Section 3 - Event Handling

Purpose

In this section you will learn about some of the MLTE API that greatly simplify the event processing in your application.

Objective

Upon completion of this session, you will be able to do the following:

- Use the following eleven MLTE event related API:

  TXNGetSleepTicks, TXNTSMCheck, TXNAdjustCursor, TXNIdle, TXNClick, TXNGrowWindow, TXNZoomWindow, TXNKeyDown, TXNUpdate, TXNActivate and TXNFocus

Sleep time for your event loop

In a cooperative processing environment, one of the decisions an application must make is how long to give to background processes.

MLTE provides a function to determine an appropriate sleep time to pass to WaitNextEvent.

```
EXTERN_API(UInt32)TXNGetSleepTicks(TXNObject iTXNObject);
```

The return value of **TXNGetSleepTicks** holds the appropriate sleep time.

Our sample's WaitNextEvent calls our function GetSleep(). At the Step 6 tag, call TXNGetSleepTicks in that function and return the result to WaitNextEvent. Your code might look something like this:

```
TXNObject   object = NULL;

window = FrontWindow();
if (IsAppWindow(window))      // if the front window is ours
{
   GetWindowProperty(window, 'GRIT', 'tObj', sizeof(TXNObject), NULL, &object);
   sleep = TXNGetSleepTicks(object); // get appropriate sleep time
}
else
   sleep = GetCaretTime();    // use some other value
return sleep;
```

Time for Input Methods

You will want your application to also give time to any input methods that the user may have activated, to ensure they can be reasonably responsive. MLTE provides a method, **TXNTSMCheck**, that you can call when WaitNextEvent returns false or there is no active TXNObject.

```
EXTERN_API(Boolean)TXNTSMCheck(TXNObject iTXNObject, EventRecord *iEvent);
```

The TXNObject parameter can be NULL, allowing an application to call this function at any time. (You no longer need to call this function if you are a Carbon application.)

Our sample's WaitNextEvent loop has a call to DoIdle if the event is a NULL event. Place a call to TXNTSMCheck so that it is called along with DoIdle. This is Step 7.

Notes

Cursor adjustment

Another task an application must take on is to determine the correct cursor to display depending on the mouse location. MLTE provides **TXNAdjustCursor**, which sets the cursor to the i-beam if the mouse is over a text area, and sets the cursor to an arrow if the mouse is over graphics, a sound, a movie, a scroll bar, or outside of the window.

```
EXTERN_API(void)TXNAdjustCursor(TXNObject iTXNObject, RgnHandle ioCursorRgn);
```

Our sample calls its own function AdjustCursor when appropriate. At Step 8, implement an appropriate call to TXNAdjustCursor, e.g.:

```
TXNObject   object = NULL;

if (IsAppWindow(window))
{
   GetWindowProperty(window, 'GRIT', 'tObj', sizeof(TXNObject), NULL, &object);
   TXNAdjustCursor(object, region);
}
```

Idle time

MLTE also helps with any necessary Idle time processing with the **TXNIdle** function. One thing TXNIdle will also so is, if a Text Service Manager document is active, it will pass a NULL event to the Text Service Manager.

```
EXTERN_API(void)TXNIdle(TXNObject iTXNObject);
```

Our sample calls its own function DoIdle when appropriate. As you have done in Steps 7 and 8, check that the front window belongs to your app and, if so, call TXNIdle. This is Step 9.

Notes

Mouse downs

MLTE helps with the processing of mouse-down events in a window's content region via the **TXNClick** function. This function takes care of scrolling, selecting text, playing sound and movies, handling drag–and-drop operations, and responding to double-clicks.(You no longer need to call this function if you are a Carbon application.)

```
EXTERN_API(void)TXNClick(TXNObject iTXNObject,const EventRecord *iEvent);
```

Our sample calls its own function DoContentClick when appropriate. As you have done earlier, check that the window belongs to your app and, if so, call TXNClick. This is Step 10. (You don't have to check if the window is the front window since this was already done by DoEvent in its "case inContent:" switch.)

Growing the window

If the application has requested a grow region, and if the TXNObject is contained in a window and not a subframe of that window, as ours is, then you can call **TXNGrowWindow** to track the cursor and grow the TXNObject's view rectangle.

```
EXTERN_API(void)TXNGrowWindow(TXNObject iTXNObject, const EventRecord *iEvent);
```

Our sample calls its own function DoGrowWindow when appropriate. As you have done earlier, check that the window belongs to your app and, if so, call TXNGrowWindow. This is Step 11. (You don't have to check if the window is the front window.)

Zooming the window

If the application has requested a zoom box, and if the TXNObject is contained in a window and not a subframe of that window, as ours is, then you can call **TXNZoomWindow** to handle mouse-down events in the zoom box.

```
EXTERN_API(void)TXNZoomWindow(TXNObject iTXNObject, short iPart);
```

Our sample calls its own function DoZoomWindow when appropriate. As you have done earlier, check that the window belongs to your app and, if so, call TXNZoomWindow. This is Step 12. (You don't have to check if the window is the front window.)

Notes

Key downs

Applications need to process keydown events. MLTE provides a function, **TXNKeyDown**, to facilitate this processing. When TXNKeyDown is called, if any Chinese-Japanese-Korean (CJK) script is installed and the current font is CJK inline, input will take place. This is always the case unless the application has requested the bottomline window or has turned off TSM (as with an initialization option).

```
EXTERN_API(void)TXNKeyDown(TXNObject iTXNObject,const EventRecord *iEvent);
```

Our sample calls its own function DoKeyDown when appropriate. As you have done earlier, check that the window belongs to your app and, if so, call TXNKeyDown. This is Step 13. (You don't have to check if the window is the front window.)

Update Events

Applications also need to handle update events (i.e. draw everything in a frame.) MLTE provides the function **TXNUpdate**, which itself calls the Toolbox BeginUpdate - EndUpdate functions for the window that was passed to TXNNewObject. This makes it inappropriate for windows that contain something else besides the TXNObject. In that case, applications should use TXNDraw to update TXNObjects.

```
EXTERN_API(void)TXNUpdate(TXNObject iTXNObject);
```

Our sample calls its own function DoUpdate when appropriate. As you have done earlier, check that the window belongs to your app and, if so, call TXNUpdate. This is Step 14. (You don't have to check if the window is the front window, but you do have be sure the port is set to your window.)

Notes

Activate events

At the time of activation we want to make the TXNObject active in the sense that it can be scrolled if it has scroll bars by calling **TXNActivate**. If the TXNScrollBarState parameter is true, then the scroll bars will be active even when the TXNObject is not focused (i.e., the insertion point is not active). This function can also be used if you have multiple TXNObjects in a window, and you want them all to be scrollable even though only one at a time can have the keyboard focus.

```
EXTERN_API(OSStatus)TXNActivate(TXNObject iTXNObject, TXNFrameID iTXNFrameID,
                                TXNScrollBarState iActiveState);
```

In addition to activating the window, you may want to "focus" the TXNObject by calling **TXNFocus**. When focus is given to a particular TXNObject, then scroll bars and the insertion caret are made active, and inactive otherwise. However, in conjunction with TXNActivate, scroll bars can remain active even though text input is not focused. This is handy for windows containing multiple text areas that are scrollable.

```
EXTERN_API(void)TXNFocus(TXNObject iTXNObject, Boolean iBecomingFocused);
```

Our sample calls its own function DoActivate when appropriate. As you have done earlier, check that the window belongs to your app and, if so, call TXNActivate and TXNFocus. This is Step 15. Note that TXNActivate takes a TXNFrameID parameter as well as a TXNObject parameter. You code may look something like this:

```
TXNObject   object = NULL;
TXNFrameID  frameID = 0;

if (IsAppWindow(window))
{
   GetWindowProperty(window, 'GRIT', 'tFrm', sizeof(TXNFrameID), NULL, &frameID);
   GetWindowProperty(window, 'GRIT', 'tObj', sizeof(TXNObject), NULL, &object);

   if (becomingActive)
   {
      TXNActivate(object, frameID, becomingActive);
      AdjustMenus();
   }
   else
      TXNActivate(object, frameID, becomingActive);
}
TXNFocus(object, becomingActive);
```

Build and run the MLTESampleShell application. Note that the events that we have just implemented are indeed handled. In particular, note how the cursor is adjusted based on the mouse location, the grow and zoom box behavior, the behavior of scroll bars, and the results of keyhits (especially if an input method is activated).