Section 2 - The TXNObject

Purpose

In this section, you will start using the MLTE API by initializing the MLTE manager and by setting up the document object.

Objective

Upon completion of this section you will be able to:

- Use the TXNInitTextension and TXNTerminateTextension API
- Use the TXNObject API: TXNNewObject, TXNDeleteObject
- Store and retrieve the TXNObject: Set/GetWindowProperty

Outline

The following topics will be covered in this section:

- Initializing MLTE
- Creating a disposing of a document object
- The available options for a document object
- Storing and retrieving the object and its frame

Initializing MLTE

The initialization of MLTE should be done as soon as possible after the Macintosh toolbox is initialized. MLTE is initialized by calling **TXNInitTextension**. This function should only be called once per context. If it is called more than once, this function returns a result code of -22012. NoErr is returned if everything initializes correctly.

```
EXTERN_API_C(OSStatus)TXNInitTextension
     (const TXNMacOSPreferredFontDescription  iDefaultFonts[], /*can be NULL*/
      ItemCount         CountDefaultFonts,
      TXNInitOptions    iUsageFlags);
```

The **TXNMacOSPreferredFontDescription** parameter is a table of font information including fontFamily ID, point size, style, and script code. The table can be NULL. Defaults are defined in MacTextEditor.h.

```
struct TXNMacOSPreferredFontDescription {
      UInt32            fontID;
      Fixed             pointSize;
      TextEncoding      encoding;
      Style             fontStyle;
};
```

For example, if you were happy with the default font, style, size, and encoding, you would call the function like this:

```
TXNMacOSPreferredFontDescription defaults;
OSStatus status = noErr;

defaults.fontID = kTXNDefaultFontName;
defaults.pointSize = kTXNDefaultFontSize;
defaults.fontStyle = kTXNDefaultFontStyle ;
defaults.encoding = kTXNSystemDefaultEncoding;
status = TXNInitTextension(&defaults, 1, 0);
```

TXNInitOptions are also passed to the function TXNInitTextension. They specify data types other than text, such as sound and movies, that you wish to support in your application.

TXNInitOptions **bit masks**, such as kTXNWantMoviesMask, are provided in MacTextEditor.h. So, if you wanted to support movies and sound, you might set up your options like this:

```
TXNInitOptions options = kTXNWantMoviesMask | kTXNWantSoundMask;
```

Terminating MLTE

It is necessary to call the **TXNTerminateTextension** function when terminating your application so that MLTE can correctly close down and do other clean up.

```
EXTERN_API(void)TXNTerminateTextension(void);
```

1) At Step 1 in the sample, add the code to initialize MLTE.

2) At Step 2 in the sample, add the code to terminate MLTE.

Notes

Creating the TXNObject

A TXNObject is an opaque structure that encapsulates an object containing private variables and functions necessary to handle text formatting at a document level. For each document, a new TXNObject is allocated and returned by the TXNNewObject function.

```
typedef struct OpaqueTXNObject* TXNObject;
```

You create a new TXNObject using **TXNNewObject**:

```
EXTERN_API_C(OSStatus)
TXNNewObject(const FSSpec                 *iFileSpec, /* can be NULL */
             WindowPtr                    iWindow,
             Rect *                       iFrame,     /* can be NULL */
             TXNFrameOptions              iFrameOptions,
             TXNFrameType                 iFrameType,
             TXNFileType                  iFileType,
             TXNPermanentTextEncodingType iPermanentEncoding,
             TXNObject *                  oTXNObject,
             TXNFrameID *                 oTXNFrameID,
             TXNObjectRefcon              iRefCon);
```

If the **iFileSpec** parameter is not NULL, the file is read to obtain the document contents. If it is NULL, you start with an empty document.

The **iWindow** parameter is required. This is the window in which the document is going to be displayed.

The **iFrame** parameter specifies the area to fill, ususally the window's portRect.

The **iFrameOptions** parameter lets you specify the options to be supported by this frame. The TXNFrameOptions **bit masks**, such as kTXNDrawGrowIconMask, are enumerated in MacTextEditor.h. For example, if you wanted support for scrollbars and a grow box, you would specify your frame options like this:

```
frameOptions = kTXNWantHScrollBarMask | kTXNWantVScrollBarMask
    | kTXNDrawGrowIconMask;
```

Another useful frameOption is `kTXNShowWindowMask`. If you specify this option, you no longer need to call ShowWindow(...) on this window; MLTE will do this for you.

Options for **iFrameType**, such as kTXNTextEditStyleFrameType, are defined in MacTextEditor.h, as are options for **iPermanentEncoding**, such as kTXNSystemDefaultEncoding.

In the **iFileType** parameter you specify the primary file type. Possibilities are:

kTextensionTextFile: a private format, can contain graphics and sound.
'TEXT': plain text files, can specify 'styl' (multiple) or 'MSPR' (single) styles in the frameOptions parameter. A plain text editor like SimpleText would specify 'styl'. An editor like BBEdit or MPW would use 'MPSR'.

On successful return, the **oTXNObject** will contain a pointer to the opaque TXNObject.

iFrameID is intended to hold, on return, a unique ID for the frame, however in version 1.0 this value is always set to 0.

You can set the **iRefCon** parameter to any value. It is retained by the TXNNewObject which can later be asked to return it.

A good place to call TXNNewObject is in the sample's DoNew() function. DoNew() is called when the application launches, in order to display a document at launch time, and also is called whenever the user selects "New" from the File menu.

1) At the Step 3 tag, add the code to create a new TXNObject. Your code might look something like this:

```
OSStatus          status = noErr;
TXNObject         object = NULL;
TXNFrameID        frameID     = 0;
Rect              *framePtr = &window->portRect;
TXNFrameOptions   frameOptions = kTXNWantHScrollBarMask | kTXNWantVScrollBarMask |
                                 kTXNDrawGrowIconMask | kTXNShowWindowMask;

status = TXNNewObject(  NULL, // start with an empty document
                        window,
                        framePtr,   // the area to fill
                        frameOptions,
                        kTXNTextEditStyleFrameType,
                        kTXNTextensionFile,
                        kTXNSystemDefaultEncoding,
                        &object,    // ptr to opaque TXNObject
                        &frameID,   // always 0 in ver 1.0
                        0);
```

You will need to be able to pass the TXNObject to many of the MLTE API. Some of the API also want the frameID. There are, of course, many ways to do this. One simple way is to store references in a window property with `SetWindowProperty(...)` and retrieve them with `GetWindowProperty(...)`. Get/SetWindowProperty were introduced in Mac OS 8.5. You can find more infomation on them in the Window Manager documentation.

```
EXTERN_API(OSStatus)
SetWindowProperty(WindowRef         window,
                  PropertyCreator   propertyCreator,
                  PropertyTag       propertyTag,
                  UInt32            propertySize,
                  void *            propertyBuffer);
EXTERN_API(OSStatus)
GetWindowProperty(WindowRef         window,
                  PropertyCreator   propertyCreator,
                  PropertyTag       propertyTag,
                  UInt32            bufferSize,
                  UInt32 *          actualSize, /* can be NULL */
                  void *            propertyBuffer);
```

2) At the Step 4 tag, add the code to store the object and the frameID as window properties. For example:

```
status = SetWindowProperty(window,'GRIT','tFrm',sizeof(TXNFrameID),&frameID);
status = SetWindowProperty(window,'GRIT','tObj',sizeof(TXNObject),&object);
```

(The propertyCreator and propertyTag are up to you.)

You can retrieve the information with a call to GetWindowProperty.

```
TNXObject   object      = NULL;
TXNFrameID  frameID     = 0;
status = GetWindowProperty(window,'GRIT','tObj',sizeof(TXNObject),NULL, &object);
status = GetWindowProperty(window, 'GRIT', 'tFrm', sizeof(TXNFrameID), NULL, &frameID);
```

Finally, now that we have created a TXNObject for our window, when we close the window we will want to delete the object and all associated data structures.

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

2) Windows are closed in the DoCloseWindow function in the sample. At the Step 5 tag, add the code to delete the TXNObject.

Notes