Demonstration Program Lists Listing

```
// *******************************************************************************************
// Lists.h                                                                CLASSIC EVENT MODEL
// *******************************************************************************************
//
// This program allows the user to open a window and a movable modal dialog by choosing the
// relevant items in the Demonstration menu.  The window and the dialog box both contain two
// lists.
//
// The cells of one list in the window, and of both lists in the dialog box, contain text.
// The cells of the second list in the window contain icons.
//
// The text lists use the default list definition function.  The list with the icons uses a
// custom list definition function.
//
// The currently active list is indicated by a keyboard focus frame, and can be changed by
// clicking in the non-active list or by pressing the tab key.
//
// The text list in the window uses the default cell-selection algorithm; accordingly,
// multiple cells, including discontiguous multiple cells, may be selected.  The cell-
// selection algorithm for the other lists is customised so as to allow the selection of only
// one cell at a time.
//
// All lists support arrow key selection.  The text list in the window and one of the lists in
// the dialog box support type selection.
//
// The window is provided with an "Extract" push button.  When this button is clicked, or when
// the user double clicks in one of the lists, the current selections in the lists are
// extracted and displayed in the bottom half of the window.  In the dialog box, the user's
// selections are displayed in static text fields embedded in placards below each list.
//
// The program utilises the following resources:
//
// •  A 'plst' resource.
//
// •  An 'MBAR' resource, and 'MENU' resources for Apple, File, Edit and Demonstration menus
//    (preload, non-purgeable).
//
// •  A 'WIND' resource (purgeable) (initially not visible).
//
// •  A'DLOG' resource (purgeable) (initially not visible) and associated 'dlgx', 'dftb', and
//    'DITL' resources (purgeable).
//
// •  'CNTL' resources (purgeable) for various controls in both the window and dialog box,
//    including the list controls for the dialog box.
//
// •  'ldes' resources associated with the list controls for the dialog box.
//
// •  'STR#' resources (purgeable) containing the text strings for the text lists and for the
//    titles of the icons.
//
// •  An icon suite (non-purgeable) containing the icons for icon list.
//
// •  An 'LDEF' resource (preload, locked, non-purgeable) containing the custom list
//    definition function  used by the icon list.
//
// •  'hrct' and 'hwin' (purgeable) resources for balloon help.
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
// *******************************************************************************************

// ............................................................................... includes

#include <Carbon.h>
```

```
// .................................................................................... defines

#define rMenubar                 128
#define mAppleApplication        128
#define  iAbout                  1
#define mFile                    129
#define  iQuit                   12
#define mDemonstration           131
#define  iHandMadeLists          1
#define  iListControlLists       2
#define rListsWindow             128
#define  cExtractButton          128
#define  cGroupBox               129
#define  cSoftwareStaticText     130
#define  cHardwareStaticText     131
#define  rTextListStrings        128
#define  rIconListIconSuiteBase  128
#define  rIconListStrings        129
#define rListsDialog             128
#define  iDateFormatList         4
#define  iWatermarkList          5
#define  iDateFormatStaticText   7
#define  iWatermarkStaticText    9
#define  rDateFormatStrings      130
#define  rWatermarkStrings       131
#define kUpArrow                 0x1e
#define kDownArrow               0x1f
#define kTab                     0x09
#define kScrollBarWidth          15
#define kMaxKeyThresh            120
#define kSystemLDEF              0
#define MAX_UINT32               0xFFFFFFFF

// ................................................................................... typedefs

typedef struct
{
  ListHandle textListHdl;
  ListHandle iconListHdl;
  ControlRef extractButtonHdl;
} docStructure, **docStructureHandle;

typedef struct
{
  RGBColor     backColour;
  PixPatHandle backPixelPattern;
  Pattern      backBitPattern;
} backColourPattern;

// ....................................................................... function prototypes

void        main                        (void);
void        doPreliminaries             (void);
OSErr       quitAppEventHandler         (AppleEvent *,AppleEvent *,SInt32);
void        doEvents                    (EventRecord *);
void        doAdjustMenus               (void);
void        doMenuChoice                (SInt32);
void        doSaveBackground            (backColourPattern *);
void        doRestoreBackground         (backColourPattern *);
void        doSetBackgroundWhite        (void);

void        doOpenListsWindow           (void);
void        doKeyDown                   (SInt8,EventRecord *);
void        doUpdate                    (EventRecord *);
void        doActivate                  (EventRecord *);
void        doActivateWindow            (WindowRef,Boolean);
void        doInContent                 (EventRecord *);
ListHandle  doCreateTextList            (WindowRef,Rect,SInt16,SInt16);
void        doAddRowsAndDataToTextList  (ListHandle,SInt16,SInt16);
void        doAddTextItemAlphabetically (ListHandle,Str255);
ListHandle  doCreateIconList            (WindowRef,Rect,SInt16,ListDefUPP);
void        doAddRowsAndDataToIconList  (ListHandle,SInt16);
void        doHandleArrowKey            (SInt8,EventRecord *,Boolean);
void        doArrowKeyMoveSelection     (ListHandle,SInt8,Boolean);
void        doArrowKeyExtendSelection   (ListHandle,SInt8,Boolean);
void        doTypeSelectSearch          (ListHandle,EventRecord *);
SInt16      searchPartialMatch          (Ptr,Ptr,SInt16,SInt16);
Boolean     doFindFirstSelectedCell     (ListHandle,Cell *);
void        doFindLastSelectedCell      (ListHandle,Cell *);
void        doFindNewCellLoc            (ListHandle,Cell,Cell *,SInt8,Boolean);
void        doSelectOneCell             (ListHandle,Cell);
void        doMakeCellVisible           (ListHandle,Cell);
void        doResetTypeSelection        (void);
void        doRotateCurrentList         (void);
void        doDrawFrameAndFocus         (ListHandle,Boolean);
void        doExtractSelections         (void);
void        doDrawSelections            (Boolean);
void        listDefFunction             (SInt16,Boolean,Rect *,Cell,SInt16,SInt16,ListHandle);
void        doLDEFDraw                  (Boolean,Rect *,Cell,SInt16,ListHandle);
void        doLDEFHighlight             (Rect *);

void        doListsDialog               (void);
Boolean     eventFilter                 (DialogPtr,EventRecord *,SInt16 *);

// *******************************************************************************************
// Lists.c
// *******************************************************************************************

// .......................................................................................... includes

#include "Lists.h"

// .................................................................................. global variables

ListDefUPP        gListDefFunctionUPP;
Boolean           gRunningOnX = false;
Boolean           gDone;
backColourPattern gBackColourPattern;

// ************************************************************************************** main

void  main(void)
{
  MenuBarHandle menubarHdl;
  SInt32        response;
  MenuRef       menuRef;
  EventRecord   eventStructure;

  // ............................................................................ do preliminaries

  doPreliminaries();

  // ........................................................... create universal procedure pointer

  gListDefFunctionUPP = NewListDefUPP((ListDefProcPtr) listDefFunction);

  // ................................................................... set up menu bar and menus

  menubarHdl = GetNewMBar(rMenubar);
  if(menubarHdl == NULL)
    ExitToShell();
  SetMenuBar(menubarHdl);
  DrawMenuBar();

  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
    {
      DeleteMenuItem(menuRef,iQuit);
      DeleteMenuItem(menuRef,iQuit - 1);
      DisableMenuItem(menuRef,0);
    }

    gRunningOnX = true;
  }

  // .......................................................................................... enter eventLoop

  gDone = false;

  while(!gDone)
  {
    if(WaitNextEvent(everyEvent,&eventStructure,MAX_UINT32,NULL))
      doEvents(&eventStructure);
  }
}

// ************************************************************************** doPreliminaries

void  doPreliminaries(void)
{
  OSErr osError;

  MoreMasterPointers(256);
  InitCursor();
  FlushEvents(everyEvent,0);

  osError = AEInstallEventHandler(kCoreEventClass,kAEQuitApplication,
                            NewAEEventHandlerUPP((AEEventHandlerProcPtr) quitAppEventHandler),
                            0L,false);
  if(osError != noErr)
    ExitToShell();
}

// *************************************************************************** doQuitAppEvent

OSErr  quitAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefcon)
{
  OSErr    osError;
  DescType returnedType;
  Size     actualSize;

  osError = AEGetAttributePtr(appEvent,keyMissedKeywordAttr,typeWildCard,&returnedType,NULL,0,
                              &actualSize);

  if(osError == errAEDescNotFound)
  {
    gDone = true;
    osError = noErr;
  }
  else if(osError == noErr)
    osError = errAEParamMissed;

  return osError;
}

// ********************************************************************************* doEvents

void  doEvents(EventRecord *eventStrucPtr)
{
  WindowRef          windowRef;
  WindowPartCode     partCode;
  docStructureHandle docStrucHdl;
  SInt8              charCode;
  switch(eventStrucPtr->what)
  {
    case mouseDown:
      partCode = FindWindow(eventStrucPtr->where,&windowRef);

      switch(partCode)
      {
        case kHighLevelEvent:
          AEProcessAppleEvent(eventStrucPtr);
          break;

        case inMenuBar:
          doAdjustMenus();
          doMenuChoice(MenuSelect(eventStrucPtr->where));
          break;

        case inContent:
          if(windowRef != FrontWindow())
            SelectWindow(windowRef);
          else
            doInContent(eventStrucPtr);
          break;

        case inDrag:
          DragWindow(windowRef,eventStrucPtr->where,NULL);
          break;

        case inGoAway:
          docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
          LDispose((*docStrucHdl)->textListHdl);
          LDispose((*docStrucHdl)->iconListHdl);
          DisposeHandle((Handle) docStrucHdl);
          DisposeWindow(windowRef);
          break;
      }
      break;

    case keyDown:
      charCode = eventStrucPtr->message & charCodeMask;
      if((eventStrucPtr->modifiers & cmdKey) != 0)
      {
        doAdjustMenus();
        doMenuChoice(MenuEvent(eventStrucPtr));
      }
      if(FrontWindow())
        doKeyDown(charCode,eventStrucPtr);
      break;

    case updateEvt:
      doUpdate(eventStrucPtr);
      break;

    case activateEvt:
      doActivate(eventStrucPtr);
      break;

    case osEvt:
      switch((eventStrucPtr->message >> 24) & 0x000000FF)
      {
        case suspendResumeMessage:
          if((eventStrucPtr->message & resumeFlag) == 1)
            SetThemeCursor(kThemeArrowCursor);
          break;
      }
      break;
  }
}

// ****************************************************************************** doAdjustMenus
```

```
void  doAdjustMenus(void)
{
  MenuRef  menuRef;

  menuRef = GetMenuRef(mDemonstration);

  if(FrontWindow())
    DisableMenuItem(menuRef,1);
  else
    EnableMenuItem(menuRef,1);
}

// ****************************************************************************** doMenuChoice

void  doMenuChoice(SInt32 menuChoice)
{
  MenuID        menuID;
  MenuItemIndex menuItem;

  menuID = HiWord(menuChoice);
  menuItem = LoWord(menuChoice);

  if(menuID == 0)
    return;

  switch(menuID)
  {
    case mAppleApplication:
      if(menuItem == iAbout)
        SysBeep(10);
      break;

    case mFile:
      if(menuItem == iQuit)
        gDone = true;
      break;

    case mDemonstration:
      if(menuItem == iHandMadeLists)
        doOpenListsWindow();
      else if(menuItem == iListControlLists)
        doListsDialog();
      break;
  }

  HiliteMenu(0);
}

// ************************************************************************** doSaveBackground

void  doSaveBackground(backColourPattern *gBackColourPattern)
{
  GrafPtr      currentPort;
  PixPatHandle backPixPatHdl;

  GetPort(&currentPort);

  GetBackColor(&gBackColourPattern->backColour);
  gBackColourPattern->backPixelPattern  = NULL;

  backPixPatHdl = NewPixPat();
  GetPortBackPixPat(currentPort,backPixPatHdl);

  if((*backPixPatHdl)->patType != 0)
    gBackColourPattern->backPixelPattern = backPixPatHdl;
  else
    gBackColourPattern->backBitPattern = *(PatPtr) (*(*backPixPatHdl)->patData);
  DisposePixPat(backPixPatHdl);
}

// ********************************************************************** doRestoreBackground

void  doRestoreBackground(backColourPattern *gBackColourPattern)
{
  RGBBackColor(&gBackColourPattern->backColour);

  if(gBackColourPattern->backPixelPattern)
    BackPixPat(gBackColourPattern->backPixelPattern);
  else
    BackPat(&gBackColourPattern->backBitPattern);
}

// ********************************************************************** doSetBackgroundWhite

void  doSetBackgroundWhite(void)
{
  RGBColor whiteColour = { 0xFFFF, 0xFFFF, 0xFFFF };
  Pattern  whitePattern;

  RGBBackColor(&whiteColour);
  GetQDGlobalsWhite(&whitePattern);
  BackPat(&whitePattern);
}

// *******************************************************************************************
// WindowList.c
// *******************************************************************************************

// .................................................................................. includes

#include "Lists.h"

// .......................................................................... global variables

ListHandle       gCurrentListHdl;
Str255           gStringArray[16];
TypeSelectRecord gTSStruct;
SInt16           gTSResetThreshold;
ListHandle       gTSLastListHit;

extern ListDefUPP        gListDefFunctionUPP;
extern backColourPattern gBackColourPattern;
extern Boolean           gRunningOnX;

// ************************************************************************ doOpenListsWindow

void  doOpenListsWindow(void)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  ControlRef         controlRef;
  Str255             software = "\pSoftware:";
  Str255             hardware = "\pHardware:";
  SInt16             fontNum;
  Rect               textListRect, pictListRect;
  ListHandle         textListHdl, iconListHdl;

  // ........... open window, attach document structure, set and save background colour/pattern

  if(!(windowRef = GetNewCWindow(rListsWindow,NULL,(WindowRef) -1)))
    ExitToShell();

  if(!(docStrucHdl = (docStructureHandle) NewHandle(sizeof(docStructure))))
    ExitToShell();
  SetWRefCon(windowRef,(SInt32) docStrucHdl);
  SetPortWindowPort(windowRef);

  SetThemeWindowBackground(windowRef,kThemeBrushDialogBackgroundActive,true);
  if(!gRunningOnX)
    doSaveBackground(&gBackColourPattern);
  else
    doSetBackgroundWhite();

  // ............................................................................ create window's controls

  CreateRootControl(windowRef,&controlRef);

  if(!((*docStrucHdl)->extractButtonHdl = GetNewControl(cExtractButton,windowRef)))
    ExitToShell();

  if(!(controlRef = GetNewControl(cSoftwareStaticText,windowRef)))
    ExitToShell();
  SetControlData(controlRef,kControlNoPart,kControlStaticTextTextTag,software[0],
               &software[1]);

  if(!(controlRef = GetNewControl(cHardwareStaticText,windowRef)))
    ExitToShell();
  SetControlData(controlRef,kControlNoPart,kControlStaticTextTextTag,hardware[0],
               &hardware[1]);

  if(!(controlRef = GetNewControl(cGroupBox,windowRef)))
    ExitToShell();

  // ...................................................................................... set window's font

  GetFNum("\pGeneva",&fontNum);
  TextFont(fontNum);
  TextSize(10);

  // ..................................... create lists, assign handles to document structure fields

  SetRect(&textListRect,21,26,151,130);
  SetRect(&pictListRect,169,26,236,130);

  textListHdl = doCreateTextList(windowRef,textListRect,1,kSystemLDEF);
  iconListHdl = doCreateIconList(windowRef,pictListRect,1,gListDefFunctionUPP);

  (*docStrucHdl)->textListHdl = textListHdl;
  (*docStrucHdl)->iconListHdl  = iconListHdl;

  // ............................... assign handles to list structure refCon fields for linked ring

  SetListRefCon(textListHdl,(SInt32) iconListHdl);
  SetListRefCon(iconListHdl,(SInt32) textListHdl);

  // ....................................................................... make text list the current list

  gCurrentListHdl = textListHdl;

  // ............................................................................................. show window

  ShowWindow(windowRef);
}

// ********************************************************************************* doKeyDown

void  doKeyDown(SInt8 charCode,EventRecord *eventStrucPtr)
{
  docStructureHandle docStrucHdl;
  Boolean            allowExtendSelect;

  docStrucHdl = (docStructureHandle) GetWRefCon(FrontWindow());

  if(charCode == kTab)
  {
    doRotateCurrentList();
  }
  else if(charCode == kUpArrow || charCode == kDownArrow)
  {
    if(gCurrentListHdl == (*docStrucHdl)->textListHdl)
      allowExtendSelect = true;
    else
      allowExtendSelect = false;

    doHandleArrowKey(charCode,eventStrucPtr,allowExtendSelect);
  }
  else
  {
    if(gCurrentListHdl == (*docStrucHdl)->textListHdl)
      doTypeSelectSearch((*docStrucHdl)->textListHdl,eventStrucPtr);
  }
}

// ********************************************************************************* doUpdate

void  doUpdate(EventRecord *eventStrucPtr)
{
  WindowRef          windowRef;
  Rect               theRect;
  docStructureHandle docStrucHdl;
  ListHandle         textListHdl, iconListHdl;
  RgnHandle          visibleRegionHdl = NewRgn();

  windowRef = (WindowRef) eventStrucPtr->message;
  SetPortWindowPort(windowRef);

  BeginUpdate(windowRef);

  SetRect(&theRect,24,188,237,381);
  EraseRect(&theRect);

  GetPortVisibleRegion(GetWindowPort(windowRef),visibleRegionHdl);
  UpdateControls(windowRef,visibleRegionHdl);

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  textListHdl = (*docStrucHdl)->textListHdl;
  iconListHdl = (*docStrucHdl)->iconListHdl;

  if(visibleRegionHdl)
  {
    if(gRunningOnX)
    {
      if(IsWindowHilited(windowRef))
        TextMode(srcOr);
      else
        TextMode(grayishTextOr);
    }

    LUpdate(visibleRegionHdl,textListHdl);
    LUpdate(visibleRegionHdl,iconListHdl);
    DisposeRgn(visibleRegionHdl);
  }

  doDrawFrameAndFocus(textListHdl,IsWindowHilited(windowRef));
  doDrawFrameAndFocus(iconListHdl,IsWindowHilited(windowRef));

  doDrawSelections(windowRef == FrontWindow());

  EndUpdate(windowRef);
}

// ******************************************************************************* doActivate
```

```
void  doActivate(EventRecord *eventStrucPtr)
{
  WindowRef windowRef;
  Boolean   becomingActive;

  windowRef = (WindowRef) eventStrucPtr->message;
  becomingActive = ((eventStrucPtr->modifiers & activeFlag) == activeFlag);
  doActivateWindow(windowRef,becomingActive);
}

// ************************************************************************* doActivateWindow

void  doActivateWindow(WindowRef windowRef,Boolean becomingActive)
{
  GrafPtr            oldPort;
  ControlRef         controlRef;
  docStructureHandle docStrucHdl;
  ListHandle         textListHdl, iconListHdl;
  RgnHandle          visibleRegionHdl = NewRgn();

  GetPort(&oldPort);
  SetPortWindowPort(windowRef);

  if(!gRunningOnX)
  {
    GetRootControl(windowRef,&controlRef);
    if(becomingActive)
      ActivateControl(controlRef);
    else
      DeactivateControl(controlRef);
  }

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  textListHdl = (*docStrucHdl)->textListHdl;
  iconListHdl = (*docStrucHdl)->iconListHdl;

  if(visibleRegionHdl)
    GetPortVisibleRegion(GetWindowPort(windowRef),visibleRegionHdl);

  if(becomingActive)
  {
    LActivate(true,textListHdl);
    LActivate(true,iconListHdl);

    if(!gRunningOnX)
    {
      TextMode(srcOr);
      if(visibleRegionHdl)
      {
        LUpdate(visibleRegionHdl,textListHdl);
        LUpdate(visibleRegionHdl,iconListHdl);
        DisposeRgn(visibleRegionHdl);
      }
    }

    doDrawFrameAndFocus(textListHdl,true);
    doDrawFrameAndFocus(iconListHdl,true);

    doResetTypeSelection();

    doDrawSelections(true);
  }
  else
  {
    LActivate(false,textListHdl);
    LActivate(false,iconListHdl);

    if(!gRunningOnX)
    {
      TextMode(grayishTextOr);
      if(visibleRegionHdl)
      {
        LUpdate(visibleRegionHdl,textListHdl);
        LUpdate(visibleRegionHdl,iconListHdl);
        DisposeRgn(visibleRegionHdl);
      }
    }

    doDrawFrameAndFocus(textListHdl,false);
    doDrawFrameAndFocus(iconListHdl,false);

    doDrawSelections(false);
  }

  SetPort(oldPort);
}

// ******************************************************************************* doInContent

void  doInContent(EventRecord *eventStrucPtr)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  ListHandle         textListHdl, iconListHdl;
  Rect               textListRect, pictListRect;
  Point              mouseXY;
  ControlRef         controlRef;
  Boolean            isDoubleClick;

  windowRef = FrontWindow();
  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  textListHdl = (*docStrucHdl)->textListHdl;
  iconListHdl = (*docStrucHdl)->iconListHdl;

  GetListViewBounds((*docStrucHdl)->textListHdl,&textListRect);
  GetListViewBounds((*docStrucHdl)->iconListHdl,&pictListRect);
  textListRect.right += kScrollBarWidth;
  pictListRect.right += kScrollBarWidth;

  SetPortWindowPort(windowRef);
  mouseXY = eventStrucPtr->where;
  GlobalToLocal(&mouseXY);

  if(PtInRect(mouseXY,&textListRect) || PtInRect(mouseXY,&pictListRect))
  {
    if((PtInRect(mouseXY,&textListRect) && gCurrentListHdl != textListHdl) ||
       (PtInRect(mouseXY,&pictListRect) && gCurrentListHdl != iconListHdl))
    {
      doRotateCurrentList();
    }

    isDoubleClick = LClick(mouseXY,eventStrucPtr->modifiers,gCurrentListHdl);
    if(isDoubleClick)
      doExtractSelections();
  }
  else if(FindControl(mouseXY,windowRef,&controlRef))
  {
    if(TrackControl(controlRef,mouseXY,NULL))
    {
      if(controlRef == (*docStrucHdl)->extractButtonHdl)
        doExtractSelections();
    }
  }
}

// ************************************************************************** doCreateTextList

ListHandle  doCreateTextList(WindowRef windowRef,Rect listRect,SInt16 numCols,SInt16 lDef)
```

```
{
  Rect       dataBounds;
  Point      cellSize;
  ListHandle textListHdl;
  Cell       theCell;

  SetRect(&dataBounds,0,0,numCols,0);
  SetPt(&cellSize,0,0);

  listRect.right = listRect.right - kScrollBarWidth;

  textListHdl = LNew(&listRect,&dataBounds,cellSize,lDef,windowRef,true,false,false,true);

  doAddRowsAndDataToTextList(textListHdl,rTextListStrings,15);

  SetPt(&theCell,0,0);
  LSetSelect(true,theCell,textListHdl);

  doResetTypeSelection();

  return textListHdl;
}

// *************************************************************** doAddRowsAndDataToTextList

void  doAddRowsAndDataToTextList(ListHandle textListHdl,SInt16 stringListID,
                                 SInt16 numberOfStrings)
{
  SInt16 stringIndex;
  Str255 theString;

  for(stringIndex = 1;stringIndex < numberOfStrings + 1;stringIndex++)
  {
    GetIndString(theString,stringListID,stringIndex);
    doAddTextItemAlphabetically(textListHdl,theString);
  }
}

// ************************************************************** doAddTextItemAlphabetically

void  doAddTextItemAlphabetically(ListHandle listHdl,Str255 theString)
{
  Boolean    found;
  ListBounds dataBounds;
  SInt16     totalRows, currentRow, cellDataOffset, cellDataLength;
  DataHandle dataHandle;
  Cell       aCell;

  found = false;

  GetListDataBounds(listHdl,&dataBounds);
  totalRows = dataBounds.bottom - dataBounds.top;
  currentRow = -1;

  while(!found)
  {
    currentRow += 1;
    if(currentRow == totalRows)
      found = true;
    else
    {
      SetPt(&aCell,0,currentRow);
      LGetCellDataLocation(&cellDataOffset,&cellDataLength,aCell,listHdl);

      dataHandle = GetListDataHandle(listHdl);
      MoveHHi((Handle) dataHandle);
      HLock((Handle) dataHandle);

      if(CompareText(theString + 1,((Ptr) (*listHdl)->cells[0] + cellDataOffset),
                    StrLength(theString),cellDataLength,NULL) == -1)
      {
        found = true;
      }

      HUnlock((Handle) dataHandle);
    }
  }

  currentRow = LAddRow(1,currentRow,listHdl);
  SetPt(&aCell,0,currentRow);

  LSetCell(theString + 1,(SInt16) StrLength(theString),aCell,listHdl);
}

// *************************************************************************** doCreateIconList

ListHandle  doCreateIconList(WindowRef windowRef,Rect listRect,SInt16 numCols,ListDefUPP lDef)
{
  Rect        dataBounds;
  Point       cellSize;
  ListHandle  iconListHdl;
  Cell        theCell;
  ListDefSpec listDefSpec;

  SetRect(&dataBounds,0,0,numCols,0);
  SetPt(&cellSize,52,52);

  listRect.right = listRect.right - kScrollBarWidth;

  listDefSpec.u.userProc = lDef;

  CreateCustomList(&listRect,&dataBounds,cellSize,&listDefSpec,windowRef,true,false,false,
                   true,&iconListHdl);

  SetListSelectionFlags(iconListHdl,lOnlyOne);

  doAddRowsAndDataToIconList(iconListHdl,rIconListIconSuiteBase);

  SetPt(&theCell,0,0);
  LSetSelect(true,theCell,iconListHdl);

  return iconListHdl;
}

// ******************************************************************* doAddRowsAndDataToIconList

void  doAddRowsAndDataToIconList(ListHandle iconListHdl,SInt16 iconSuiteBase)
{
  ListBounds dataBounds;
  SInt16     rowNumber, suiteIndex, index = 0;
  Handle     iconSuiteHdl;
  Cell       theCell;

  GetListDataBounds(iconListHdl,&dataBounds);
  rowNumber = dataBounds.bottom;

  for(suiteIndex = iconSuiteBase;suiteIndex < (iconSuiteBase + 10);suiteIndex++)
  {
    GetIconSuite(&iconSuiteHdl,suiteIndex,kSelectorAllLargeData);

    rowNumber = LAddRow(1,rowNumber,iconListHdl);
    SetPt(&theCell,0,rowNumber);
    LSetCell(&iconSuiteHdl,sizeof(iconSuiteHdl),theCell,iconListHdl);

    rowNumber += 1;
  }
}
```

```
// *************************************************************************** doHandleArrowKey

void  doHandleArrowKey(SInt8 charCode,EventRecord *eventStrucPtr,Boolean allowExtendSelect)
{
  Boolean moveToTopBottom = false;

  if(eventStrucPtr->modifiers & cmdKey)
    moveToTopBottom = true;

  if(allowExtendSelect && (eventStrucPtr->modifiers & shiftKey))
    doArrowKeyExtendSelection(gCurrentListHdl,charCode,moveToTopBottom);
  else
    doArrowKeyMoveSelection(gCurrentListHdl,charCode,moveToTopBottom);
}

// ********************************************************************** doArrowKeyMoveSelection

void  doArrowKeyMoveSelection(ListHandle listHdl,SInt8 charCode,Boolean moveToTopBottom)
{
  Cell currentSelection, newSelection;

  if(doFindFirstSelectedCell(listHdl,&currentSelection))
  {
    if(charCode == kDownArrow)
      doFindLastSelectedCell(listHdl,&currentSelection);

    doFindNewCellLoc(listHdl,currentSelection,&newSelection,charCode,moveToTopBottom);

    doSelectOneCell(listHdl,newSelection);
    doMakeCellVisible(listHdl,newSelection);
  }
}

// ******************************************************************** doArrowKeyExtendSelection

void  doArrowKeyExtendSelection(ListHandle listHdl,SInt8 charCode,Boolean moveToTopBottom)
{
  Cell currentSelection, newSelection;

  if(doFindFirstSelectedCell(listHdl,&currentSelection))
  {
    if(charCode == kDownArrow)
      doFindLastSelectedCell(listHdl,&currentSelection);

    doFindNewCellLoc(listHdl,currentSelection,&newSelection,charCode,moveToTopBottom);

    if(!(LGetSelect(false,&newSelection,listHdl)))
      LSetSelect(true,newSelection,listHdl);

    doMakeCellVisible(listHdl,newSelection);
  }
}

// ************************************************************************** doTypeSelectSearch

void  doTypeSelectSearch(ListHandle listHdl,EventRecord *eventStrucPtr)
{
  Cell          theCell;
  ListSearchUPP searchPartialMatchUPP;

  if((gTSLastListHit != listHdl) || ((eventStrucPtr->when - gTSStruct.tsrLastKeyTime) >=
     gTSResetThreshold) || (StrLength(gTSStruct.tsrKeyStrokes) == 63))
    doResetTypeSelection();

  gTSLastListHit = listHdl;
  gTSStruct.tsrLastKeyTime = eventStrucPtr->when;

  TypeSelectNewKey(eventStrucPtr,&gTSStruct);
  SetPt(&theCell,0,0);

  searchPartialMatchUPP  = NewListSearchUPP((ListSearchProcPtr) searchPartialMatch);

  if(LSearch(gTSStruct.tsrKeyStrokes + 1,StrLength(gTSStruct.tsrKeyStrokes),
             searchPartialMatchUPP,&theCell,listHdl))
  {
    LSetSelect(true,theCell,listHdl);
    doSelectOneCell(listHdl,theCell);
    doMakeCellVisible(listHdl,theCell);
  }

  DisposeListSearchUPP(searchPartialMatchUPP);
}

// ************************************************************************ searchPartialMatch

SInt16  searchPartialMatch(Ptr searchDataPtr,Ptr cellDataPtr,SInt16 cellDataLen,
                           SInt16 searchDataLen)
{
  SInt16 result;

  if((cellDataLen > 0) && (cellDataLen >= searchDataLen))
    result = IdenticalText(cellDataPtr,searchDataPtr,searchDataLen,searchDataLen,NULL);
  else
    result = 1;

  return result;
}

// ******************************************************************* doFindFirstSelectedCell

Boolean  doFindFirstSelectedCell(ListHandle listHdl,Cell *theCell)
{
  Boolean result;

  SetPt(theCell,0,0);
  result = LGetSelect(true,theCell,listHdl);

  return result;
}

// ******************************************************************** doFindLastSelectedCell

void  doFindLastSelectedCell(ListHandle listHdl,Cell *theCell)
{
  Cell    aCell;
  Boolean moreCellsInList;

  if(doFindFirstSelectedCell(listHdl,&aCell))
  {
    while(LGetSelect(true,&aCell,listHdl))
    {
      *theCell = aCell;
      moreCellsInList = LNextCell(true,true,&aCell,listHdl);
    }
  }
}

// ************************************************************************** doFindNewCellLoc

void  doFindNewCellLoc(ListHandle listHdl,Cell oldCellLoc,Cell *newCellLoc,SInt8 charCode,
                       Boolean moveToTopBottom)
{
  ListBounds dataBounds;
  SInt16     listRows;

  GetListDataBounds(listHdl,&dataBounds);
  listRows = dataBounds.bottom - dataBounds.top;
  *newCellLoc = oldCellLoc;

  if(moveToTopBottom)
  {
    if(charCode == kUpArrow)
      (*newCellLoc).v = 0;
    else if(charCode == kDownArrow)
      (*newCellLoc).v = listRows - 1;
  }
  else
  {
    if(charCode ==  kUpArrow)
    {
      if(oldCellLoc.v != 0)
        (*newCellLoc).v = oldCellLoc.v - 1;
    }
    else if(charCode == kDownArrow)
    {
      if(oldCellLoc.v != listRows - 1)
        (*newCellLoc).v = oldCellLoc.v + 1;
    }
  }
}

// ************************************************************************** doSelectOneCell

void  doSelectOneCell(ListHandle listHdl,Cell theCell)
{
  Cell    nextSelectedCell;
  Boolean moreCellsInList;

  if(doFindFirstSelectedCell(listHdl,&nextSelectedCell))
  {
    while(LGetSelect(true,&nextSelectedCell,listHdl))
    {
      if(nextSelectedCell.v != theCell.v)
        LSetSelect(false,nextSelectedCell,listHdl);
      else
        moreCellsInList = LNextCell(true,true,&nextSelectedCell,listHdl);
    }

    LSetSelect(true,theCell,listHdl);
  }
}

// ************************************************************************ doMakeCellVisible

void  doMakeCellVisible(ListHandle listHdl,Cell newSelection)
{
  ListBounds visibleRect;
  SInt16     dRows;

  GetListVisibleCells(listHdl,&visibleRect);

  if(!(PtInRect(newSelection,&visibleRect)))
  {
    if(newSelection.v > visibleRect.bottom - 1)
      dRows = newSelection.v - visibleRect.bottom + 1;
    else if(newSelection.v < visibleRect.top)
      dRows = newSelection.v - visibleRect.top;

    LScroll(0,dRows,listHdl);
  }
}

// ********************************************************************** doResetTypeSelection

void  doResetTypeSelection(void)
{
  TypeSelectClear(&gTSStruct);
  gTSLastListHit = NULL;
  gTSResetThreshold = 2 * LMGetKeyThresh();
  if(gTSResetThreshold > kMaxKeyThresh)
    gTSResetThreshold = kMaxKeyThresh;
}

// ********************************************************************** doRotateCurrentList

void  doRotateCurrentList(void)
{
  ListHandle oldListHdl, newListHdl;

  oldListHdl = gCurrentListHdl;

  newListHdl = (ListHandle) GetListRefCon(gCurrentListHdl);
  gCurrentListHdl = newListHdl;

  doDrawFrameAndFocus(oldListHdl,true);
  doDrawFrameAndFocus(newListHdl,true);
}

// ********************************************************************** doDrawFrameAndFocus

void  doDrawFrameAndFocus(ListHandle listHdl,Boolean inState)
{
  Rect borderRect;

  GetListViewBounds(listHdl,&borderRect);
  borderRect.right += kScrollBarWidth;

  if(!gRunningOnX)
    doRestoreBackground(&gBackColourPattern);
  else
    InvalWindowRect(FrontWindow(),&borderRect);

  DrawThemeFocusRect(&borderRect,false);

  if(inState)
    DrawThemeListBoxFrame(&borderRect,kThemeStateActive);
  else
    DrawThemeListBoxFrame(&borderRect,kThemeStateInactive);

  if(listHdl == gCurrentListHdl)
    DrawThemeFocusRect(&borderRect,inState);

  if(!gRunningOnX)
    doSetBackgroundWhite();
}

// ********************************************************************** doExtractSelections

void  doExtractSelections(void)
{
  docStructureHandle docStrucHdl;
  ListHandle         textListHdl, iconListHdl;
  SInt16             a, cellIndex, offset, dataLen;
  ListBounds         dataBounds;
  Cell               theCell;
  Rect               theRect;

  docStrucHdl = (docStructureHandle) GetWRefCon(FrontWindow());
  textListHdl = (*docStrucHdl)->textListHdl;
  iconListHdl = (*docStrucHdl)->iconListHdl;

  for(a=0;a<16;a++)
    gStringArray[a][0] = 0;

  GetListDataBounds(textListHdl,&dataBounds);
  for(cellIndex = 0;cellIndex < dataBounds.bottom;cellIndex++)
  {
    SetPt(&theCell,0,cellIndex);
    if(LGetSelect(false,&theCell,textListHdl))
    {
      LGetCellDataLocation(&offset,&dataLen,theCell,textListHdl);
      LGetCell(gStringArray[cellIndex] + 1,&dataLen,theCell,textListHdl);
      gStringArray[cellIndex][0] = (SInt8) dataLen;
    }
  }

  SetPt(&theCell,0,0);
  LGetSelect(true,&theCell,iconListHdl);
  GetIndString(gStringArray[15],rIconListStrings,theCell.v + 1);

  SetRect(&theRect,24,181,233,380);
  InvalWindowRect(FrontWindow(),&theRect);
}

// *********************************************************************** doDrawSelections

void  doDrawSelections(Boolean inState)
{
  Rect        theRect;
  SInt16      a, nextLine = 190;
  CFStringRef stringRef;

  if(inState == kThemeStateActive)
    TextMode(srcOr);
  else
    TextMode(grayishTextOr);

  SetRect(&theRect,22,179,235,382);
  EraseRect(&theRect);

  for(a=0;a<15;a++)
  {
    if(gStringArray[a][0] != 0)
    {
      stringRef = CFStringCreateWithPascalString(NULL,gStringArray[a],
                                          kCFStringEncodingMacRoman);
      SetRect(&theRect,36,nextLine,240,nextLine + 15);
      DrawThemeTextBox(stringRef,kThemeSmallSystemFont,0,false,&theRect,teJustLeft,
                       NULL);
        nextLine += 12;
    }
  }

  stringRef = CFStringCreateWithPascalString(NULL,gStringArray[15],
                                          kCFStringEncodingMacRoman);
  SetRect(&theRect,170,190,240,205);
  DrawThemeTextBox(stringRef,kThemeSmallSystemFont,0,false,&theRect,teJustLeft,
                   NULL);

  TextMode(srcOr);
}

// ************************************************************************ listDefFunction

void listDefFunction(SInt16 message,Boolean selected,Rect *cellRect,Cell theCell,
                     SInt16 dataOffset,SInt16 dataLen,ListHandle theList)
{
  switch(message)
  {
    case lDrawMsg:
      doLDEFDraw(selected,cellRect,theCell,dataLen,theList);
      break;
    case lHiliteMsg:
      doLDEFHighlight(cellRect);
      break;
  }
}

// ******************************************************************************* doLDEFDraw

void  doLDEFDraw(Boolean selected,Rect *cellRect,Cell theCell,SInt16 dataLen,
                 ListHandle theList)
{
  GrafPtr oldPort;
  Rect    drawRect;
  Handle  iconSuiteHdl;
  Str255  theString;

  GetPort(&oldPort);

  SetPort(GetListPort(theList));

  EraseRect(cellRect);

  drawRect = *cellRect;

  drawRect.top += 2;
  drawRect.left += 10;
  drawRect.right -= 10;
  drawRect.bottom -= 18;

  if(dataLen == sizeof(Handle))
  {
    LGetCell(&iconSuiteHdl,&dataLen,theCell,theList);

    if(GetListActive(theList))
      PlotIconSuite(&drawRect,kAlignNone,kTransformNone,iconSuiteHdl);
    else
      PlotIconSuite(&drawRect,kAlignNone,kTransformDisabled,iconSuiteHdl);
  }

  GetIndString(theString,129,theCell.v + 1);
  SetRect(&drawRect,drawRect.left - 10,drawRect.top + 36,drawRect.right + 10,
          drawRect.bottom + 16);
  TETextBox(&theString[1],theString[0],&drawRect,teCenter);

  if(selected)
    doLDEFHighlight(cellRect);

  SetPort(oldPort);
}

// ************************************************************************** doLDEFHighlight

void  doLDEFHighlight(Rect *cellRect)
{
  UInt8  hiliteVal;

  hiliteVal = LMGetHiliteMode();
  BitClr(&hiliteVal,pHiliteBit);
  LMSetHiliteMode(hiliteVal);

  InvertRect(cellRect);
}

// ******************************************************************************************
// DialogLists.c
// ******************************************************************************************

// .......................................................................................... includes
```

```
#include "Lists.h"

// ***************************************************************************** doListsDialog

void  doListsDialog(void)
{
  DialogPtr      dialogPtr;
  GrafPtr        oldPort;
  ModalFilterUPP eventFilterUPP;
  ControlRef     dateFormatControlRef, watermarkControlRef, controlRef;
  ListHandle     dateFormatListHdl, watermarkListHdl;
  SInt16         itemHit;
  Cell           theCell;
  SInt16         dataLen, offset;
  Str255         dateFormatString, watermarkString;
  Boolean        wasDoubleClick = false;

  // .............................. explicitly deactivate front window if it exists, create dialog

  if(FrontWindow())
    doActivateWindow(FrontWindow(),false);

  if(!(dialogPtr = GetNewDialog(rListsDialog,NULL,(WindowRef) -1)))
    ExitToShell();

  GetPort(&oldPort);
  SetPortDialogPort(dialogPtr);

  // ................................................................ set default and cancel push buttons

  SetDialogDefaultItem(dialogPtr,kStdOkItemIndex);
  SetDialogCancelItem(dialogPtr,kStdCancelItemIndex);

  // ................................ create universal procedure pointer for event filter function

  eventFilterUPP = NewModalFilterUPP((ModalFilterProcPtr) eventFilter);

  // .............. add rows to lists, store data in their cells, modify cell selection algorithm

  GetDialogItemAsControl(dialogPtr,iDateFormatList,&dateFormatControlRef);
  GetControlData(dateFormatControlRef,kControlEntireControl,kControlListBoxListHandleTag,
                 sizeof(dateFormatListHdl),&dateFormatListHdl,NULL);

  doAddRowsAndDataToTextList(dateFormatListHdl,rDateFormatStrings,17);

  SetListSelectionFlags(dateFormatListHdl,lOnlyOne);

  SetPt(&theCell,0,0);
  LSetSelect(true,theCell,dateFormatListHdl);

  GetDialogItemAsControl(dialogPtr,iWatermarkList,&watermarkControlRef);
  GetControlData(watermarkControlRef,kControlEntireControl,kControlListBoxListHandleTag,
                 sizeof(watermarkListHdl),&watermarkListHdl,NULL);

  doAddRowsAndDataToTextList(watermarkListHdl,rWatermarkStrings,12);

  SetListSelectionFlags(watermarkListHdl,lOnlyOne);

  SetPt(&theCell,0,0);
  LSetSelect(true,theCell,watermarkListHdl);

  // ................................................................ show dialog and set keyboard focus

  ShowWindow(GetDialogWindow(dialogPtr));

  SetKeyboardFocus(GetDialogWindow(dialogPtr),watermarkControlRef,1);
  SetKeyboardFocus(GetDialogWindow(dialogPtr),dateFormatControlRef,1);

  // ........................................................................................ enter ModalDialog loop
  do
  {
    ModalDialog(eventFilterUPP,&itemHit);

    if(itemHit == iDateFormatList)
    {
      SetPt(&theCell,0,0);
      LGetSelect(true,&theCell,dateFormatListHdl);
      LGetCellDataLocation(&offset,&dataLen,theCell,dateFormatListHdl);
      LGetCell(dateFormatString + 1,&dataLen,theCell,dateFormatListHdl);
      dateFormatString[0] = (SInt8) dataLen;

      GetDialogItemAsControl(dialogPtr,iDateFormatStaticText,&controlRef);
      SetControlData(controlRef,kControlEntireControl,kControlStaticTextTextTag,
                     dateFormatString[0],&dateFormatString[1]);
      Draw1Control(controlRef);

      GetControlData(dateFormatControlRef,kControlEntireControl,
                     kControlListBoxDoubleClickTag,sizeof(wasDoubleClick),&wasDoubleClick,
                     NULL);
    }
    else if(itemHit == iWatermarkList)
    {
      SetPt(&theCell,0,0);
      LGetSelect(true,&theCell,watermarkListHdl);
      LGetCellDataLocation(&offset,&dataLen,theCell,watermarkListHdl);
      LGetCell(watermarkString + 1,&dataLen,theCell,watermarkListHdl);
      watermarkString[0] = (SInt8) dataLen;

      GetDialogItemAsControl(dialogPtr,iWatermarkStaticText,&controlRef);
      SetControlData(controlRef,kControlEntireControl,kControlStaticTextTextTag,
                     watermarkString[0],&watermarkString[1]);
      Draw1Control(controlRef);

      GetControlData(watermarkControlRef,kControlEntireControl,
                     kControlListBoxDoubleClickTag,sizeof(wasDoubleClick),&wasDoubleClick,
                     NULL);
    }

  } while(itemHit != kStdOkItemIndex && wasDoubleClick == false);

  // ..................................................................................... clean up

  DisposeDialog(dialogPtr);
  DisposeModalFilterUPP(eventFilterUPP);

  SetPort(oldPort);
}

// ******************************************************************************* eventFilter

Boolean  eventFilter(DialogPtr dialogPtr,EventRecord *eventStrucPtr,SInt16 *itemHit)
{
  Boolean    handledEvent;
  GrafPtr    oldPort;
  WindowRef  windowRef;
  SInt8      charCode;
  ControlRef controlRef, focusControlRef;
  ListHandle watermarkListHdl;

  handledEvent = false;

  windowRef = GetDialogWindow(dialogPtr);

  if((eventStrucPtr->what == updateEvt) && ((WindowRef) eventStrucPtr->message != windowRef))
  {
    doUpdate(eventStrucPtr);
  }
  else
  {
    GetPort(&oldPort);
    SetPortDialogPort(dialogPtr);

    if(eventStrucPtr->what == keyDown)
    {
      charCode = eventStrucPtr->message & charCodeMask;

      if(charCode != kUpArrow && charCode != kDownArrow && charCode != kTab)
      {
        GetDialogItemAsControl(dialogPtr,iWatermarkList,&controlRef);
        GetControlData(controlRef,kControlEntireControl,kControlListBoxListHandleTag,
                   sizeof(watermarkListHdl),&watermarkListHdl,NULL);
        GetKeyboardFocus(GetDialogWindow(dialogPtr),&focusControlRef);
        if(controlRef == focusControlRef)
        {
          doTypeSelectSearch(watermarkListHdl,eventStrucPtr);
          Draw1Control(controlRef);
        }

        handledEvent = true;
      }
    }
    else
    {
      handledEvent = StdFilterProc(dialogPtr,eventStrucPtr,itemHit);
    }

    SetPort(oldPort);
  }

  return handledEvent;
}

// *******************************************************************************************
```

Demonstration Program Lists Comments

When this program is run, the user should open the window and movable modal dialog box by choosing the relevant items in the Demonstration menu. The user should manipulate the lists in the window and dialog box, noting their behaviour in the following circumstances:

- Changing the active list (that is, the current target of mouse and keyboard activity) by clicking in the non-active list and by using the Tab key to cycle between the two lists.
- Scrolling the active list using the vertical scroll bars, including dragging the scroll box and clicking in the scroll arrows and gray areas.
- Clicking, and clicking and dragging, in the active list so as to select a particular cell, including dragging the cursor above and below the list to automatically scroll the list to the desired cell.
- Pressing the Up-Arrow and Down-Arrow keys, noting that this action changes the selected cell and, where necessary, scrolls the list to make the newly-selected cell visible.
- In the lists in the window:
  - Double-clicking on a cell in the active list.
  - Pressing the Command-key as well as the Up-Arrow and Down-Arrow keys, noting that, in both the text list and the picture list, this results in the top-most or bottom-most cell being selected.
- In the text list in the window:
  - Shift-clicking and dragging in the list to make contiguous multiple cell selections.
  - Command-clicking and dragging in the list to make discontiguous multiple cell selections, noting the differing effects depending on whether the cell initially clicked is selected or not selected.
  - Shift-clicking outside a block of multiple cell selections, including between two fairly widely separated discontiguous selected cells.
  - Pressing the Shift-key as well as the Up-Arrow and Down-Arrow keys, noting that this results in multiple cell selections.
- When the text list in the window, or the right hand list in the dialog, is the active list, typing the text of a particular cell so as to select that cell by type selection, noting the effects of any excessive delay between keystrokes.

The user should also send the program to the background and bring it to the foreground again, noting the list deactivation/activation effects.

List.h

defines

rListsWindow represents the resource ID of the window's 'WIND' resource. The following four constants represent the resource IDs of the window's controls. The next three constants represent the resource IDs of 'STR#' resource containing the strings for the window's text list, the icon suite for the icon list, and the 'STR#' resource containing the strings for the icon titles.

rListsDialog represents the resource ID of the dialog's 'DLOG', 'dlgx', and 'dftb' resources. The following four constants represent the item numbers of items in the dialog's 'DITL' resource. The next two constants represent the resource IDs of the 'STR#' resources containing the strings for the dialog's lists.

The next three constants represent the character codes returned by the Up Arrow, Down Arrow, and Tab keys. kScrollBarWidth represents the width of the lists' vertical scroll bars. kMaxKeyThresh is used in the type selection function. kSystemLDEF represents the resource ID of the default list definition function.

typedefs

The docStructure type which will be used to store the handles to the two list structures for the window and the reference to the window's push button. The handle to this structure will be stored in the window object.

The backColourPattern data type will be used to save and restore the background colour and pattern.

Lists.c

Lists.c is simply the basic "engine" supporting the demonstration. There is little in this file which has not featured in previous demonstration programs.

main

A universal procedure pointer is created for the custom list definition function used by the second list in the window.

doEvents

At the inGoAway case in the mouseDown case, a handle to the window's document structure is retrieved so as to be able to pass the handles to the window's two list structures in the two calls to LDispose. LDispose disposes of all memory associated with the specified list.

doSaveBackground, doRestoreBackground, and doSetBackgroundWhite

doSaveBackground and doRestoreBackground save and restore the background colour and the background bit or pixel pattern. doSetBackgroundWhite sets the background colour to white and the background pattern to the pattern white.

WindowList.c

WindowList.c contains the functions pertaining to the lists in the window.

Global Variables

gCurrentListHandle will be assigned the handle to the list structure associated with the currently active list in the window. gStringArray will be assigned strings representing the selections from the lists. The next three global variables are associated with the type selection functions.

doOpenListsWindow

doOpenListsWindow creates the window and its controls, and calls the functions which create the two lists for the window.

The call to GetNewCWindow creates the window. The call to NewHandle creates a block for the window's document structure. SetWRefCon associates the document structure with the window.

SetThemeWindowBackground is called to set the window's background colour/pattern and, if the program is running on Mac OS 8/9, doSaveBackground is called to save this background colour/pattern for later use in the function doDrawFrameAndFocus. The call to doSetBackgroundWhite at this point is required only on Mac OS X to ensure that the background within the list frames is drawn in white.

CreateRootControl creates a root control for the window so as to simplify the task of activating and deactivating the window's controls. (This call is not required on Mac OS X because, on Mac OS X, the root control will be created automatically for windows which have at least one control.) The window's remaining controls are then created.

The calls to doCreateTextList and doCreateIconList create the lists. First, the rectangles in which the lists are to be displayed are defined. These are then passed in the calls to doCreateTextList and doCreateIconList. The handles to the list structures returned by these functions are then assigned to the relevant fields of the window's document structure.

The next block assigns the icon list's handle to the refCon field of the text list's list structure and the text list's handle to the refCon field of the icon list's list structure. This establishes the "linked ring" which will be used to facilitate the rotation of the active list via Tab key presses.

The penultimate line establishes the text list as the currently active list. ShowWindow is then called to display the window.

doKeyDown

The first line gets the handle to the document structure.

If the key pressed was the Tab key, doRotateCurrentList is called to change the currently active list.

If the key pressed was either the Up Arrow or the Down Arrow key, and if the current list is the text list, a variable which specifies whether multiple cell selections via the keyboard are permitted is set to true. If the current list is the icon list, this variable is set to false. This variable is then passed as a parameter in a call to doHandleArrowKey, which further processes the Arrow key event.

If the key pressed was neither the Tab key, the Up Arrow key, or the Down Arrow key, and if the active list is the text list, the event is passed to doTypeSelectSearch function for further processing.

doUpdate

doUpdate handles update events. Between the usual calls to BeginUpdate and EndUpdate, the calls to LUpdate update (that is, redraw) the lists and the calls to doDrawFrameAndFocus draw the focus rectangles in the appropriate state. The call to doDrawSelections simply draws the current list selections in the rectangle at the bottom of the window.

The calls to TextMode if the program is running on Mac OS X are required because of certain machinations in the function doDrawFrameAndFocus.

doActivateWindow

doActivateWindow activates and deactivates the content area of the window.

If the program is running on Mac OS 8/9 the root control is activated or deactivated, as appropriate, thus activating or deactivating all the controls in the window. (This is done automatically on Mac OS X.)

If the window is becoming active, the following occurs. For both lists, LActivate is called with true passed in the first parameter so as to highlight the currently selected cells. The calls to TextMode and LUpdate when the program is running on Mac OS 8/9 are required for cosmetic purposes only. LUpdate causes a redraw of all of the list's text in the srcOr mode. The calls to doDrawFrameAndFocus draw the list box frames in the active state and ensure that a keyboard focus frame is redrawn around the currently active list. The call to doResetTypeSelection resets certain variables used by the type selection function. (This latter is necessary because it is possible that, while the application was in the background, the user may have changed the "Delay Until Repeat" setting in the Keyboard control panel (Mac OS 8/9) or System Preferences/Keyboard (Mac OS X), a value which is used in the type selection function.) doDrawSelections redraws the current list selections in the srcOr mode.

Except for the call to doResetTypeSelection, much the same occurs if the window is becoming inactive, except that LActivate removes highlighting from the currently selected cells, LUpdate redraws the lists' text in the grayishTextOr mode (on Mac OS 8/9), doDrawFrameAndFocus removes the keyboard focus frame from the active list and draws the list box frames in the inactive state, and doDrawSelections redraws the current list selections in the grayishTextOr mode.

doInContent

doInContent further processes mouse-down events in the content region of the window.

In the first block, handles to the two lists are retrieved. The first two lines of the next block get copies of the lists' display rectangles. Since these rectangles do not include the scroll bars, they are then expanded to the right to encompass the scroll bar areas.

The next block converts the mouse coordinates of the mouse-down to local coordinates required by the following call to PtInRect.

If the mouse click was in one of the list rectangles, and if that rectangle is not the current list's rectangle, doRotateCurrentList is called to change the currently active list. Next, LClick is called to handle all user action until the mouse-button is released. If LClick returns true, a double-click occurred, in which case doExtractSelections is called to extract the contents of the currently selected cells.

If the mouse-down event was not within one of the list rectangles, FindControl is called to determine if there is an enabled control under the mouse cursor. If so, TrackControl is called to handle user action until the mouse button is released. If the cursor is still within the control when the mouse button is released, and if the control is the window's single push button, doExtractSelections is called to extract the contents of the currently selected cells.

doCreateTextList

doCreateTextList, supported by the two following functions, creates the text list.

SetRect sets the rectangle which will be passed as the rDataBnds parameter of the LNew call to specify one column and (initially) no rows. SetPt sets the variable that will be passed as the cellSize parameter so as to specify that the List Manager should automatically calculate the cell size. The next line adjusts the received list rectangle to eliminate the area occupied by the vertical scroll bar.

The call to LNew creates the list. The parameters specify that the List Manager is to calculate the cell size, the default list definition function is to be used, automatic drawing mode is to be enabled, no room is to be left for a size box, the list is not to have a horizontal scroll bar, and the list is to have a vertical scroll bar.

The next line calls doAddRowsAndDataToTextList, which adds rows to the list and stores data in its cells.

The next two lines set the cell at the topmost row as the initially-selected cell. doResetTypeSelection calls a function which initialises certain variables used by the type selection function. The last line returns the handle to the list.

doAddRowsAndDataToTextList

doAddRowsAndDataToTextList adds rows to the text list and stores data in its cells. The data is retrieved from a 'STR#' resource.

The for loop copies the strings from the specified 'STR#' resource and passes each string as a parameter in a call to doAddTextItemAlphabetically, which inserts a new row into the list and copies the string to that cell.

Note at this point that the strings in the 'STR#' resource are not arranged alphabetically.

doAddTextItemAlphabetically

doAddTextItemAlphabetically does the heavy work in the process of adding the rows to the text list and storing the text. The bulk of the code is concerned with building the list in such a way that the cells are arranged in alphabetical order.

The first line sets the variable found to false. The next line sets the variable totalRows to the number of rows in the list. (In this program, this is initially 0.) The next line sets the variable currentRow to -1.

The while loop executes until the variable found is set to true.

Within the loop, the first line increments currentRow to 0. The first time this function is called, currentRow will equal totalRows at this point and the loop will thus immediately exit to the first line below the loop. The call to LAddRow at this line adds one row to the list, inserting it before the row specified by currentRow. The list now has one row (cell (0,0)). LSetCell copies the string to this cell. The function then exits, to be re-called by doAddRowsAndDataToTextList for as many times as there are remaining strings.

The second time the function is called, the first line in the while loop again sets currentRow to 0. This time, however, the if block does not execute because totalRows is now 1. Thus SetPt sets the variable aCell to (0,0) and LGetCellDataLocation retrieves the offset and length of the data in cell (0,0). This allows the string in this cell to be alphabetically compared with the "incoming" string using CompareText. If the incoming string is "less than" the string in cell (0,0), CompareText returns -1, in which case:

- The loop exits. LaddRow inserts one row before cell(0,0) and the old cell (0,0) thus becomes cell(0,1). The list now contains two rows.

- SetPt sets cell (0,0) and LSetCell copies the "incoming" string to that cell. The "incoming" string, which was alphabetically "less than" the first string, is thus assigned to the correct cell in the alphabetical sense.

- The function then exits, to be re-called for as many times as there are remaining strings.

If, on the other hand, CompareText returns 0 (strings equal) or 1 ("incoming" string "greater than" the string in cell (0,0), the loop repeats. At the first line in the loop, currentRow is incremented to 1, which is equal to totalRows. Accordingly, the loop exits immediately, LAddRow inserts a row before cell (0,1) (that is, cell (0,1) is created), LSetCell copies the "incoming" string to that cell, and the function exits, to be re-called for as many times as there are remaining strings.

The ultimate result of all this is an alphabetically ordered list.

doCreateIconList

doCreateIconList, supported by the following function, creates the icon list. This list uses a custom list definition function; accordingly, CreateCustomList, rather than LNew, is used to create the list.

SetRect sets the rectangle which will be passed as the dataBounds parameter of the CreateCustomList call to specify one column and (initially) no rows. SetPt sets the variable which will be passed as the cellSize parameter so as to specify that the List Manager should make the cell size of all cells 52 by 52 pixels. The next line adjusts the list rectangle to reflect the area occupied by the vertical scroll bar.

The line after that assigns the universal procedure pointer to the custom list definition function to the userProc field of the variable of type ListDefSpec that will be passed in the theSpec parameter of the CreateCustomList call.

The call to CreateCustomList creates the list. The parameters specify that the List Manager is to make all cell sizes 52 by 52 pixels, a custom list definition function is to be used, automatic drawing mode is to be enabled, no room is to be left for a size box, the list is not to have a horizontal scroll bar, and the list is to have a vertical scroll bar.

The next line assigns lOnlyOne to the selFlags field of the list structure, meaning that the List manager's cell selection algorithm is modified so as to allow only one cell to be selected at any one time.

The next line calls doAddRowsAndDataToIconList, which adds rows to the list and stores data in its cells.

The next two lines select the cell at the topmost row as the initially-selected cell. The last line returns the handle to the list.

doAddRowsAndDataToIconList

doAddRowsAndDataToIconList adds eight rows to the icon list and stores a handle to an icon suite in each of the eight cells.

The first two lines set the variable rowNumber to the current number of rows, which is 0.

The for loop executes eight times. Each time through the loop, the following occurs:

- GetIconSuite creates a new icon family and fills it with icons with the specified resource ID and of the types specified in the last parameter (that is, large icons only).

- LAddRow inserts a new row in the list at the location specified by the variable rowNumber. SetPt sets this cell and LSetCell stores the handle to the icon suite as the cell's data. The last line increments the variable rowNumber, which is passed in the SetPt call.

doHandleArrowKey

doHandleArrowKey further processes Down Arrow and Up Arrow key presses. This is the first of eleven functions dedicated to the handling of key-down events.

Recall that doHandleArrowKey's third parameter (allowExtendSelect) is set to true by the calling function (doKeyDown) only if the text list is the currently active list.

The first line sets the variable moveToTopBottom to false, which can be regarded as the default. At the next two lines, if the Command key was also down at the time of the Arrow key press, this variable is set to true.

If the text list is the currently active list, and if the Shift key was down, doArrowKeyExtendSelection is called; otherwise, otherwise doArrowKeyMoveSelection is called.

doArrowKeyMoveSelection

doArrowKeyMoveSelection further processes those Arrow key presses which occurred when either list was the currently active list but the Shift key was not down. The effect of this function is to deselect all currently selected cells and to select the appropriate cell according to, firstly, which Arrow key was pressed (Up or Down) and, secondly, whether the Command key was down at the same time.

The if statement calls doFindFirstSelectedCell, which searches for the first selected cell in the specified list. That function returns true if a selected cell is found, or false if the list contains no selected cells.

If true is returned by that call, the variable currentSelection will hold the first selected cell. However, this could be changed by the second line within the if block if the key pressed was the Down-Arrow. doFindLastSelectedCell finds the last selected cell (which could, of course, well be the same cell as the first selected cell if only one cell is currently selected). Either way, the variable currentSelection will now hold either the only cell currently selected, the first cell selected (if more than one cell is currently selected and the key pressed was the Up Arrow), or the last cell selected (if more than one cell is currently selected and the key pressed was the Down Arrow).

With that established, doFindNewCellLoc determines the next cell to select, which will depend on, amongst other things, whether the Command key was down at the time of the key press (that is, on whether the moveToTopBottom parameter is true or false). The variable newSelection will contain the results of that determination.

doSelectOneCell then deselects all currently selected cells and selects the cell specified by the variable newSelection.

It is possible that the newly-selected cell will be outside the list's display rectangle. Accordingly, doMakeCellVisible, if necessary, scrolls the list until the newly-selected cell appears at the top or the bottom of the display rectangle.

doArrowKeyExtendSelection

doArrowKeyExtendSelection is similar to the previous function except that it adds additional cells to the currently selected cells. This function is called only when the text list is the currently active list and the Shift key was down at the time of the Arrow key press.

By the fifth line, the variable currentSelection will hold either the only cell currently selected, the first cell selected (if more than one cell is currently selected and the key pressed was the Up Arrow), or the last cell selected (if more than one cell is currently selected and the key pressed was the Down Arrow).

doFindNewCellLoc determines the next cell to select, which will depend on, amongst other things, whether the Command key was down at the time of the key press (that is, on whether the moveToTopBottom parameter is true or false). The variable newSelection will contain the results of that determination. The similarities between this function and doArrowKeyMoveSelection end there.

At the next line, LGetSelect is called to check whether the cell specified by the variable newSelection is selected. If it is not, LSetSelect selects it. (This check by LGetSelect is advisable because, for example, the first-selected cell as this function is entered might be cell (0,0), that is, the very top row. If the Up-Arrow was pressed in this circumstance, and as will be seen, doFindNewCellLoc returns cell (0,0) in the newSelection variable. There is no point in selecting a cell which is already selected.)

It is possible that the newly-selected cell will be outside the list's display rectangle. Accordingly, doMakeCellVisible, if necessary, scrolls the list until the newly-selected cell appears at the top or the bottom of the display rectangle.

doTypeSelectSearch

doTypeSelectSearch is the main type selection function. It is called from doKeyDown whenever a key-down or auto-key event is received and the key pressed is not the Tab key, the Up Arrow key or the Down Arrow key.

The global variables gTSStruct, gTSResetThreshold, and gTSLastListHit are central to the operation of doTypeSelectSearch. gTSStruct holds the current type selection search string entered by the user and the time in ticks of the last key press. gTSResetThreshold holds the number of ticks which must elapse before type selection resets, and is dependent on the value the user sets in the "Delay Until Repeat" setting in the Keyboard control panel (Mac OS 8/9) or System Preferences/Keyboard (Mac OS X). gTSLastListHit holds a handle to the last list that type selection affected.

The first block will cause doResetTypeSelection to be called to reset type selection if either of the following situations prevail: if the list which is the target of the current key press is not the same as the list which was the target of the previous key press; if a number of ticks since the last key press is greater than the number stored in gTSResetThreshold; if the current length of the type selection string is 63 characters.

The next line stores the handle to the list which is the target of the current key press in gTSLastListHit so as to facilitate the comparison at the first if block the next time the function is called. The next line stores the time of the current key press in gTSLastKeyTime for the same purpose.

The call to TypeSelectNewKey extracts the character code from the message field of the event structure and adds the character to the tsrKeyStrokes field of gTSStruct. That field now holds all the characters received since the last type selection reset.

SetPt sets the variable theCell to represent the first cell in the list. This is passed as a parameter in the LSearch call, and specifies the first cell to examine. LSearch examines this cell and all subsequent cells in an attempt to find a match to the type selection string. If a match exists, the cell in which the first match is found will be returned in theCell parameter, LSearch will return true and the following three lines will execute.

Of those three lines, ordinarily only the call to LSetSelect (which deselects all currently selected cells and selects the specified cell) and the last line (which, if necessary, scrolls the list so that the newly-selected cell is visible in the display rectangle) would be necessary. However, because doSelectOneCell has no effect unless there is currently at least one selected cell in the list, the call to doSelectOneCell is included to account for the situation where the user may have deselected all of the text list cells using Command-clicking or dragging.

The actual matching task is performed by the match (callback) function the universal procedure pointer to which is passed in the third parameter to the LSearch call. Note that the default match function has been replaced by the custom callback function doSearchPartialMatch.

doSearchPartialMatch

doSearchPartialMatch is the custom match function called by LSearch, in the previous function, to attempt to find a match to the current type selection string. For the default function to return a match, the type selection string would have to match an entire cell's text. doSearchPartialMatch, however, only compares the characters of the type selection string with the same number of characters in the cell's text. For example, if the type selection string is currently "fr" and a cell with the text "Fractal Painter" exists, doSearchPartialMatch will report a match.

A comparison by IdenticalText (which returns 0 if the strings being compared are equal) is only made if the cell contains data and the length of that data is greater than or equal to the current length of the type selection string. If these conditions do not prevail, doSearchPartialMatch returns 1 (no match found). If these conditions do prevail, IdenticalText is called with, importantly, both the third and fourth parameters set to the current length of the type selection string. IdenticalText will return 0 if the strings match or 1 if they do not match.

doFindFirstSelectedCell

doFindFirstSelectedCell and the following four functions are general utility functions called by the previous Arrow key handling and type selection functions. doFindFirstSelectedCell searches for the first selected cell in a list, returning true if a selected cell is found and providing the cell's coordinates to the calling function.

SetPt sets the starting cell for the LGetSelect call. Since the first parameter in the LGetSelect call is set to true, LGetSelect will continue to search the list until a selected cell is found or until all cells have been examined.

doFindFirstSelectedCell returns true when and if a selected cell is found.

doFindLastSelectedCell

doFindLastSelectedCell finds the last selected cell in a list (which could, of course, also be the first selected cell if only one cell is selected).

If the call to doFindFirstSelectedCell reveals that no cells are currently selected, doFindlastSelectedCell simply returns. If, however, doFindFirstSelectedCell finds a selected cell, that cell is passed as the starting cell in the LGetSelect call.

As an example of how the rest of this function works, assume that the first selected cell is (0,1), and that cell (0,4) is the only other selected cell. LGetSelect examines this cell and returns true, causing the loop to execute. The first line in the while loop thus assigns (0,1) to theCell and the next line increments aCell to (0,2). LGetSelect starts another search using (0,2) as the starting cell. Because cells (0,2) and (0,3) are not selected, LGetSelect advances to cell (0,4) before it returns. Since it has found another selected cell, LGetSelect again returns true, so the loop executes again. aCell now contains (0,4), and the first line in the while loop assigns that to theCell. Once again, LNextCell increments aCell, this time to (0,5).

This time, however, LGetSelect will return false because neither cell (0,5) nor any cell below it is selected. The loop thus terminates, theCell containing (0,4), which is the last selected cell.

doFindNewCellLoc

doFindNewCellLoc finds the new cell to be selected in response to Arrow key presses. That cell will be either one up or one down from the cell specified in the oldCellLoc parameter (if the Command key was not down at the time of the Arrow key press) or the top or bottom cell (if the Command key was down).

The first line gets the number of rows in the list. (Recall that the List Manager sets the dataBounds.bottom coordinate to one more than the vertical coordinate of the last cell.)

If the Command key was down (moveToTopBottom is true) and the key pressed was the Up Arrow, the new cell to be selected is the top cell in the list. If the key pressed was the Down Arrow key, the new cell to be selected is the bottom cell in the list.

If the Command key was not down and the key pressed was the Up Arrow key, and if the first selected cell is the top cell in the list, the new cell to be selected remains as set at the second line in the function; otherwise, the new cell to be selected is set as the cell above the first selected cell. If the key pressed was the Down Arrow key, and if the last selected cell is the bottom cell in the list, the new cell to be selected remains as set at the second line in the function; otherwise, the new cell to be selected is set as the cell below the last selected cell.

doSelectOneCell

doSelectOneCell deselects all cells in the specified list and selects the specified cell.

If no cells in the list are selected, the function returns immediately. Otherwise, the first selected cell is passed as the starting cell in the call to LGetSelect.

The while loop will continue to execute while a selected cell exists between the starting cell specified in the LGetSelect call and the end of the list. Within the loop, if the current LGetSelect starting cell is not the cell specified for selection, that cell is deselected. When the loop exits, LSetSelect selects the cell specified for selection.

Note that defeating the de-selection of the cell specified for selection if it is already selected (the if statement within the while loop) prevents the unsightly flickering which would occur as a result of that cell being deselected inside the loop and then selected again after the loop exits.

doMakeCellVisible

doMakeCellVisible checks whether a specified cell is within the list's display rectangle and, if not, scrolls the list until that cell is visible.

The first line gets a copy of the rectangle that encompasses the currently visible cells. (Note that this rectangle is in cell coordinates.) The if statement tests whether the specified cell is within this rectangle. If it is not, the list is scrolled as follows:

- If the specified cell is "below" the bottom of the display rectangle, the variable dRows is set to the difference between the cell's v coordinate and the value in the bottom field of the display rectangle, plus 1. (Recall that the List Manager sets the bottom field to one greater than the v coordinate of the last visible cell.)

- If the specified cell is "above" the top of the display rectangle, the variable dRows is set to the difference between the cell's v coordinate and the value in the top field of the display rectangle.

With the number of cells to scroll, and the direction to scroll, established, LScroll is called to effect the scroll.

doResetTypeSelection

doResetTypeSelection resets the global variables which are central to the operation of the type selection function doTypeSelectSearch.

The first line resets the tsrKeyStrokes and tsrLastKeyTime fields of gTSStruct to NULL and 0 respectively. The next line sets the variable which holds the handle to the list which is the target of the current key press to NULL. The next line sets the variable which holds the type selection reset threshold to twice the value stored in the low memory global variable KeyThresh. However, if this value is greater than the value represented by the constant kMaxKeyThresh, the variable is made equal to kMaxKeyThresh.

doRotateCurrentList

doRotateCurrentList rotates the currently active list in response to the Tab key and to mouse-downs in the non-active list.

The first line saves the handle to the currently active list. The next line retrieves the handle to the new list to be activated from the refCon field of the currently active list's list structure. The third line makes the new list the currently active list.

The last two lines cause the keyboard focus frame to be erased from the previously current list, the list box frame to be drawn around the previously current list, and the keyboard focus frame to be drawn around the new current list.

doDrawFrameAndFocus

doDrawFrameAndFocus is called by doUpdate, doActivateWindow, and doRotateCurrentList to draw or erase the keyboard focus frame from the specified list, and to draw the list box frame in either the activated or deactivated state.

The second and third lines get the list's rectangle from the rView field of the list structure and expand it to the right by the width of the scroll bar.

The machinations at the next four lines are for cosmetic purposes only. If the program is running on Mac OS 8/9, the current background colour and pattern will be white, so the saved background colour/pattern must be restored before the first call to DrawThemeFocusRect, which erases the keyboard focus frame to the background colour/pattern.

Depending on the value received in the inState formal parameter, the list box frame is drawn in either the activated or deactivated state. If the specified list is the current list, DrawThemeFocusRect is called again, this time to draw the keyboard focus frame.

If the program is running on Mac OS 8/9, the last two lines reset the backgound colour/pattern to white.

doExtractSelections

doExtractSelections is called when the user clicks the Extract push button or double clicks an item in a list.

The first block gets the handles to the lists. The next two lines initialise the Str255 array that will be used to hold the extracted strings.

The next block copies the data from the selected cells in the text list to the Str255 array. The for loop is traversed once for each cell in the list. SetPt increments the v coordinate of the variable theCell. If the specified cell is selected (LGetSelect), LGetCellDataLocation is called to get the length of the data in the cell, and LGetCell is called to copy the cell's data into an element of the Str255 array.

The next block gets the selected cell in the icon list, retrieves the related string from the specified STR# resource, and assigns it to the 15th element of the Str255 array. SetPt sets the starting cell for the LGetSelect search.

The last two lines force an update event which will cause the function doDrawSelections to draw the contents of the Str255 array in the group box at the bottom of the window.

doDrawSelections

doDrawSelections is called by doUpdate and DoActivateWindow to draw the contents of the Str255 array "filled in" by the function doExtractSelections.

listDefFunction

listDefFunction the custom list definition function (LDEF) used by the window's icon list.

The List Manager sends a list definition function four types of messages in the message parameter. Only two of these are relevant to this list definition function. listDefFunction calls the appropriate function to handle each message type.

doLDEFDraw

doLDEFDraw handles the lDrawMsg message, which relates to a specific cell.

The first two lines save the current graphics port and set the graphics port to the port in which the list is drawn.

EraseRect erases the cell rectangle. The next line gets a copy of the 52 pixel by 52 pixel cell rectangle. The next four lines adjust this rectangle to the size of a 32 by 32 pixel icon.

The if statement checks whether the cell's data is 4 bytes long (the size of a handle). If it is, LGetCell is called to get the cells's data into the variable iconSuiteHdl and PlotIconSuite is called to draw the icon within the specified rectangle. If the list is active, kTransformNone is passed in the transform parameter, otherwise kTransformDisabled is passed. This latter causes the icon to be drawn in the disabled (dimmed) state.

GetIndString is then called to get the string corresponding to the icon. The rectangle used to draw the icon is adjusted and passed, together with the string, in a call to TETextBox. TETextBox draws the string, with centre justification, underneath the icon.

If the lDrawMsg message indicated that the cell was selected, the cell highlighting function is called. The previously saved graphics port is then restored

doLDEFHighlight

doLDEFHighlight handles the lHiliteMsg message and may also be called from doLDEFDraw.

A copy of the value in the low memory global HiliteMode is acquired, BitClr is called to clear the highlight bit, and HiliteMode is set to this new value. The last line highlights the cell.

DialogList.c

doListsDialog contains the main functions pertaining to the lists in the movable modal dialog box.

doListsDialog

doListsDialog creates a movable modal dialog box using 'DLOG', 'dlgx', 'dftb', and 'DITL' resources. The 'DITL' resource contains, amongst other items, two list controls. Each list control is supported by an 'ldes' resource. Both 'ldes' resources specify no rows, one column, a cell height of 14 pixels, a vertical scroll bar, and the system LDEF. The 'dftb' resource specifies the small system font for the list controls.

At the first block, the window, if open, is explicitly deactivated. The dialog is then created. At the next block, the Dialog Manager is told which items are the default and Cancel items.

A custom event filter function is used. The call to NewModalFilterProc creates the associated universal procedure pointer.

At the next block, and for each list control, the handle to the list control is obtained, the handle to the associated list structure is obtained, doAddRowsAndDataToTextList is called to add the specified number of rows and the data to the list's cells, the cell-selection algorithm is customised to allow the selection of one cell only, and the first cell is selected.

ShowDialog is then called to display the dialog. The first call to SetKeyboardFocus is made to force the "Watermark" list's scroll bar to be drawn. The second call is to set the keyboard focus to the "Date Format" list.

Within the do-while loop, ModalDialog retains control until an enabled item is hit. If the push buttons are hit, or if the last click in one of the list boxes was a double-click, the loop exits.

If the item hit is the "Date Format" list, SetPt sets the variable theCell to represent the first cell in the list. This is passed as a parameter in the LGetSelect call, which searches the list until it finds a cell that is selected. LGetDataLocation is called to get the length of the data in that cell and LGetCell is called to copy the data (a string) to a local Str255 variable.

At the next block, a reference to the static text control associated with this list is obtained and its text is set with the string obtained by LGetCell. Draw1Control is then called to draw the static text field control with this newly-set text.

The last action is to check whether the last click in the list box was a double-click. If the last click was a double-click, the variable wasDoubleClick is set to true, causing the loop to exit.

The same general procedure is followed in the event of a hit on the "Watermark" list.

When the OK or Cancel push button is hit, or one of the lists has been double-clicked, the dialog and the routine descriptor are disposed of.

eventFilter

A custom event filter function is used to intercept keyDown events so as to support type selection in the "Watermark" list.

If the event is a keyDown event, the character code is extracted from the event structure's message field.

If the key hit was not the Up-Arrow, Down-Arrow, or tab key, the following occurs. GetDialogItemAsControl is called to get the reference to the "Watermark" list control, GetControlData is called to get the handle to the associated list, and GetKeyboardFocus is called to get the reference to the control with keyboard focus. If the "Watermark" list control currently has the focus, doTypeSelectSearch is called (to handle type selection) and Draw1Control is called on the list control to ensure that the type-selected item is highlighted. handledEvent is then set to true to inform ModalDialog that the filter function handled the event.

Apart from supporting type-selection in the "Watermark" list, this arrangement means that the only keyDown events received by ModalDialog in respect of the "Date Format" list will be Up-Arrow, Down-Arrow, and tab key events.