ABOUT THIS CHAPTER

This chapter describes the extended list manager, written by Jack Herrington of the University of Miami School of Medicine. Which supports row and size functions for all cells, fonts support for the entire list or by cell, different handlers for different cells which allows for *multiple LDEFs*, and many other features

ABOUT THE ENHANCED LIST MANAGER

The enhanced list manager supports all of the features of the original with these exceptions:

- It does not support standard list manager LDEFS (they are replaced).
- It's scroll bars are measured in pixels and not in cells.
- When you give the new list call a box it creates the scrollbars *inside* the box, instead of on the outside.
- When the list is deactivated the scrollbars maximum is set to zero to better conform to the Human Interface Guide specifications.
- It doesn't support the standard clikLoop field, but adds something better.

It does however support these new features:

- Individual sizes for every column and row.
- Flags on every cell and on the entire list to define how the cell should be boxed (on the right, left, top or bottom, and in grey or black)
- Flags on every cell and on the entiere list for font, size, and face.
- Flags on every cell to tell which routine should handle the drawing, a DrawString call, a TextBox call, a TEUpdate call, or one of thirteen definable userRoutines.
- Simpler updating on the entire list
- A built-in framing utiltity to frame the list.
- X and Y offset that are added onto the list to make mandatory blank space (good for lists where the order of the items is changable).
- A built-in frame around the list with scrolling titles.
- A built-in routine to resize cell boundaries on a hit on a cell side.
- A routine to let you know what part of the list is being hit before you hit it.
- Built-in textEdit on every cell, intelligent enough to keep running through cell resizes and moves.
- Built-in character parsing to make sure the user doesn't type text into a number field or however you want it.
- Switch to make the autoScrolling either cell by cell or incremental (the farther the pointer is away from the list the more the list scrolls).

You should be familiar with the original list manager before embarking to use this ones features.

ENHANCED LIST RECORDS

The enhanced list manager has a list record which is absolutely nothing like the original. It includes:

- The font, size and face of the entire list.
- The boxing parameters for the list.
- The display box for the list.
- The x and y slush amount.
- The choice of whether to frame it.
- The choice of whether to draw with every set command.
- The column and row widths.
- The column and row starting positions.
- Handles to the scrollbars.
- The handler defintions.
- The identity of the cell just above and left of the upper left-hand corner.
- The identity of the cell in the bottom right-hand corner.
- A rectangle defining the visible cells.
- The data boundaries.
- A handle to the cells.

In general you shouldn't mess around with this, you should let the routines, but hey!

The enhanced list structure

```
typedef struct
{
    Rect                visBox;                 /* Box */
    int                 visBoxXLength;          /* X Length of box */
    int                 visBoxYLength;          /* Y Length of box */
    int                 visXLength;             /* X Length in pixels */
    int                 visYLength;             /* Y Length in pixels */

    int                 xAddative;              /* Addition to X length */
    int                 yAddative;              /* Addition to Y length */

    int                 frameX;                 /* Frame width */
    int                 frameY;                 /* Frame height */

    int                 drawIt;                 /* Draw flag */

    ListFontSpec        fontSpecs;              /* Font specs */
    int                 boxFlags;               /* Boxflags */

    int                 columnWidth[MaxCol];    /* Width of column */
    int                 columnStart[MaxCol];    /* Start of the column */
    unsigned char       columnSel[MaxCol];      /* Column selected flag */
    int                 columnFil[MaxCol];      /* Column filter */

    int                 rowWidth[MaxRow];       /* Width of row */
    int                 rowStart[MaxRow];       /* Start of the column */
    unsigned char       rowSel[MaxCol];         /* Row selected flag */
```

```
    void                (*frameDraw)();         /* Frame drawing routine */

    ControlHandle       XScroll;                /* X Scroll */
    ControlHandle       YScroll;                /* Y Scroll */

    Boolean             active;                 /* Active */

    Handler             handler[MaxHand];       /* Handlers */

    Point               firstCell;              /* First cell visible */
    Point               lastCell;               /* Last cell visible */
    Rect                visArea;                /* Visible area */

    int                 dataXMax;               /* X Amount of cells */
    int                 dataYMax;               /* Y Amount of cells */
    Handle              cells;                  /* Cell data */

    char                selFlags;               /* Selection flags */
    char                listFlags;              /* Scrolling flags */

    long                lastClickTime;          /* Last click time */
    Point               lastClickPos;           /* Last click position */

    char                autoScrollMethod;       /* Auto scroll method */

    TEHandle            cellEditText;           /* TextEdit handle to cell */
    Point               cellEdit;               /* Cell being edited */
} NewList,*NewListPtr,**NewListHandle;
```

The cells are just an array of dataXMax * dataYMax and they include this information:

- Font specifications for the cell.
- The boxing information for the cell.
- The number of the handler.
- A flag to indicate selection.
- A flag to indicate whether the data handle was allocated by the list manager (in response to a SetCell call).
- A handle to the data.

The Cell Structure

```
typedef struct
{
    SmallListFontSpec   fontSpecs;              /* Font specifics */
    int                 boxFlags;               /* Boxing flags */
    unsigned char       handler;                /* Handler */
    unsigned char       selected;               /* Selected flag */
    unsigned char       dataLocal;              /* Local creation flag */
    Handle              data;                   /* Data */
} NewCell,*NewCellPtr;
```

This method is really costly in handles and a set width, set font list probably should be done in the original, but this system adds a lot more flexilibility.

USING THE ENHANCED LIST MANAGER PACKAGE

This is the same as in the original list manager.

ENHANCED LIST MANAGER PACKAGE ROUTINES

One of the most important rules to follow is that all the new routines have the same names as the old ones (and the same parameters) except that the new ones have replace the 'L' in the beginning with 'List' (which makes more sense). So for instance 'LNew' is 'ListNew'.

Creating and Disposing of Lists

NewListHandle ListNew(Rect *rView,Rect *dataBounds,Point cSize,int theProc,WindowPtr theWindow,int drawIt,int hasGrow,int scrollHoriz,int scrollVert); **(original)**

This creates a new list inside 'rView'. The manager uses the bottom and right values of 'dataBounds' to get the maximum rows and columns. The 'cSize' gives the strarting width and height of the cells. The 'theProc' variable is ignored (simply as a matter of taste). The 'theWindow' window is used when the manager creates the scrollBars. The 'drawIt' variable tells whether drawing is on or off. The 'hasGrow' is ignored. The 'scrollHoriz' and 'scrollVert' decsribe where the list is going to be scrollable.

void ListDispose(NewListHandle listHandle); **(original)**

This kills a list (obviously it's much faster than deleting one row at a time!)

Adding and Deleting Rows and Columns

int ListAddColumn(int count,int colNum,NewListHandle listHandle); **(original)**

Adds 'count' columns starting at 'colNum', with that column's size to the list. colNum is the return value.

int ListAddRow(int count,int rowNum,NewListHandle listHandle); **(original)**

Adds 'count' rows starting at 'rowNum', with that row's size to the list. rowNum is the return value.

void ListDelColumn(int count,int colNum,NewListHandle listHandle); **(original)**

This kills 'count' columns starting at 'colNum'. Follows all the same rules as the original.

void ListDelRow(int count,int rowNum,NewListHandle listHandle); **(original)**

This kills 'count' rows starting at 'rowNum'. Follows all the same rules as the original.

Operations on cells

void ListAddToCell(unsigned char *dataPtr,int dataLen,Point theCell,NewListHandle listHandle);
(original)

Adds 'dataPtr' of length 'dataLen' to the cell 'theCell' in list 'listHandle'.

void ListClrCell(Point theCell,NewListHandle listHandle); **(original)**

Clears the contents of the data in the cell and sets the data value to 0L.

void ListGetCell(unsigned char *dataPtr,int *dataLen,Point theCell,NewListHandle listHandle);
(original)

Gets the data in the cell, returns the length in 'dataLen' and the data in 'dataPtr'.

void ListSetCell(unsigned char *dataPtr,int dataLen,Point theCell,NewListHandle listHandle); **(original)**

Uses 'dataPtr' and 'dataLen' to set the data in the cell destroying anything that may already be there.

void ListCellSize(Point cSize,NewListHandle listHandle); **(original)**

Resets the size of every row and column in the entire list to 'cSize'.

int ListGetSelect(Boolean next,Point *theCell,NewListHandle listHandle); **(original)**

Acts like the original (read Inside Macintosh IV-273).

void ListSetSelect(Boolean setIt,Point theCell,NewListHandle listHandle); **(original)**

Set's the select value of the cell using the 'setIt'.

void ListCleanList(NewListHandle listHandle); **(enhanced)**

Cleans the entire list of any selected cells.

void ListSetBox(int boxFlags,Point cell,NewListHandle listHandle); **(enhanced)**

Set's the boxing parameters of the cell 'cell'. The boxFlags can be any combination of the following:

```
#define     ListLeft        0x0001
#define     ListRight       0x0002
#define     ListTop         0x0004
#define     ListBottom      0x0008
#define     ListGray        0x0010
#define     ListDouble      0x0020
```

The left, right, top and bottom parameters tell where lines are to go. The gray flag tells the drawing routine to draw the lines in 'gray' and the double increseases the width of *all* of the lines to 2.

void ListSetHandler(int handler,Point cell,NewListHandle listHandle); **(enhanced)**

This sets the handler number of the cell described by 'cell' to 'handler'. There are three defined handlers to start with:

```
#define     DrawStringHandler       0
#define     TextBoxHandler          1
#define     TEUpdateHandler         2
```

The DrawStringHandler is the default handler, it uses the Quickdraw routine DrawString to draw the string (which means it has to be a pascal style string). The TextBoxHandler uses a C style string and uses the 'TextBox' routine decsribed in the TextEdit chapter of volume I. The TEUpdateHandler takes the 'data' in the cell to be a TEHandle to a valid textEdit record. It changes the destRect and viewRect to that of the current location of the cell and updates the record (see **ListCalTextWidth** for more information).

You can give the handlerNum as any number (except 0,1, or 2) as the cell handler as long as you have established a drawing routine (see **ListEstablishHandler**).

void ListSetRowWidth(int rowNum,int rowWidth,NewListHandle listHandle); **(enhanced)**

This sets the width of the row 'rowNum' to 'rowWidth' and redraws the list if drawing is on.

void ListSetColumnWidth(int colNum,int colHeight,NewListHandle listHandle); **(enhanced)**

This sets the height of the column 'colNum' to 'colHeight' and redraws the list if drawing is on.

void ListCellFont(Point cell,int font,int size,int face,NewListHandle listHandle); **(enhanced)**

This sets the font, size, and face of the cell to the passed values and redraws the cell. For size reasons the values are actually stored as unsigned characters so large font numbers and sizes may be clipped, this doesn't happen in the global font, size, and face listing.

void ListSetGlobalFont(int font,int size,int face,NewListHandle listHandle); **(enhanced)**

This sets the font, size, and face of the whole list to the passed values and redraws the current visible area.

void ListSetGlobalBox(int boxFlags,NewListHandle listHandle); **(enhanced)**

This sets the global value of the boxFlags to boxFlags then redraws the current visible area. The global value of boxFlags takes precedence on undefined cells, if the cell has a defined boxFlag it takes precedence.

void ListSetAddative(int xAddative,int yAddative,NewListHandle listHandle); **(enhanced)**

This adds a slush area to the whole list of 'xAddative' pixels on the columns and 'yAddative' on the row axis. This is convenient when you are dragging around a gray image of a cell in order to place it somewhere else. The slush area allows you to make a bottom zone that is always around where you can put the gray image to indicate that you want it moved to the bottom.

void ListEstablishHandler(int handlerNum,void (*handler)(),NewListHandle listHandle); **(enhanced)**

This adds a cell drawing routine to the handler slot referenced by 'handlerNum'. This could be anything from 3-16. The cell drawing routines parameters are described later on in 'Cell Draw userRoutines'.

void ListSetData(Handle hand,Point cell,NewListHandle listHandle); **(enhanced)**

This sets the data handle in the cell 'cell' to the handle given and marks that the handle was not allocated locally (so that when dispose is called it doesn't dispose of it!) This is good for sending along the TEHandle of the textEdit record to a handler of TEUpdateHandler.

void ListGetVisible(NewListHandle listHandle,Rect *visArea); **(enhanced)**

This places the rectangle of visible cells in 'visArea'. The rectangle of visible cells is defined as all of the cells that are on the screen in all or in part.

int ListDrawStatus(NewListHandle listHandle); **(enhanced)**

This returns the value of the current drawing status of the list.

int ListValidCell(Point cell,NewListHandle listHandle); **(enhanced)**

Returns if the cell passed is within the dataBounds of the list. If not the routine returns FALSE, if it is the routine returns TRUE.

void ListGetDataMax(int *xmax,int *ymax,NewListHandle listHandle); **(enhanced)**

This returns the X maximum and the Y maximum data boundaries of the list in the integer pointers you supply.

int ListCalTextWidth(Point cell,NewListHandle listHandle); **(enhanced)**

This returns the length in pixels of the TextEdit cell pointed to by cell. This should then be passed to **ListSetRowWidth** to set the cell to the correct width to display the whole textEdit area. This should be used at the beginning when the list is created and everytime any **ListSetColumnWidth** is done on the cell containing the textEdit record.

Mouse Location

int ListGetCellAt(Point pt,NewListHandle listHandle,Point *newCell); **(enhanced)**

This returns to cell that contains the point 'pt' in the list. If there is a point the routine returns TRUE if not the routine returns FALSE.

int ListClick(Point pt,int modifiers,NewListHandle listHandle); **(original)**

This tracks a mouseclick on the list starting at the point 'pt' with the modifiers 'modifiers'.

int ListClickEhanced(Point pt,int modifiers,NewListHandle listHandle,void (*clikLoop)()); **(enhanced)**

This is the real 'ListClick', the original is now simply a macro to call this with the emulating clikLoop. The specifics on writing a clikLoop routine are in the 'clikLoop userRoutines' section of this document.

Point ListLastClick(NewListHandle listHandle); **(original)**

This returns the last cell clicked on in the last ListClick call.

int ListPart(Point pt,Point *cell,NewListHandle listHandle) **(enhanced)**

This returns the part of the list that is at the local point 'pt'. A return of zero states that the is no part where the hit is, otherwise the parts are this:

```
#define     inCell          1   /* In the cell area */
#define     inHorizScroll   2   /* In the horizontal scroll */
```

```
#define     inVertScroll      3   /* In the vertical scroll */
#define     inHorizSizer      4   /* In the horizontal sizer */
#define     inVertSizer       5   /* In the vertical sizer */
#define     inHorizFrame      6   /* In the horizontal frame */
#define     inVertFrame       7   /* In the vertical frame */
#define     inCornerFrame     8   /* In the corner frame */
#define     inTextEditCell    9   /* In the cell being edited */
```

If the routine returns anything other than 'inHorizScroll' , 'inVertScroll' or 'inCornerFrame' the cell has the location of the affected cell.

void ListCellSizer(Rect dragArea,int part,Point cell,NewListHandle nlh); **(enhanced)**

Pass the part and the cell you that was returned with the the listPart command. The dragArea rectangle is similiar to the rectangle you pass to the window resizer routine where the top and bottom integers limit the top and bottom Y limits, and the left and right define the X boundaries. This automically changes the cursor to one of the two resizing cursors names 'ResizeHCursor' and 'ResizeVCursor' if it can find them, and then back to the arrow again when it leaves.

List Display

void ListDraw(Point cell,NewListHandle listHandle); **(original)**

Draw the cell 'cell' of the list.

void ListDoDraw(Boolean drawIt,NewListHandle listHandle); **(original)**

Sets the drawing status on or off.

void ListScroll(int dCols,int dRows,NewListHandle listHandle); **(original)**

This scrolls the list by dCols and dRows in cells if the autoScrollMethod is by cells. If the autoScrollMethod is incremental then dCols and dRows are taken as pixel values.

void ListAutoScroll(NewListHandle listHandle); **(original)**

This takes the first selected cells and justifies it to the upper left hand corner of the screen.

void ListUpdate(RgnHandle theRgn,NewListHandle listHandle); **(original)**

Updates the portion of the list included the 'theRgn'.

void ListActivate(Boolean act,NewListHandle listHandle); **(original)**

This activates or deactivates a list.

void ListUpdateWhole(NewListHandle listHandle); **(enhanced)**

This creates a region the size of the list and uses ListUpdate to update the region.

USERROUTINE DEFINITIONS

The routine to draw a cell that you pass as a 'handler' is defined as:

```
void DrawCell(NewCellPtr theCell,unsigned char *theData,long length, Rect *visBox); (enhanced)
```

When called the the port is already setup to clip to the visBox. Any lines on the cell have already been done and the font, face, and size are already set. All you have to do is draw. Cell inversion is done automatically.

The routine that you send to ListClickEnhanced goes something like this:

```
void clikLoop(int message,Point *oldCell,Point *newCell, NewListHandle listHandle,int dataXMax,int
        dataYMax,char selFlags,int mod); (enhanced)
```

The message send to the routine can be:

```
#define     startClikLoop       0
#define     newCellLocation     1
```

StartClikLoop allows you to preset the oldCell to whatever you like, newCell is also passed as the first selected cell. listHandle is the list, dataXMax and dataYMax are the two extreme sides of the data, selFlags are any selection flags and mod is the current event modifier. newCellLocation is passed after every cycle. Therefore you should be prepared to compare the new to the old and make sure you don't do any more work than you need to. The listHandle comes to you unlocked.

This is the template for the routine that you send to

```
void MyDrawFrame(int message,Rect *visBox,int cell); (enhanced)
```

Message has one of two possible values:

```
#define     HorizCell       0
#define     VertCell        1
```

If the message is 'HorizCell' then the 'cell' value is the horizontal column heading, if the message is 'VertCell' then the 'cell' value is the vertical row value. VisBox is the rectangle where you draw the heading. Clipping has already been setup and you shouldn't redo it, and the font and size have been set as well. The face is always bold. Frame inverting is done automatically on selection.

UTTERLY POINTLESS SECTION

In true Apple tradition I give you now totally pointless advice:

- Bigger cells scroll faster.
- Deleting rows or columns makes room for new ones.
- Disposing of the list is faster than deleting each row.
- Turn off drawing before adjusting the size of all of the columns/rows.
- For speed use the buit-in drawing handlers before using your own.
- Turn off the frame for faster movement.
- There may be bugs in the code.
- Calling cells with the wrong parameters may cause a system crash.

MANAGERIAL ABUSE

Although the new list manager is ripe for abuse you should restrain yourself somewhat to around at maximum a 100x10 grid of cells. It's editing abilities does however make it acceptable for spreadsheet style input sheets, etc. So go for it. Apple's List Manager was so bad, by having one cell size, and other limiting factors that abuse was almost impossible.

As for Apple's document on Managerial abuse, first off, who is going to view 10,000 cells? As for TextEdit abuse, who has real documents under 32k (other than this). And as for the dialog manager, it is the easiest way to setup a window and create it with buttons, otherwise the code behind a large window is ridiculous.