Demonstration Program Miscellany Listing

```
// *******************************************************************************************
// Miscellany.h                                                           CLASSIC EVENT MODEL
// *******************************************************************************************
//
// This program demonstrates:
//
// •  The use of the Notification Manager to allow an application running in the background to
//    to communicate with the foreground application.
//
// •  The use of the determinate progress indicator control to show progress during a time-
//    consuming operation, together with scanning the event queue for Command-period key-down
//    events for the purpose of terminating the lengthy operation  before it concludes of its
//    own accord.
//
// •  The use of the Color Picker to solicit a choice of colour from the user.
//
// •  Image drawing optimisation in a multi-monitors environment.
//
// •  Help tags for controls and windows with minimum and maximum content.
//
// 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 visible) for a window in which graphics and
//    information relevant to the demonstrations is displayed.
//
// •  A 'DLOG' resource (purgeable), and associated 'DITL', 'dlgx', and 'dftb' resources
//    (purgeable), for a dialog box in which the progress indicator is displayed.
//
// •  'CNTL' resources (purgeable) for the progress indicator dialog.
//
// •  'icn#', 'ics4', and 'ics8' resources (non-purgeable) which contain the application icon
//    shown in the Application menu during the Notification Manager demonstration.
//
// •  A 'snd ' resource (non-purgeable) used in the Notification Manager demonstration.
//
// •  A 'STR ' resource (non-purgeable) containing the text displayed in the alert box invoked
//    by the Notification Manager.
//
// •  'STR#' resources (purgeable) containing the label and narrative strings for the
//    notification-related alert displayed by Miscellany and the minimum and maximum Help tag
//    content.
//
// •  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  iNotification      1
#define  iProgress          2
```

```
#define  iColourPicker      3
#define  iMultiMonitors     4
#define  iHelpTag           5
#define rWindow             128
#define rDialog             128
#define  iProgressIndicator 1
#define rIconFamily         128
#define rBarkSound          8192
#define rString             128
#define rAlertStrings       128
#define  indexLabel         1
#define  indexNarrative     2
#define rPicture            128
#define topLeft(r)          (((Point *) &(r))[0])
#define botRight(r)         (((Point *) &(r))[1])

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

void  main                         (void);
void  doPreliminaries              (void);
OSErr quitAppEventHandler          (AppleEvent *,AppleEvent *,SInt32);
void  doEvents                     (EventRecord *);
void  doMenuChoice                 (SInt32 menuChoice);

void  doSetUpNotification          (void);
void  doPrepareNotificationStructure (void);
void  doIdle                       (void);
void  doOSEvent                    (EventRecord *);
void  doDisplayMessageToUser       (void);

void  doProgressIndicator          (void);

void  deviceLoopDraw               (SInt16,SInt16,GDHandle,SInt32);

void  doColourPicker               (void);
void  doDrawColourPickerChoice     (void);
char  *doDecimalToHexadecimal      (UInt16 n);

void  doHelpTagControl             (void);
void  doHelpTagWindow              (void);

// *******************************************************************************************
// Miscellany.c
// *******************************************************************************************

#include "Miscellany.h"

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

DeviceLoopDrawingUPP gDeviceLoopDrawUPP;
Boolean              gDone;
WindowRef            gWindowRef;
ControlRef           gBevelButtonControlRef;
ProcessSerialNumber  gProcessSerNum;
Boolean              gMultiMonitorsDrawDemo = false;
Boolean              gColourPickerDemo  = false;
Boolean              gHelpTagsDemo = false;
RGBColor             gWhiteColour = { 0xFFFF, 0xFFFF, 0xFFFF };
RGBColor             gBlueColour  = { 0x6666, 0x6666, 0x9999 };
Rect                 controlRect = { 65,10,155,100 };

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

void  main(void)
{
  MenuBarHandle menubarHdl;
  SInt32        response;
  MenuRef       menuRef;
  EventRecord   eventStructure;
  // ........................................................................ do preliminaries

  doPreliminaries();

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

  gDeviceLoopDrawUPP = NewDeviceLoopDrawingUPP((DeviceLoopDrawingProcPtr) deviceLoopDraw);

  // ................................................................... 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);
    }
  }

  // ............................................................................... open window

  if(!(gWindowRef = GetNewCWindow(rWindow,NULL,(WindowRef)-1)))
    ExitToShell();

  SetPortWindowPort(gWindowRef);
  TextSize(10);

  // ......................................................................... create help tags

  CreateBevelButtonControl(gWindowRef,&controlRect,CFSTR("Control"),
                           kControlBevelButtonNormalBevel,kControlBehaviorPushbutton,
                           NULL,0,0,0,&gBevelButtonControlRef);
  HideControl(gBevelButtonControlRef);

  doHelpTagControl();
  doHelpTagWindow();
  HMSetHelpTagsDisplayed(false);

  // ...................................................... get process serial number of this process

  GetCurrentProcess(&gProcessSerNum);

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

  gDone = false;

  while(!gDone)
  {
    if(WaitNextEvent(everyEvent,&eventStructure,1,NULL))
      doEvents(&eventStructure);
    else
    {
      if(eventStructure.what == nullEvent)
        doIdle();
    }
  }
}

// ************************************************************************** doPreliminaries
```

```
void  doPreliminaries(void)
{
  OSErr osError;

  MoreMasterPointers(64);
  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)
{
  WindowPartCode partCode, zoomPart;
  WindowRef      windowRef;
  SInt32         userData;
  Rect           constraintRect, mainScreenRect, portRect;
  Point          standardStateHeightAndWidth;
  RgnHandle      regionHdl;

  switch(eventStrucPtr->what)
  {
    case kHighLevelEvent:
      AEProcessAppleEvent(eventStrucPtr);
      break;

    case mouseDown:
      partCode = FindWindow(eventStrucPtr->where,&windowRef);

      switch(partCode)
      {
        case inMenuBar:
          doMenuChoice(MenuSelect(eventStrucPtr->where));
          break;

        case inContent:
          if(windowRef != FrontWindow())
            SelectWindow(windowRef);
          break;
        case inDrag:
          DragWindow(windowRef,eventStrucPtr->where,NULL);
          doHelpTagWindow();
          break;

        case inGrow:
          constraintRect.top  = 302;
          constraintRect.left = 445;
          constraintRect.bottom = constraintRect.right = 32767;
          ResizeWindow(windowRef,eventStrucPtr->where,&constraintRect,NULL);

          GetWindowPortBounds(windowRef,&portRect);
          InvalWindowRect(windowRef,&portRect);
          doHelpTagWindow();
          break;

        case inZoomIn:
        case inZoomOut:
          GetAvailableWindowPositioningBounds(GetMainDevice(),&mainScreenRect);
          standardStateHeightAndWidth.v = mainScreenRect.bottom;
          standardStateHeightAndWidth.h = mainScreenRect.right;

          if(IsWindowInStandardState(windowRef,&standardStateHeightAndWidth,NULL))
            zoomPart = inZoomIn;
          else
            zoomPart = inZoomOut;

          if(TrackBox(windowRef,eventStrucPtr->where,zoomPart))
          {
            ZoomWindowIdeal(windowRef,zoomPart,&standardStateHeightAndWidth);
            doHelpTagWindow();
          }
          break;
      }
      break;

    case keyDown:
      if((eventStrucPtr->modifiers & cmdKey) != 0)
        doMenuChoice(MenuEvent(eventStrucPtr));
      break;

    case updateEvt:
      windowRef = (WindowRef) eventStrucPtr->message;

      BeginUpdate(windowRef);

      if(gMultiMonitorsDrawDemo)
      {
        RGBBackColor(&gWhiteColour);
        userData = (SInt32) windowRef;
        regionHdl = NewRgn();
        if(regionHdl)
        {
          GetPortVisibleRegion(GetWindowPort(windowRef),regionHdl);
          DeviceLoop(regionHdl,gDeviceLoopDrawUPP,userData,0);
          DisposeRgn(regionHdl);
        }
      }
      else if(gColourPickerDemo )
      {
        RGBBackColor(&gBlueColour);
        GetWindowPortBounds(windowRef,&portRect);
        EraseRect(&portRect);
        doDrawColourPickerChoice();
      }
      else
      {
        RGBBackColor(&gBlueColour);
        GetWindowPortBounds(windowRef,&portRect);
        EraseRect(&portRect);
        if(gHelpTagsDemo)
        {
          Draw1Control(gBevelButtonControlRef);
          RGBForeColor(&gWhiteColour);
          MoveTo(10,20);
          DrawString("\pHover the cursor in the window, and over the bevel button, ");
          DrawString("\puntil the Help tag appears.");
          MoveTo(10,35);
          DrawString("\pPress the Command key to invoke the maximum content.");
          MoveTo(10,50);
          DrawString("\pDrag the window to a new location.");
        }
      }

      EndUpdate(windowRef);
      break;

    case osEvt:
      doOSEvent(eventStrucPtr);
      break;
  }
}

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

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

  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)
        ExitToShell();
      break;

    case mDemonstration:
      gMultiMonitorsDrawDemo = gColourPickerDemo = gHelpTagsDemo = false;
      if(HMAreHelpTagsDisplayed)
        HMSetHelpTagsDisplayed(false);
      HideControl(gBevelButtonControlRef);
      GetWindowPortBounds(gWindowRef,&portRect);

      switch(menuItem)
      {
        HideControl(gBevelButtonControlRef);

        case iNotification:
          RGBBackColor(&gBlueColour);
          EraseRect(&portRect);
          doSetUpNotification();
          break;

        case iProgress:
          RGBBackColor(&gBlueColour);
          EraseRect(&portRect);
          doProgressIndicator();
          break;

        case iColourPicker:
          gColourPickerDemo = true;
          doColourPicker();
          break;

        case iMultiMonitors:
          gMultiMonitorsDrawDemo = true;
          InvalWindowRect(gWindowRef,&portRect);
          break;

        case iHelpTag:
          gHelpTagsDemo = true;
          InvalWindowRect(gWindowRef,&portRect);
          ShowControl(gBevelButtonControlRef);
          HMSetHelpTagsDisplayed(true);
          break;
      }

      break;
  }

  HiliteMenu(0);
}

// *******************************************************************************************
// Notification.c
// *******************************************************************************************

#include "Miscellany.h"

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

NMRec                      gNotificationStructure;
long                       gStartingTickCount;
Boolean                    gNotificationDemoInvoked;
Boolean                    gNotificationInQueue;
extern WindowRef           gWindowRef;
extern ProcessSerialNumber gProcessSerNum;
extern RGBColor            gWhiteColour;
extern RGBColor            gBlueColour;

// ******************************************************************** doSetUpNotification

void  doSetUpNotification(void)
{
  doPrepareNotificationStructure();
  gNotificationDemoInvoked = true;

  gStartingTickCount = TickCount();

  RGBForeColor(&gWhiteColour);
  MoveTo(10,279);
  DrawString("\pPlease click on the desktop now to make the Finder ");
  DrawString("\pthe frontmost application.");
  MoveTo(10,292);
  DrawString("\p(This application will post a notification 10 seconds from now.)");
}

// *********************************************************** doPrepareNotificationStructure

void  doPrepareNotificationStructure(void)
{
  Handle      iconSuiteHdl;
  Handle      soundHdl;
  StringHandle stringHdl;
  GetIconSuite(&iconSuiteHdl,rIconFamily,kSelectorAllSmallData);
  soundHdl = GetResource('snd ',rBarkSound);
  stringHdl = GetString(rString);

  gNotificationStructure.qType    = nmType;
  gNotificationStructure.nmMark   = 1;
  gNotificationStructure.nmIcon   = iconSuiteHdl;
  gNotificationStructure.nmSound  = soundHdl;
  gNotificationStructure.nmStr    = *stringHdl;
  gNotificationStructure.nmResp   = NULL;
  gNotificationStructure.nmRefCon = 0;
}

// ************************************************************************************ doIdle

void  doIdle(void)
{
  ProcessSerialNumber frontProcessSerNum;
  Boolean             isSameProcess;
  Rect                portRect;

  if(gNotificationDemoInvoked)
  {
    if(TickCount() > gStartingTickCount + 600)
    {
      GetFrontProcess(&frontProcessSerNum);
      SameProcess(&frontProcessSerNum,&gProcessSerNum,&isSameProcess);

      if(!isSameProcess)
      {
        NMInstall(&gNotificationStructure);
        gNotificationDemoInvoked = false;
        gNotificationInQueue = true;
      }
      else
      {
        doDisplayMessageToUser();
        gNotificationDemoInvoked = false;
      }

      GetWindowPortBounds(gWindowRef,&portRect);
      EraseRect(&portRect);
    }
  }
}

// ********************************************************************************* doOSEvent

void  doOSEvent(EventRecord *eventStrucPtr)
{
  switch((eventStrucPtr->message >> 24) & 0x000000FF)
  {
    case suspendResumeMessage:
      if((eventStrucPtr->message & resumeFlag) == 1)
      {
        if(gNotificationInQueue)
          doDisplayMessageToUser();
      }
      break;
  }
}

// ******************************************************************** doDisplayMessageToUser

void  doDisplayMessageToUser(void)
{
  Rect                  portRect;
  AlertStdAlertParamRec paramRec;
  Str255                labelText;
  Str255                narrativeText;
  SInt16                itemHit;

  if(gNotificationInQueue)
  {
    NMRemove(&gNotificationStructure);
    gNotificationInQueue = false;
  }

  GetWindowPortBounds(gWindowRef,&portRect);
  EraseRect(&portRect);

  paramRec.movable       = true;
  paramRec.helpButton    = false;
  paramRec.filterProc    = NULL;
  paramRec.defaultText   = (StringPtr) kAlertDefaultOKText;
  paramRec.cancelText    = NULL;
  paramRec.otherText     = NULL;
  paramRec.defaultButton = kAlertStdAlertOKButton;
  paramRec.cancelButton  = 0;
  paramRec.position      = kWindowDefaultPosition;

  GetIndString(labelText,rAlertStrings,indexLabel);
  GetIndString(narrativeText,rAlertStrings,indexNarrative);

  StandardAlert(kAlertNoteAlert,labelText,narrativeText,&paramRec,&itemHit);

  DisposeIconSuite(gNotificationStructure.nmIcon,false);
  ReleaseResource(gNotificationStructure.nmSound);
  ReleaseResource((Handle) gNotificationStructure.nmStr);
}

// *******************************************************************************************
// ProgressIndicator.c
// *******************************************************************************************

#include "Miscellany.h"

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

extern WindowRef gWindowRef;
extern RGBColor  gWhiteColour;

// ************************************************************************** doProgressBar

void  doProgressIndicator(void)
{
  DialogRef  dialogRef;
  RgnHandle  visRegionHdl = NewRgn();
  ControlRef progressBarRef;
  SInt16     statusMax, statusCurrent;
  SInt16     a, b, c;
  Handle     soundHdl;
  Rect       portRect, theRect;
  RGBColor   redColour = { 0xFFFF, 0x0000, 0x0000 };

  if(!(dialogRef = GetNewDialog(rDialog,NULL,(WindowRef) -1)))
    ExitToShell();

  SetPortDialogPort(dialogRef);
  GetPortVisibleRegion(GetWindowPort(GetDialogWindow(dialogRef)),visRegionHdl);
  UpdateControls(GetDialogWindow(dialogRef),visRegionHdl);
  QDFlushPortBuffer(GetDialogPort(dialogRef),NULL);

  SetPortWindowPort(gWindowRef);
  GetWindowPortBounds(gWindowRef,&portRect);

  GetDialogItemAsControl(dialogRef,iProgressIndicator,&progressBarRef);
  statusMax = 3456;
  statusCurrent = 0;
  SetControlMaximum(progressBarRef,statusMax);

  for(a=0;a<9;a++)
  {
    for(b=8;b<423;b+=18)
    {
      for(c=8;c<286;c+=18)
      {
        if(CheckEventQueueForUserCancel())
        {
          soundHdl = GetResource('snd ',rBarkSound);
          SndPlay(NULL,(SndListHandle) soundHdl,false);
          ReleaseResource(soundHdl);
          DisposeDialog(dialogRef);

          EraseRect(&portRect);
          MoveTo(10,292);
          RGBForeColor(&gWhiteColour);
          DrawString("\pOperation cancelled at user request");

          return;
        }

        SetRect(&theRect,b+a,c+a,b+17-a,c+17-a);
        if(a < 3)                 RGBForeColor(&gWhiteColour);
        else if(a > 2 && a < 6)  RGBForeColor(&redColour);
        else if(a > 5)           RGBForeColor(&gWhiteColour);
        FrameRect(&theRect);

        QDFlushPortBuffer(GetWindowPort(gWindowRef),NULL);
        QDFlushPortBuffer(GetDialogPort(dialogRef),NULL);

        SetControlValue(progressBarRef,statusCurrent++);
      }
    }
  }

  DisposeRgn(visRegionHdl);
  DisposeDialog(dialogRef);
  EraseRect(&portRect);
  MoveTo(10,292);
  RGBForeColor(&gWhiteColour);
  DrawString("\pOperation completed");
}

// *******************************************************************************************
// ColourPicker.c
// *******************************************************************************************

#include "Miscellany.h"

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

RGBColor         gInColour = { 0xCCCC, 0x0000, 0x0000 };
RGBColor         gOutColour;
Boolean          gColorPickerButton;
extern WindowRef gWindowRef;
extern RGBColor  gWhiteColour;
extern RGBColor  gBlueColour;

// ************************************************************************** doColourPicker

void  doColourPicker(void)
{
  Rect   portRect, theRect;
  Point  where;
  Str255 prompt = "\pChoose a rectangle colour:";

  GetWindowPortBounds(gWindowRef,&portRect);
  theRect = portRect;

  RGBBackColor(&gBlueColour);
  EraseRect(&theRect);
  InsetRect(&theRect,55,55);
  RGBForeColor(&gInColour);
  PaintRect(&theRect);

  where.v = where.h = 0;

  gColorPickerButton = GetColor(where,prompt,&gInColour,&gOutColour);

  InvalWindowRect(gWindowRef,&portRect);
}

// ******************************************************************* doDrawColorPickerChoice

void  doDrawColourPickerChoice(void)
{
  Rect portRect;
  char *cString;

  GetWindowPortBounds(gWindowRef,&portRect);
  InsetRect(&portRect,55,55);

  if(gColorPickerButton)
  {
    RGBForeColor(&gOutColour);
    PaintRect(&portRect);

    RGBForeColor(&gWhiteColour);

    MoveTo(55,22);
    DrawString("\pRequested Red Value: ");
    cString = doDecimalToHexadecimal(gOutColour.red);
    MoveTo(170,22);
    DrawText(cString,0,6);

    MoveTo(55,35);
    DrawString("\pRequested Green Value: ");
    cString = doDecimalToHexadecimal(gOutColour.green);
    MoveTo(170,35);
    DrawText(cString,0,6);

    MoveTo(55,48);
    DrawString("\pRequested Blue Value: ");
    cString = doDecimalToHexadecimal(gOutColour.blue);
    MoveTo(170,48);
    DrawText(cString,0,6);
  }
  else
  {
    RGBForeColor(&gInColour);
    PaintRect(&portRect);

    RGBForeColor(&gWhiteColour);
    MoveTo(55,48);
    DrawString("\pCancel button was clicked.");
  }
}

// ******************************************************************** doDecimalToHexadecimal

char  *doDecimalToHexadecimal(UInt16 decimalNumber)
{
  static char cString[] = "0xXXXX";
  char        *hexCharas = "0123456789ABCDEF";
  SInt16      a;

  for (a=0;a<4;decimalNumber >>= 4,++a)
    cString[5 - a] = hexCharas[decimalNumber & 0xF];

  return cString;
}

// *******************************************************************************************
// MultiMonitor.c
// *******************************************************************************************

#include "Miscellany.h"

// ************************************************************************** deviceLoopDraw

void  deviceLoopDraw(SInt16 depth,SInt16 deviceFlags,GDHandle targetDeviceHdl,SInt32 userData)
{
  RGBColor  oldForeColour;
  WindowRef windowRef;
  Rect      portRect;
  RGBColor  greenColour  = { 0x0000, 0xAAAA, 0x1111 };
  RGBColor  redColour    = { 0xAAAA, 0x4444, 0x3333 };
  RGBColor  blueColour   = { 0x5555, 0x4444, 0xFFFF };
  RGBColor  ltGrayColour = { 0xDDDD, 0xDDDD, 0xDDDD };
  RGBColor  grayColour   = { 0x9999, 0x9999, 0x9999 };
  RGBColor  dkGrayColour = { 0x4444, 0x4444, 0x4444 };

  GetForeColor(&oldForeColour);

  windowRef = (WindowRef) userData;
  GetWindowPortBounds(windowRef,&portRect);
  EraseRect(&portRect);

  if(((1 << gdDevType) & deviceFlags) != 0)
  {
    InsetRect(&portRect,50,50);
    RGBForeColor(&greenColour);
    PaintRect(&portRect);
    InsetRect(&portRect,40,40);
    RGBForeColor(&redColour);
    PaintRect(&portRect);
    InsetRect(&portRect,40,40);
    RGBForeColor(&blueColour);
    PaintRect(&portRect);
  }
  else
  {
    InsetRect(&portRect,50,50);
    RGBForeColor(&ltGrayColour);
    PaintRect(&portRect);
    InsetRect(&portRect,40,40);
    RGBForeColor(&grayColour);
    PaintRect(&portRect);
    InsetRect(&portRect,40,40);
    RGBForeColor(&dkGrayColour);
    PaintRect(&portRect);
  }

  RGBForeColor(&oldForeColour);
}

// *******************************************************************************************
// HelpTag.c
// *******************************************************************************************

#include "Miscellany.h"
#include <string.h>
```

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

extern ControlRef gBevelButtonControlRef;
extern WindowRef  gWindowRef;

// ............................................................................................................. doHelpTagControl

void  doHelpTagControl(void)
{
  HMHelpContentRec helpContent;

  memset(&helpContent,0,sizeof(helpContent));

  HMSetTagDelay(50);
  helpContent.version = kMacHelpVersion;
  helpContent.tagSide = kHMOutsideBottomLeftAligned;

  helpContent.content[kHMMinimumContentIndex].contentType = kHMStringResContent;
  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmResID = 129;
  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 1;
  helpContent.content[kHMMaximumContentIndex].contentType = kHMStringResContent;
  helpContent.content[kHMMaximumContentIndex].u.tagStringRes.hmmResID = 129;
  helpContent.content[kHMMaximumContentIndex].u.tagStringRes.hmmIndex = 2;

  HMSetControlHelpContent(gBevelButtonControlRef,&helpContent);
}

// ............................................................................................................. doHelpTagWindow

void  doHelpTagWindow(void)
{
  Rect             hotRect;
  HMHelpContentRec helpContent;

  memset(&helpContent,0,sizeof(helpContent));

  HMSetTagDelay(500);
  helpContent.version = kMacHelpVersion;
  helpContent.tagSide = kHMOutsideRightCenterAligned;

  GetWindowPortBounds(gWindowRef,&hotRect);
  LocalToGlobal(&topLeft(hotRect));
  LocalToGlobal(&botRight(hotRect));
  helpContent.absHotRect = hotRect;

  helpContent.content[kHMMinimumContentIndex].contentType = kHMStringResContent;
  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmResID = 129;
  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 3;
  helpContent.content[kHMMaximumContentIndex].contentType = kHMStringResContent;
  helpContent.content[kHMMaximumContentIndex].u.tagStringRes.hmmResID = 129;
  helpContent.content[kHMMaximumContentIndex].u.tagStringRes.hmmIndex = 4;

  HMSetWindowHelpContent(gWindowRef,&helpContent);
}

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

Demonstration Program Miscellany Comments

When this program is run, the user should make choices from the Demonstration menu, taking the following actions and making the following observations:

- Choose the Notification item and, observing the instructions in the window, click the desktop immediately to make the Finder the foreground application. A notification will be posted by Miscellany about 10 seconds after the Notification item choice is made. Note that, when about 10 seconds have elapsed, the Notification Manager invokes an alert (Mac OS 8.6), floating window (Mac OS 9.x), or system movable modal alert (Mac OS X) and alternates the Finder and Miscellany icons in the OS 8/9 Application menu title. Observing the instructions in the alert/floating window/system movable modal alert:
  - Dismiss the alert (Mac OS 8.6 only).
  - On Mac OS 8/9, then choose the Miscellany item in the OS 8/9 Application menu, noting the ♦ mark to the left of the item name. When Miscellany comes to the foreground, note that the icon alternation concludes and that an alert (invoked by Miscellany) appears. Dismiss this second alert.
  - On Mac OS X, click on the application's icon in the Dock.
- Choose the Notification item again and, this time, leave Miscellany in the foreground. Note that only the alert invoked by Miscellany appears on this occasion.
- Choose the Notification item again and, this time, click on the desktop and then in the Miscellany window before 10 seconds elapse. Note again that only the alert invoked by Miscellany appears.
- Choose the Determinate Progress Indicator item, noting that the progress indicator dialog is automatically disposed of when the (simulated) time-consuming task concludes.
- Choose the Determinate Progress Indicator item again, and this time press the Command-period key combination before the (simulated) time-consuming task concludes. Note that the progress indicator dialog is disposed of when the Command-period key combination is pressed.
- Choose the Colour Picker item and make colour choices using the various available modes. Note that, when the Colour Picker is dismissed by clicking the OK button, the requested RGB colour values for the chosen colour are displayed in hexadecimal, together with a rectangle in that colour, in the Miscellany window.
- Choose the Multiple Monitors Draw item, noting that the drawing of the simple demonstration image is optimised as follows:
  - On a monitor set to display 256 or more colours, the image is drawn in three distinct colours. The luminance of the three colours is identical, meaning that, if these colours are drawn on a grayscale screen, they will all appear in the same shade of gray.
  - On a monitor set to display 256 shades of gray, the image is drawn in three distinct shades of gray.
- Choose the Help Tags item, hover the cursor over the window and, when the Help tag appears, press the Command key to observe the maximum content version of the tag. Repeat this while hovering the cursor over the bevel button control.

Miscellany.c

Global Variables

gDeviceLoopDrawUPP will be assigned a universal procedure pointer to the application-defined image-optimising drawing function called by DeviceLoop. gProcessSerNum will be assigned the process serial number of the Miscellany application.

main

The call to NewDeviceLoopDrawingProc creates a universal procedure pointer for the image-optimising drawing function deviceLoopDraw.

A bevel button control is created, following which the calls to doHelpTagControl and doHelpTagWindow create Help tags for the bevel button control and the window. HMSetHelpTagsDisplayed is called to disable the tags until the Help Tags item is chosen from the Demonstration menu.

GetCurrentProcess gets the process serial number of this process.

Within the event loop, note that the call to doIdle is relevant to the notification demonstration only.

doEvents

Within the updateEvt case, if the Multiple Monitors Draw item in the Demonstration menu has been chosen (gMultiMonitorsDrawDemo is true), a call is made to DeviceLoop and the universal procedure pointer to the application-defined drawing function deviceLoopDraw is passed as the second parameter.

doMenuChoice

When the Multiple Monitors Draw item in the Demonstration menu is chosen, the window's port rectangle is invalidated so as to force an update event and consequential call to DeviceLoop.

Notification.c

doSetUpNotification

doSetUpNotification is called when the user chooses Notification from the Demonstration menu.

The first line calls a function which fills in the relevant fields of a notification structure. The next line assigns true to a global variable which records that the Notification item has been chosen by the user.

The next line saves the system tick count at the time that the user chose the Notification item. This value is used later to determine when 10 seconds have elapsed following the execution of this line.

doPrepareNotificationStructure

doPrepareNotificationStructure fills in the relevant fields of the notification structure.

First, however, GetIconSuite creates an icon family based on the specified resource ID and the third parameter, which limits the family to 'ics#', 'ics4' and 'ics8' icons. The GetIconSuite call returns the handle to the suite in its first parameter. The call to GetResource loads the specified 'snd ' resource. GetString loads the specified 'STR ' resource.

The first line of the main block specifies the type of operating system queue. The next line specifies that the ♦ mark is to appear next to the application's name in the Application menu. The next three lines assign the icon suite, sound and string handles previously obtained. The next line specifies that no response function is required to be executed when the notification is posted.

doIdle

doIdle is called from the main event loop when a null event is received. Recall that the canBackground flag is set in the application's 'SIZE' resource, meaning that the application will receive null events when it is in the background.

If the user has not just chosen the Notification item in the Demonstration menu (gNotificationDemoInvoked is false), doIdle simply returns immediately.

If, however, that item has just been chosen, and if 10 seconds (600 ticks) have elapsed since that choice was made, the following occurs:

- The calls to GetFrontProcess and SameProcess determine whether the current foreground process is Miscellany. If it is not, the notification request is installed in the notification queue by NMInstall and a global variable is set to indicate that a request has been placed in the queue by Miscellany. Also, gNotificationDemoInvoked is set to false so as to ensure that the main if block only executes once after the Notification item is chosen.

- If, however, the current foreground process is Miscellany, doDisplayMessageToUser is called to present the required message to the user, via an alert box, in the normal way. Once again gNotificationDemoInvoked is reset to false so as to ensure that the main if block only executes once after the Notification item is chosen.

doOSEvent

doOSEvent handles operating system events.

If the event is a resume event (that is, Miscellany is coming to the foreground) and if the notification request is still in the notification queue (gNotificationInQueue is true), the function doDisplayMessageToUser is called to remove the notification request from the queue and have Miscellany convey the required message to the user via an alert box.

doDisplayMessageToUser

doDisplayMessageToUser is called by doOSEvent and doIdle in the circumstances previously described.

If a Miscellany notification request is in the queue, NMRemove removes it from the queue and gNotificationInQueue is set to false to reflect this condition. (Recall that, if the nmResp field of the notification structure is not assigned -1, the application itself must remove the queue element from the queue.)

Regardless of whether there was a notification in the queue or not, Miscellany then presents its alert. When the alert is dismissed, the notification's icon suite, sound and string resources are released/disposed of.

ProgressIndicator.c

doProgressIndicator

doProgressIndicator is called when the user chooses Determinate Progress Indicator from the Demonstration menu.

GetNewDialog creates a modal dialog box. The call to UpdateControls draws the dialog box's controls. The call to GetDialogItemAsControl a reference to the dialog's progress indicator control. SetControlMaximum sets the control's maximum value to equate to the number of steps in a simulated time-consuming task.

The main for loop performs the simulated time-consuming task, represented to the user by the drawing of a large number of coloured rectangles in the window. The task involves 3456 calls to FrameRect.

Within the inner for loop, CheckEventQueueForCancel is called to check whether the user has pressed the Command-period key. If so, a 'snd ' resource is loaded, played, and released, the dialog is disposed of, an advisory message in drawn in the window, and the function returns.

Within the inner for loop, the rectangles are drawn. Each time round this inner loop, a progress indicator control's value is incremented.

When the outer loop exits (that is, when the Command-period key combination is not pressed before the simulated time-consuming task completes), the dialog is disposed of.

ColourPicker.c

doColourPicker

doColourPicker is called when the user chooses Colour Picker from the Demonstration menu.

The first block erases the window's content area and paints a rectangle in the colour which will be passed in GetColor's inColor parameter.

The next line assigns 0 to the fields of the Point variable to be passed in GetColor's where parameter. ((0,0) will cause the Colour Picker dialog box to be centred on the main screen.)

The call to GetColor displays the Colour Picker's dialog box. GetColor retains control until the user clicks either the OK button or the Cancel button, at which time the port rectangle is invalidated, causing the function doDrawColourPickerChoice to be called.

doDrawColourPickerChoice

If the user clicked the OK button, a filled rectangle is painted in the window in the colour returned in GetColor's outColor parameter, and the values representing the red, green, and blue components of this colour are displayed at the top of the window in hexadecimal. Note that the function doDecimalToHexadecimal is called to convert the decimal (UInt32) values in the fields of the RGBColor variable outColor to hexadecimal.

If the user clicks the Cancel button, a filled rectangle is painted in the window in the colour passed in GetColor's inColor parameter.

doDecimalToHexadecimal

doDecimalToHexadecimal converts a UInt16 value to a hexadecimal string.

MultiMonitor.c

deviceLoopDraw

deviceLoopDraw is the image-optimising drawing function the universal procedure pointer to which is passed in the second parameter in the DeviceLoop call in the function doEvents. (Recall that the DeviceLoop call is made whenever the Multiple Monitors Draw item in the Demonstration menu has been selected and an update event is received.) DeviceLoop scans all active video devices, calling deviceLoopDraw whenever it encounters a device which intersects the drawing region, and passing certain information to deviceLoopDraw.

The second line casts the SInt32 value received in the userData parameter to a WindowRef. The window's content area is then erased.

If an examination of the device's attributes, as received in the deviceFlags formal parameter, reveals that the device is a colour device, three rectangles are painted in the window in three different colours. (The luminance value of these colours is the same, meaning that the rectangles would all be the same shade of gray if they were drawn on a monochrome (grayscale) device.)

If the examination of the device's attributes reveals that the device is a monochrome device, the rectangles are painted in three distinct shades of gray.

HelpTag.c

doHelpTagControl and doHelpTagWindow

doHelpTagControl and doHelpTagWindow create Help tags for the bevel button control and the window.

The call to memset clears the specified block of memory. The call to HMSetTagDelay sets the delay, in milliseconds, before the tag opens.

For the bevel button, the tagSide field of the HMHelpContentRec structure is assigned a value which will cause the control's tag to be displayed below the control with its left side aligned with the left side of the button. For the window, the tagSide field is assigned a value which will cause the control's tag to be displayed on the window's right, centered vertically.

The main block sets the content type and retrieves and assigns the minimum and maximum content strings from a 'STR#' resource. The calls to HMSetControlHelpContent and HMSetWindowHelpContent install the Help tags on the control and window.