SUMMARY

vListMngr is a package of Pascal routines which implement creation and management of Lists with variable column width, and up to four lines of header text per column. The column headers behave like spreadsheet headings, staying fixed on the screen while the cells scroll vertically, and tracking the columns when cells are scrolled horizontally. vListMngr also provides utilities for multiple cell selection and copying and editing cell text. The programming interface to vListMngr is very similar to that of the standard List Manager, and existing programs will require very little modification to incorporate the new features. However, this version of vListMngr does not support some standard List Manager capabilities, such as custom LDEF's, the Search function, or selection flags.

The archive contains a 68K demo program, documentation, and THINK Pascal source code for vListMngr and the demo program. The package was written to provide flexible entry of fairly large tables of data for scientific instrument control systems and has been tested on several 68000 - 68040 machines under System 6, System 7.1, and System 7.5. The code may be used, modified, and distributed freely provided that no profit is derived therefrom and the author is duly credited.

INTRODUCTION

The Macintosh OS List Manager is a very handy tool for displaying and manipulating relatively large amounts of data in a table format. However, the standard List Manager has limitations which make it ineffective in certain cases. Only fixed size data cells can be used, which makes display of data elements which vary significantly in size rather awkward. Standard Lists lack headings which let the user keep track of what the various columns and rows in the list represent. There is also no built-in mechanism for editing cell contents. The latter two problems can be overcome by supplying routines which coordinate display of headings with the list display, and routines which keep Text Edit record(s) coordinated with the list selection(s), but these are cumbersome programming tasks. vListMngr was written because I needed to be able to enter large amounts of tabular data in programs for scientific instrument control systems, and I needed variable-width columns and column headings to produce a functional user interface. Built-in support of text editing and provision for multi-cell copy/paste were added for convenience.

THE CODE

I started with the data structures and routine descriptions for List Manager given in Inside Macintosh. New data fields were added and code was written to implement the described routines. Although the code is completely original, the functionality and syntax of the standard List Manager calls has been preserved wherever possible. vListMngr can therefore support most existing List Manager code fairly transparently, and only the additional capabilities needed for the application need to be learned and used. However, not all List Manager calls have been implemented in this version of vListMngr (see routine descriptions below). In addition, the selection algorithms, user LDEF's, and user LClickLoop routines are NOT supported. All the routines available are listed below, but only the new or altered calls are described in detail. Please refer to Inside Macintosh for descriptions of the standard List Manager calls. The unimplemented calls are present as stubs, so they will compile but will simply return without taking any action. The code is sparsely commented and gives some clue to what is going on.

SUPPORT/COMMENTS

As will no doubt be apparent from the code, I am not a professional programmer. I will gladly provide any assistance within my capabilities, including bug fixes, improvements, and extensions to the package. The demo has been tested on a number of 68K machines under system 7.1 and 7.5. The routines don't do anything fancy, so they should perform appropriately in other environments. Please send communications to:

Jeff Brewster
jbrewster@arserrc.gov

USING vListMngr

Minimal Operation

1. Create a new vListRecord by calling vLNew.
   The list created will have uniform cell dimensions, will use the existing font, and will have no headings or built-in Text Edit record.
2. Process mouse clicks in the list or its scroll bars by calling vLClick.
3. Process update events in the window by calling vLUpdate.
4. When done with the list call vLDispose

Full Operation

1. Create a new vListRecord by calling vLNew.
   The list created will have uniform cell dimensions, will use the existing font, and will have no headings or built-in Text Edit record.
2. Set up the headings by calling vLSetHeadings.
3 Set the list and heading fonts with calls to vLSetListFont and vLSetHeadFont.
4. Set the heading widths by calling vLSetWidths OR let vListMngr calculate the cell widths by calling vLCalcCellWidths before calling vLSetWidths.
5. Associate a TextEdit Record with the selected cell by calling vLTENew.
6. Call vLNewScrap to create a new vLScrapHandle to support cutting and pasting.
7. Process mouse clicks in the list by calling vLClick.
8. Process keypresses in the window by calling vLKey
9. Process update events in the window by calling vLUpdate.
10. When done with the list call vLDispose

Summary of the vListMngr Package

vListMngr Data

const
```
    maxCols = 31;
    maxChars = 16000;
    maxOffsets = 4095;
```

type
```
    Cell = Point;

    EditModeType = (editCell, editTE);

    DataArray = packed array[0..maxChars] of CHAR;
    DataPtr = ^DataArray;
    DataHandle = ^DataPtr;

    OffsetArray = array[0..maxOffsets] of INTEGER;

    lHeadArray = array[1..4] of STR255;

    WidthArray = array[0..maxCols] of INTEGER;
```

```
modifierType = (none, command, option, control, shift);

vLScrapHandle = ^vLScrapPtr;
vLScrapPtr = ^vListScrapRec;
```

```
vListScrapRec = record
     scrapBounds: RECT;
     scrapData: dataArray;
     scrapOffsets: offsetArray;
 end;

vListHandle = ^vListPtr;
vListPtr = ^vListRec;
vListRec = record
     rView: RECT;                    {rectangle in which list heading and are viewed}
     port: GrafPtr;                  {Grafport that owns us}
     indent: Point;                  {Indent pixels in cell}
     cellSize: Point;                {Cell width and height (width is generally ignored)}
     visible: RECT;                  {visible row/column bounds}
     vScroll: ControlHandle;         {vertical scroll bar (or NIL)}
     hScroll: ControlHandle;         {horizontal scroll bar (or NIL)}
     selFlags: SignedByte;           {defines selection characteristics}
     LActive: Boolean;               {active or not}
     LReserved: SignedByte;          {internally used flags}
     listFlags: SignedByte;          {other flags}
     clikTime: LONGINT;              {save time of last click}
     clikLoc: Point;                 {save position of last click}
     mouseLoc: Point;                {current mouse position}
     lClikLoop: Ptr;                 {routine called repeatedly during ListClick}
     lastClick: Cell;                {the last cell clicked in}
     refCon: LONGINT;                {reference value}
     listDefProc: HANDLE;            {Handle to the defProc}
     userHandle: HANDLE;             {General purpose handle for user}
     dataBounds: RECT;               {Total number of rows/columns}
     cells: DataHandle;              {Handle to data}
     maxIndex: INTEGER;              {index past the last element = length of cellArray}
     cellArray: OffsetArray;         {offsets to elements  up to 32 cols and 128 rows}
     cellWidth: WidthArray;          {array of cell widths}
     lView: RECT;                    {Rect in which list alone is viewed}
     frameWidth: INTEGER;            {width of frame around cells}
     just: INTEGER;                  {text justification mode}
     lFont, lSize: INTEGER;          {list font info}
     lFace: Style;                   {list font info}
     nhRows: Integer;                {# rows in heading}
     lHead: lHeadArray;              {heading text}
     hFont, hSize: INTEGER;          {heading font info}
     hFace: Style;                   {heading font info}
     hCellHeight: INTEGER;           {height of heading cell}
     listTE: TEHandle;               {edit TE}
     lActiveTE: BOOLEAN;             {is the TE active?}
     lEditMode: BOOLEAN;             {are we editing a TE?}
 end;
```

Creating and Disposing of Lists

Standard

```
FUNCTION        vLNew        (plView, pdatabounds: RECT; cellSize:
                POINT; procID: INTEGER; theWindow: WindowPtr;
```

```
    drawIt, hasGrow, scrollHoriz, scrollVert: Boolean): vListHandle;
    PROCEDURE                              vLDispose     (vlHandle: vListHandle);
```

Adding and Deleting Rows and Columns

Standard

```
    FUNCTION                               vLAddRow      (count, rowNum: Integer; vlHandle:
                                           vListHandle): Integer;
    PROCEDURE                              vLDelRow      (count, rowNum: Integer; vlHandle:
                                           vListHandle);
```

Unimplemented

```
    FUNCTION                               vLAddColumn        (count, colNum: Integer; vlHandle:
                                           vListHandle): Integer;
    PROCEDURE                              vLDelColumn (count, colNum: Integer; vlHandle:
                                           vListHandle);
```

Operations on Cells

Standard

```
    PROCEDURE                              vLClrCell     (theCell: Cell; vlHandle: vListHandle);
    PROCEDURE                              vLGetCell     (dataPtr: Ptr; var dataLen: Integer; theCell:
                                           Cell; vlHandle: vListHandle);
    PROCEDURE                              vLSetCell     (dataPtr: Ptr; dataLen: Integer; theCell: Cell;
                                           vlHandle: vListHandle);
    FUNCTION                               vLGetSelect   (next: Boolean; var theCell: Cell; vlHandle:
                                           vListHandle): Boolean;
    PROCEDURE                              vLSetSelect   (setIt: Boolean; theCell: Cell; vlHandle:
                                           vListHandle);
```

New

```
    PROCEDURE                              vLCalcCellWidths     (var newWidth: widthArray; var
                                           nCol: INTEGER; vlHandle: vListHandle);
    PROCEDURE                              vLSetHeadings        (nRows: INTEGER; headings:
                                           lHeadArray; vlHandle: vListHandle);
    PROCEDURE                              vLSetWidths   (widths: widthArray; vlHandle:
                                           vListHandle);
    PROCEDURE                              vLKey (ch: CHAR; modifiers: Integer; vlHandle:
                                           vListHandle; vScrap: vLScrapHandle);
    PROCEDURE                              vLTENew       (active: BOOLEAN; whatCell: Cell;
                                           vlHandle: vListHandle);
    PROCEDURE                              vLTEDispose  (vlHandle: vListHandle
```

Unimplemented

```
    PROCEDURE                              vLAddToCell  (dataPtr: Ptr; dataLen: Integer; theCell: Cell;
                                           vlHandle: vListHandle);        ***
```

Mouse Location

```
    FUNCTION                               vLClick       (pt: Point; modifiers: Integer; vlHandle:
                                           vListHandle): Boolean;
    FUNCTION                               vLLastClick   (vlHandle: vListHandle): Cell;
```

Accessing Cells

Standard

```
    FUNCTION            vLNextCell   (hNext, vNext: Boolean; var theCell: Cell;
                        vlHandle: vListHandle): Boolean;
    PROCEDURE           vLFind(var offset, len: Integer; theCell: Cell; vlHandle:
                        vListHandle);
    PROCEDURE           vLRect(var cellRect: Rect; theCell: Cell; vlHandle:
                        vListHandle);
```

New

FUNCTION	vLEncloseSel (vlHandle: vListHandle): Rect;
PROCEDURE	vLCellsToScrap (whatCells: RECT; vlHandle: vListHandle; hScrap: vLScrapHandle);
PROCEDURE	vLScrapToCells (whatCells: RECT; vlHandle: vListHandle; hScrap: vLScrapHandle);
Unimplemented	
FUNCTION	vLSearch (dataPtr: Ptr; dataLen: Integer; SearchProc: Ptr; var theCell: Cell; vlHandle: vListHandle): Boolean;

List Display

Standard	
PROCEDURE	vLActivate (act: Boolean; vlHandle: vListHandle);
PROCEDURE	vLDoDraw (drawIt: Boolean; vlHandle: vListHandle);
PROCEDURE	vLDraw (theCell: Cell; vlHandle: vListHandle);
PROCEDURE	vLScroll (dRows, dCols: Integer; vlHandle: vListHandle);
PROCEDURE	vLSize (listWidth, listHeight: Integer; vlHandle: vListHandle);
PROCEDURE	vLUpdate (theRgn: RgnHandle; vlHandle: vListHandle);
New	
PROCEDURE	vLDrawHeading (vlHandle: vListHandle);
PROCEDURE	vLFont(myFont, mySize: INTEGER; myFace: Style; vlHandle: vListHandle);
PROCEDURE	vLFrame (frameWidth: INTEGER; var vlHandle: vListHandle);
PROCEDURE	vLIndent (indent: POINT; var vlHandle: vListHandle);
PROCEDURE	vLInsetList (dH, dV: Integer; vlHandle: vListHandle);
PROCEDURE	vLJust (just: INTEGER; var vlHandle: vListHandle);
PROCEDURE	vLUpdateSelRect (selRect: Rect; vlHandle: vListHandle);
Unimplemented	
PROCEDURE	vLAutoScroll (vlHandle: vListHandle);

ROUTINE DESCRIPTIONS

Creating and Disposing of Lists

FUNCTION vLNew (plView, pdatabounds: RECT; cellSize: POINT; procID: INTEGER; theWindow: WindowPtr; drawIt, hasGrow, scrollHoriz, scrollVert: Boolean): vListHandle;

This function returns a handle to a new vList record. The list font defaults to theWindow's font, and the number of heading rows is set to 0. If cellSize.h is 0, the cell width is calculated by dividing the width of plView by databounds.right. If cellSize.v is 0, the cell height is calculated from the font information.

Operations on Cells

PROCEDURE vLCalcCellWidths (var newWidth: widthArray; var nCol: INTEGER; vlHandle: vListHandle);

vLCalcCellWidths returns in nCol the number of columns and in newWidths the widths of columns as calculated from the current heading string(s) (set previously by vLSetHeadings). The calculation is based on parsing the heading string into substrings using the '|' character as a delimiter. The width of each substring in pixels is determined using the StringWidth function and the current heading font, and the resulting value is returned as the width of the respective column. If a substring is blank, then the width of that column is not changed from its previous value. Thus headings can be supplied to only a few columns. If there are multiple heading rows, the width of the widest row in the column is used. If an integer n is the first non-space character of a heading substring, that substring is centered over n columns in the row(s) beneath. Otherwise the heading is centered in its column.

For example, the following heading strings:

```
TTHead1 = 'Time  |        2Power      |       5Valves           |';
TTHead2 = ' min   | Volts | mAmps | A | B | C | D | E |';
```

are parsed to yield this heading:

```
Time       Power          Valves
min    Volts   mAmps    A  B  C  D  E
```

The values returned by vLCalcCellWidths generally give an attractive appearance to the table. To actually change the column widths it is necessary to call vLSetWidths with the returned array.

```
PROCEDURE   vLSetHeadings         (nRows: INTEGER; headings: lHeadArray; vlHandle:
            vListHandle);
```

Sets the heading strings and number of heading rows appropriately. The top of lView is recalculated so as to fit the heading into rView. The list and headings are redrawn.

```
PROCEDURE   vLTENew       (active: BOOLEAN; whatCell: Cell; vlHandle: vListHandle);
```

Creates a new TERec associated with the list and used to edit cell contents. If active is TRUE, then the contents of whatCell are displayed in and cell editing is enabled.

```
PROCEDURE   vLTEDispose  (vlHandle: vListHandle);
```

Disposes of the TERec associated with the list and sets listTE to nil.

```
PROCEDURE   vLKey (ch: CHAR; modifiers: Integer; vlHandle: vListHandle; vScrap:
            vLScrapHandle);
```

vLKey handles keypresses in the list, performing editing functions (copy, cut, paste) on selected cells and/or list navigation (return, enter, tab, arrow keys). If a TERec is associated with the list, and cell text editing is activated, then most keypresses are passed to TEKey. In this mode, return or enter cause the contents of the TERec to be transferred to the active cell, and selection of the next cell in the list. In this mode, tab and arrow keys cause the original contents of the cell to be restored, followed by appropriate navigation activity.

Mouse Location

```
FUNCTION    vLClick       (pt: Point; modifiers: Integer; vlHandle: vListHandle): Boolean;
```

vLClick handles mouse-down events in the list and its scroll bars, returning TRUE if there was a double click in a cell, and FALSE otherwise. Cells are selected according the the current mouse position as follows:

Shift Click	Add clicked cell to existing selection.
Shift Drag	Extend selection to the currently selected cell.
Option Click	Select the clickedrow.
Control Click	Select the clicked column.

If there is a TERec associated with the list (listTE <> nil) then editing of of cell contents is activated by double clicking in a cell. Once activated, further clicks in the cell are passed to TEClick. Clicking outside the active cell terminates editing, restores the original cell contents, and selects the clicked cell.

Accessing Cells
FUNCTION vLEncloseSel (vlHandle: vListHandle): RECT;

vLEncloseSel returns the smallest rectangle which encloses all selected cells. This is useful for processing multiple cell selections outside of vListMngr.

PROCEDURE vLCellsToScrap (whatCells: RECT; vlHandle: vListHandle; hScrap: vLScrapHandle);

vLCellsToScrap copies the data bounded by whatCells to hScrap. The transferred data are within whatCells, i.e. whatCells.right is 1 greater than rightmost cell index. The cell data are not neccesarilly contiguous. This routine is generally called when a Cut or Copy command is being processed on a group of selected cells. It can also be used to transfer data from a list into a file for storage.

PROCEDURE vLScrapToCells (whatCells: RECT; vlHandle: vListHandle; hScrap: vLScrapHandle);

vLScrapToCells copies the data in hScrap to the the cells bounded by whatCells. The transferred data are within whatCells, i.e. whatCells.right is 1 greater than rightmost cell index. This routine is generally called when a Paste command is being processed on a group of selected cells. It can also be used to transfer data from a file into a list.

List Display
PROCEDURE vLDrawHeading (vlHandle: vListHandle);

Draws the heading(s) using the current heading font and style.

PROCEDURE vLFont(myFont, mySize: INTEGER; myFace: Style; vlHandle: vListHandle);

Set the List font.

PROCEDURE vLFrame (frameWidth: INTEGER; var vlHandle: vListHandle);

Set the width of the frame drawn around cells. The frame rectangle is drawn so that
the top and left edges are "inside" the cell rectangle, while the bottom and right edges begin one pixel "outside" the cell rectangle. The view rectangle should be sized to allow drawing of the frame to get the best appearance.

PROCEDURE vLIndent (indent: POINT; var vlHandle: vListHandle);

Set the cell indent.

```
PROCEDURE    vLInsetList      (dH, dV: Integer; vlHandle: vListHandle);
```

Change the size of the display rectangle by dH, dV. Similar to vLSize, but simpler to use when resizing windows.

PROCEDURE   vLJust (just: INTEGER; var vlHandle: vListHandle);

Set the justification mode for cell text.

PROCEDURE   vLUpdateSelRect   (selRect: Rect; vlHandle: vListHandle);

Selects all cells bounded by selRect and unselects all other cells in the list. Redraws any cells which change select status}