**LClick** Process mouse-down for list dragging and selection

#include <Lists.h> **List Manager Package**

```
Boolean     LClick(localPt, modifiers, theList );
Point       localPt ;       click location, in local coordinates
short       modifiers ;     shift, cmd, option, etc., from event record
ListHandle  theList ;       handle leading to a ListRec
            returns         Did double-click occur?
```

**LClick** handles all mouse tracking and clicking for a list. Call it from your event loop when a mousedown occurs inside the list viewing area of a window. By default, **LClick** keeps control until the button is released, but you can write a custom click loop routine to monitor the action.

*localPt* is a Point, expressed local coordinates local to a list window. Normally, it is the where field of an EventRecord when a mouseDown occurs in the display area of *theList*. Use **GlobalToLocal** to convert from global coordinates of the EventRecord to the local coordinates needed here.

*modifiers* is the state of the modifier keys (such as , Shift, and Option) at the time of the click. Use the value obtained in the modifiers field of the EventRecord.

*theList* is a handle leading to a variable-length ListRec structure. It is a value previously obtained via **LNew**.

**Returns**: a Boolean; it identifies whether or not a double-click occurred. It is one of:

| | |
|---|---|
| FALSE | No double-click. |
| TRUE | Double-click. The previous click occurred in the same cell within a short time span (see **GetDblTime**). |

---

Notes: **LClick** is at the heart of List Manager handling. After drawing the list, your application will simply idle in your main event loop (see **GetNextEvent**). When a mouseDown occurs in the content region of a list window(see **FindWindow**), call **LClick** to process the action.

If the click is outside of the list's viewing area and scroll bars, **LClick** returns immediately. Otherwise, it retains control, handling all screen updating and mouse tracking, including maintenance of the scroll bars. If the user attempts to drag outside the visible area, it automatically scrolls the list contents.

**LClick** performs cell selection according to the criteria established by the bits of ListRec.selFlags. This important field lets you specify how to handle shift- and - clicks and drags, and lets you limit selection to one cell at a time. See ListRec for details.

Here's a skeletal example:

```
GetNextEvent( everyEvent, &theEvent );

/*  . . . if it's a mouseDown event . . . */

FindWindow( theEvent.where, &whichWindow );

/*. . .  if it's in the content region of a list's window . . . */

GlobalToLocal( &theEvent.where );

isDbl = LClick( theEvent.where, theEvent.modifiers, theList );
if ( isDbl ) {
    /* . . . do something about the double click . . . */
}
```

See **LNew** for another example of usage.

## Lists and Dialogs

When using a list in a modeless dialog window, you will typically put the list in a rectangle defined as a userItem (i.e., when you create the dialog via **GetNewDialog**, use **LNew** to create the list in the rectangle obtained from **GetDItem**).

When **IsDialogEvent** (in your event loop) is TRUE, call **DialogSelect** and when it determines that the list's userItem was clicked, call **LClick** to process subsequent mouse action.

For a modal dialog, create and display the list right after calling **GetNewDialog**. When you call **ModalDialog**, your *filterProc* should call **LClick** to process hits on the list's userItem. Remember to call **LDispose** before you dispose of the dialog window.

## The lClikLoop routine

If you want some control over what happens inside **LClick**, you can store the address of a callback routine into ListRec.lClikLoop. **LClick** calls this routine repeatedly (with no parameters), as long as the mouse button remains pressed. Since scrolling is automatic, the most common use of such a "click loop" routine is moot. However, you can take other actions, such as displaying information about a cell as the mouse drags over it.

Your custom routine should be declared as:

```
pascal Boolean myLClikLoop()
{
    //. . . do whatever you like;  the list's window is active and frontmost  . .
    //. . . the mouse is at (*theList)->mouseLoc  (in local coordinates). . .
    //. . . use LRect and PtInRect to help find which cell is at that location

    return (TRUE)                    /* return FALSE to abort LClick */
}
```