Section 4 - Edit Commands

Purpose

In this section, you will use MLTE's edit command API. MLTE supports the Undo, Redo, Cut, Copy, Paste, Clear, and Select All items in the Edit menu.

Objective

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

- Use these twelve MLTE editing related API:

  TXNUndo, TXNRedo, TXNCanUndo, TXNCanRedo, TXNIsSelectionEmpty, TXNCut, TXNCopy, TXNClear, TXNIsScrapPastable, TXNPaste, TXNDataSize, and TXNSelectAll

Outline

The following topics will be covered in this section:

- Checking for Undo and Redo-ability
- TXNActionKeys
- Checking for Cut, Copy, Paste, and Clear-ability
- Checking for Selection-ability

Undo and Redo

MLTE provides strong support for Undo and Redo actions.

The undo level in MLTE 1.0 is 32 levels deep. That is to say, Undoable actions are collected until the total count is 32. If a user undoes two actions she will need to do redo twice to get back to the original state. If more than 32 actions are performed the oldest actions are forgotten as each new action takes place. The undo and redo functions are:

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

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

The TXNCanUndo/Redo functions can tell you if an action is un/redoable, and also provide a keycode which is the type of action that can be un/redone:

```
EXTERN_API(Boolean)TXNCanUndo(TXNObject iTXNObject, TXNActionKey* iActionKey);

EXTERN_API(Boolean)TXNCanRedo(TXNObject iTXNObject, TXNActionKey* iActionKey);
```

The optional action key codes (if you are not interested, use NULL as a parameter) in TXNCanUndo/Redo identify the action that can be un/redone. You are be responsible for mapping keycodes to appropriate localized strings for display. For example, if the value of iActionKey was kTXNTyping, you could then map that value to a localized string such as “Redo Typing” in the Edit menu. The keycodes are defined in MacTextEditor.h.

There are two things to do when coding an Edit command. First is to decide whether the command item should be enabled in the Edit menu, and the second is to carry out the command if the user actually chose that item.

Menu enabling and disabling is carried out in our sample app's AdjustMenus function. At Step 16, write the checks to see if the Undo and Redo menu items should be enabled. You code may look something like this:

```
GetWindowProperty(window, 'GRIT', 'tObj', sizeof(TXNObject), NULL, &object);
if (TXNCanUndo(object, NULL))
   undo = true;
if (TXNCanRedo(object, NULL))
   redo = true;
```

Finally, call the TXNUndo(...) and TXNRedo(...) functions in Steps 17 and 18 in the sample app's DoMenuCommand function.

Cut, Copy, Clear

MLTE provides functions for Cut, Copy, and Clear Edit menu actions:

```
EXTERN_API(OSStatus)TXNCut(TXNObject iTXNObject);
```

Cut the current selection to the MLTE private clipboard.

```
EXTERN_API(OSStatus)TXNCopy(TXNObject iTXNObject);
```

Copy the current selection to the MLTE private clipboard.

```
EXTERN_API(OSStatus)TXNClear(TXNObject iTXNObject);
```

Clear the current selection from the TXNObject.

Note that each of these functions operates on selected text. One way to determine if the user has selected some text is to use the TXNIsSelectionEmpty function:

```
EXTERN_API(Boolean)TXNIsSelectionEmpty(TXNObject iTXNObject);
```

If a selection is non-empty, then we want to enable the Cut, Copy, Clear menu items.

In our sample app's AdjustMenus function, at Step 19, check to see if the Cut, Copy, and Clear menu items should be enabled. You can do this by checking whether TXNIsSelectionEmpty returns false (meaning some text is indeed selected).

Also, call the TXNCut(...), TXNCopy(...), and TXNClear(...) functions in Steps 20, 21, and 22 in the sample app's DoMenuCommand function.

Notes

Paste

MLTE provides functions to deal with the scrap, facilitating the pasting of text:

```
EXTERN_API(OSStatus)TXNPaste(TXNObject iTXNObject);
```

Paste the clipboard into the TXNObject.

You can also test to see if the current scrap contains data that is supported by MLTE. TXNIsScrapPastable can be used to determine if the Paste item in the Edit menu should be active or inactive.

```
EXTERN_API(Boolean)TXNIsScrapPastable(void);
```

In our sample app's AdjustMenus function, at Step 23, check to see if the Paste menu item should be enabled by checking the return value of TXNIsScrapPastable.

Also, call the TXNPaste(...) function at Step 24 in the sample app's DoMenuCommand function.

Selections

MLTE provides functions to deal with selections:

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

Selects all data belonging to the TXNObject.

You can test whether the Select All menu item should be enabled by asking whether the TXNObject contains any data:

```
EXTERN_API(ByteCount)TXNDataSize(TXNObject iTXNObject);
```

Return the size in bytes of the characters in a given TXNObject.

In our sample app's AdjustMenus function, at Step 25, check to see if the Select All menu item should be enabled by checking whether the return value of TXNDataSize is non-zero.

Also, call the TXNSelectAll function at Step 26 in the sample app's DoMenuCommand function.

Build and run the MLTESampleShell application. Note when the Edit menu items are enabled and disabled. Try out the Edit menu commands.