XText 0.9

Introduction

I was unhappy to discover after switching from my NeXTStation to an Intel P5 running NeXTStep that XText 0.8 broke. This is because version 0.8 uses keyboard specific "key
codes".

XText 0.9 is a hardware independent version of XText, with a few extra goodies thrown in for fun. It uses "character codes", which are NeXT/Adobe's generalization of ASCII codes.
XText reads character codes from keyboard events and allows the user considerable flexability in redefining keystrokes. This is achieved by Mike Dixon's remarkable "action parsing" code, where text strings are convered into XText method calls.

XText 0.8 was written by Mike Dixon:

Mike Dixon
Xerox PARC
3333 Coyote Hill Rd.
Palo Alto, CA 94304
mdixon@parc.xerox.com

Much of the discussion below is based on the original XText0.8 ;XText0.8.rtf;;¬README file. Major changes to XText are:

a. An expanded version of XTDemo saves keybinding files, and XText reads keybinding files.

b. Hardware independent modifiers:

- l = NX_ALPHASHIFTMASK
  - Alpha Lock is set and Shift key is NOT* down
- s = NX_SHIFTMASK
  - a shift key is down
- c = NX_CONTROLMASK
  - Control key is down
- a = NX_ALTERNATEMASK
  - Alternate key is down
- m = NX_COMMANDMASK
  - Command key is down
- n = NX_NUMERICPADMASK
  - Key is on numeric keypad
- h = NX_HELPMASK
  - Help Key

c. Keycodes can be entered by Capitalized HEX numbers.

Basic Concepts

XText lets users execute complex text editing and formating commands by redefining keystrokes. It does this by letting the
user associate, for every key, text object methods. For

example, to associate the "control a" key with the   method

```
        [xtext replaceSel:"Hello World\n"];
```

The user, or application programmer, constructs a keybinding format

```
        c'a = replaceSel: "Hello World\n"'
```

XText can parse, in this way,   methods with at most two arguments, which must be either integers or strings.   XText has an expanded set of useful methods, that give users "emacs like" control over their keystrokes.   The expanded list is given below.

XText enables programmers to easily incorporate this keyboard functionality into their applications.   Keybindings can be loaded at run time,   using an XText method which parses strings of keybindings, or (new to version 0.9) a method which reads a file of keybinding strings.   The file reader method was added to work with the concept of an application wrapper. (That is, programmers can save keybindings as a file   in their .app directory.)

XText 0.9 methods for keybindings

The following methods were written for easy keybinding

construction. All of the cursor-movement methods take a
`mode' argument, which may be

0	just move the point to new location
1	delete to new location
2	cut to new location
3	extend selection to new location

The methods for cursor-movement are:

goto:end:mode: implements all movement; second argument specifies
the other end of the selection when mode != 0

moveWord:mode: move n words forward from point (back if n<0)

moveChar:mode: move n chars forward from point (back if n<0)

moveLine:mode: move n lines down from point (up if n<0)

lineBegin: move to beginning of current line

lineEnd:
move to end of current line

docBegin:	move to beginning of document
docEnd:	move to end of document
collapseSel: (dir<0), end of (dir=0)	move to beginning of selection selection (dir>0), or active end of sel
transChars	transpose characters around point
openLine	insert new line after point
scroll::	scroll window n pages + m lines
scrollIfRO::	scroll window n pages + m lines if doc is read-only; returns nil if doc is editable
insertChar:	inserts the character associated with a key event
insertNextChar key event will be	sets nextAction so that the next interpreted as a character

Methods for c program formatting:

autoIndent
creates a new line with space and tab indentation equal to the current line

match:"LR";
Finds previous correctly nested matched character L and
briefly displays it; then prints R.   Useful for "()" "{}"
and "[]".

Character Codes

Keyboard independent character codes which XText 0.9 uses
are constructed using simple rules.   They can be found in the
insertKeyCombination: method of XText.

a. Type the following characters to denote modifier keys:

c	control key down,
s	shift key down,
a	alt key down,
m	command key down,
n	a numeric keypad character, on my `101' keyboard, the arrow keys and the keypad,
l	caps-lock key down, and shift key NOT pressed,
h	help key down

Note: Use "l" if you want a charater code to
be active only if the caps-lock key is down.   Other
character codes work with the caps-lock key
on or off.

b. Add the ' (right quote) key to denote the beginning of the
actual key.

c. If the key will print, type it.

Note: Alt characters are generally mapped to the
upper 127 chacters of the Adobe/NeXT extended character
set. Being non-ASCII characters, they look unusual.

d. If it a non-printing character like space, tab, or return, type
its hexadecimal character code. Character codes are found
in Appendix C of the AppKit Documentation (Figure C1).

e. Hexadecimal codes are denoted by the characters 1-9, A-F.
(The uppercase in the hex numbering is important.)

f. If it is a control character, type the key of the character; i.e.
"control a" = c'a, "control shift A" = cs'A etc.

Binding Specifications (Putting it all together)

A binding spec is a comma separated list of key codes,
followed by an equal sign, followed by an action (a method
above with input data). For example,

c'w, a'ã = moveWord:-1 mode:1
(control w and alt h = delete last word)

c'b=moveChar:-1 mode:0; c'B=moveChar:-1 mode:3

Comment lines in the example files below begin with `#'

;arrow.keys.rtf;arrow.keys;¬arrow.keys
;emacs.keys.rtf;emacs.keys;¬emacs.keys
;keypad.keys.rtf;keypad.keys;¬keypad.keys
;programmer.keys.rtf;programmer.keys;¬programmer.keys
;readonly.keys.rtf;readonly.keys;¬readonly.keys

Program Implimentation

1. Copy the XText subproject into your application. Include the line

```
#import "XText.subproj/XText.h"
```

In files that create XText and XTScroller objects.

2. Occurrences of [Text alloc] must be replaced with

[XText alloc].

If you're using IB to construct your Text
objects it currently provides no clean way to make a
ScrollView containing something other than a Text, so
there is a support class XTScroller that provides just
that --
simply replace your ScrollViews with XTScroller custom
views and the XTexts will be constructed automatically.
These newly-created XText objects will behave just like
Text objects; in particular, they will have no key
bindings..

3a. Initialize an "action", which stores and parses
(interprets strings into method calls) keybindings:

```
demoAction =
[[XTDispatchAction alloc] init];
```

3b. Or initialize using a default keybinding table:

```
                    action = [[XTDispatchAction alloc]
                    initBase:NXGetDefaultValue("myApp",
   "KeyBase") estream:nil];
```

(NOTE: in XText0.9, the emacs table
has been removed.   The advent of
".app" wrappers in NS 3.0 makes

storage of keybindings as files
in application directories a much
more elegant approach.   See 4b below.)

4a. Add any dwrite type user-defined bindings

```
                    [action addBindings:
                    NXGetDefaultValue("myApp",
"KeyString")
                    estream:nil];
```

4b. Or add any file of user-defined bindings.

```
                    [action loadFromFile:
                    NXGetDefaultValue("myApp", "KeyFile")
```

loads bindings from a file.   Comments are lines in the file beginning with `#'.   This method enables developers to load keybinding files from their .app wrapper directories, via

```
    [[NXBundle mainBundle]
        getPath:path
        forResource:"KeyBindingFiles"
        ofType:""];
```

5.   Attach the action to the text object.

[myXText setInitalAction:action];

6. See XTDemo.app   for more example code.

;XText.rtfd;format;¬**The Format of Binding Specifications**

The format used to specify bindings is:

A *binding spec* is a sequence of zero or more *bindings*, separated by `;'s

A *binding* is a key spec, followed by an `=', followed by an *action*

A *key spec* is a sequence of one or more *key combinations*, separated by `,'s

A *key combination* is a sequence of zero or more *modifiers*, followed by a *key*

A *modifier* is

c	(control),
s	(shift),
a	(alt),
m	(command),
n	(numeric keypad),
l	(caps-lock),
h	(help key)

A *key* is a `'' followed by any character (designates the key
 that generates that character), or a 2-digit hex key code, as
 documented   in

NextLibrary/Documentation/NextDev/GeneralRef/
_ApC_KeyboardEvents/KeyInfo.rtfd

An *action* is a *message*, or a sequence of *actions* separated
by
 `;'s and enclosed in `{}'s

A *message* is something like
`moveWord:-1 mode:1'   or
`replaceSel: "hi there\n"'
(at most two arguments, which must be either integers or
 strings)

Paul Griffin, 7/95