List Manager Reference

(Not Recommended)

Carbon > User Experience

2007-12-11

Contents

List Manager Reference (Not Recommended)

Framework:	Carbon/Carbon.h
Declared in	Lists.h

Overview

Important: The List Manager is deprecated in Mac OS X version 10.5 and later. The replacement API is the Data Browser. For more information, see *Data Browser Programming Guide*.

In Mac OS 9 and earlier, the List Manager allowed applications to create, manipulate, and display scrolling lists of data items in a window. The List Manager was included in Carbon to facilitate the porting of legacy applications to Mac OS X. For Carbon applications, the Data Browser provides a more convenient way to present data for browsing and to create easily customized lists whose columns can be sorted, moved, and resized.

You should not use the List Manager in new application development.

Functions by Task

Accessing and Manipulating Cell Data

`LAddToCell` (page 42) Deprecated in Mac OS X v10.5
    Appends data to the data already contained in a cell.

`LClrCell` (page 45) Deprecated in Mac OS X v10.5
    Clears the data contained in a cell.

`LGetCell` (page 48) Deprecated in Mac OS X v10.5
    Copies a cell's data.

`LGetCellDataLocation` (page 49) Deprecated in Mac OS X v10.5
    Finds the memory location of cell data.

`LSetCell` (page 55) Deprecated in Mac OS X v10.5
    Changes the data contained in a cell.

Adding and Deleting Columns and Rows To and From a List

`LAddColumn` (page 41) Deprecated in Mac OS X v10.5
: Adds one or more columns to a list.

`LAddRow` (page 42) Deprecated in Mac OS X v10.5
: Adds one or more rows to a list.

`LDelColumn` (page 45) Deprecated in Mac OS X v10.5
: Deletes one or more columns from a list.

`LDelRow` (page 46) Deprecated in Mac OS X v10.5
: Deletes one or more rows from a list.

Changing the Size of Cells and Lists

`LCellSize` (page 44) Deprecated in Mac OS X v10.5
: Changes the size of cells in a list.

`LSize` (page 58) Deprecated in Mac OS X v10.5
: Changes the size of a list.

Creating and Disposing of Lists

`LDispose` (page 47) Deprecated in Mac OS X v10.5
: Disposes of the memory associated with a list.

`LNew` (page 51) Deprecated in Mac OS X v10.5
: Creates a new list in a window.

Creating and Managing Universal Procedure Pointers

`DisposeListClickLoopUPP` (page 30) Deprecated in Mac OS X v10.5
: Disposes of the universal procedure pointer (UPP) to a list click loop callback function.

`DisposeListDefUPP` (page 30) Deprecated in Mac OS X v10.5
: Disposes of the universal procedure pointer (UPP) to a list definition callback function.

`DisposeListSearchUPP` (page 30) Deprecated in Mac OS X v10.5
: Disposes of the universal procedure pointer (UPP) to a list search callback function.

`InvokeListClickLoopUPP` (page 38) Deprecated in Mac OS X v10.5
: Calls your list click loop callback function.

`InvokeListDefUPP` (page 39) Deprecated in Mac OS X v10.5
: Calls your list definition callback function.

`InvokeListSearchUPP` (page 40) Deprecated in Mac OS X v10.5
: Calls your list search callback function

`NewListClickLoopUPP` (page 59) Deprecated in Mac OS X v10.5
: Creates a new universal procedure pointer (UPP) to a list click loop callback function.

`NewListDefUPP` (page 59) Deprecated in Mac OS X v10.5
: Creates a new universal procedure pointer (UPP) to a list definition callback function.

`NewListSearchUPP` (page 60) Deprecated in Mac OS X v10.5
Creates a new universal procedure pointer (UPP) to a list search callback function.

Determining or Changing the Selection

`LGetSelect` (page 50) Deprecated in Mac OS X v10.5
Gets information about which cells are selected.

`LSetSelect` (page 57) Deprecated in Mac OS X v10.5
Selects or deselects a cell.

Getting Information About Cells

`LLastClick` (page 51) Deprecated in Mac OS X v10.5
Determines the coordinates of the last cell clicked in a particular list.

`LNextCell` (page 52) Deprecated in Mac OS X v10.5
Finds the next cell in a given row, in a given column, or in an entire list.

`LRect` (page 53) Deprecated in Mac OS X v10.5
Finds a rectangle that encloses a cell.

Modifying a List's Appearance

`LAutoScroll` (page 43) Deprecated in Mac OS X v10.5
Scrolls a list so that the first selected cell is in the upper-left corner of the list's visible rectangle.

`LDraw` (page 48) Deprecated in Mac OS X v10.5
Draws a cell in a list.

`LScroll` (page 54) Deprecated in Mac OS X v10.5
Scrolls a list a specified number of rows and columns.

`LSetDrawingMode` (page 56) Deprecated in Mac OS X v10.5
Changes the automatic drawing mode specified when creating a list.

Responding to Events Affecting Lists

`LActivate` (page 40) Deprecated in Mac OS X v10.5
Activates or deactivates a list.

`LClick` (page 44) Deprecated in Mac OS X v10.5
Processes a mouse-down event in a list.

`LUpdate` (page 58) Deprecated in Mac OS X v10.5
Responds to an update event.

Searching a List for a Particular Item

`LSearch` (page 55) Deprecated in Mac OS X v10.5
 Finds a cell whose data matches data that you specify.

Miscellaneous

`CreateCustomList` (page 29) Deprecated in Mac OS X v10.5

`GetListActive` (page 31) Deprecated in Mac OS X v10.5

`GetListCellIndent` (page 31) Deprecated in Mac OS X v10.5

`GetListCellSize` (page 32) Deprecated in Mac OS X v10.5

`GetListClickLocation` (page 32) Deprecated in Mac OS X v10.5

`GetListClickLoop` (page 32) Deprecated in Mac OS X v10.5

`GetListClickTime` (page 33) Deprecated in Mac OS X v10.5

`GetListDataBounds` (page 33) Deprecated in Mac OS X v10.5

`GetListDataHandle` (page 34) Deprecated in Mac OS X v10.5

`GetListDefinition` (page 34) Deprecated in Mac OS X v10.5

`GetListFlags` (page 34) Deprecated in Mac OS X v10.5

`GetListHorizontalScrollBar` (page 35) Deprecated in Mac OS X v10.5

`GetListMouseLocation` (page 35) Deprecated in Mac OS X v10.5

`GetListPort` (page 36) Deprecated in Mac OS X v10.5

`GetListRefCon` (page 36) Deprecated in Mac OS X v10.5

`GetListSelectionFlags` (page 36) Deprecated in Mac OS X v10.5

`GetListUserHandle` (page 37) Deprecated in Mac OS X v10.5

`GetListVerticalScrollBar` (page 37) Deprecated in Mac OS X v10.5

`GetListViewBounds` (page 38) Deprecated in Mac OS X v10.5

`GetListVisibleCells` (page 38) Deprecated in Mac OS X v10.5

`RegisterListDefinition` (page 60) Deprecated in Mac OS X v10.5

`SetListCellIndent` (page 61) Deprecated in Mac OS X v10.5

`SetListClickLoop` (page 61) Deprecated in Mac OS X v10.5

`SetListClickTime` (page 62) Deprecated in Mac OS X v10.5

`SetListFlags` (page 62) Deprecated in Mac OS X v10.5

`SetListLastClick` (page 63) Deprecated in Mac OS X v10.5

`SetListPort` (page 63) Deprecated in Mac OS X v10.5

`SetListRefCon` (page 63) Deprecated in Mac OS X v10.5

`SetListSelectionFlags` (page 64) Deprecated in Mac OS X v10.5

`SetListUserHandle` (page 64) Deprecated in Mac OS X v10.5

`SetListViewBounds` (page 65) Deprecated in Mac OS X v10.5

Callbacks

ListClickLoopProcPtr

Defines a pointer to a list click loop callback function. Your list click loop callback function overrides the standard click-loop function that is used to select cells and automatically scroll a list.

```
typedef Boolean (*ListClickLoopProcPtr)
(
);
```

If you name your function `MyListClickLoopProc`, you would declare it like this:

```
Boolean MyListClickLoopProc ();
```

Parameters

Return Value
A value indicating whether the `LClick` function should continue tracking the mouse. Your function should return `TRUE` if you wish `LClick` to continue to track the mouse, and `FALSE` if `LClick` should stop and return immediately.

Discussion
If your application defines a custom click-loop function, then the `LClick` (page 44) function repeatedly calls the function until the user releases the mouse button. A click-loop function may perform any processing desired when it is executed.

Because no parameters are passed to the click-loop function, your click-loop function probably needs to access a global variable that contains a handle to the list record, which contains information about the location of the cursor and other information potentially of interest to a click-loop function. You might also create a global variable that stores the state of the modifier keys immediately before a call to the `LClick` function. You would need to set these global variables immediately before calling `LClick`.

The pointer to your function, which you provide in the list record structure, should be a universal procedure pointer (UPP). The definition of the UPP data type for your list click loop function is as follows:

```
typedef (ListClickLoopProcPtr) ListClickLoopUPP;
```

Before using your list click loop function, you must first create a new universal procedure pointer to it, using the `NewListClickLoopUPP` (page 59) function, as shown here:

```
ListClickLoopUPP MyListClickLoopUPP;
MyListClickLoopUPP = NewListClickLoopUPP(&MyListClickLoopProc)
```

You then use `MyListClickLoopUPP` in the `lClickLoop` field of the `ListRec` (page 20) structure for your list. The `LClick` (page 44) function calls your list click loop function while the user holds down the mouse button. If you wish to call your own list click loop function, you can use the `InvokeListClickLoopUPP` (page 38) function:

```
continueTracking = InvokeListClickLoopUPP(MyListClickLoopUPP);
```

When you are finished using your list click loop callback function, you should dispose of the universal procedure pointer associated with it, using the `DisposeListClickLoopUPP` (page 30) function.

```
DisposeListClickLoopUPP(MyListClickLoopUPP);
```

A click-loop function does not execute at interrupt time. Instead, it is called directly by the `LClick` function. Thus, a click-loop function can allocate memory, and it does not need to adjust the value contained in the A5 register.

Special Considerations

A click-loop function does not execute at interrupt time. Instead, it is called directly by the `LClick` function. Thus, a click-loop function can allocate memory, and it does not need to adjust the value contained in the A5 register.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Lists.h`

ListDefProcPtr

Defines a pointer to a list definition callback function. Your list definition callback function defines a custom list display.

```
typedef void (*ListDefProcPtr) (
    SInt16 lMessage,
    Boolean lSelect,
    Rect *lRect,
    Cell lCell,
    SInt16 lDataOffset,
    SInt16 lDataLen,
    ListHandle lHandle
);
```

If you name your function `MyListDefProc`, you would declare it like this:

```
void MyListDefProc (
    SInt16 lMessage,
    Boolean lSelect,
    Rect * lRect,
    Cell lCell,
    SInt16 lDataOffset,
    SInt16 lDataLen,
    ListHandle lHandle
);
```

Parameters

`lMessage`
: A value that identifies the operation to be performed. See "List Definition Constants" (page 25).

`lSelect`
: Indicates whether the cell specified by the `lCell` parameter should be highlighted. This parameter is defined only for the `lDrawMessage` and `lHiliteMsg` messages.

`lRect`
: A pointer to the rectangle (in local coordinates of the list's graphics port) that encloses the specified cell. Although this parameter is defined as a pointer, your list definition function must not change the coordinates of the rectangle. This parameter is defined only for the `lDrawMessage` and `lHiliteMsg` messages.

`lCell`
: The coordinates of the cell to be drawn or highlighted. This parameter is defined only for the `lDrawMessage` and `lHiliteMsg` messages.

`lDataOffset`
: The location of the cell data associated with the specified cell. The location is specified as an offset from the beginning of the relocatable block referenced by the `cells` field of the list record. This parameter is defined only for the `lDrawMessage` and `lHiliteMsg` messages.

`lDataLen`
: The length in bytes of the cell data associated with the specified. This parameter is defined only for the `lDrawMessage` and `lHiliteMsg` messages.

`lHandle`
: A handle to the list for which a message is being sent. Your application can access the list's list record, or it can call List Manager functions to manipulate the list.

Discussion

Your application can write a list definition function to customize list display. For example, you can write a list definition function to support the display of color icons. A custom list definition function must be compiled as a code resource of type `'LDEF'` and added to the resource file of the application that needs to use it.

The List Manager calls your list definition function whenever an application using the function creates a new list with the `LNew` (page 51) function, needs a cell to be drawn, needs a cell's highlighting state to be reversed, or has called the `LDispose` (page 47) function to dispose of a list.

The pointer to your list definition function should be a universal procedure pointer (UPP). The definition of the UPP data type for your definition function is as follows:

```
typedef (ListDefProcPtr) ListDefUPP;
```

Before using your list definition function, you must first create a new universal procedure pointer to it, using the `NewListDefUPP` (page 59) function, as shown here:

```
ListDefUPP MyListDefUPP;
MyListDefUPP = NewListDefUPP(&MyListDefProc)
```

The List Manager automatically invokes your list definition function when a new list is created. If you wish to call your own list definition callback function, you can use the `InvokeListDefUPP` (page 39) function:

```
InvokeListDefUPP(lMessage, lSelect, &lRect, lCell, lDataOffset,
        lDataLen, lHandle, MyListDefUPP)
```

When you are finished with your list definition function, you should dispose of the universal procedure pointer associated with it, using the `DisposeListDefUPP` (page 30) function.

```
DisposeListDefUPP(MyListDefUPP);
```

Because a list definition function is stored in a code resource, it cannot have its own global variables that it accesses through the A5 register. (Some development systems, however, may allow code resources to access global variables through some other register, such as A4. See your development system's documentation for more information.) If your list definition function needs access to global data, it might store a handle to such data in the `refCon` or `userHandle` fields of the list record; however, applications would not then be able to use these fields for their own purposes.

Special Considerations

Because a list definition function is stored in a code resource, it cannot have its own global variables that it accesses through the A5 register. (Some development systems, however, may allow code resources to access global variables through some other register, such as A4. See your development system's documentation for more information.) If your list definition function needs access to global data, it might store a handle to such data in the `refCon` or `userHandle` fields of the list record; however, applications would not then be able to use these fields for their own purposes.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Lists.h`

ListNotificationProcPtr

```
typedef void (*ListNotificationProcPtr)
(
    ListHandle theList,
    ListNotification notification,
    SInt32 param
);
```

If you name your function `MyListNotificationProc`, you would declare it like this:

```
void MyListNotificationProc (
    ListHandle theList,
    ListNotification notification,
    SInt32 param
);
```

Parameters

theList

notification

param

ListSearchProcPtr

Defines a pointer to a list search callback function. Your list search callback function compares data in a search field to the contents of a list cell.

```
typedef SInt16 (*ListSearchProcPtr) (
    Ptr aPtr,
    Ptr bPtr,
    SInt16 aLen,
    SInt16 bLen
);
```

If you name your function `MyListSearchProc`, you would declare it like this:

```
short MyListSearchProc (
    Ptr aPtr,
    Ptr bPtr,
    short aLen,
    short bLen
);
```

Parameters

aPtr

A pointer to the data contained in a cell.

bPtr

A pointer to the data for which you are searching.

aLen

The number of bytes of data contained in the cell.

bLen

The number of bytes of data for which you are searching.

Return Value
If the cell data matches the search data, your function should return 0. Otherwise, your search function should return 1.

Discussion
You can pass a pointer to your search function as the third parameter to the `LSearch` function. A search function must compare the data defined by the `aPtr` and `aLen` parameters with the data defined by the `bPtr` and `bLen` parameters. Your function can use any technique you choose to compare the data.

If you do not wish to create your own search function, your application can specify `NULL` as a parameter to `LSearch`, in place of a pointer to your function. `LSearch` then uses the Text Utilities function `IUMagIDString`, the default search function. The `IUMagIDString` function returns 0 if the search data exactly matches the cell data, but `IUMagIDString` considers the strings `'Rose'` and `'rosé'` to be equivalent. If your application simply needs a search function that works like `IUMagIDString` but considers `'Rose'` to be different from `'rosé'`, the Text Utilities provides the case-sensitive comparison function `IUMagString`. Instead of writing a custom function, your application can simply pass `@IUMagString` as the third parameter to the `LSearch` function.

The pointer which you pass to the `LSearch` function should be a universal procedure pointer (UPP). The definition of the UPP data type for your search function is as follows:

```
typedef (ListSearchProcPtr) ListSearchUPP;
```

Before using your search function, you must first create a universal procedure pointer to it, using the `NewListSearchUPP` `NewListSearchUPP` (page 60) function, as shown here:

```
ListSearchUPP MyListSearchUPP;
MyListSearchUPP = NewListSearchUPP(&MyListSearchProc)
```

You then pass `MyListSearchUPP` to the `LSearch` function, which will call your custom search function on each cell it searches. If you wish to call your own list search function, use the `InvokeListSearchUPP` (page 40) function:

```
isMatch = InvokeListSearchUPP(aPtr, bPtr, aLen, bLen,
                     MyListSearchUPP);
```

When you are finished with your list search callback function, you should dispose of the universal procedure pointer associated with it, using the `DisposeListSearchUPP` (page 30) function:

```
DisposeListSearchUPP(MyListSearchUPP);
```

A search function does not execute at interrupt time. Instead, it is called directly by the `LSearch` function. Thus, a search function can allocate memory, and it does not need to adjust the value contained in the A5 register.

Special Considerations

A search function does not execute at interrupt time. Instead, it is called directly by the `LSearch` function. Thus, a search function can allocate memory, and it does not need to adjust the value contained in the A5 register.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Lists.h`

Data Types

Cell

```
typedef Point Cell;
```

Discussion
The `Cell` data type defines a cell record. The functions `LGetSelect` (page 50) , `LSetSelect` (page 57) , `LSetCell` (page 55) , `LAddToCell` (page 42) , `LClrCell` (page 45) , `LGetCellDataLocation` (page 49) , `LGetCell` (page 48) , `LDraw` (page 48) , `LSearch` (page 55) , `LNextCell` (page 52) , `LRect` (page 53) , and `LLastClick` (page 51) use the `Cell` data type to specify the coordinates of a cell in a list.

Note that column and row numbers are 0-based. Also note that this reference designates cells using the notation (column–1, row–1), so that a cell with coordinates (2,5) is in the third column and sixth row of a list. You specify a cell with coordinates (2,5) by setting the cell's `h` field to 2 and its `v` field to 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Lists.h`

DataArray

```
typedef  DataArray[32001];
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Lists.h`

DataHandle

```
typedef  DataPtr * DataHandle;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Lists.h`

DataPtr

```
typedef  char * DataPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Lists.h`

ListBounds

```
typedef Rect ListBounds;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Lists.h`

ListClickLoopUPP

```
typedef ListClickLoopProcPtr ListClickLoopUPP;
```

Discussion
For more information, see the description of the ListClickLoopUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Lists.h`

ListDefSpec

```
struct ListDefSpec {
    ListDefType defType
    union {
        ListDefUPP userProc;
    } u;
};
typedef struct ListDefSpec ListDefSpec;
typedef ListDefSpec * ListDefSpecPtr;
```

Fields
`defType`
`ListDefUPP`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Lists.h`

ListDefType

```
typedef UInt32 ListDefType;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Lists.h`

ListDefUPP

```
typedef ListDefProcPtr ListDefUPP;
```

Discussion
For more information, see the description of the ListDefUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Lists.h`

ListNotification

```
typedef SInt32 ListNotification;
```

ListNotificationUPP

```
typedef ListNotificationProcPtr ListNotificationUPP;
```

ListRec

```
struct ListRec {
    Rect rView;
    GrafPtr port;
    Point indent;
    Point cellSize;
    ListBounds visible;
    ControlRef vScroll;
    ControlRef hScroll;
    SInt8 selFlags;
    Boolean lActive;
    SInt8 lReserved;
    SInt8 listFlags;
    long clikTime;
    Point clikLoc;
    Point mouseLoc;
    ListClickLoopUPP lClickLoop;
    Cell lastClick;
    long refCon;
    Handle listDefProc;
    Handle userHandle;
    ListBounds dataBounds;
    DataHandle cells;
    short maxIndex;
    short cellArray[1];
};
typedef struct ListRec ListRec;
typedef ListRec * ListPtr;
typedef ListPtr * ListHandle
```

Fields

`rView`

The rectangle in which the list's visible rectangle is located, in local coordinates of the graphics port specified by the `port` field. Note that the list's visible rectangle does not include the area needed for the list's scroll bars. The width of a vertical scroll bar (which equals the height of a horizontal scroll bar) is 15 pixels.

`port`

The graphics port of the window containing the list.

`indent`

The location, relative to the upper-left corner of a cell, at which drawing should begin. List definition functions should set this field to a value appropriate to the type of data that a cell in a list is to contain.

`cellSize`
: The size in pixels of each cell in the list. When your application creates a list, it can either specify the cell size or let the List Manager calculate the cell size. You should not change the `cellSize` field directly; if you need to change the cell size after creating a list, use the `LCellSize` (page 44) function.

`visible`
: The cells in a list that are visible within the area specified by the `rView` field. The List Manager sets the `left` and `top` fields of `visible` to the coordinates of the first visible cell; however, the List Manager sets the `right` and `bottom` fields so that each is 1 greater than the horizontal and vertical coordinates of the last visible cell. For example, if a list contains 4 columns and 10 rows but only the first 2 columns and the first 5 rows are visible (that is, the last visible cell has coordinates (1,4)), the List Manager sets the `visible` field to (0,0,2,5).

`vScroll`
: A control handle for a list's vertical scroll bar, or `NULL` if a list does not have a vertical scroll bar.

`hScroll`
: A control handle for a list's horizontal scroll bar, or `NULL` if a list does not have a horizontal scroll bar.

`selFlags`
: Indicates the selection flags for a list. When your application creates a list, the List Manager clears the `selFlags` field to 0. This defines the List Manager's default selection algorithm. To change the default behavior for a particular list, set the desired bits in the list's `selFlags` field. See "Selection Flags" (page 28).

`lActive`
: Indicates whether the list is active (`TRUE` if active, `FALSE` if inactive).

`lReserved`
: Reserved.

`listFlags`
: Indicates whether the List Manager should automatically scroll the list if the user clicks the list and then drags the cursor outside the list display rectangle. See "List Flags" (page 26) for the values used in this field.

: By default, the List Manager enables horizontal autoscrolling for a list if the list includes a horizontal scroll bar, and enables vertical autoscrolling for a list if the list includes a vertical scroll bar.

`clikTime`
: The time in ticks of the last click in the list. If your application depends on the value contained in this field, then your application should update the field if the application selects a list item in response to keyboard input.

`clikLoc`
: The location in local coordinates of the last click in the list.

`mouseLoc`
: Indicates the current location of the cursor in local coordinates. This value is continuously updated by the `LClick` function after the user clicks a list.

`lClickLoop`
: A universal procedure pointer to your click loop callback function, which is repeatedly called by the `LClick` (page 44) function, or NULL if the default click-loop function is to be used.

`lastClick`
: The coordinates of the last cell in the list that was clicked. This may not be the same as the last cell selected if the user selects a range of cells by Shift-dragging or Command-dragging. If your application depends on the value contained in this field, then your application should update the field whenever your application selects a list item in response to keyboard input.

`refCon`
: 4 bytes for use by your application.

`listDefProc`
: A handle to the code for the list definition function that defines how the list is drawn.

`userHandle`
: 4 bytes that your application can use as needed. For example, your application might use this field to store a handle to additional storage associated with the list. However, the `LDispose` (page 47) function does not automatically release this storage when disposing of the list.

`dataBounds`
: The range of cells in a list. When your application creates a list, it specifies the initial bounds of the list. As your application adds rows and columns, the List Manager updates this field. The List Manager sets the `left` and `top` fields of `dataBounds` to the coordinates of the first cell in the list; the List Manager sets the `right` and `bottom` fields so that each is 1 greater than the horizontal and vertical coordinates of the last cell. For example, if a list contains 4 columns and 10 rows (that is, the last cell in the list has coordinates (3,9)), the List Manager sets the `dataBounds` field to (0,0,4,10).

`cells`
: A handle to a relocatable block used to store cell data. Your application should not change the contents of this relocatable block directly.

`maxIndex`
: Used internally.

`cellArray`
: Offsets to data that indicate the location of different cells' data within the data handle specified by the `cells` parameter. Your application should not access this field directly.

Discussion

Functions in the List Manager interface use the `ListHandle` datatype to identify a list. The `ListHandle` type uses a `ListRec` structure to maintain information about a list. The `ListRec` data type defines a list record.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Lists.h`

ListRef

```
typedef ListHandle ListRef;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Lists.h`

ListSearchUPP

```
typedef ListSearchProcPtr ListSearchUPP;
```

Discussion
For more information, see the description of the ListSearchUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Lists.h`

StandardIconListCellDataRec

```
struct StandardIconListCellDataRec {
    Handle iconHandle;
    short font;
    short face;
    short size;
    Str255 name;
};
typedef struct StandardIconListCellDataRec StandardIconListCellDataRec;
typedef StandardIconListCellDataRec * StandardIconListCellDataPtr;
```

Fields

`iconHandle`

`font`

`face`

`size`

`name`

Discussion

Version Notes

Carbon Porting Notes

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Lists.h`

Constants

kListDefProcPtr

```
enum {
    kListDefProcPtr = 0,
    kListDefUserProcType = kListDefProcPtr,
    kListDefStandardTextType = 1,
    kListDefStandardIconType = 2
};
```

Constants

`kListDefProcPtr`
Available in Mac OS X v10.0 and later.

Declared in `Lists.h`.

`kListDefUserProcType`
Available in Mac OS X v10.0 and later.

Declared in `Lists.h`.

`kListDefStandardTextType`
Available in Mac OS X v10.0 and later.

Declared in `Lists.h`.

`kListDefStandardIconType`
Available in Mac OS X v10.0 and later.

Declared in `Lists.h`.

lDrawingModeOff

```
enum {
    lDrawingModeOff = 8,
    lDoVAutoscroll = 2,
    lDoHAutoscroll = 1
};
```

Constants

`lDrawingModeOff`
Available in Mac OS X v10.0 and later.

Declared in `Lists.h`.

`lDoVAutoscroll`
Set this bit to 1 if you wish to allow automatic vertical scrolling.

Available in Mac OS X v10.0 and later.

Declared in `Lists.h`.

`lDoHAutoscroll`
Set this bit to 1 if you wish to allow automatic horizontal scrolling.

Available in Mac OS X v10.0 and later.

Declared in `Lists.h`.

lDrawingModeOffBit

```
enum {
    lDrawingModeOffBit = 3,
    lDoVAutoscrollBit = 1,
    lDoHAutoscrollBit = 0
};
```

Constants

`lDrawingModeOffBit`

Available in Mac OS X v10.0 and later.

Declared in `Lists.h`.

`lDoVAutoscrollBit`

Available in Mac OS X v10.0 and later.

Declared in `Lists.h`.

`lDoHAutoscrollBit`

Available in Mac OS X v10.0 and later.

Declared in `Lists.h`.

List Definition Constants

```
enum {
    lInitMsg = 0,
    lDrawMsg = 1,
    lHiliteMsg = 2,
    lCloseMsg = 3
};
```

Constants

`lInitMsg`

In response to the `lInitMsg` message, your list definition function should perform any special initialization needed for a list. For example, the function might set fields of the list record, such as the `cellSize` and `indent` fields, to appropriate values. Your list definition function does not necessarily need to do anything in response to the initialization message. If it does nothing, then memory is still allocated for the list, and fields of the list record are set to the same values as they would be set to if the default list definition function were being used.

Available in Mac OS X v10.0 and later.

Declared in `Lists.h`.

`lDrawMsg`

Your list definition function should draw the cell specified by the `theCell` parameter after receiving an `lDrawMsg` message. The function must ensure that it does not draw anywhere but within the rectangle specified by the `cellRect` parameter. If the `selected` parameter is `TRUE`, then your list definition function should draw the cell in its highlighted state; otherwise, it should draw the cell without highlighting. When drawing, your list definition function should take care not to permanently change any characteristics of the drawing environment.

Available in Mac OS X v10.0 and later.

Declared in `Lists.h`.

`lHiliteMsg`
: Your list definition function should respond to the `lHiliteMsg` message by reversing the selection status of the cell contained within the rectangle specified by the `cellRect` parameter. If a cell is highlighted, your list definition function should remove the highlighting; if a cell is not highlighted, your list definition function should highlight it.

  Available in Mac OS X v10.0 and later.

  Declared in `Lists.h`.

`lCloseMsg`
: The List Manager sends your list definition function an `lCloseMsg` message before it disposes of a list and its data. Your list definition function need only respond to this message if additional memory has been allocated for the list. For example, your list definition function might allocate a relocatable block in response to the `lInitMsg` message. In this case, your list definition function would need to dispose of this relocatable block in response to the `lCloseMsg` message. Or, if your list definition function defines cells simply to contain pointers or handles to data stored elsewhere in memory, it would need to dispose of that memory in response to the `lCloseMsg` message.

  Available in Mac OS X v10.0 and later.

  Declared in `Lists.h`.

Discussion
The List Manager passes these values to your `ListDefProcPtr` (page 13) function to identify the operation to be performed.

List Flags

Constants

Discussion
The following constants define bits in the `listFlags` field of the `ListRec` (page 20) structure that determine whether horizontal autoscrolling and vertical autoscrolling are enabled:.

listNotifyNothing

```
enum {
    listNotifyNothing = 'nada',
    listNotifyClick = 'clik',
    listNotifyDoubleClick = 'dblc',
    listNotifyPreClick = 'pclk'
};
```

Constants

`listNotifyNothing`
`listNotifyClick`
`listNotifyDoubleClick`
`listNotifyPreClick`

lOnlyOneBit

```
enum {
    lOnlyOneBit = 7,
    lExtendDragBit = 6,
    lNoDisjointBit = 5,
    lNoExtendBit = 4,
    lNoRectBit = 3,
    lUseSenseBit = 2,
    lNoNilHiliteBit = 1
};
```

Constants

`lOnlyOneBit`

Available in Mac OS X v10.0 and later.

Declared in `Lists.h`.

`lExtendDragBit`

Available in Mac OS X v10.0 and later.

Declared in `Lists.h`.

`lNoDisjointBit`

Available in Mac OS X v10.0 and later.

Declared in `Lists.h`.

`lNoExtendBit`

Available in Mac OS X v10.0 and later.

Declared in `Lists.h`.

`lNoRectBit`

Available in Mac OS X v10.0 and later.

Declared in `Lists.h`.

`lUseSenseBit`

Available in Mac OS X v10.0 and later.

Declared in `Lists.h`.

`lNoNilHiliteBit`

Available in Mac OS X v10.0 and later.

Declared in `Lists.h`.

Selection Flags

```
enum {
    lOnlyOne = -128,
    lExtendDrag = 64,
    lNoDisjoint = 32,
    lNoExtend = 16,
    lNoRect = 8,
    lUseSense = 4,
    lNoNilHilite = 2
};
```

Constants

`lOnlyOne`

Specify this value if you wish to allow only one item to be selected at once.

Available in Mac OS X v10.0 and later.

Declared in `Lists.h`.

`lExtendDrag`

Specify this value if you wish to enable selection of multiple items by dragging without the Shift key.

Available in Mac OS X v10.0 and later.

Declared in `Lists.h`.

`lNoDisjoint`

Specify this value if you wish to prevent discontinuous selections using the Command key.

Available in Mac OS X v10.0 and later.

Declared in `Lists.h`.

`lNoExtend`

Specify this value if you wish to prevent extending Shift key selections. All items are deselected before responding to Shift-click.

Available in Mac OS X v10.0 and later.

Declared in `Lists.h`.

`lNoRect`

Specify this value if you wish to select all items in the cursor's path during Shift-drag.

Available in Mac OS X v10.0 and later.

Declared in `Lists.h`.

`lUseSense`

Specify this value if you wish to allow the user to deselect one or more items using the Shift key

Available in Mac OS X v10.0 and later.

Declared in `Lists.h`.

`lNoNilHilite`

Specify this value if you wish to disable the highlighting of empty cells.

Available in Mac OS X v10.0 and later.

Declared in `Lists.h`.

Discussion

The `ListRec` (page 20) structure uses these values in the `selFlags` field to indicate the List Manager's default selection algorithm. Use these values additively to select more than one selection option.

Deprecated List Manager Reference (Not Recommended) Functions

A function identified as deprecated has been superseded and may become unsupported in the future.

Deprecated in Mac OS X v10.5

CreateCustomList

(Deprecated in Mac OS X v10.5.)

```
OSStatus CreateCustomList (
   const Rect *rView,
   const ListBounds *dataBounds,
   Point cellSize,
   const ListDefSpec *theSpec,
   WindowRef theWindow,
   Boolean drawIt,
   Boolean hasGrow,
   Boolean scrollHoriz,
   Boolean scrollVert,
   ListHandle *outList
);
```

Parameters

rView

dataBounds

cellSize

theSpec

theWindow

drawIt

hasGrow

scrollHoriz

scrollVert

outList

Return Value
A result code.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Lists.h`

DisposeListClickLoopUPP

Disposes of the universal procedure pointer (UPP) to a list click loop callback function. (Deprecated in Mac OS X v10.5.)

```
void DisposeListClickLoopUPP (
   ListClickLoopUPP userUPP
);
```

Parameters

userUPP

Discussion

See the `ListClickLoopProcPtr` (page 11) callback for more information.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`Lists.h`

DisposeListDefUPP

Disposes of the universal procedure pointer (UPP) to a list definition callback function. (Deprecated in Mac OS X v10.5.)

```
void DisposeListDefUPP (
   ListDefUPP userUPP
);
```

Parameters

userUPP

Discussion

See the `ListDefProcPtr` (page 13) callback for more information.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`Lists.h`

DisposeListSearchUPP

Disposes of the universal procedure pointer (UPP) to a list search callback function. (Deprecated in Mac OS X v10.5.)

```
void DisposeListSearchUPP (
   ListSearchUPP userUPP
);
```

Parameters

userUPP

Discussion

See the `ListSearchProcPtr` (page 15) callback for more information.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`Lists.h`

GetListActive

(Deprecated in Mac OS X v10.5.)

```
Boolean GetListActive (
   ListHandle list
);
```

Parameters

list

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

GetListCellIndent

(Deprecated in Mac OS X v10.5.)

```
Point * GetListCellIndent (
   ListHandle list,
   Point *indent
);
```

Parameters

list

indent

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Lists.h`

GetListCellSize

(Deprecated in Mac OS X v10.5.)

```
Point * GetListCellSize (
   ListHandle list,
   Point *size
);
```

Parameters

`list`

`size`

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

GetListClickLocation

(Deprecated in Mac OS X v10.5.)

```
Point * GetListClickLocation (
   ListHandle list,
   Point *click
);
```

Parameters

`list`

`click`

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

GetListClickLoop

(Deprecated in Mac OS X v10.5.)

```
ListClickLoopUPP GetListClickLoop (
   ListHandle list
);
```

Parameters

list

Return Value

See the description of the `ListClickLoopUPP` data type.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

GetListClickTime

(Deprecated in Mac OS X v10.5.)

```
SInt32 GetListClickTime (
   ListHandle list
);
```

Parameters

list

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

GetListDataBounds

(Deprecated in Mac OS X v10.5.)

```
ListBounds * GetListDataBounds (
   ListHandle list,
   ListBounds *bounds
);
```

Parameters

list

bounds

Return Value

See the description of the `ListBounds` data type.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Lists.h`

GetListDataHandle

(Deprecated in Mac OS X v10.5.)

```
DataHandle GetListDataHandle (
   ListHandle list
);
```

Parameters

`list`

Return Value
See the description of the `DataHandle` data type.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Lists.h`

GetListDefinition

(Deprecated in Mac OS X v10.5.)

```
Handle GetListDefinition (
   ListHandle list
);
```

Parameters

`list`

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Lists.h`

GetListFlags

(Deprecated in Mac OS X v10.5.)

```
OptionBits GetListFlags (
   ListHandle list
);
```

Parameters

list

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

GetListHorizontalScrollBar

(Deprecated in Mac OS X v10.5.)

```
ControlRef GetListHorizontalScrollBar (
   ListHandle list
);
```

Parameters

list

Return Value

See the Control Manager documentation for a description of the `ControlRef` data type.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

GetListMouseLocation

(Deprecated in Mac OS X v10.5.)

```
Point * GetListMouseLocation (
   ListHandle list,
   Point *mouse
);
```

Parameters

list

mouse

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Lists.h`

GetListPort

(Deprecated in Mac OS X v10.5.)

```
CGrafPtr GetListPort (
   ListHandle list
);
```

Parameters

`list`

Return Value
See the QuickDraw Manager documentation for a description of the `CGrafPtr` data type.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Lists.h`

GetListRefCon

(Deprecated in Mac OS X v10.5.)

```
SInt32 GetListRefCon (
   ListHandle list
);
```

Parameters

`list`

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Lists.h`

GetListSelectionFlags

(Deprecated in Mac OS X v10.5.)

```
OptionBits GetListSelectionFlags (
   ListHandle list
);
```

Parameters

list

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Lists.h`

GetListUserHandle

(Deprecated in Mac OS X v10.5.)

```
Handle GetListUserHandle (
   ListHandle list
);
```

Parameters

list

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Lists.h`

GetListVerticalScrollBar

(Deprecated in Mac OS X v10.5.)

```
ControlRef GetListVerticalScrollBar (
   ListHandle list
);
```

Parameters

list

Return Value
See the Control Manager documentation for a description of the `ControlRef` data type.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Lists.h`

GetListViewBounds

(Deprecated in Mac OS X v10.5.)

```
Rect * GetListViewBounds (
   ListHandle list,
   Rect *view
);
```

Parameters
`list`

`view`

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Lists.h`

GetListVisibleCells

(Deprecated in Mac OS X v10.5.)

```
ListBounds * GetListVisibleCells (
   ListHandle list,
   ListBounds *visible
);
```

Parameters
`list`

`visible`

Return Value
See the description of the `ListBounds` data type.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Lists.h`

InvokeListClickLoopUPP

Calls your list click loop callback function. (Deprecated in Mac OS X v10.5.)

```
Boolean InvokeListClickLoopUPP (
   ListClickLoopUPP userUPP
);
```

Parameters

userUPP

Discussion

See the `ListClickLoopProcPtr` (page 11) callback for more information.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`Lists.h`

InvokeListDefUPP

Calls your list definition callback function. (Deprecated in Mac OS X v10.5.)

```
void InvokeListDefUPP (
   short lMessage,
   Boolean lSelect,
   Rect *lRect,
   Cell lCell,
   short lDataOffset,
   short lDataLen,
   ListHandle lHandle,
   ListDefUPP userUPP
);
```

Parameters

lMessage

lSelect

lRect

lCell

lDataOffset

lDataLen

lHandle

userUPP

Discussion

See the `ListDefProcPtr` (page 13) callback for more information.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`Lists.h`

InvokeListSearchUPP

Calls your list search callback function (Deprecated in Mac OS X v10.5.)

```
short InvokeListSearchUPP (
   Ptr aPtr,
   Ptr bPtr,
   short aLen,
   short bLen,
   ListSearchUPP userUPP
);
```

Parameters

aPtr

bPtr

aLen

bLen

userUPP

Return Value

Discussion

See the `ListSearchProcPtr` (page 15) callback for more information.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`Lists.h`

LActivate

Activates or deactivates a list. (Deprecated in Mac OS X v10.5.)

```
void LActivate (
   Boolean act,
   ListHandle lHandle
);
```

Parameters

act

Indicates whether the list should be activated. Specify `TRUE` to activate the list. Specify `FALSE` to deactivate the list.

lHandle

The list to be activated or deactivated.

Discussion

If a list is being deactivated, this function removes highlighting from selected cells and hides the scroll bars. If a list is being activated, the function highlights selected cells and shows the scroll bars.

This function has no effect on a list's size box, if one exists.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Lists.h`

LAddColumn

Adds one or more columns to a list. (Deprecated in Mac OS X v10.5.)

```
short LAddColumn (
   short count,
   short colNum,
   ListHandle lHandle
);
```

Parameters

`count`
The number of columns to add.

`colNum`
The column number of the first column to add.

`lHandle`
The list to which to add the columns.

Return Value
The column number of the first column added, which is equal to the value specified by the `colNum` parameter if that value is a valid column number. If the column number specified by `colNum` is not already in the list, then new last columns are added. The value returned by this function thus has significance only in this case.

Discussion
This function inserts columns starting at the column specified by the `colNum` parameter. If there is insufficient memory in the heap to add the new columns, this function may fail to add the new columns although it returns a positive function result. Be sure there is enough memory in the heap to allocate the new columns before calling `LAddColumn`.

Columns whose column numbers are initially greater than `colNum` have their column numbers increased by `count`.

If the automatic drawing mode is enabled and the columns added by the function are visible, then the list (including its scroll bars) is updated. New cells created by a call to this function are initially empty.

You may add columns to a list that does not yet have rows. The `dataBounds` field of the list record reflects that the list has columns, but you can only access cells when both rows and columns have been added.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

LAddRow

Adds one or more rows to a list. (Deprecated in Mac OS X v10.5.)

```
short LAddRow (
   short count,
   short rowNum,
   ListHandle lHandle
);
```

Parameters

count

The number of rows to add.

rowNum

The row number of the first row to add.

lHandle

The list to add the rows to.

Return Value

The row number of the first row added, which is equal to the value specified by the `rowNum` parameter if that value is a valid row number. If the row number specified by `rowNum` is not already in the list, then new last rows are added. The value returned by this function thus has significance only in this case.

Discussion

This function inserts rows starting at the row specified by the `rowNum` parameter. If there is insufficient memory in the heap to add the new rows, the function may fail to add the new rows although it returns a positive function result. Be sure there is enough memory in the heap to allocate the new rows before calling this function.

Rows whose row numbers are initially greater than `rowNum` have their row numbers increased by `count`.

If the automatic drawing mode is enabled and the rows added by this function are visible, then the list (including its scroll bars) is updated. New cells created by a call to this function are initially empty.

You may add rows to a list that does not yet have columns. The `dataBounds` field of the list record reflects that the list has rows, but you can only access cells when both rows and columns have been added.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

LAddToCell

Appends data to the data already contained in a cell. (Deprecated in Mac OS X v10.5.)

```
void LAddToCell (
   const void *dataPtr,
   short dataLen,
   Cell theCell,
   ListHandle lHandle
);
```

Parameters

dataPtr

A pointer to the data to be appended.

dataLen

The length in bytes of the data to be appended.

theCell

The coordinates of the cell to which the data should be appended.

lHandle

The list containing the cell given in the `theCell` parameter.

Discussion

If the cell coordinates specified by the parameter `theCell` are invalid, then this function does nothing.

If the data of a visible cell is changed and the automatic drawing mode is enabled, the function updates the list.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

LAutoScroll

Scrolls a list so that the first selected cell is in the upper-left corner of the list's visible rectangle. (Deprecated in Mac OS X v10.5.)

```
void LAutoScroll (
   ListHandle lHandle
);
```

Parameters

lHandle

The list to be scrolled.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

LCellSize

Changes the size of cells in a list. (Deprecated in Mac OS X v10.5.)

```
void LCellSize (
   Point cSize,
   ListHandle lHandle
);
```

Parameters

cSize

The new size of each cell in the list. This function sets the `cellSize` field of the list record of the list to the value of the `cSize` parameter. That is, the list's new cells will be of width `cSize.h` and of height `cSize.v`.

All cells in a list must be the same size.

lHandle

The list whose cells' size is being changed.

Discussion

The function updates the list's visible rectangle to contain cells of the specified size. However, it does not redraw any cells.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

LClick

Processes a mouse-down event in a list. (Deprecated in Mac OS X v10.5.)

```
Boolean LClick (
   Point pt,
   EventModifiers modifiers,
   ListHandle lHandle
);
```

Parameters

pt

The location in local coordinates of the mouse-down event. Your application can simply call `GlobalToLocal(myEvent.where)` and then pass `myEvent.where` in this parameter.

If the `pt` parameter specifies a portion of the list's visible rectangle, then cells are selected with an algorithm that depends on the list's selection flags and on the `modifiers` parameter. If the user drags the cursor above or below the list's visible rectangle and vertical autoscrolling is enabled, then the List Manager vertically autoscrolls the list. If the user drags the cursor to the right or the left of the list's visible rectangle and horizontal autoscrolling is enabled, then the List Manager horizontally autoscrolls the list.

If the `pt` parameter specifies a point within the list's scroll bar, then the List Manager calls the scroll bar's control definition function to track the cursor and it scrolls the list appropriately.

`modifiers`

An integer value corresponding to the `modifiers` field of the event record.

`lHandle`

The list in which the mouse-down event occurred.

Return Value

`TRUE` if the click was a double-click, or `FALSE` otherwise.

Discussion

The `LClick` function handles all user interaction until the user releases the mouse button.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

LClrCell

Clears the data contained in a cell. (Deprecated in Mac OS X v10.5.)

```
void LClrCell (
   Cell theCell,
   ListHandle lHandle
);
```

Parameters

`theCell`

The coordinates of the cell to be cleared.

`lHandle`

The list containing the cell given in the `theCell` parameter.

Discussion

If the cell coordinates specified by the `theCell` parameter are invalid, then the function does nothing.

If the data of a visible cell is cleared and the automatic drawing mode is enabled, the function updates the list.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

LDelColumn

Deletes one or more columns from a list. (Deprecated in Mac OS X v10.5.)

```
void LDelColumn (
   short count,
   short colNum,
   ListHandle lHandle
);
```

Parameters

count

The number of columns to delete, or 0 to delete all columns.

colNum

The column number of the first column to delete.

lHandle

The list from which to delete the columns.

Discussion

This function deletes columns starting at the column specified by the `colNum` parameter. If the column specified by `colNum` is invalid, then nothing is done.

Your application can quickly delete all columns from a list (and thus delete all cell data) simply by setting the `count` parameter to 0. The number of rows is left unchanged. Your application can achieve the same effect by setting the `colNum` parameter to `(**lHandle).dataBounds.left` and setting the `count` parameter to a value greater than `(**lHandle).dataBounds.right - (**lHandle).dataBounds.left`.

Columns whose column numbers are initially greater than `colNum` have their column numbers decreased by `count`.

If the automatic drawing mode is enabled and one or more of the columns deleted by this function are visible, then the list (including its scroll bars) is updated.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

LDelRow

Deletes one or more rows from a list. (Deprecated in Mac OS X v10.5.)

```
void LDelRow (
   short count,
   short rowNum,
   ListHandle lHandle
);
```

Parameters

count

The number of rows to delete, or 0 to delete all rows.

rowNum

The row number of the first row to delete.

`lHandle`
: The list from which to delete the rows.

Discussion

This function deletes rows starting at the row specified by the `rowNum` parameter. If the row specified by `rowNum` is invalid, then nothing is done.

Your application can quickly delete all rows from a list (and thus delete all cell data) simply by setting the `count` parameter to 0. The number of columns is left unchanged. Your application can achieve the same effect by setting the `rowNum` parameter to `(**lHandle).dataBounds.top` and setting the `count` parameter to a value greater than `(**lHandle).dataBounds.bottom - (**lHandle).dataBounds.top`.

Rows whose row numbers are initially greater than `rowNum` have their row numbers decreased by `count`.

If the automatic drawing mode is enabled and one or more of the rows deleted by the function are visible, then the list (including its scroll bars) is updated.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

LDispose

Disposes of the memory associated with a list. (Deprecated in Mac OS X v10.5.)

```
void LDispose (
   ListHandle lHandle
);
```

Parameters

`lHandle`
: The list to be disposed of.

Discussion

This function releases all memory allocated by the List Manager in creating a list. First, it issues a close request to the list definition function and calls the Control Manager function `DisposeControl` for the list's scroll bars (if any). The function then uses the Memory Manager to free the memory referenced by the cells field, then disposes of the list record itself.

Because it disposes of data associated with cells in your list, there is no need to clear the data from list cells or to delete individual rows and columns before calling this function.

This function does not dispose of any memory associated with a list that the List Manager has not allocated. In particular, it does not dispose of any memory referenced by the `userHandle` field of the list record. Your application is responsible for deallocating any memory it has allocated through the `userHandle` field before calling this function.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

LDraw

Draws a cell in a list. (Deprecated in Mac OS X v10.5.)

```
void LDraw (
   Cell theCell,
   ListHandle lHandle
);
```

Parameters

theCell

The cell to draw.

lHandle

The list containing the cell identified by the parameter `theCell`.

Discussion

The List Manager makes the list's graphics port the current port, sets the clipping region to the cell's rectangle, and calls the list definition function to draw the cell. It restores the clipping region and port before exiting.

Ordinarily, you should only need to use this function when the automatic drawing mode has been disabled.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

LGetCell

Copies a cell's data. (Deprecated in Mac OS X v10.5.)

```
void LGetCell (
   void *dataPtr,
   short *dataLen,
   Cell theCell,
   ListHandle lHandle
);
```

Parameters

dataPtr

A pointer to the location to which to copy the cell's data.

dataLen

On input, a pointer to the maximum number of bytes to copy. On return, a pointer to the number of bytes actually copied.

theCell

The cell whose data is to be copied.

lHandle

The list containing the cell specified by the parameter `theCell`.

Discussion

If the cell data is longer than `dataLen`, only `dataLen` bytes are copied and the `dataLen` parameter is unchanged. If the cell data is shorter than `dataLen`, then the function sets `dataLen` to the length in bytes of the cell's data.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

LGetCellDataLocation

Finds the memory location of cell data. (Deprecated in Mac OS X v10.5.)

```
void LGetCellDataLocation (
   short *offset,
   short *len,
   Cell theCell,
   ListHandle lHandle
);
```

Parameters

offset

On return, a pointer to the offset of the cell's data, specified from the beginning of the data handle referenced by the `cells` field of the list record.

len

On return, a pointer to the length of the cell's data in bytes.

theCell

The cell whose data's location is sought.

lHandle

The list containing the cell specified by the parameter `theCell`.

Discussion

Your application can use this function to read cell data. The `cells` field of the list record contains a handle to a relocatable block used to store all cell data. When this function returns, the `offset` parameter contains the offset of the specified cell's data in this relocatable block, and the `len` parameter specifies the length in bytes of the cell's data. In other words, the first byte of cell data is located at `Ptr(ORD4(lHandle^^.cells^) + offset)`, and the last byte of cell data is located at `Ptr(ORD4(lHandle^^.cells^) + offset + len)`. Your application should not modify the contents of the `cells` field directly. To change a cell's data, use the `LSetCell` (page 55) function or the `LAddToCell` (page 42) function.

If the cell coordinates specified by the parameter `theCell` are invalid, then the function sets the `offset` and `len` parameters to –1.

This function is also available as the `LFind` function.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Lists.h`

LGetSelect

Gets information about which cells are selected. (Deprecated in Mac OS X v10.5.)

```
Boolean LGetSelect (
   Boolean next,
   Cell *theCell,
   ListHandle lHandle
);
```

Parameters

next

Indicates whether the function should check only the cell specified by the parameter `theCell`, or whether it should try to find the next selected cell. If `next` is `TRUE`, then the function searches the list for the first selected cell beginning at the cell specified by `theCell`. (In particular, it first checks cells in row `theCell.v`, and then cells in the next row, and so on.)

If `next` is `FALSE`, then the function checks only the cell specified by the parameter `theCell`.

theCell

On input, a pointer to the first cell whose selection status should be checked. If `next` is `TRUE`, then, on return this parameter is a pointer to the next selected cell greater than or equal to the cell specified on input. Otherwise, this parameter remains unchanged.

lHandle

The list in which the selection is being checked.

Return Value
`TRUE` if `next` is `TRUE` and the function finds a selected cell, or if `next` is `FALSE` and the cell specified by `theCell` is selected. If this function does not find a selected cell, `FALSE`.

Special Considerations

This function is contained in a resource of resource type `'PACK'`. Calling it could result in the loading of the package resource and the allocation of memory. Thus, your application should not call this function from within an interrupt, such as in a completion function or VBL task.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Lists.h`

LLastClick

Determines the coordinates of the last cell clicked in a particular list. (Deprecated in Mac OS X v10.5.)

```
Cell LLastClick (
   ListHandle lHandle
);
```

Parameters

lHandle

The list to be checked for the last cell clicked.

Return Value

The cell coordinates of the last cell clicked. If the user has not clicked a cell since the creation of the list, then both the `h` and `v` fields of the cell returned contain negative numbers. See the description of the `Cell` data type.

Discussion

Note that the last cell clicked is not necessarily the last cell selected. The user could Shift-click in one cell and then drag the cursor to select other cells.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

LNew

Creates a new list in a window. (Deprecated in Mac OS X v10.5.)

```
ListHandle LNew (
   const Rect *rView,
   const ListBounds *dataBounds,
   Point cSize,
   short theProc,
   WindowRef theWindow,
   Boolean drawIt,
   Boolean hasGrow,
   Boolean scrollHoriz,
   Boolean scrollVert
);
```

Parameters

rView

A pointer to the rectangle in which to display the list, in local coordinates of the window specified by the `theWindow` parameter. This rectangle does not include the area to be taken up by the list's scroll bars.

dataBounds

A pointer to the initial data bounds for the list. By setting the `left` and `top` fields of this rectangle to (0,0) and the `right` and `bottom` fields to (`kMyInitialColumns, kMyInitialRows`), your application can create a list that has `kMyInitialColumns` columns and `kMyInitialRows` rows.

`cSize`
: The size of each cell in the list. If your application specifies (0,0) and is using the default list definition function, the List Manager sets the `v` coordinate of this parameter to the sum of the ascent, descent, and leading of the current font, and it sets the `h` coordinate using the following formula:

```
cSize.h = (rView.right - rView.left) / (dataBounds.right - dataBounds.left).
```

`theProc`
: The resource ID of the list definition function to use for the list. To use the default list definition function, which supports the display of unstyled text, specify a resource ID of 0.

`theWindow`
: A pointer to the window in which to install the list.

`drawIt`
: Indicates whether the List Manager should initially enable the automatic drawing mode. When the automatic drawing mode is enabled, the List Manager automatically redraws the list whenever a change is made to it. You can later change this setting using the `LSetDrawingMode` (page 56) function. Your application should leave the automatic drawing mode disabled only for short periods of time when making changes to a list (by, for example, adding rows and columns).

`hasGrow`
: Indicates whether the List Manager should leave room for a size box. The List Manager does not actually draw the grow icon. Usually, your application can draw it with the Window Manager's `DrawGrowIcon` function.

`scrollHoriz`
: Indicates whether the list should contain a horizontal scroll bar. Specify `TRUE` if your list requires a horizontal scroll bar; specify `FALSE` otherwise.

`scrollVert`
: Indicates whether the list should contain a vertical scroll bar. Specify `TRUE` if your list requires a vertical scroll bar; specify `FALSE` otherwise.

Return Value
A handle to the newly created list, or if the function cannot allocate the list, `NULL`. This might happen if there is not enough memory available or if the function cannot load the resource specified by the `theProc` parameter. If it returns successfully, then all of the fields of the list record referenced by the returned handle are correctly set. See the description of the `ListHandle` data type.

Discussion
If the list contains a horizontal or vertical scroll bar and the window specified by the parameter `theWindow` is visible, this function draws the scroll bar for the new list in the window just outside the list's visible rectangle specified by the `rView` parameter. This function does not, however, draw a 1-pixel border around the list's visible rectangle.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Lists.h`

LNextCell

Finds the next cell in a given row, in a given column, or in an entire list. (Deprecated in Mac OS X v10.5.)

```
Boolean LNextCell (
   Boolean hNext,
   Boolean vNext,
   Cell *theCell,
   ListHandle lHandle
);
```

Parameters

hNext

Indicates whether the function should check columns other than the current column. To get the next cell in a row, set this parameter to TRUE and set vNext to FALSE. The function then tries to find a cell whose coordinates are greater than those of the cell specified in theCell parameter but that is in the same row as theCell.

To get the next cell in a column, set this parameter to FALSE and set vNext to TRUE. The function then tries to find a cell whose coordinates are greater than those of the cell specified in theCell but that is in the same column as theCell.

To get the next cell in a list, set both this parameter and vNext to TRUE. This function then tries to find a cell whose coordinates are greater than those of the cell specified in the parameter theCell.

vNext

Indicates whether the function should check rows other than the current row. To get the next cell in a row, set this parameter to FALSE and set hNext to TRUE. The function then tries to find a cell whose coordinates are greater than those of the cell specified in theCell parameter but that is in the same row as theCell.

To get the next cell in a column, set this parameter to TRUE and set hNext to FALSE. The function then tries to find a cell whose coordinates are greater than those of the cell specified in theCell but that is in the same column as theCell.

To get the next cell in a list, set both this parameter and hNext to TRUE. This function then tries to find a cell whose coordinates are greater than those of the cell specified in the parameter theCell.

theCell

A pointer to the coordinates of the current cell. On return, a pointer to the next cell in the list, column or row being searched. If there are no more cells in the list, column or row, this parameter remains unchanged.

lHandle

The list in which to find the next cell.

Return Value

TRUE, if the function finds the next cell in the list, column or row being searched. FALSE, if the cell initially specified by theCell is the last in the row, column or list being searched. Also FALSE when both hNext and vNext are FALSE.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

Lists.h

LRect

Finds a rectangle that encloses a cell. (Deprecated in Mac OS X v10.5.)

```
void LRect (
   Rect *cellRect,
   Cell theCell,
   ListHandle lHandle
);
```

Parameters

cellRect

On return, a pointer to the rectangle enclosing the cell, specified in local coordinates of the list's graphics port. This rectangle is not necessarily within the list's rectangle.

theCell

The cell for which an enclosing rectangle is sought. This function does not check whether the cell is actually contained within the list's visible rectangle.

If this parameter specifies cell coordinates not contained within the list, this function sets the `cellRect` parameter to (0,0,0,0).

lHandle

The list containing the cell specified by the parameter `theCell`.

Discussion

Because the List Manager automatically draws cells, few applications need to call this function directly.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

LScroll

Scrolls a list a specified number of rows and columns. (Deprecated in Mac OS X v10.5.)

```
void LScroll (
   short dCols,
   short dRows,
   ListHandle lHandle
);
```

Parameters

dCols

The number of columns to scroll. Specify a positive number to scroll down (that is, each cell moves up), and a negative number to scroll up.

dRows

The number of rows to scroll. Specify a positive number to scroll right (that is, each cell moves left), and a negative number to scroll left.

lHandle

The list to be scrolled.

Discussion

The List Manager will not scroll beyond the data bounds of the list. If the automatic drawing mode is enabled, this function does all necessary updating of the list.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Lists.h`

LSearch

Finds a cell whose data matches data that you specify. (Deprecated in Mac OS X v10.5.)

```
Boolean LSearch (
   const void *dataPtr,
   short dataLen,
   ListSearchUPP searchProc,
   Cell *theCell,
   ListHandle lHandle
);
```

Parameters

dataPtr
: A pointer to the data being searched for.

dataLen
: The length in bytes of the data being searched for.

searchProc
: A pointer to a function to be used to compare the data being searched for with cell data. If `NULL`, the Text Utilities Package function `IUMagIDString` is used.

: If either the function pointed to by `searchProc` or `IUMagIDString` returns 0, `LSearch` has found a match; otherwise, it checks the next cell in the list.

theCell
: A pointer to the first cell to be searched. If the function finds a match, this parameter is, on return, a pointer to the coordinates of the first cell whose data matches the data being searched for.

lHandle
: The list to be searched.

Return Value
If the function finds a match, `TRUE`. Otherwise, `FALSE`.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Lists.h`

LSetCell

Changes the data contained in a cell. (Deprecated in Mac OS X v10.5.)

```
void LSetCell (
   const void *dataPtr,
   short dataLen,
   Cell theCell,
   ListHandle lHandle
);
```

Parameters

`dataPtr`
: A pointer to the new data for a cell.

`dataLen`
: The length in bytes of the new data.

`theCell`
: The coordinates of the cell to hold the new data.

`lHandle`
: The list containing the cell given in the `theCell` parameter.

Discussion

Any previous cell data in `theCell` is replaced. If there is insufficient memory in the heap, the function may fail to set the cell's data. If the cell coordinates specified by the `theCell` parameter are invalid, the function does nothing.

If the data of a visible cell is changed and the automatic drawing mode is enabled, the function updates the list.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

LSetDrawingMode

Changes the automatic drawing mode specified when creating a list. (Deprecated in Mac OS X v10.5.)

```
void LSetDrawingMode (
   Boolean drawIt,
   ListHandle lHandle
);
```

Parameters

`drawIt`
: Indicates whether the List Manager should enable the automatic drawing mode. Specify `TRUE` to enable the automatic drawing mode. Specify `FALSE` to disable the automatic drawing mode.

`lHandle`
: The list whose drawing mode is being changed.

Discussion

Your application can use the `LSetDrawingMode` function to enable or disable automatic drawing of lists. If your application uses `LSetDrawingMode` to temporarily disable list drawing, then it must call the `LDraw` (page 48) function to draw a cell when its appearance changes, or when new rows or columns are added to the list. .

While the automatic drawing mode is turned off, all cell drawing and highlighting are disabled, and the scroll bar does not function properly. Thus, your application should disable the automatic drawing mode only for short periods of time. After enabling it, your application should ensure that the list is redrawn.

This function is also available as the `LDoDraw` function.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

LSetSelect

Selects or deselects a cell. (Deprecated in Mac OS X v10.5.)

```
void LSetSelect (
   Boolean setIt,
   Cell theCell,
   ListHandle lHandle
);
```

Parameters

`setIt`

Indicates whether the function should select or deselect the specified cell. Specify `TRUE` to select the cell. If the cell is already selected, the function does nothing. Specify `FALSE` to deselect the cell. If the cell is already deselected, the function does nothing.

`theCell`

The cell to be selected or deselected.

`lHandle`

The list containing the cell to be selected or deselected.

Discussion

If a cell's selection status is changed and the cell is visible, `LSetSelect` redraws the cell.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

LSize

Changes the size of a list. (Deprecated in Mac OS X v10.5.)

```
void LSize (
   short listWidth,
   short listHeight,
   ListHandle lHandle
);
```

Parameters

listWidth

The new width (in pixels) of the list's visible rectangle.

listHeight

The new height (in pixels) of the list's visible rectangle.

lHandle

The list whose size is being changed.

Discussion

This function adjusts the lower-right side of the list so that the list's visible rectangle is the width and height specified by the `listWidth` and `listHeight` parameters.

Because the list's visible rectangle does not include room for the scroll bars, your application should make `listWidth` 15 pixels less than the desired width of the list if it contains a vertical scroll bar, and it should make `listHeight` 15 pixels less than the desired height of the list if it contains a horizontal scroll bar.

The contents of the list and the scroll bars are adjusted and redrawn as necessary. However, this function does not draw a border around the list's rectangle. Also, it does not erase any portions of the old list that may still be visible. This approach should not be a problem if your application only calls `LSize` after the user resizes a window containing a list in its lower-right corner.

Usually, you need to call this function only after calling the Window Manager function `SizeWindow`.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

LUpdate

Responds to an update event. (Deprecated in Mac OS X v10.5.)

```
void LUpdate (
   RgnHandle theRgn,
   ListHandle lHandle
);
```

Parameters

theRgn

The visible region of the list's port after a call to the Window Manager's `BeginUpdate` function.

lHandle
The list to be updated.

Discussion
This function redraws all visible cells in the list specified by the `lHandle` parameter that intersect the region specified by the parameter `theRgn`. It also redraws the scroll bars if they intersect the region.

You should bracket calls to `LUpdate` by calls to the Window Manager functions `BeginUpdate` and `EndUpdate`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

NewListClickLoopUPP

Creates a new universal procedure pointer (UPP) to a list click loop callback function. (Deprecated in Mac OS X v10.5.)

```
ListClickLoopUPP NewListClickLoopUPP (
   ListClickLoopProcPtr userRoutine
);
```

Parameters
userRoutine

Return Value
See the description of the `ListClickLoopUPP` data type.

Discussion
See the `ListClickLoopProcPtr` (page 11) callback for more information.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Lists.h`

NewListDefUPP

Creates a new universal procedure pointer (UPP) to a list definition callback function. (Deprecated in Mac OS X v10.5.)

```
ListDefUPP NewListDefUPP (
   ListDefProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value
See the description of the `ListDefUPP` data type.

Discussion
See the `ListDefProcPtr` (page 13) callback for more information.

Carbon Porting Notes

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In
`Lists.h`

NewListSearchUPP

Creates a new universal procedure pointer (UPP) to a list search callback function. (Deprecated in Mac OS X v10.5.)

```
ListSearchUPP NewListSearchUPP (
   ListSearchProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value
See the description of the `ListSearchUPP` data type.

Discussion
See the `ListSearchProcPtr` (page 15) callback for more information.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In
`Lists.h`

RegisterListDefinition

(Deprecated in Mac OS X v10.5.)

```
OSStatus RegisterListDefinition (
   SInt16 inResID,
   ListDefSpecPtr inDefSpec
);
```

Parameters

inResID

inDefSpec

Return Value

A result code.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

SetListCellIndent

(Deprecated in Mac OS X v10.5.)

```
void SetListCellIndent (
   ListHandle list,
   Point *indent
);
```

Parameters

list

indent

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

SetListClickLoop

(Deprecated in Mac OS X v10.5.)

```
void SetListClickLoop (
   ListHandle list,
   ListClickLoopUPP clickLoop
);
```

Parameters

list

clickLoop

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

SetListClickTime

(Deprecated in Mac OS X v10.5.)

```
void SetListClickTime (
   ListHandle list,
   SInt32 time
);
```

Parameters

list

time

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

SetListFlags

(Deprecated in Mac OS X v10.5.)

```
void SetListFlags (
   ListHandle list,
   OptionBits listFlags
);
```

Parameters

list

listFlags

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Lists.h`

SetListLastClick

(Deprecated in Mac OS X v10.5.)

```
void SetListLastClick (
   ListHandle list,
   Cell *lastClick
);
```

Parameters

list

lastClick

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Lists.h`

SetListPort

(Deprecated in Mac OS X v10.5.)

```
void SetListPort (
   ListHandle list,
   CGrafPtr port
);
```

Parameters

list

port

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Lists.h`

SetListRefCon

(Deprecated in Mac OS X v10.5.)

```
void SetListRefCon (
   ListHandle list,
   SInt32 refCon
);
```

Parameters

list

refCon

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

SetListSelectionFlags

(Deprecated in Mac OS X v10.5.)

```
void SetListSelectionFlags (
   ListHandle list,
   OptionBits selectionFlags
);
```

Parameters

list

selectionFlags

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

SetListUserHandle

(Deprecated in Mac OS X v10.5.)

```
void SetListUserHandle (
   ListHandle list,
   Handle userHandle
);
```

Parameters

list

userHandle

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Lists.h`

SetListViewBounds

(Deprecated in Mac OS X v10.5.)

```
void SetListViewBounds (
   ListHandle list,
   const Rect *view
);
```

Parameters

`list`

`view`

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

Document Revision History

This table describes the changes to *List Manager Reference*.

Date	Notes
2007-12-11	Added deprecation information.
2003-02-01	Updated formatting and linking.

Index