsFolder(row) : Boolean setter and getter determining if a row is a folder
   - RowDepth(row) as Integer: Used to determine indent level of the row

   These classes supported some of this functionality for prior versions of RealStudio/Xojo under different names.

   To be consistent with the new functionality for these Classes the method :

   - IsFolder(row as integer) as Boolean was renamed RowIsFolder to be consistent with the new function. In addition a RowAsFolder setter that works on all versions of RealStudio/Xojo supported by these classes.
   - IndentLevel(row) as integer was renamed to RowDepth(row as Integer) as Integer

5) For hierarchical listboxes, several bugs with drawing the nodes connecting lines were fixed including:

   - A big which resulted in exceptions if line drawing was turned off in certain cases was fixed.
   - Fixing the InhibitConnector method to work in all cases

3. Public Classes

3.1. Class: ListboxModule.ListboxClass Super: ListBox

3.1.1. Features:

1) Supports a background color for the listbox as well as alternating row colors via the property pane
2) Optionally supports wrapping text in a cell. Does **NOT** use g.DrawString with the WrapWidth parameter. Only displays lines that are fully visible and if all text is not displayed last line will have an ellipsis even if that line fits (end of a paragraph)
3) Support for multi-line cell editing on all platforms
4) For multi-line text, supports specifying line spacing and line by line horizontal alignment
5) Allows aligning text (single or multi-line) vertically as well as horizontally
6) Provides enhanced cell text attribute and border assignment methods
7) Hierarchical Listbox support:
   - Uses the standard Xojo listbox hierarchical API.
   - Optionally draws lines connecting folders and their children
   - Enhanced UI for Expanding/Collapsing Folders:
     - Option Click on a closed disclosure widget will open all sublevels
     - SpaceBar toggles disclosure widget. If closed Option-SpaceBar will open all sublevels
     - Option-UpArrow closes an open folder. Option-DownArrow opens a folder. If closed, Shift-Option-DownArrow will open all sublevels
   - Provides methods to:
     - Determine if a row is a folder
     - Return the indent level of a row
     - Open all sublevels of a folder
     - Close all open folders
     - Inhibit the drawing of the horizontal connector line to a child row. This is useful for visual grouping when related data is presented in multiple child rows at the same indent level. In other words data for one logical node is displayed in multiple consecutive rows. (See Lot Testing Demo)

3.1.2. Addtional/Changed Properties:

WrapByDefault As Boolean = True

Settable in the IDE for the listbox. If True text will wrap in all cells as needed. If False, the text will be displayed on a single line. See the CellTextPaint Event to change this behavior for individual cells

BackColor As Color

If AltBackColor not specified it is the background color for all unselected cells. If black (&c000000) then ignored. If AltBackColor is specified then it is the background color for odd numbered rows only

AltBackColor As Color

Background color for even numbered unselected rows. If black (&c000000) then is ignored

DrawNodeLines As Boolean = True

If True, in a hierarchical listbox, lines will be drawn connecting a parent folder to it's child nodes.

NodeLineColor As Color = &cA2A2A2

In a hierarchical listbox this is the color used to draw the lines connecting child rows to their parent folder.

Computed **isActive** As Boolean

Returns True if Listbox has the focus and the window is active

3.1.3. Additional/Changed Public Methods:

Shared Function **ContrastingColor**(InputColor as Color) As Color

Returns the color (either white or black) for which text should be most readable when displayed on an InputColor background.

Shared Function **BackgroundColor**(isSelected as Boolean,
Active as Boolean) As Color

Returns the standard listbox background color. If IsSelected is true the color is the standard selected highlight color. Active determines if the color is that of an active listbox or not. Active here means the listbox has the focus and the window is active. Updated to use Cocoa declares for Cocoa builds

Function **BackgroundColor**(Row as integer, isSelected as Boolean,
Active as Boolean) As Color

Returns appropriate background color for the rowconsidering the values of the BackColor and AltBackColor properties

Sub **CellAlignment**(row as integer, column as integer,
Assigns Alignment as Integer)

Sub **CellAlignmentOffset**(row as integer, column as integer,
Assigns Offset as Integer)

Sub **CellBold**(row as integer, column as integer,
Assigns Value as Boolean)

Sub **CellItalic**(row as integer, column as integer,
Assigns Value As Boolean)

Sub **CellUnderline**(row as integer, column as integer,
Assigns Value As Boolean)

If row = -1 then all cells in that column for all rows are assigned that value for that cell attribute. If column is -1 then all cells in the specified row are assigned that value. If both are -1 then that attribute is set for all cells in the listbox.

Sub **CellBorderTop**(row as integer, column as integer,
rows as integer = 1,columns as integer = 1,
Assigns BorderValue As Integer)

Sub **CellBorderBottom**(row as integer, column as integer,
rows as integer = 1, columns as integer = 1,
Assigns BorderValue As integer)

Sub **CellBorderLeft**(row as integer, column as integer,
rows as integer = 1, columns as integer = 1,
Assigns BorderValue As Integer)

Sub **CellBorderRight**(row as integer, column as integer,
rows as integer = 1, columns as integer = 1,
Assigns BorderValue As Integer)

These routines make it easier to draw multi-cell borders. If the optional parameters are specified the borders extends that number of cells to the right or down from the specified cell.

If row = -1 then all cells in that row are assigned that border value and rows is ignored. If column is -1 then all cells in the column are assigned that value and columns is ignored. If both row and column are -1 then that border is set for all cells in the listbox and both optional parameters are ignored.

Function **CellStyle**(row As Integer, column As Integer) As CellStyle

Sub **CellStyle**(row As Integer, column As Integer,
Assigns Data As CellStyle)

Gets and Sets a Cell Style with an instance of the CellStyle which is defined as :

Class CellStyle

Properties

Alignment As Integer

AlignmentOffset As Integer

Bold As Boolean

BorderBottom As Integer

BorderLeft As Integer

BorderRight As Integer

BorderTop As Integer

Italic As Boolean

Type As Integer

Underline As Boolean

End Class

Function **CellDefinition**(row As Integer, column As Integer) As CellDefinition

Sub **CellDefinition** (row As Integer, column As Integer,
Assigns Data As CellDefinition)

Gets and sets all of the values for a cell. CellDefinition is a subclass of CellStyle that adds the following properties:

State As CheckBox.CheckedStates

Tag As Variant

Tag As Variant

Function **RowIsFolder**(row as integer) As Boolean

If true then the row is a folder. In prior versions was named **IsFolder**. Note: The setter form is now supported for all versions of RealStudio and Xojo that these classes support.

Function **RowDepth**(row As Integer) As Integer

Returns the indent level of a row in a hierarchical ListBox. In prior versions was named **IndentLevel**.

Sub **InhibitConnector**(row As Integer)

For a hierarchical ListBox inhibits the drawing of the horizontal connector line to a child row. May only be called from the ExpandRow event. This is sued for logically grouping multiple child rows into a single node visually.

Sub **ExpandFully**(row As Integer = -1)

For a hierarchical ListBox, if the row is a folder, expands not only that level but all sublevels as well. A value of -1 expands all levels for all rows in the listbox

Sub **CollapseAll**()

For a hierarchical ListBox Collapses all open folders

3.1.4. New/Changed Events:

Event **CellBackgroundPaint**(g as graphics, row as integer,
column as integer) As Boolean

Analogous to the behavior of CellTextpaint, g.forecolor is set to the expected background color before the event thus called making it available to be used or changed in the event.

Changing it in the event will result in changing the background color if you do not return true. Regardless of what is returned, changing it MAY effect TextColor (this is the current default behavior - see class notes).

In that case, if you need to change it within the event for drawing, but do not wish to have it effect the text color, cache g.forecolor in a local variable at the beginning of the event and restore that value before returning

Event **CellTextPaint**(g as graphics, row as integer, column as integer,
x as integer, y as Integer,
ByRef VertAlign As Integer,
ByRef LineSpacing as integer) As Boolean

g.forecolor is set before the event is called to draw text and always takes into consideration the use of custom background colors to ensure the text will be readable on it. It MAY also take into account a custom background color that is set in CellBackgroundPaint. (See Notes)

X,Y: Position for text drawing, the values are incorrect for multi-line text

VertAlign determines the vertical alignment of the text of the cell. Values are:

Listbox. AlignDefault = 0 (Center)

ListboxModule.ListboxClass.AlignTop = 1

Listbox. AlignCenter = 2

ListboxModule.ListboxClass.AlignBottom = 3

LineSpacing is the spacing between text lines in pixels.

A value of: ListboxModule.ListboxClass.NoWrap = -1 (default)
Prevents line wrapping

Event **MakeCellTag**(row as Integer, column As Integer) As ItemInfo

Used internally in conjunction with the overridden CellTag setter and getter methods to support subclassing the listbox where the subclass requires the use of CellTags but still needs to make the normal CellTag API available transparently. (see notes for more details)

Event **MakeRowTag**(row as Integer) As RowInfo

Used internally in conjunction with the overridden RowTag setter and getter methods to support subclassing the listbox where the subclass requires the use of RowTags but still needs to make the normal RowTag API available transparently. (see notes for more details)

Event **SupressTextPaint**(row as Integer,column As integer) As Boolean

For internal Use Only

3.1.5. Class Notes

Hierarchical Support:

The use of InsertRow and InsertFolder with the indent parameter specified is not supported

Paint Events and Text Color:

By default the background color (which you may set in the CellBackgroundPaint or by using the BackColor properties) is cached and used to calculate a text color which will be readable on that background without you having to worry about it.

The ability to account for a change of background color in text drawing depends on the undocumented event order of the listbox paint events. It assumes that for a given cell CellTextPaint will fire after CellBackgroundPaint without CellBackgroundPaint firing for any other cell inbetween. While I believe this is highly unlikely to ever change, it may in a future version of Xojo.

If you do not wish to depend on this feature, change the constant ListboxModule.UseCachedBkgColor to False. Then while text color will still be adjusted for the 2 Backcolor properties, if you change the background color in the CellBackgroundPaint event, you will be responsible for ensuring that any text will be readable on it, by putting complementary code in CellTextPaint.

CellTags And RowTags:

A subclass of this class that you create may require the use of CellTags and/or RowTags, yet still need to have CellTags and RowTags available with the standard API. To enable this both methods have been overridden.

To do this one must create a subclass of the classes and implement the events shown below. If you wish to enable subclasses of THAT class to have the same capability, you should add that specified event to that class and call to it in the implementation of the event. Hopefully the information below will make that clear.

RowTags:

A subclass which requires the internal use of Rowtags should implement the new event:

```
Event MakeRowTag(row as Integer) As RowInfo
```

RowInfo is the required superclass for the subclass that will hold your class specific information which the event returns. That subclass maybe as simple as just adding a single property to store class specific row data.
If you wish to allow subclasses of your class to support custom row data transparently while maintaining the capability to support RowTags, you should define a new event of exactly the same form in the class. For such a situation your code should look something like:

```
Function MakeRowTag(row as Integer) As RowInfo
   Dim Info as RowInfo = MakeRowTag(row)
   If Info Is NIL then
       Return New MyRowInfoSubclass
   Else
       Return Info
   End if
End Function
```

There are protected methods for use internally in subclasses to support this mechanism:

Protected **RealRowTag**(row As integer) As/Assigns Variant

Since Super.RowTag only calls the parent method, not that of the superclasses further up the chain, these methods are protected wrappers for the Listbox.RowTag methods to be able to get the actual RowTag when required

Protected Function **GetAssignRowTagSubclass**(row As integer) As RowInfo

This methods first retrieves the true RowTag for that row. If it is a

RowInfo, that is returned. If not, it calls MakeRowTag to get an instance. If that is nil, it creates an instance of RowInfo. In ether case it saves the “true” tag value in that instance.

Once you have retrieved an instance of your RowInfo subclass by either method you may assign or read class specific data to it by casting to that class.

CellTags:

A subclass which requires the internal use of CellTags should implement the new event:

```
Event MakeCellTag(  row as Integer, column as integer) As ItemInfo
```

ItemInfo is the required superclass for the subclass which the event returns that will hold your class specific information. That subclass may be as simple as just adding a single property to store class specific cell data. If you wish to allow subclasses of your class to support custom cell data transparently, while maintaining the capability to support CellTags, you should define a new event of exactly the same form in the class. For such a situation your code should look something like:

```
Function MakeCellTag(row as Integer, column as integer) As ItemInfo
    Dim Info as ItemInfo = MakeCellTag(row,collumn)
    If Info Is NIL then
        Return New MyItemInfoSubclass
    Else
        Return Info
    End if
End Function
```

There are also protected methods for use internally in subclasses to support this mechanism:

Protected **RealCellTag**(row As integer, column as integer)
As/Assigns Variant

Since Super.CellTag only calls the parent method, not that of the superclasses further down the chain, these methods are protected wrappers for the Listbox.CellTag methods to be able to get the actual CellTag when required

Protected Function **GetAssignCellTagSubclass**(row As integer,
column as integer)
As ItemInfo

This methods first retrieves the true CellTag for that row, if it is a RowInfo that is returned. If not it calls MakeCellTag to get a new instance. If that is nil, it creates an instance of Item and in ether case it saves the “true” tag value in that instance.

3.2. Class: ListboxModule.ListboxMergable **Super: ListBoxClass**

3.2.1. Additional Features:

1) Supports all new features of the ListboxBoxClass above
2) Merging and unmerging of cells across both rows and columns
3) Column resizing across merged areas is supported
4) Editing merged cells
5) Using a Vertical Scrollbar for in-line editing in any cell
6) Rows may be inserted or deleted within merged area that cross rows. The merged area is either expanded or contracted as expected.
7) Several methods added for dealing with merged cells
8) The listbox may be hierarchal and rows which are folders may be merged as long as the folder is the top row or last row. (see class notes)

3.2.2. Additional Public Properties:

ActiveCell As TextEdit

It was necessary to shadow Listbox.ActiveCell with a computed property to support the editing of merged cells and having scrollbars while editing any cell. Be aware that if you cast a ListBoxMergable to listbox you will not have access to the TextArea used for merged cell through this property. Also the TextEdit returned is only guaranteed to return correct object in the CellGotFocus event and when one is actually editing a cell.

CurrentRow As integer
CurrentColumn As integer

When cells are merged across rows an columns, in the CellPaint events the row and column passed in are those of the upper right cell of the group. Sometimes in paint events it is helpful to know the true cell being drawn. That information is available in these properties during those events.

3.2.3. Additional Public Methods:

Sub **MergeCells**(TopRow as Integer, LeftColumn as integer,
Rows as integer, Columns as integer)

TopRow, LeftColumn: Top left cell of area to be merged
Rows, Columns: Numbers of rows and columns to merge

The text and text attributes of the top/left cell are used for the whole merged area. When the specified area overlaps an existing merged area, the result is a single rectangular merged area that encompasses both regions. In that case the text and text attributes of the originally specified top left cell will be used even if the area is merged with another. Border styles: The top and left borders of the top left cell become the borders of those edges. The right border of the right most top cell becomes the right border for the merged area and the bottom border of the bottom left cell becomes the bottom border.

Sub **UnMergeCells**(row as Integer, column as Integer)

Unmerges a merged area if the cell at row and column is within the merged area. If row = -1 then all merged areas that have cells within the specified column are unmerged. If column = -1 then all merged areas that have cells within the specified row are unmerged. If both = -1 then all merged areas in the listbox are unmerged

Cell Border Getters and Setters:

For the CellBorder methods for merged areas, when specifying an individual cell (row and column > -1) can set the cell border by either using the row and column for the top left cell, or any cell on the correct outer edge for that border type.
Attempting to set the border properties for other cells within the merged area, will have no effect. When setting borders for multiple cells using -1 for rows or columns or the optional parameters, only those cells that fall on the appropriate OUTER BORDER only will be changed and all on that outer border will be changed if one is.

Cell Text/Text Attribute Getters and Setters:

For merged cells getting cell text and other cell text attribute, for cells internal to a merged cell will get or set the value for the merged area as a whole.

Function **SelectedCell**(row As integer, column As Integer) As Boolean

Returns true if that cell is selected (row and column may be internal to the merged area). A merged cell is “selected”/highlighted if any row that intersects it is highlighted. This means Listbox.Selected(row) can not be used to determine if a merged cell is selected. Use this function instead

Function **CellInfo**(row as integer, column As integer)
As ListboxModule.CellInfo

Returns information about the ‘size of cell’ and if it’s selected. All cells internal to a merged cell will report the same information

Function **CellHeight**(row as integer, column As integer) As Integer

Returns the cell ‘height’ in rows. All cells internal to a merged cell will report the same information

Function **CellWidth**(row as integer, column As integer) As Integer

Returns the cell ‘width’ in columns. . All cells internal to a merged cell will report the same information

Function **CellTop**(row as integer, column As integer) As Integer

Returns the row of the top right cell of a merged area. All cells internal to a merged cell will report the same information

Function **CellLeft**(row as integer, column As integer) As Integer

Returns the column of the left most cell of a merged area. All cells internal to a merged cell will report the same information

Function **NextCell**(row as Integer, column as integer, Direction As Integer,
DoInvalidate as Boolean = False) As Integer

Returns the top row or left column of the next cell in the direction specified skipping over the internal cells of merged area and optionally invalidates that cell. Used when iterating over a row or column.
Direction constant values are:

ListboxMerged.Direction_Right = 0

ListboxMerged.Direction_Left = 1

ListboxMerged.Direction_Up = 2

ListboxMerged.Direction_Down = 3

Sub **InvalidateCell**(row as Integer, column as Integer)

Extended to invalidate all cells within a merged area

Sub **InvalidateRow**(row as Integer)

Invalidates all cells within row a row including all cells of any merged areas that include the row

Sub **InvalidateSelectedRows**()

Same as invalidate row but applied to all currently selected rows

3.2.4. Additional/Changed Events:

Event **EditingMode**(row as Integer, column as Integer)
As ListboxModule.EditMode

This event, along with the configuration(s) of the MergedLBEF instances used if any (See Class notes), determines if a specific cell should be edited using a scrollbar or not. This event is called at the start of editing before, BEFORE the CellGotFocus event.

Returns ENUM ListboxModule**.EditMode** which has the following values:

EditMode.Default	: Edit Without Scrollbar if possible
EditMode.WithVerticalScrollBar	:Edit With Scrollbar if possible
EditMode.Custom :	Edit All Cells using an MergedLBEF instance if possible, and without scrollbar if available.

The Class notes contain a full explanation of the interaction of cell editing with MergedLBEF instances and this event.

Event **CellBackgroundPaint**(g as graphics,row as integer,
column as integer) As Boolean

For Merged Cells g.Width and g.height is the sum of the actual column widths and row heights for the merged cells. This does not take into account partially visible cells at edges of the listbox. It will all ways be sum of the full size of the merged cells.

For unmerged cells the values are the same as g.width and g.height and do take into considerations clipping at box edges.

Event **CellTextPaint**(g as graphics, row as integer, column as integer,
x as integer, y as Integer,
ByRef VertAlign As Integer,
ByRef LineSpacing as integer) As Boolean

Same considerations in terms dimension except that it is only for the area text may be drawn into.

```
Event CompareRows( Row1 as integer, Cell1Left as integer,
                   Row2 as Integer, Cell2Left as Integer,
                   MergedState as Integer, column as integer,
                   ByRef result As integer) As Boolean
```

Sorting gets complicated very quickly. There is no way to support sorting of cells merged across rows. However sorting single rows which have cells merged that do not cross rows is possible, though the logic would be very case specific. To support this possibility the CompareCells has been modified.

Row1, Row 2 , column and result are the normal values passed into the event. Cell1Left and Cell2Left are the left most cells of the merged area. If the cell is not part of a merged area they will always be the same as column, however being the same does not guarantee that the cell is part of merged area. That is handled by MergedCells parameter

Merged Cells values:	0 = neither cell is merged
	1 = the row1 cell is merged
	2 = the row2 cell is merged
	3 = Both are merged

3.2.5. Class Notes:

Inserting/Deleting Rows:

Inserting row into the interior of a merged area will increase or decrease it’s size as expected.

Editing Merged and Unmerged Cells:

You can always edit unmerged cells , but for editing of merged cells to be possible, at least one an instance of the subclass ListboxModule.MergedLBEF must be placed on top of the listbox at design time. This gives added flexibility because if you wish you could subclass MergedLBEF giving you more control over editing than is possible with the normal listbox.

This design decision also makes editing configuration a little more involved. The added complexity could have been hidden using a container control, but that would prevented using MergedLBEF subclasses.

How editing of cells takes place depends on both the instances of MergedLBEF on the listbox and the value returned by the EditingMode event. Editing merged cells always uses an instance of MergedLBEF. If one is not paced on the listbox you will NOT be able to edit merged cells.

Editing of non merged cells **MAY** use an instance of MergedLBEF if one is present.

To be able to edit cells using a vertical scrollbar one of the MergedLBEF instances on the Listbox must have MergedLBEF.ScrollbarVertical set to true in the IDE. If that is only instance of MergedLBEF present, the all merged cells will be edited using a scrollbar. Editing of non merged cells still depends on the value returned by the EditingMode event. In this case returning EditMode.WithVerticalScrollBar or EditMode.Custom

For the EditingMode event to always control if a cell being edited has a scrollbar or not, 2 instances of MergedLBEF must be placed on the Window and one should have MergedLBEF.ScrollbarVertical set to true in the IDE, and the other should have it set to False. In that case returning EditMode.WithVerticalScrollBar will cause both merged and unmerged cells to be edited

If there is only one instance of MergedLBEF on the listbox, if it is set to have a vertical scrollbar in the IDE, all merged cells will be edited using a vertical scrollbar regardless of what the EditWithVerticalScrollbar event returns, however the event result would still control if non merged cells had a scrollbar during editing. If ScrollBarVertical was not enabled for the single instance of MergedLBEF then no cell would have a scrollbar during editing and non merged cells would just use standard listbox TextEdit object.

Having more than one instances of MergedLBEF with the same ScrollBarVertical setting will not cause an error. All but the last will be ignored

Hierarchical Support:

If cell are merged across rows, folders may be in the top or bottom row and only one merged area should exist on the row. Folders should not be inserted internally in to a merged area.

UseFocusRing:

This may be set to true freely on the Macintosh as the focus ring is external to the listbox. On other platforms it should only be set to true if cells are not merged across rows as the focus ring is by row and will be seen internally in the merged area.

Unsupported Listbox Features:

DragReorderRows

4. Utility Modules

4.1. Module: KEA_Wrap

KEA_Wrap is module some methods useful for wrapping and drawing/ “flowing” text. The Listbox classes do not require it be present however they may useful for drawing multi-line text in merged cells.

4.1.1. Global Methods:

Function **ContrastingTextColor**(InputColor as Color) As Color

Returns the color (either white or black) for which text should be most readable when displayed on an InputColor background.

Function **WrapTextLines**(Extends g as graphics, Text as String,
Width As Integer = 0) As String()

Returns a String array which contains the text wrapped to the specified optional width. If width is omitted the Width of the graphics context is used. This is analogous to using g.DrawString with the WrapWidth parameter but allows you more flexibility. As with DrawString with condense = false, no ellipsis is added as no height is specified.

Function **WrapTextBlock**(Extends g as graphics , Text as String,
Width as Integer, Height As Integer,
LineSpacing As Integer = 0) As String()
Function **WrapTextBlock**(Extends g as graphics , Text as String,
LineSpacing As Integer = 0) As String()

Returns a string array which contains the text wrapped to a specific width and height. Does NOT draw the strings. If the form is used that omits width and height, the width and height of the graphics context is used. In this case if the text would not fit in the specified space the last line will have an ellipsis. LineSpacing allows you to specify the space between lines in pixels if used.

```
Function FlowTextBlock( Extends g as graphics, Text as String,
                        x as integer, y as Integer,
                        Width as Integer, Height as integer,
                        AlignH As integer = 0, AlignV As Integer = 0,
                        LineSpacing as integer = 0) As String()
Function FlowTextBlock( Extends g as graphics, Paragraphs() as String,
                        x as integer, y as Integer,
                        Width as Integer, Height as integer,
                        AlignH As integer = 0, AlignV As Integer = 0,
                        LineSpacing as integer = 0) As String()
Function FlowTextBlock( Extends g as graphics , Text as String,
                        AlignH As integer = 0, AlignV As Integer = 0,
                        LineSpacing as integer = 0) As String()
Function FlowTextBlock( Extends g as graphics , Paragraphs() as String,
                        AlignH As integer = 0, AlignV As Integer = 0,
                        LineSpacing as integer = 0) As String()
```

These routines draw the passed text wrapped into a specified rectangular area (block) at a specified position and return a string array that contains the remaining paragraphs of text that did not fit in the block. The text is input is either in the form of a string or a string array containing the text divided by paragraphs. The remaining text is always returned as an array of paragraphs.

The first 2 forms require explicit specification of block position (x, y) and size (width, height) within the graphics context. The second two forms assume you wish to start at 0,0 and use the entire width and height of the graphics context. This would be most useful for clipped areas. These routines are useful if you want to flow text into different sized blocks (see Demo)

AlignH and AlignV are the horizontal (line by line) and vertical (by block) text alignments. The values are the same as for the listbox classed above, except that decimal alignment is not supported.

Line Spacing is the spacing between lines in pixels.

```
Sub DrawTextBlock(Extends g as graphics, Text as String,
                  x as integer, y as Integer,
                  Width as Integer, Height as integer,
                  AlignH As integer = 0, AlignV As Integer = 0,
                  LineSpacing as integer = 0)
Sub DrawTextBlock(Extends g as graphics, Paragraphs() as String,
                  x as integer, y as Integer,
                  Width as Integer, Height as integer,
                  AlignH As integer = 0, AlignV As Integer = 0,
                  LineSpacing as integer = 0)
Sub DrawTextBlock(Extends g as graphics , Text as String,
                  AlignH As integer = 0, AlignV As Integer = 0,
                  LineSpacing as integer = 0)
```

These methods are the same as the FlowTextBlock methods except that if the text does not fit entirely within the block, the last line will have an ellipsis. The remaining text is not returned, These methods are for use when the block specified is the last/only one and so needs to be ended.