TextORama

by Sharon Zakhour, NeXT Developer Support Team

Valid for 3.0

Overview

This MiniExample illustrates two things:

· How to create a TextField which supports one or more of these features:
- Restricting text length (both when typing and pasting).
- Autojumping to another TextField when maximum length is reached.
- Interpreting a carriage return literally rather than as an indication to end editing.

· How to implement emacs key binding support for the Text object.

Topics Of Interest

TurboTextField:

There are several ways available to modify the behavior for entering text. These include:
Character filter
Text filter
Text delegate methods such as **textDidGetKeys:isEmpty:**

Two standard character filters are provided by NeXT -- NXFieldFilter (used primarily by TextFields) and NXEditorFilter (used primarily for standard text editing). [See reference information below for further information on the Text class.] There is no default text filter. Both types of filters are implemented as function calls with a pre-defined argument protocol.

They cannot be subclassed but they can be replaced using the **setCharFilter:** and **setTextFilter:** methods of Text.

How does a character filter differ from a text filter?
A character filter is typically appropriate for very straightforward filtering -- trapping the character typed by the user and replacing it with another. A character filter has no awareness of where it sits in the grand scheme of things. A text filter is slightly more sophisticated having knowledge of the Text object and the insertion point for the intended text, for example. A text filter is ideal for implementing a date field, a social security field or a phone number field -- text with a specific format. Unfortunately both character and text filters have one drawback -- they are not called during the paste mechanism. This can be counteracted with the Text delegate method **textDidGetKeys:isEmpty:**.

How do all of these filters and methods work together?
Assuming that the full battery of filters and such have been implemented, they are deployed in the following order when any key is entered:

charFilter
textFilter
textDidGetKeys:isEmpty:

So what's the point of this MiniExample?
This example attempts to combine several often-requested TextField behaviors into one class. These behaviors are:

· The ability to end editing after a maximum number of characters have been entered for a specific field -- "maxLength".
·€The ability to cause the insertion point to automatically jump to the next TextField when the maximum number has been met -- "autoJump".
· The ability to interpret the Return character literally -- instead of as a signal to move to the

next TextField -- "acceptsReturn".
· The ability to employ any combination of the above features -- a non-autojumping, non-return accepting TurboTextField of maxLength 0 will behave as any standard TextField.
· The ability for all of the above to work consistently with the Text paste mechanism.

How to implement this?
A character filter is required to implement autoJump -- say you want to jump after 5 characters: this is done by posting a minutely delayed faked event containing the 6th character typed (so that it doesn't get lost) immediately after returning a TAB in place of that character. This implementation is not perfect -- you cannot backspace to the previous TextField and there is a slight lag because the cursor does not jump to the next TextField until the user actually types the overflow character.

A character filter is also required to implement acceptsReturn. When a RETURN key is detected and acceptsReturn is in effect, the standard NXEditorFilter() is employed (because it handles Returns literally) otherwise the default NXFieldFilter() is used.

The lengthWatching "maxLength" mechanism requires hooks in the character filter (for the autoJump mechanism), the text filter *and* **textDidGetKeys:isEmpty:** (for pasting). All of these mechanisms must work together in order to keep the TextField on or below it's maximum length.

It is necessary to subclass TurboTFCell primarily in order to override the **select:inView:editor:delegate:start:length:** and **edit:inView:editor:delegate:event:** methods -- it's important for the filters and the text delegate methods to know which Cell is currently being edited -- one of these two methods is always called whenever a TextField becomes the firstResponder and overriding these methods allows an internal static variable to be reset to the current Cell. Since each Cell maintains information about it's maxLength, autoJumping, textFilter, etc. it's essential that all of the filters can get access to this information via class methods. Methods such as **setAutoJump:forLength:** and

setAcceptsReturn: have been added to this class. Identical methods have been added to the TurboTextField class which call these TurboTFCell methods, so the programmer need only worry about the TurboTextField class and allow this class to do it's thing invisibly. The character and text filters have also been implemented in this class. For those wishing to customize the text filter for a specific Cell, the **setCustomFilter:** method is provided.

The three TurboTextFields in the nib file install their own custom filter in this way: the Date Field, the Social Security Number Field and the Phone Number Field.

TextField also must be subclassed in order to install TurboTFCell as it's cell class. This is also where **textDidGetKeys:isEmpty:** is implemented. When a user pastes an illegally long string into the TurboTextField, the previous contents of the field will be restored -- the TurboTFCell maintains an instance variable "originalText" with this information.

EmacsText:

Implemented as a subclass of Text using a table-driven approach. The **keyDown:** method for the Text subclass has been overridden. In the new method, each key stroke is examined to determine whether it is an emacs character, in which case it is interpreted, or whether the key should be passed through to the superclass's **keyDown:** method. This object was created in IB as a custom view (retyped to EmacsText) which was put into a ScrollView using the Layout/Group In ScrollView feature. No other IB connections were required.

Other Files

SoapStory.rtf An RTF file loaded into the Emacs window.

SEE ALSO:

· "The Text Class", p. 9-4 (Chapter 9: User-InterfaceObjects) of the NeXTstep Concepts Manual.
· The Text Class spec sheet, p. 2-557 of the NeXTstep Reference Manual, volume 2.

KUDOS:

Thanks to:
·€Julie Zelenski for the Emacs binding support code.
·€Jayson Adams for the "Auto Jump" idea.

Change History:

· 4/14/92 sz: Found and fixed a crasher in the date filter (TextORama.m) which was due to a statically allocated string. [The crash would occur if you pasted in a string longer than 10 characters into the date field.]

· 4/27/92 sz: Changed the mechanism used to set the cell class for the TurboTextField. Previously I was setting the cell class in +initialize but this may create problems if you attempt to create any other instances of TextField in your application (depending upon the order of events). This is an inherent problem to the way TextField is implemented (and the other controls as well). Bug #16741 has been filed against the documentation to explain this more fully.

The TextField class maintains a static variable indicating the cell class. (Objective-C does not support class variables.) When you set this in +initialize it changes that class for everything -- TextField as well as TextField subclasses. So once you changed it for your TextField subclass, it will continue to effect all other TextFields created everywhere!

The way to workaround this is to *not* set the cell class in the +initialize method in the TurboTextField class and instead, set it temporarily when init'ing a textField of your type and set it back afterwards:

```
@implementation TurboTextField

- initFrame:(const NXRect *)frameRect
{
    id newTextField;

    [[self class] setCellClass: [TurboTFCell class]];
    newTextField = [super initFrame:frameRect];
    [[self class] setCellClass: [TextFieldCell class]];
    return newTextField;
}

...

@end
```

· 7/23/92 sz: Updated to 3.0. Primary changes were:

- Replaced all of the "#import <appkit/Class.h>" lines with "#import <appkit/appkit.h> in the Header files.
- Modified the **loadEmacsScrollView** method of TextORama to read the text file from the main bundle.
- Under 3.0 you can no longer initialize a variable using the "@selector(foo)" syntax at compile time. I added a +initialize method to EmacsText to take care of this.