# XText: An Extensible Text Object

This package defines a subclass of the Appkit's Text object that is designed to allow the easy addition of new key bindings, both by application programmers and end users. It also provides a reasonably comprehensive set of initial key bindings, based largely on those of emacs.

## What it looks like to the end user

Here's a scenario of the functionality XText could provide to the user of a hypothetical mail-reading program.

First, the user can specify custom key bindings that will be available in all programs based on XText. For example, if you wanted ctrl-shift-K to delete to the beginning of the current line, and alt-: to add a `:' and new line after the next word, you could say

```
dwrite -g KeyBindings "c'K=lineBegin:1; a':={moveWord:1 mode:0;
                                                replaceSel:\":\n\"}"
```

Of course, if you want key bindings that are specific to a particular application you can replace the `-g' with the name of the application. (Read the man page for dwrite if you're unfamiliar with it.)

If you would rather define all your own key bindings from scratch, rather than starting with the emacs base set, you can say

```
dwrite -g KeyBase none
```

Second, the mail program might define its own subclasses of XText with methods on them specialized to the particular requirements of a mail program, and use XText to assign appropriate bindings to each. For example, it might define one subclass for the message display window, and bind `n' and `p' to move to the next and previous messages. (Since this XText object is read-only, it will already have space and delete bound to forward and backward page scrolling.)

Finally, the user can specify additional bindings specific to each of these window types, that invoke their specialized methods. For example, to bind ctrl-c to the sendMessage method, you could just say

```
dwrite Mail SendWindowBindings "c'c=sendMessage"
```

## What it looks like to the application programmer

There are a few steps required by the application programmer to use XText in this scenario.

First, 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 Ð simply replace your ScrollViews with XTScroller custom views and the XTexts will be constructed automatically. (This could probably also have been handled by a custom palette, but I haven't tried to figure those out yet). These newly-created XText objects will behave just like Text objects; in particular, they will have no key bindings yet.

Second, you need to construct a ªdispatch actionº to store the key bindings in; the code will look something like this:

```
id action = [[XTDispatchAction alloc]
```

```
                    initBase:NXGetDefaultValue("myApp","KeyBase")
                    estream:nil];
```

The second argument to initBase:estream: is an object of class ErrorStream; this allows you to control the reporting of errors, but the default error stream (which just pops up an alert panel with the message) is usually adequate.

Third, you want to add in the user's custom key bindings:

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

To have special bindings for some XTexts (like the message window), copy this action and add them in:

```
        id msgAction = [[action copy]
                    addBindings:"'n=changeMsg:1; 'p=changeMsg:-1"
                    estream:nil];
```

(This assumes you've defined a subclass of XText with a changeMsg: method.)  Then add in any custom user bindings for message windows:

```
        [msgaction addBindings:NXGetDefaultValue("myApp","MsgWindowBindings")
                estream:nil];
```

Finally, attach these actions to the appropriate XText objects (each action can be shared by many XTexts):

```
        [simple_XText setInitialAction:action];
        [msg_XText setInitialAction:msgAction];
```

¼ and you're done.

One more thing: if you've got windows with TextFields, you probably want the key bindings to work in them too.  To arrange this you'll have to provide a delegate for your window, with a windowWillReturnFieldEditor:toObject: method that looks like this:

```
- windowWillReturnFieldEditor:sender toObject:client
{
        return [XText newFieldEditorFor:sender
                        initialAction:action
                        estream:nil];
}
```

## 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), or `m' (command)
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/Summaries/06_KeyInfo
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)

Some examples:
c'w, a'h = moveWord:-1 mode:1
c'b=moveChar:-1 mode:0; c'B=moveChar:-1 mode:3
(c'B could also have been written as cs'b, or as cs35).
csam49={docBegin:0; moveWord:5 mode:2; docEnd:0; paste:0}
(makes ctrl-shift-alt-command-escape move the first five words to the end of the document!)

## A Simple Testbed

This distribution also includes a very simple demonstration program, called XTDemo. XTDemo puts up a single window with an XText to play with, and an XText-backed text field in which you can enter new key bindings.

In addition, XTDemo adds a custom key binding so that ctrl-shift-Q inserts the key code for the next key you hit; for example, ctrl-shift-Q ctrl-alt-escape inserts the string `ca49'.

## The Emacs base set

The key bindings provided in the default base set are:

Movement

| | |
|---|---|
| ctrl-f, ctrl-b | move one character forward / back |
| alt-f, alt-b | move one word forward / back |
| ctrl-n, ctrl-p | move one line down / up |

| | |
|---|---|
| ctrl-a, ctrl-e | move to beginning / end of line |
| alt-<, alt-> | move to beginning / end of document |

Deletion

| | |
|---|---|
| ctrl-d, del (or ctrl-h) | delete next / previous character |
| alt-d, alt-del (or alt-h) | delete next / previous word |
| ctrl-k | delete to end of line |

Selection

| | |
|---|---|
| ctrl-shift-F, ctrl-shift-B | extend selection one character forward / back |
| alt-shift-F, alt-shift-B | extend selection one word forward / back |
| ctrl-shift-N, ctrl-shift-P | extend selection one line down / up |
| ctrl-shift-A, ctrl-shift-E | extend selection to beginning / end of line |

Scrolling

| | |
|---|---|
| ctrl-v, alt-shift-downarrow | scroll one page forward |
| alt-v, alt-shift-uparrow | scroll one page back |
| ctrl-shift-V, alt-shift-V | scroll four lines forward / back |
| ctrl-alt-uparrow | scroll to beginning of document |
| ctrl-alt-downarrow | scroll to end of document |
| ctrl-l | scroll to selection |

Additional scrolling when editing disabled

| | |
|---|---|
| space, del | scroll one page forward / back |
| shift-space, shift-del | scroll four lines forward / back |

Miscellaneous

| | |
|---|---|
| ctrl-t | transpose characters |
| ctrl-o | insert new line after caret |
| ctrl-space | collapse selection |
| ctrl-q | quote next key |
| ctrl-alt-q | really quote next key |

(Ctrl-q causes the next character to be handled directly by the underlying Text object, with no XText-supplied rebinding; for example, ctrl-q alt-b inserts a sigma. Ctrl-alt-q goes one step further and avoids any special handling that Text normally supplies for that key; for example, ctrl-alt-q downarrow causes a downarrow character to be inserted (you'll probably want to be in the symbol font), and ctrl-alt-q return allows you to insert a newline in a text field.)

## XText Status and Future

This should be thought of as a beta-test version of XText; although it has no known bugs, it has not been very heavily exercised. In particular, I have not yet built it into any non-trivial programs. I plan to maintain & use XText, but I also have a job and this isn't it.

XText is freeware; you are welcome to use it, modify it, and distribute it without restriction (although I would appreciate having my name kept on it). It is copyrighted by my employer (Xerox), but only to prevent someone from claiming that it belongs to them.

Please do send me bug reports and suggestions, and let me know if you find it useful. This is my first experience with objective-C and Interface Builder, so there is certainly room for improvement.

Anyone want to build a replacement for Edit using XText?

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