49

How to Design a Bumble Bee, or

a Curl Code Editor that will Fly...

by

Jon S. Heiner

Submitted to the Department of Electrical Engineering
and Computer Science in partial fulfillment of the
Requirements for the degree of Master of Engineering in
Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 7, 1998

Author .....
Department of Electrical Engineering andComputer Science
August 7, 1998

Certified by .....
Professor Stephen A. Ward
Department of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

How to Design a Bumble Bee, or a Curl Code Editor that will Fly...

by

Jon Heiner

Submitted to the

Deptartment of Electrical Engineering and Computer Science

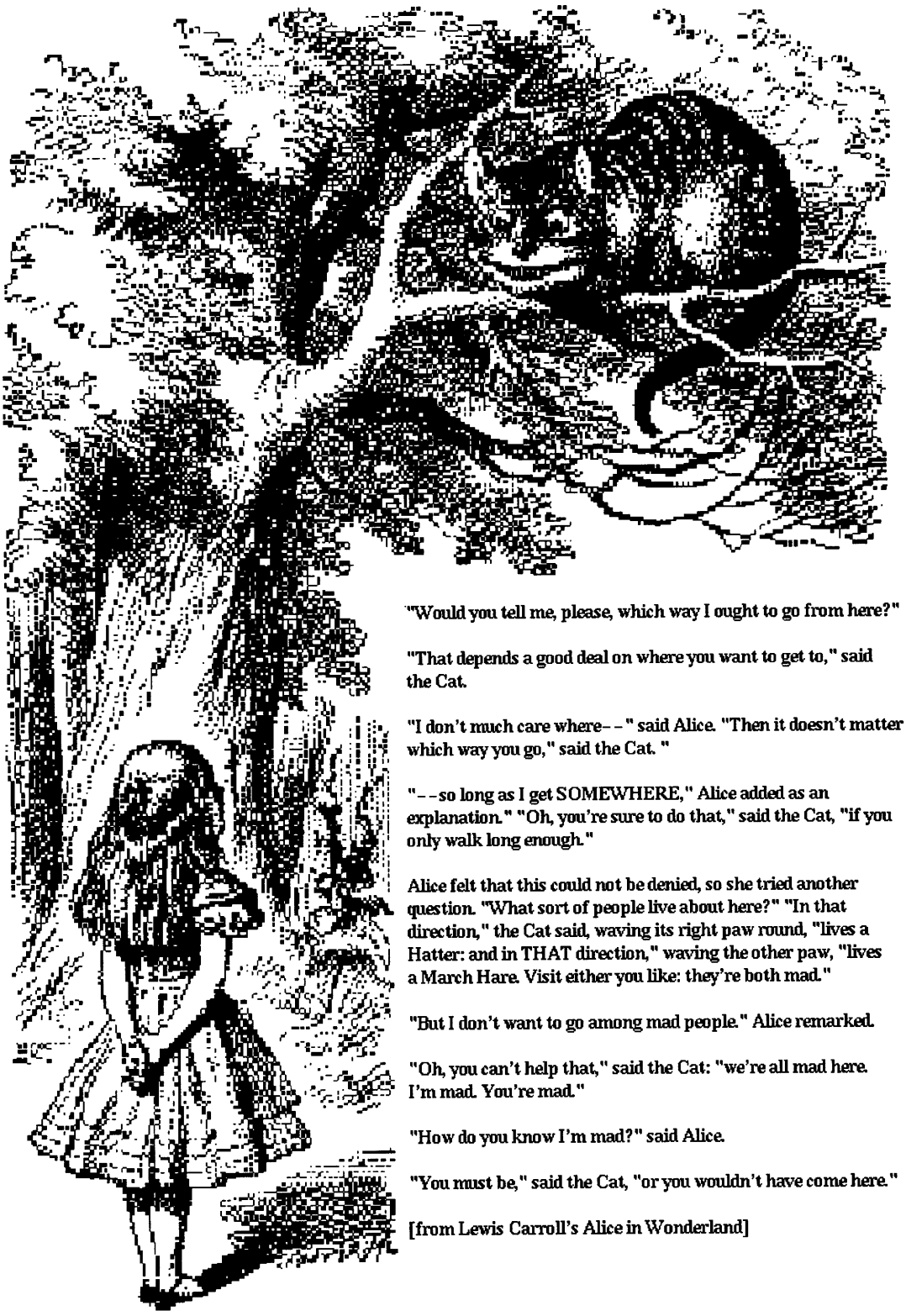
August 7, 1998

In Partial Fulfillment of the Requirements
for the Degree of Master of Engineering in
Electrical Engineering and Computer Science.

Abstract

The Curl language is a new web development language being written in the Computer Architecture Group at the MIT Laboratory for Computer Science. This thesis describes my design considerations for writing a code editor expressly for the language Curl. The motivation, design philosophy and process, architecture, and implementation details are provided here. The overall goal is to is to create an editor which expediates the process of writing Curl code while providing a pleasant environment in which to work with and learn this "gentle-slope" language.

Thesis Supervisor: Stephen A. Ward
Title: Professor, EECS



"Would you tell me, please, which way I ought to go from here?"

"That depends a good deal on where you want to get to," said the Cat.

"I don't much care where--" said Alice. "Then it doesn't matter which way you go," said the Cat. "

"--so long as I get SOMEWHERE," Alice added as an explanation." "Oh, you're sure to do that," said the Cat, "if you only walk long enough."

Alice felt that this could not be denied, so she tried another question. "What sort of people live about here?" "In that direction," the Cat said, waving its right paw round, "lives a Hatter: and in THAT direction," waving the other paw, "lives a March Hare. Visit either you like: they're both mad."

"But I don't want to go among mad people." Alice remarked.

"Oh, you can't help that," said the Cat: "we're all mad here. I'm mad. You're mad."

"How do you know I'm mad?" said Alice.

"You must be," said the Cat, "or you wouldn't have come here."

[from Lewis Carroll's Alice in Wonderland]

Acknowledgements

I wish to thank my family for their constant support during all my trials at MIT, this one being the last and the most satisfying to finish, for it is a closure. Thanks to them for seeing me through to the end, giving sound advice and moral support from thousands of miles away, and always putting up with the only "techie" in the family.

I wish to thank Irena Asmundson who probably was affected directly by my thesis more than anyone else (other than myself.) Thanks Ii for your support, your friendship and above all, your understanding, much love.

I wish to thank the Curl group, first at the Laboratory of C omputer Science at MIT and now on their own wild adventure at Curl, Inc. Special thanks to my Thesis Advisor Professor Stephen Ward for his patience. Thanks to Dr. David Kranz, Chris Metcalf, Anne Hunter, Patrick LoPresti and Marty Wagner. Best of luck.

Thanks to rms and marc, both of whom I know better by their usernames than by their faces, for their relevant help on my thesis: rms: for creating Emacs and doggedly pursuing the ideal of free software; marc: for his "intuitively obvious" algorithm.

There are many other people who I wish to thank for their support, wisdom, advice, distractions, friendship and comfort: Trevor Stricker who provides the vibe, my instructors at the Somerville School of Chung Moo Doe and higher belts, my brothers at ATO, my brothers, sisters and others at No.6, the on-line crew @ Lambda without whom I would either never have finished or finished much earlier, my officemates Mark "Wally" Herschberg and Arthur E Housinger for the many deep philosophical meanderings and for restocking the refrigerator, Boca Grande for sustenance, and Jasmine for always being there when she needed me.

Thanks all,

Jon Heiner

Table of Contents

Chapter 1

Introduction

1.1 Curl

Curl is a new language under development at the MIT Laboratory for Computer Science. Its goal is to provide a rich development base for the rapidly growing requirements of the developing World Wide Web.

> Curl is a new language for creating web documents with almost any sort of content, from simple formatted text to complex interactive applets.
>
> Curl provides a rich set of formatting operations similar to those implemented by HTML tags. Unlike HTML, the Curl formatter can be extended by users to provide additional functionality, from simple macros (e.g., to provide a convenient way to switch to a particular font, size and color) to direct control over the positioning of subcomponents (e.g., as in a TeX-like equation formatter). Several packages of useful formatting extensions are currently under development.
>
> Using a TK-like interface toolkit of interactive components, Curl makes it easy to build simple interactive web pages. One can view interactive objects like buttons or editable fields as extensions to the basic formatting operations provided above -- one uses the same easy-to-learn syntax to create interactive documents as to create regular text documents. There's no need to learn a separate scripting language!
>
> Other components of an interactive document may require more sophisticated mechanisms than are provided by the interface toolkit. These components can also be developed using Curl since, at its heart, Curl is really an object-oriented programming language. Curl expressions embedded in the web document are securely compiled to native code by the built-in on-the-fly compiler and then executed without the need for any sort of interpreter. Curl provides many of the features of a modern object-oriented programming language: multiple inheritance, extensible syntax, a strong type system that includes a dynamic "any"

> type, safe execution through encapsulation of user code and extensive checking performed both at compile and run time. Almost all of the Curl system and compiler are written in Curl.
>
> Curl is intended to be a gentle slope system, accessible to content creators at all skill levels ranging from authors new to the web to experienced programmers. By using a simple, uniform language syntax and semantics, Curl avoids the discontinuities experienced by current web users who have to juggle HTML, JavaScript, Java, Perl, etc. to create today's exciting sites. Our hope is that the single environment provided by Curl will be an attractive alternative for web developers.
>
> (from http://www.cag.lcs.mit.edu/curl/)

1.2 A Curl Programming Environment

The Curl Language and its associated utilities are changing daily. At present, it has a browser, an inspector, a rudimentary equation editor, a slide presenter, and a host of other demo applications. Notably, it is missing any coding environment. Almost all the code is currently written using Emacs (see below) and then compiled and run at the command line. Bugs must be handled "off-line" by editing files in a separate text editor and then recompiling them over and over again.

Although there is nothing wrong with this system, there are many obvious advantages to having an editing environment built right in to the language base. First, by writing the editor in Curl, the editor will have access to all kinds of run-time information, including Curl's extensive database of documentation. Also, the editor can be opened as needed for debugging. Run-time errors can be quickly edited without having to leave the Curl program. The editor itself is a validation of the programming language as it is a non-trivial application. Also, the creation of such an editor requires the establishment of a whole new set of useful classes that can be used in other projects and applications besides the editing

environment. Possible applications include: an email program, a news reader, an outline editor, an on-line documentation editor, an assignment composer for an electronic text-book and so on.

1.3 Emacs

Most of the Curl code written to date has been done in Emacs. Emacs (originally chosen as an abbreviation of Editor MACros) is the quintessential coding editor, describing itself as "the extensible, customizable, self-documenting real-time display editor."

Emacs is extraordinarily powerful and estensible, and despite the steep learning curve, provides long-time users with a comfortable, optimized environment. Many of the features in Emacs are set up in such a way to be user-motivated rather than automatic. A good example of this is tab-indention. When the user presses the tab key in an editing buffer, that line will be correctly indented according to the lines prior to it. This allows the user to write code without being distracted by intermittent formatting events.

Despite its many strengths, Emacs has several notable drawbacks. Emacs is cpu-intensive. It eats up a lot of processor cycles because of its complexity. It has a large memory footprint. It has a steep learning curve. Above all, it is certainly not within the gentle slope paradigm to which Curl aspires.

1.4 Bumble Bee Anectdote

There is a well-known urban myth about how a scientist proved that bumble bees cannot fly. The tale was popularized during the 1930's in the German technical universities by the students of the aerodynamicist Ludwig Prandtl at Göttingen. The story he tells is that a noted Swiss aerodynamicist, whom he does not name, was talking to a biologist at dinner. The biologist asked about the flight of bees and the Swiss gentleman did a back-of-the-napkin calculation. It showed simply that the size of the bumble bee's wings would not

support the relative volume and mass of the bumble bee's body. In fewer words, the bumble bee cannot fly. Unfortunately for the scientist, everyone knows that a bumble bee can fly, and with later developments in aerodynamic models and high-speed photography, it was shown that certain features of how the wings were moving allowed for this seeming miracle.

Similarly, with many modern programming languages, the editing facilities available are either too small and thus limited in usefulness (e.g.: vi, TeachText), or too bulky and result in overkill. (e.g.: emacs, Microsoft Word v.5.0) As a consequence, the developer either can get little work done or must spend a considerable amount of time getting up to speed in an essentially extraneous application. Often, there are only a small set of necessary features which are actually useful to the efficiency of the developers. By examing which features impact efficiency, we can maintain the simplicity of the editor while keeping it creative and powerful. At first glance, this may seem limiting for a language with so much power and size, but with careful application, it can allow the developer to truly fly. The lesson of the bumble bee is the simple one of quality over quantity.

1.5 Overview of Thesis

This thesis is divided into six major chapters, of which this ***introduction*** is the first.

The next three chapters present the design phase of the Curl Code Editor. Chapter Two is a discussion of the ***design philosophy*** which I chose in approaching the problem of writing a text editor. This section includes my research, including discussions with Richard Stallman, the originator of Emacs. Chapter Three is on ***high-level design***. It presents the conclusions I made from my research about what the important feature set is for a functional editor. The last of these three chapters, Chapter Four is on the ***low-level design***. It

describes the programming design architecture that I determined would be necessary to provide for the high-level feature set.

Chapter Five covers the ***implementation*** details that the primary foundation modules iterated through and goes on to cover the major features and how they were layered on top. Chapter Six presents closing comments and ***conclusions*** about the editor.

Chapter 2

Design Philosophy

2.1 Design is a Two-Pronged Process

My belief about design is that it is an innately two-pronged process. One must start at the highest and lowest level and effectively move in until the design itself is "squashed" out between the resulting specifications. With the editor, it was necessary to begin first with the high-level ideals that one would like to achieve while also considering the low-level classes that are necessary to provide even the most basic functionality. This idea of polarized design can be seen in even the most simple of engineering tasks--such as making a sandwich for lunch. This example will be used to illustrate what I mean.

High-level design is my way of expressing that it is important to have an overall picture before beginning. One must have a mind's eye view of the final project before trying to create it. [This is not to say that the final incarnation of the project will not be significantly different as elements are changed or substituted due to what is learned through the implementation process.] From the sandwich example, when you go make lunch, you decide what the final sandwich will be before you start. This means deciding on how big it should be, what ingredients are necessary and which are extraneous, and how much effort you are willing to put into it before just heading over to McBurgers. For the Curl Code Editor, this means figuring out what the final application will look like, how it will run, and what features will be needed to achieve a final product that is appealing and sharp.

Low-level design is often referred to as systems architecture or class hierarchy. Either way, it is simply a preparatory measure of what raw needs one has in order to accomplish some engineering goal. What do we have when we start? What else do we need? What would be nice? What is essential? Again, when you look in the refrigerator to make a

sandwich, what you make is often determined by what is there. Sometimes you have to go to the store to get what you want. In the case of the Curl Code Editor, this involved understanding what underlying classes are necessary to provide the high-level functionality. In this particular case, the cupboard was almost bare.

Before doing any design, however there is a more fundamental question which should be answered. ***Is there any reason to even build an editor?*** This thought led me to a discussion with Richard Stallman, the originator of Emacs.

2.2 Discussions with Richard Stallman, creator of Emacs

A lunch discussion with Richard Stallman, the eminent creator of Emacs, taught me a number of things, among which is that RMS (as he prefers to be called) really likes soup. Beyond this personal fact, the topic of our conversation ranged over various editor optimizations, the internals of emacs memory management, and philosophical views on the utility of text editors.

The main point that was raised in our talk is that traditionally, people will often use tools/technology for purposes for which they were not originally intended. This near tautology implies that when one designs a new tool, it should be as robust as possible in order that its usefulness can be stretched to the fullest. Always include provisions for customization and extension in order to extend the life and utility of your tool. For a text editor, this means that it could potentially be used for almost anything. RMS pointed out that if both the editor and the language are successful then people are going to begin using the editor inevitably for things it is not intended. It will become a mail editor and a news browser and so forth. Further, to not account for this situation is folly. His statements can be neatly summarized. ***Any editor that works with any text must first and foremost be a text editor, and a popular text editor will be used with all kinds of text.***

2.3 A correct tool for each job

RMS raises an insightful point, which is that the success of the language will necessitate a robust general-purpose editor and perhaps, vice versa. His point is a general validation for the need for a text editor within the Curl language. The focus of my thesis, however is for a ***code*** editor. A code editor is the first step on the road which RMS indicates. I am providing a tool for a specific group of people (programmers) to do a clearly defined task, not a multi-purpose *Uber*-application. But, the modules that comprise the editor have been designed so that they can be reused to enable future needed applications.

The Curl Code Editor is expressely design to work only with Curl code. It is my feeling that although Stallman is correct in his belief that technology is often used in ways for which it is not initially intended, there is nevertheless a correct tool for each job. By maintaining this one-to-one correspondence between problems and solutions, the ramp-up time needed to understand the solution (in this case that is the time needed to learn an editor) is significantly lower than it would be for a infinitely large application like Emacs. This comment is intended not as a slant to Emacs, but rather as a validation of the Curl Code Editor design tack.

With this in mind, we can also limit our overall conception of the editor to include only those functions/features which contribute to code efficiency, as opposed to the potentially infinite uses a more general editor maintains. Regardless, the classes which comprise the Curl Code Editor can be reused to create more general editors. As a result, the first short-term goal is to discretize the feature set. Once we have the feature set, we can determine what minimal underlying classes and structures are necessary to support them. Finally, with these base classes formulated, the internal structure of the application, that is the code specifications are clearly determinable.

Chapter 3

High-Level Design

3.1 What programmers need from a code editor

Programmers, experienced and otherwise are most often stymied by the syntax of a language. They may know the semantics of what the program should do (i.e.: the meaning of the code) but then get caught up in the syntactic web. The motivation behind an editor which is intended for working with a specific language is to improve the relationship between the programmer and their code. With this in mind, we set forth to make coding:

faster -- By faster, I mean that the programmer should not be hindered by the interface to the computer. Requiring the typist to use a mouse for movement or selection, limiting editing functionality, not being able to search, not having macros will all hinder a programmer's ablility to work.

simpler -- Programming is simplest when the user focuses the majority of their time on actually writing code and less time searching through documentation, source trees, and libraries; debugging easily avoidable errors; or trying to recollect strange syntax for infrequently used code forms.

more accurate -- By accuracy, I mean accuracy in terms of format, spelling, variable usage, etc. in one's code. A forgiving editor will enable a programmer to work faster if they can avoid obvious compile time errors. Accuracy really is a measure of semantic exactness, the correlation between what the programmer wants the program to do (semantics) and what the programmer enters into the editor (syntax.)

more readable -- Code readability often has a lot to do with a particular programmer's "writing" style. Nevertheless, there are many basic services that an editor can provide which may aid in reading code. These include effective use of color to categorize various

components of a file, consistent line indention for grouping text, and the use of a clear, mono-space font. Readability is even more important when inspecting or editing another programmer's code.

efficient -- This is the internal efficiency of the editor. At a basic level, this affects the previous four interface issues. We must be able to handle access to many files, to large files, editing of these files, intense memory requirements and still maintain the above goals. The efficiency of the editor at this internal level is really a question of low level memory management and less an interface consideration. It is mentioned here for completeness but does not directly affect the interface.

3.2 Correlative Features

The first four areas mentioned above [speed, simplicity, accuracy, and readability] yield our set of desirable features. These features are described below with an indication as to how they help the Curl programmer.

Flexible Editing Commands: The editing commands comprise a set of the basic commands familiar to most editors. These are coupled with access to the keybase mappings itself which allows for a fully customizable control interface.

Clickable Key Words: Especially when reading another person's work, one runs across code whose purpose is either ambiguous, evasive or simply unknown. Like the inspector option of the Curl browser, the editor supports right-button clickable text which brings up correlating documentation. This documentation may be in the form of a tutorial, an inspector, another editor to code, etc.

Curly Matching: Curl derives its name from the curly braces used to delineate code segments, etc. By hilighting the matching curly brace when closing them, accuracy and sanity is preserved.

Smart Tab Indentation: Indentation based on the scope of a code segment improves readability of code. As previously mentioned, indention should be user driven as opposed to automatic.

Colorization: Explicit coloring of the text based on its syntactic type has been found to aid in not only the general readability of the code, but accuracy.

Debugger Hooks: In order to make general interaction with the Curl language as simple as possible, the editor supports hooks which allow it to be called from other applications. Thus, debuggers, profilers, inspectors, etc. can open files for editing dynamically.

Code Fragments/Ghost Templates:* Templates allow users to have a clear indication of the semantics of their programming language. By filling in place holders for non-existent code, the accuracy and readability of the code is improved. (e.g.: {if {> a var2] cons alt] where the underlined words are just placesholders.) Another case where this is particularly useful to the users is when they are learning or accessing functions with which they are not completely familiar. The editor simplifies the situation by indicating the code flow. A good example of this in Curl is the distinction between the `if` and `when` statements. While `if` has an alternate clause, `when` does not. This can destroy the semantics of a user's code if they try to use them interchangeably.

Tutorial Package:* The tutorial package is an estensible set of documentation which includes instruction on how to use the editor itself. By providing a framework for other tutorials, however, we provide the potential for simplifying the learning process for the language as well.

*features which were later removed from design.

Chapter 4

Low-Level Design

4.1 The Three Main Low-Level Classes

With the given feature set, we begin to reason about the internal structure of the editor. Although there are design considerations which one cannot foresee and which become apparent only during the implementation process, these major modules constitute a complete functional set. The three modules are defined below and described in detail in this chapter. In addition, the major features are described in detail as they fit into these classes.

CTETextBuffer -- Memory Management Module: The CTETextBuffer class handles the maintenance of the ASCII text information from the files. It is not a visual classes but is rather linked to Views and other Graphic classes in order to provide an efficient abstraction layer. This also allows the module to be a stand-alone set unto itself, useful for other projects and applications. It incorporates file access, buffers and general editing; however, the most important concept the CTETextBuffer class encapsulates is the editing algorithm referred to as ***the Bubble*** (termed in the Emacs code and documentation as the gap.) The Bubble allows very large files to be efficiently changed around without demolishing memory. This is particularly important, because one will often have several or more large code files open simultaneously.

CTEInputFilter -- Keymap and Input Module: The CTEInputFilter class acts as the filter which catches all input from the user and routes it to the appropriate editing action. (Note that although this includes Mouse events, they are handled differently because they must get information through the graphic classes.) The keymap filter must provide access to both the multi-layered, customizable input command set and the updating functions

which are called at every keystroke. The CTEInputFilter directly associates with a CTE-TextBuffer which maintains the data being modified.

CTETextDisplay -- Graphics and Text Display Module: The visual classes for the CTE are basic. They only need display fixed-width font text with several font-sizes. Although there was initially considerable discussion about supporting many fonts, images, etc., it makes sense to not support any more visual attributes than are necessary for best viewing code. In particular, the CTETextDisplay does automatic text formatting of any code placed inside of it, as well as providing colorization and graphics. The CTETextDisplay inherits from the CTEInputFilter class in order to pass all of its input events up for parsing and handling. [This way, the CTETextDisplay can opt to have memory handled by its parent class, the CTEInputFilter, or to micromanage it, which is what I chose to do in the final implementation. As always, final implementations can be quite different than planned designs.]

4.2 The Text Buffer Class

The CTETextBuffer Class is the memory management component of the editor. It does not specificially provide extensive features, but more importantly it provides the solid foundation necessary for the basic functionality of editing a file efficiently. This class must maintain rapid and consistent access to the file being edited, while providing a simple interface for the visual classes to access.

The fundamental problem with code files is that they are often large. In addition, code files are rarely accessed individually, but are instead opened in groups. This possibility is a function of the nature of programming and requires that memory is handled in such a way as to not overload a machine's resources in this common case.

In order to handle memory access in the editor the ***right*** way, I chose to design a cache-based design. This cache incorporates two design components termed Segments and the Bubble[1].

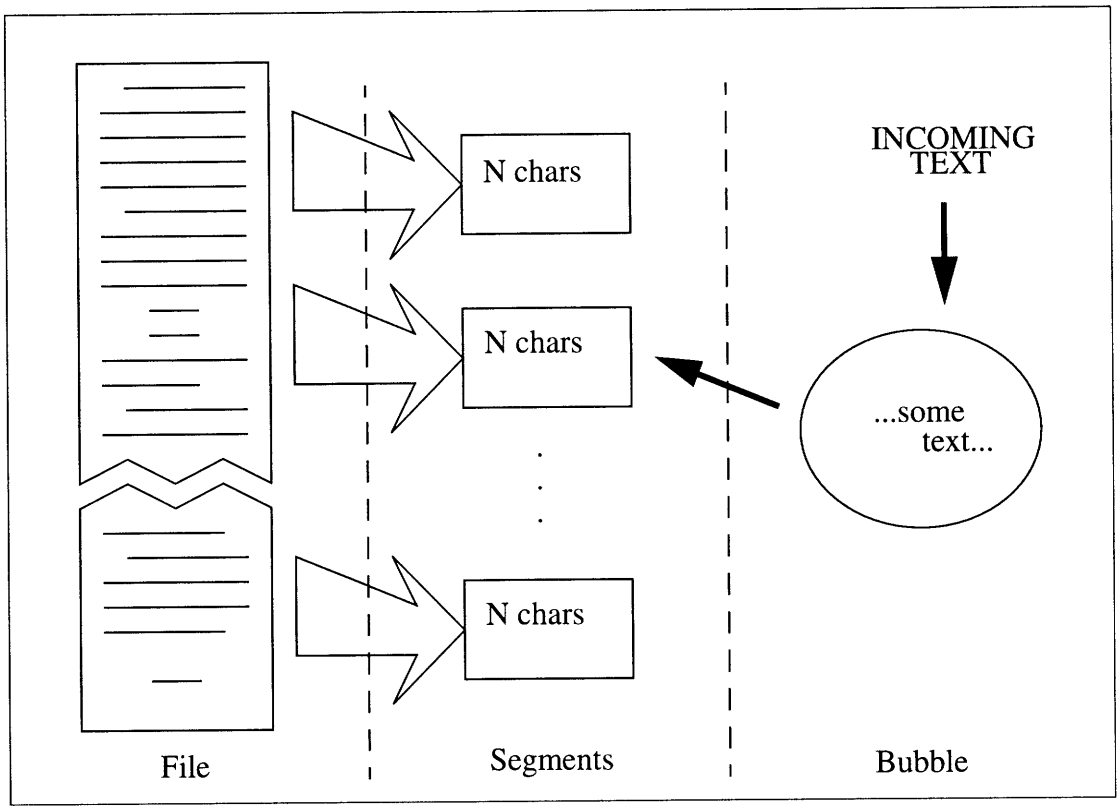


Figure 4.1: The Text Buffer has a caching system comprised of a list of Segments and a Bubble.

Text/characters written to the Text Buffer go initially to the "Bubble." The purpose of the Bubble is to prevent the necessity of memory updates on every keystroke and to allow sequential text input to be written to memory in chunks. These chunks, however can only be so large. The Bubble is a small cache which fills up quickly and, when full "pops." When the Bubble pops, its contents are moved to the currently active Segment.

Segments are larger arrays of characters (text) associated with some ***segment*** of the file being accessed. The Text Buffer maintains a finite list of Segments. Thus, only a por-

1. Emacs has a correlative component to the Bubble termed the Gap.

tion of the file (unless the file is relatively small) is kept in memory at any given time. The Text Buffer begins by slurping up the first N characters in the file and putting them in one segment. If the user moves to some portion of the file which is not currently in memory (in one of the segments), the TextBuffer will slurp up the surrounding portion of that file and put it in a different segment which does not overlap any of the current segments. Because there are only a finite total number of segments, one of the segments must be periodically thrown out in order to make space for a new segment being created. This happens by random selection. We could implement a LRU (Least Recently Used) replacement strategy, but the overhead [in practice] is not worthwhile.[1] In this way, the TextBuffer is able to edit an infinitely large file with efficiency. because edits are generally clustered. The Segments encapsulate this clustering.

It turns out that the interaction between the file, Segments and Bubble is non-trivial. As such, this balancing act needs to be isolated from the rest of the program while still providing some extra functionality which is not necessarily part of the memory management (e.g.: searching and replace.) Fundamentally, the Text Buffer provides an interface for characters or text [character arrays] to be inserted, deleted, and movement to occur. Movement in the buffer will always pop the Bubble should it be non-empty. In addition, for complicated movement (especially searches which often will access locations in the file not currently in memory) there needs to be a way to convey location. In order to encapsulate the idea of location in an infinite length file, the Point Class was created.

The Point Class provides an optimized representation for an infinite number which represents the ***point*** in the file that we are editing. Curl does not currently provide large number support, so this class was a necessary creation. The internal structure is discussed in the implementation section. The Point Class interface provides a complete set of opti-

1. Ward. Computational Structures. pp. 480-486.

mized arithmetic operators so that various modules of the Curl Code Editor can use the Point Class to communicate.

4.3 The Input Filter Class

The CTEInputFilter Class directly inherits from the EventHandler class and provides the *flexible editing commands* which are one of the main features of the Curl Code Editor. There are several different ways to approach this portion of the design, however first we should discuss what is a key sequence.

"A ***key sequence*** is a sequence of input events that are meaningful as a unit--as 'a single command.'"[1] The use of multiple keystrokes to access a single command is a common paradigm for Emacs users (i.e.: C-x C-f is a reasonable command to open a new file.) Consequently the Curl Code Editor should support this style of commands. However, many users [myself included] find this approach slow, non-intuitive and ultimately strenuous on the user's hands. With this is mind, we should also support rapid single-key based commands as well as multiple editing modes, as in `vi`[2]. In total, the Input Filter must be able to handle incoming events and call functions within the Curl Code Editor based upon its current state, as determined by previous key sequences. With this in mind, we can consider the creation of the Input Filter Class.

One of the exciting things about writing the Curl Code Editor in Curl is that we have access to our own code. Readers familiar with the LISP language will recognize the LISP initiated concept of "the interpretation of code as data and vice-versa." [HJF: LISP] This means that the Curl Code Editor can edit its own internals. This gives us an interesting option for optimizing key sequence changes in the input filter. Essentially, once we have a

---

1. from the Emacs Info page. C-h i: Emacs::Keys::
2. `vi` is a screen-oriented display editor based on `ex` and is common to most Unix systems.

coherent data structure to encapsulate key sequences and their corresponding functions, we can place this structure in a file and allow the user to edit it manually.

Also in the "grand LISP tradition", we will use a list data structure to store the functions and their key sequences. This is nice for several reasons. First, list data structures are clearly readable. Pairs of items can be put in any order with clear delimiters, curly-brackets in this case separating the groupings. Also, lists work well with non-uniform values. In this case, each key sequence can be a different length. A list data structure allows us to simlpy `cons` [construct] a longer list for more complicated key sequences. Lastly, list data structures are easy to operate on. For example, we can optimize the key sequence data structure so that functions that have the same first key stroke can be grouped together easily. Below is a sample list structure for key sequences. Each entry is a pair. The left half is a textual representation of a key sequence ("C-x" represents control and the x key together.) The right half is a symbolic representation of the corresponding function. So, in this example, to quit, press control-x and then control-c.

```
| key sequence - function symbol
{{"C-x" {"C-k" cte-edit-key-sequences}
        {"C-f" cte-open-file}
        {"C-c" cte-quit-editor}
        {"C-s" cte-save-file}}
{"~" cte-toggle-capitals} | single letter command
{"C-a" cte-goto-line-beginning}
{"C-e" cte-goto-line-end}
{"C-k" cte-delete-line}}
```

In sticking with the Curl gentle-slope philosophy, the key sequences which correspond with functions should have ***sane defaults***. The need to be able to change the run time environment though can be met simply by adding an additional function that does just this. Although the user will be encouraged to use the editor to edit the `cte-keySe-`

quences.curl file, temporary changes to the work space can be made in a simple and clever way. Oftentimes, a user may be uncertain as to how to represent a particular key or key sequence. It is easier, as well as considerably more intuitive, to refer to a function and then enter the desired key sequence to be associated with it. The application then has simply to cons up the key events it received and place them at the appropriate location in the key sequence data structure.

One final design consideration must be raised with regards to the Input Filter. It is undesirable to have the same key sequence associated with multiple functions. Although this cannot be avoided [due to the fact that the user now has access to the actual code of the editor], the application can handle this situation gracefully by executing only the first match to an entered key sequence. On the other hand, it is desirable to be able to associate multiple key sequences with the same function. The list-based keymap data structure handles this easily, but care must be taken that this feature is documented clearly and included in the function for doing temporary key sequence changes.

4.4 The Text Display Class

The Text Display Class must provide a number of critical functions to the Curl Code Editor. These include: Clickable Key Words, Parenthesis/Curly Matching, Smart-Tab Indention, and Colorization. Because these different areas are essentially orthogonal (mutually independent), I will discuss each need individually in terms of its relationship to the Text Display Class. The primary function of the Text Display Class, however, is displaying text.

Correctly displaying text can be non-trivial problem. There are a number of design choices we can make that will simplify this problem. The first choice is to display only fixed-width text. This effectively turns the screen into an NxM grid of characters. This simplifies the character display, the process of determining a specific word/character given

an (x,y) pair in screen coordinates, determining screen coordinates from a location in a file, and cursor movement.

Clickable Key Words: An integrated documentation database is an extremely useful tool. There are many times when a programmer is unclear about certain sections of code or details of a particular object which they are using. This occurs most frequencly when editing another person's code. At such times, quick access to reference information makes the process of writing and editing code easier, more enjoyable, and generally less painful. Curl provides, in its run-time environment, access to exactly such a resource.

The implementation of this part of the Text Display Class is straight-forward. Received mouse-click events are processed to determine the word which was accessed and then, an appropriate call to the documentation resource is made. If a word is not found, a beep sounds and function returns to normal. If the word is located, the appropriate documentation dialogue is displayed. This implementation is clean, because there is no information necessary beyond what is directly displayed on the screen.

This feature has one drawback. It limits the use of the mouse in text selection. Although I personally prefer to touch the mouse as little as possible [and the design of the editor reflects this preference] many feel it is useful. On the other hand, learning a language is incredible pleasurable when you can simple type out those commands which you are uncertain about and then click them in order with documentation popping up to aid you. Curl is an object-oriented language, and each class has its own set of specialized methods. When reading another persons code or designing new modules, the ability to bring up class method references is remarkably convenient. [Actually, as the editor was being written, this was one of the first functions to be implemented, because it so greatly aided the rest of the coding.]

Curly Matching: As already mentioned, Curl is a LISP-ish language. By this, it is meant that it uses a prefix notation and a singular delimiter character (the curly braces from which the name of the language derives.) These characteristics result in code that can be difficult to read and debug, often because of misplaced delimiters. It has become common practice with such languages to provide for matching the opening and closing occurences of such delimiters--curly brackets.

The most common way of matching, or balancing, is to move the cursor momentarily to the site of the previous match. Not only is this slow and inefficient, it can be disruptive to the user. The Curl Code Editor takes a new approach. Since we are constructing the Text Display at a low level, we have direct access to the draw method. In order to indicate curly matching, a colored line can be drawn underneath the current text to provide the programmer with the matching information. This type of visualization is fast, intuitive, and less disruptive than earlier methods.

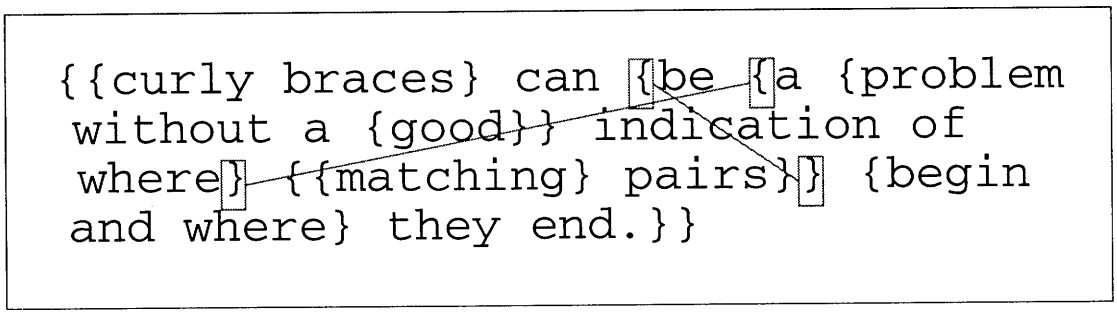


Figure 4.2: A new way to visualize curly bracket balancing; two examples are shown.

Smart Tab Indention: Another accepted practice when working with LISP-ish code is to indent each line in such a way that each line begins at an offset based on the information in that line. This process, taken in combination with the Colorization feature described below, is often referred to as grinding, specifically "To prettify hardcopy of code, especially LISP code, by reindenting lines, printing keywords and comments in distinct fonts (if available), etc." [HJF: grind] Grinding greatly improves the readability of code.

The algorithm we use for tab indention in the Text Display Class is a simple recursive algorithm for each pair of lines, the second line in the pair being indented with respect to the first. First, check how many leading spaces there are on the first line. Add to this two spaces for every unbalanced open curly bracket.

```
first line        ──►  {define {function i:int}
1 open curly      ────►{if {> i 0}
2 open curlies    ──────►{output i " is positive"}
2 open curlies    ──────►{output i " is not positive"}
2 open curlies    ──────►}
1 open curly      ────►}
```

Figure 4.3: An example of correctly indented Curl Code (notice that each line is indented depending on how many unbalanced open curlies there are)

This line-by-line algorithm turns out to be the same as counting up the number of unbalanced open curlies (versus a balanced pair of an open curly and close curly) and indenting each line accordingly. To do this as a batch job for the entire file, simply remove all leading whitespace from the first line of the file and then begin counting curlies. For every open curly, add one to the count. For every close curly, subtract one. Indent each line by twice the count. This algorithm is restated in the following pseudo-code.

- **init:** goto first line; delete leading whitespace; set **count** to 0
- **foreach** line
    - indent line (2 * **count**) spaces
    - **foreach** (open curly) in line, increment **count**
    - **foreach** (close curly) in line, decrement **count**

Colorization: The Text Display Class has a flexible and straightforward colorization utility. All code is automatically hilighted in the display. The user has the ability to create their own word sets and corresponding colors or to use those provided. The Text Display class handles two major areas of formatting. First, comments are set to green, which when on a white background is readable, but not visually distracting. Secondly, Reserved words are individually hilighted according to the user's preference.

Like the keyword sequences, the Curl Code Editor uses a list structure to maintain its reserved word colorization scheme. The default color for reserve words is red. Each word in the list of reserved words is located in the file and hilighted. Words can be flanked on each side by any of the following characters: { (open curly), } (close curly), ^j (linefeed), ^m (carriage return), (tab), (space), " (double quotes), and | (comment indicators). This collection distinguishes the set of all characters which occur syntactically next to key words but which are not actually a part of them. Here is an example of hilighted code.

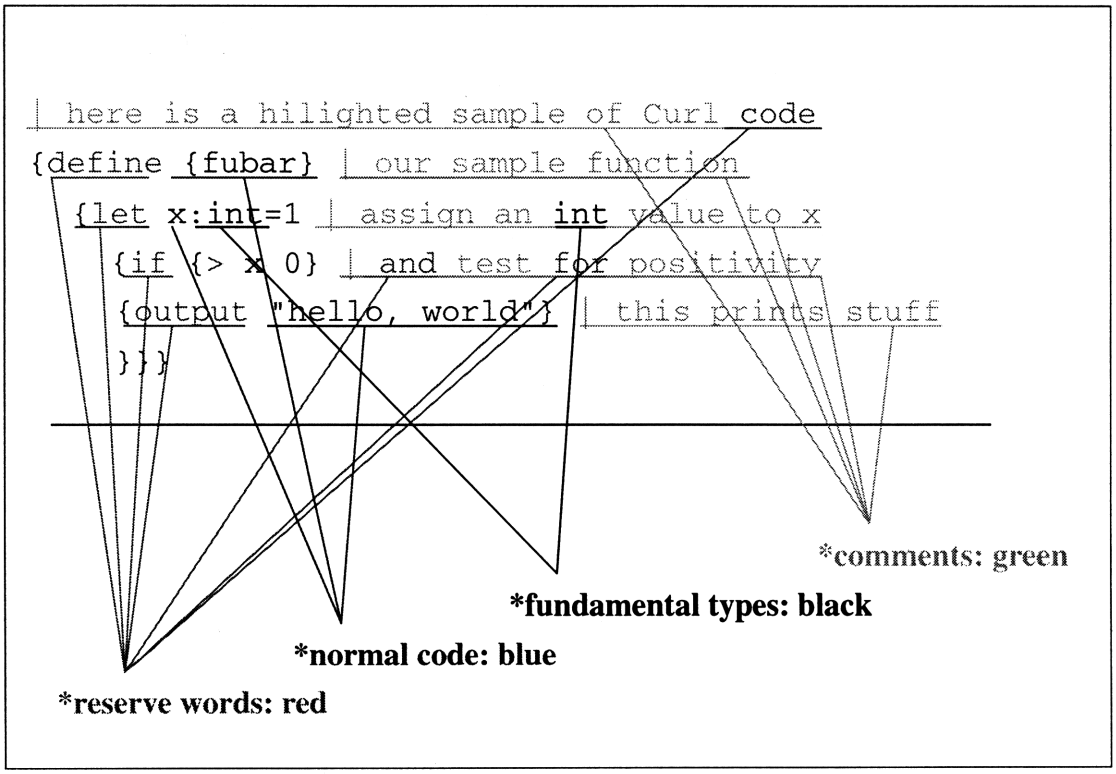


Figure 4.4: Hilighted code sample

As can be seen in the figure, the list structure provides for sophisticated color schemes. The default color for code text is blue; comments show up in green. In this example, which corresponds to the default editor behavior, reserved words are hilighted in red and basic types [`int, int8, int16, char, float, bit, any, void, text, vector`] appear in black. The editor maintains an internal list which contains sets of reserve words prefaced by a color symbol. Such a list looks like the following.

```
'{ | begin list of reserved word
black  | for base types
    {
      "int" "int8" "int16"
      "char"
      "float"
      "bit"
      "any"
      "void"
      "text"
      "vector"
      }
red  | for reserve words
    {
      "define"
      "let"
      "if"
...
```

Any words that the user finds relevant can be added easily to this list in addition to alternative colors. Despite being more visually appealing, grinding is of high value to programmers. By de-emphasizing comments, they can be ignored when writing code and hilighted when reading it. Thus, the programmer knows where to look and when. Having hilighted reserved words proves even more useful. Mistyped reserve words are not a serious problem as they are easily caught by the debugger, but having them change colors provides an instant visual cue that decreases compile time bugs and reassures the programmer

about the correctness of his or her code. In addition, highlighted reserve words result in code that is structurally clearer, because the reserve words are able to serve as both visual and functional anchor points for the lingual semantics of the written program.

4.5 Miscellaneous Design Considerations

Although most of the major code editing functionality is incorporated in the above classes, there are a few more design considerations not entirely supported:

File System Accessors: The CTETextBuffer will need to have access to the file system. This should be able to be handled simply through file input/output ports.

Code Fragment/Ghost Template Support:* Templates for functions need to be handled carefully so as to be helpful, rather than obnoxious. Since they are essentially just ghosts (i.e.: they are not ever actually present in the buffer but just visual cues) they do not affect any of the other classes.

Tutorial Package Support:* The tutorial is really just a file provided with the editor. There should be two versions, one that is intended to be viewed by the Curl Code Editor and on to be viewed as a Curl Document (i.e.: in a Curl Browser.)

*features which were later removed from design.

Chapter 5

Implementation

5.1 Implementation Phases

The implementation of the Curl Code Editor came in four major phases. Classes were implemented from low-level functionality to high-level needs and finally to feature addition. As the implementation progressed there were several major changes in the design. One of these was an eventual deactivation of the memory handling facilities. (see below) Another was the removal of some features.

Two features, the Tutorial Package, and the Ghost Templates were removed from the design during the implementation phase. The Tutorial Package is not a demonstration of the strengths of the Curl language. It is a demonstration of a markup language and is essentially just a convenience. Further, the language changes so rapidly that it would unfortunately become quickly outdated. For aide in using the Curl Code Editor, there is of course a help file (see appendices.) The Ghost Template feature was removed because it resides on the opposite side of the spectrum from the Tutorial Package. It is too difficult. The Ghost Template feature requires specialized case-by-case parsing of code with sophisticated semantic interpretation. This greatly increases the code base and slows down the utility of the editor. With the language constantly changing and the initial absence of an operational semantic for the Curl language, the feature had to be abandoned. It would have been nice, but the trade off to having a lean, fast editor is worthwhile.

5.2 Low-level memory handling implementation: CTETextBuffer

The CTETextBuffer encapsulates the memory handling component of the Curl Code Editor. In doing so, it is divided into two parts. The first the interface or accessors which are used to interract with the CTETextBuffer class in order to maintain a consistent represen-

tation for the text. The second is the internal portion which actually manages memory. As described in the design section, the internal memory representation for the CTETextBuffer maintains three layers of memory, the bubble, the segments and the file.

The Interface: The CTETextbuffer has a simple, but complete set of accessor functions to create, edit and move within the memory structure. These functions, listed below, are mostly self-explanatory. In cases where multiple accessors are presented, it is less for convenience and more for optimization. The CTEPoint class is a class which represents a location in a potentially infinite length file.

```
{init f:text}
{setPoint n:CTEPoint}:CTEPoint
{getPoint}:CTEPoint
{writeChar l:char}:void
{writeText t:text}:int
{writeTextWLen t:text len:int}:void
{read n:int}:text
{delete n:any}:void
{backspaceCharDelete}:void
{find pattern:text forward?:bool=true movePoint?:bool=false}:CTEPoint
```

The Bubble: The bubble is represented by a `text` array and a length indicator which is a simple `int`. When characters are written to the CTETextBuffer, they go directly into the bubble (unless it is a section of text which is larger than the maximum bubble size in which case it is directed to the active segment.) When the bubble is full, or a movement event occurs, the bubble pops and its output is flushed to the current active segment at the position in the segment.

The Segments: The segments are represented internally as a list. The first element of this list is an array of segments. The next two values indicate which segment is the active segment and the total number of segments currently in memory. If a maximum number of segments is reached, then segments are removed according to a random replacement strat-

egy. The segment array is comprised of some number of vectors. The vectors contain the text array, its size, a dirty bit, and start and end indicators for where in the overal file each segment is positioned.

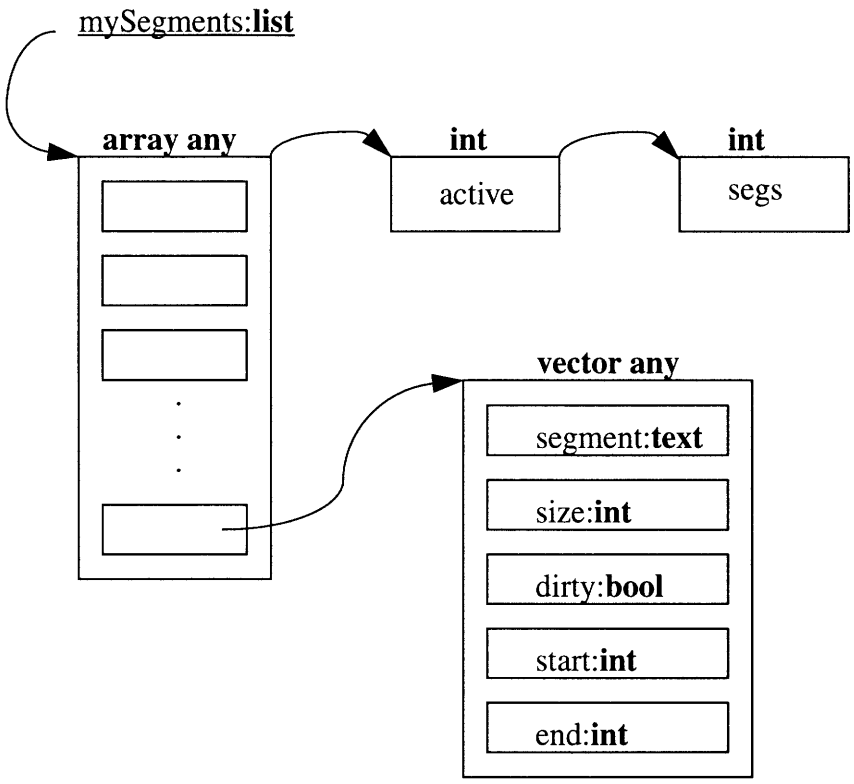


Figure 5.1: The Segment Data Structure

Compromise: The internal memory system set up here was created because of an absence of a robust low level memory management module to handle text arrays for the Curl language at the start of this thesis. Later, I found that there was in fact a low level StringBuffer class of which I was not aware. It was not exactly what was needed but could be adapted to work. I considered using it, but opted to proceed with the CTETextBuffer class in the hope that it would be efficient and useful. Later though, when I began work on the CTETextDisplay class, I found that the internal design of the windowing system required that a copy of the entire text array was kept in the display class. In order to avoid

having two redundant copies of the text around, I disabled the CTETextBuffer class in order to continue work on the CTETextDisplay class. The CTETextBuffer is not directly accessed from the CTETextDisplay class, so this does not affect the visual system. The CTETextBuffer is accessed instead through methods in the parent class of the CTETextDisplay, which is the CTEInputFilter. In order to disable the memory management, all calls to the super methods in the CTETextDisplay class were commented out or not introduced at all once feature implementation was under way.

This was my compromise to the internal functioning of the Curl windowing sytsem, the `boxes` package. The CTETextBuffer specification is still viable and to use it, the programmer has simply to inherit from the CTEInputFilter class (described next) and handle memory through the super methods in that class.

5.3 User Input: CTEInputFilter

The CTEInputFilter serves two purposes. First, it acts as an entry point to access the intrenal representation of the text. Second and most important, it provides the framework that makes flexible editing commands, in the form of key sequences, possible. The CTEInputFilter accesses the key sequences list structure which resides in the `cte-keysequences.curl` file and can be edited directly, providing complete control over the editor's functions.

The CTEInputFilter class inherits from the EventHandler class and overrides the `key-press` method to get access to key strokes. Because of this, we cannot actually get the key stroke but only the event associated with each key stroke. Each event is the `int` associated with an ASCII value for the keystroke. This is a bit complicated, because different keys can map to the ***same*** ASCII value. This is especially true with control characters. For example, control-a and control-A, although different key strokes, are

indistinguishable as key events through the event method used by the Curl language. However, the user may wish to represent a key stroke in the key sequences list as either `"C-a"` or `"C-A"` (both valid representations for that key event.) In order to remedy this situation, there is a *translate* method implemented as a large switch statement that returns the correct key event for each valid string representation used in the key sequences file. Rather than comparing an ASCII value to each event, the events are compared with values generated from the key sequences file by the *translate* function.

Key Sequences: The key sequences are kept in the `cte-keysequences.curl` file which is comprised of a large list, the format of which was described in the design section. The CTEInputFilter class takes each element in this list, translates each string representation (the first part of each element of the list), and compares it to the received key sequence. When it finds a match, it checks to see if the second part of that element is a function symbol. If so, that symbol is placed in a method variable (`mySymbol`) and everything is reset. If the second element is a list, then that list is placed in a method variable (`myKSList`) and searched the next time a key is entered. In this way, the list that is searched can be setup to allow multi-level commands such as C-x C-c. The pseudo-code for the above procedure is given here.

- **init:** set **mySymbol** to 'null; set **myKSList** to the key sequence list
- **foreach** event **k**
- set **i** to **myKSList**
- **while** (**i** is not the empty list)
    - **if** (**k** is the same as *translate* {car {car **i**}}) [the string representation in the list **i**]
    - **then**
        - **if** ({car {cdr {car **i**}}} is a symbol) [we have found a function symbol]
        - **then** set **mySymbol** to {car {cdr {car **i**}}}; reset **myKSList**
        - **else** set **mySymbol** to 'null; set **myKSList** {cdr {car **i**}} [multi-level list]
- **else** set **i** to {cdr **i**}

For completeness, the key sequence list is augmented by another list in the `cte-keybase.curl` file. This is a list of all the possible key strokes that can be entered. This list merely containts the correlative ASCII value for each key sequence (see appendices.)

The CTEInputFilter does not actually evaluate the symbol that it finds through the above procedure until later. A method called `doFunction` is provided for this purpose. The function is not implemented in the CTEInputFilter class itself, but is intended to be overridden in the child class. The `doFunction` method accesses the `mySymbol` class variable once it has been set by the above procedure. In this way, the `doFunction` command can be used to implement a command history, undo capability or repeat last function calls. Of these, the last is actually implemented in the Curl Code Editor.

When the user chooses to change key bindings for functions, he or she simply edits the `cte-keysequences.curl` file. There is a binding [defaults to `C-x C-k`] provided to quickly access this file and open it in the Curl Code Editor for adjustments. It was my hope to allow changes to this file to be dynamically loaded into the running Curl Code Editor after they were made. Although the Curl language does have access to its own run-time environment, this feature could not be implemented. This is unfortunate, because it requires the user to shut down the editor and restart it each time a new set of key bindings is introduced. On the other hand, it is still superior to giving the user direct access to the CTEInputFilter's `myKSList` class variable. This alternative, though tempting, creates the possibility that the user could strand themself in their own work environment at a particularly inconvenient time (for example in the middle of a large coding project.)

5.4 Displaying the text: CTETextDisplay

The CTETextDisplay class provides a visualization of the text being edited. This means that it is responsible for providing not only a clear presentation of that text but also one

that is consistent with the internal representation. In other words, the cursor location must be consistent with the location at which text is being inserted and so forth. In addition, this class is responsible for providing supplemental contextual clues to the code such as color information, indents, curly matching, etc.

Basic Text Display: The CTETextBuffer takes advantage of the multiple-inheritance features of the Curl language by descending from both the CTEInputFilter class described above and the CodeText class. The CodeText class is a graphical class which can display basic text. It requires an internal representation of the text in the form of a text array. Although this causes redundancy with the memory management system, the trade off is to disable the more complex memory management and use the display properties of the CodeText parent class which are straightforward to use and visually appealing. The CTE-TextDisplay class is associated with a file upon initialization. The contents of this file are then assigned to the internal text array using the class method `set-text`. At this point, we have direct access to the textual representation through accessor functions. The draw method is heavily hacked to incorporate a cursor, sophisticated text coloring and more.

Key Events: When a key is pressed, the aptly named `key-press` method is called. The first thing that this method does is call its parent method in the CTEInputFilter class. As previously described, that method parses the event and determines the symbol which corresponds to the desired function. The `doFunction` method is then passed this function symbol. The `doFunction` method does two things. First, it searches through a list of symbols and calls the appropriate function when found. Then, it records that function call in the class variable `function` so that it can be recalled through the `repeat` command or perhaps entered into a history list. The only exception to this is if the command called is the `repeat` command. This exception avoids the infinite loop which would oth-

erwise result. After the `doFunction` calls the appropriate procedure, the `invalidate` method is called in order to update the display.

Basic Movement: Basic movement in the CTETextDisplay constitutes being able to move forward, backward, up and down. Once these basic building blocks of movement are implemented, all other movement functionality can be represented by these. Unfortunately, these four functions are simple in theory, but challenging to implement correctly.

First, let us discuss the functions for moving forward and backward in the file. The actual movement in the text array is done through the functions `goR` and `goL`. These functions take an optional argument which is the number of times to move in the indicated direction. The default is one character. These functions first test for boundary conditions to be certain that we are not going beyond the ends of the file. Then, they adjust the current position in the text array and call the appropriate cursor movement function (either `moveCursorRight` or `moveCursorLeft`.) The cursor is displayed on the screen according to the value of the class variables `cursorx` and `cursory`. Thus, cursor movement simply increments or decrements the `cursorx` value. When the boundary of a line is reached, either at the beginning or end, the `cursory` value is also adjusted correspondingly. To reiterate, these functions are simple in theory, but tricky to make work correctly.

The functions which move up and down in the file are even more delicate. They are implemented by calling the goR and goL methods to move to the correct location on the next or previous line. In order to locate where lines begin and end, two helper functions are used. These are the `chars-till-next-newline` and `chars-till-prev-newline` methods. Their names are reasonably self-explanatory. By knowing how many characters lie between the current location and the newline in either direction, we can know exactly where we are on the current line so that we can correctly position ourselves when we move. These functions are particularly tricky and if the reader is interested in

them, he or she is directed to the appendices. Suffice it to say that the `goLinedown` function, for example, has four distinct cases which must be handled. Basic Movement functions include the following.

{moveCursorRight}:void - move the cursor to the right one
{moveCursorLeft}:void - move the cursor to the left one
{goCharForward}:void - move forward one character
{goR i:int=1}:void - move forward i characters
{goCharBackward}:void - move backward one character
{goL i:int=1}:void - move backward i characters
{goLineDown}:void - move to the next line
{goLineUp}:void - move to the previous line
{chars-till-prev-newline}:int - get thenumber of characters to previous newline
{chars-till-next-newline}:int - get the number of characters till next newline

---

Basic Editing: Basic editing in the CTETextDisplay is simple. To insert a character, call the `insert-char` method (inherited from the CodeText class) and call the `moveCursorRight` method to maintain consistency. To backspace delete, call the `delete-char` method (inherited from the CodeText class) and call `moveCursorLeft`. To delete the character following the cursor, simply move our current location to the right and call `delete-char`. There is no need to move the cursor. These methods are listed below.

{insertChar k:char}:void - insert character k at current position
{delPrevChar}:void - delete character before current position
{delNextChar}:void - delete character after current position

The Clipboard (Cutting and Pasting): The CTETextDisplay supports basic cut and paste editing. The class variable `clipboard` is a text array which is used to store copied text. Currently, there are two methods which are used to demonstrate the cut and paste capabilities of the CTETextDisplay. These are `delLine` and `yank`. The `delLine` func-

tion determines the distance to the next newline. It then clips the text from the current position to that newline. In the case that the preceding command (stored in the function class variable) was also a `delLine` call, the text is appended to the current `clipboard` rather than replacing it. [This corresponds to the kill-line operation in Emacs.] The `yank` command simply inserts the current clipboard at the current position in the file. Together, these two functions are sufficient to demonstrate the CTETextDisplay's ability to handle cut and paste editing. Cut and Paste methods include the following.

{delLine}:void - delete from current position to next newline and copy to clipboard
{yank}:void - insert characters from clipboard

Searching: Because of the absence of `regexps` from the Curl language, we can only search for normal text strings. The `findPattern` method will put up a dialogue box inquiring for a new pattern. This is set to the class variable `pattern` and then the `nextPattern` method is called. The `nextPattern` and `prevPattern` methods move to the appropriate location in the buffer if the pattern is found. The `prevPattern` function is a little obtuse, because matching is done forward, not backward. To remedy this, the `prevPattern` function searches from the beginning of the buffer until a match is found and then continues searching until back at the current location. Then, provided at least one match was found, the cursor is moved to the position of the last match found. In all of the above cases, a bell sounds if the pattern is not found and the cursor remains at the current position. Search methods include the following.

{findPattern}:void - prompt for new pattern and call nextPattern
{nextPattern}:void - search forward from current position for pattern
{prevPattern}:void - search backward from current position for pattern

---

Miscellaneous Class Methods: A list of ofther methods used in the CTETextDisplay class is given below. This is not a complete list. The curious reader is referred to the appendices.

{gotoLineStart}:void - move current location to start of line
{gotoLineEnd}:void - move current location to end of line
{gotoFileStart}:void - move current location to start of buffer
{gotoFileEnd}:void - move current location to end of buffer
{toggleCapital}:void - do uppercase/lowercase switch
{repeat}:void - repeat last command
{help}:void - prompt to save and open help file in buffer
{editKeySequences}:void - prompt to save and open key sequences in buffer
{editReserveWords}:void - prompt to save and open reserve words in buffer
{indent}:void - indent this line; iterative tab indention (see below)
{indentAll}:void - indent entire file; batch tab indention (see below)
{fileSave}:void - save current buffer
{fileOpen}:void - prompt for new file to open
{quitEditor}:void - prompt to save and exit

5.5 Features

There are four major feature implementations that have not yet been discussed. They are Clickable Key Words, Curly Matching, Smart Tab Indentation and Colorization The code for these features is in the CTETextDisplay class, however they are discussed here separately.

Clickable Key Words: The clickable key words are accessed through the `pointer-press` method which is overridden from the EventHandler class. The `pointer-press` method first requests that the CTETextDisplay become the recipient of future key events. Then it looks to see if a key word was pressed. This is done by transforming the pointer event into the local frame and then passing it off to the `goto-word-doc` function which

opens a documentation browser if a key word exists at that coordinate. These functions are available globally as part of the Curl language documentation system.

Curly Matching: The Curly Matching feature operates directly at the level of the `draw` routine. It activates on the condition the our current position contains either an open or close curly bracket. When this is the case, the CTETextDisplay attempts to draw a box around each of curlies in the balanced pair with a line connecting them. One of these boxes is at the current cursor position. All that remains is to locate the balancing curly.

Depending on whether the curly we are starting on is an open curly or a close curly, search forward or backward respectively for the balancing curly. The balancing curly can be found by keeping a count (initialized to one) during the search and incrementing the count when a curly of the same type as you started on is found and decrementing the count on the opposite case. When the count reaches zero, the balancing curly is located. If the count never gets to zero, then we started on an unbalanced curly.

The search is done by first saving all the values associated with the current position. The use the `goL/goR` functions to incrementally search for the balancing curly. When found, draw the curly matching boxes and restore the position values. Similar to the basic movement methods for the CTETextDisplay class, this implementation is simple in theory, but chanllenging to get the implementation to work exactly right.

Smart Tab Indention: Tab Indention is implemented in two separate class methods: `indent` and `indentAll`. Each is a straightforward implementation of the algorithm discussed in the design section. Although there are a number of tricky places where the location shuffles around newlines and such, the procedures are exactly those described. It is particularly simple because it uses available methods such as `gotoLineEnd`, `gotoLineStart`, `gotoFileStart`, etc. Lastly, the two indention methods are separate. Although the `indentAll` method could iterate each line with the `indent` method, the

former is optimized to use internal methods instead. Thus, two distinct methods are provided.

Colorization: Like the Curly Matching feature, Colorization is implemented directly into the `draw` routine. At this stage, the draw routine must seem like a mess, but it is not. First, the cursor is drawn at its location on the screen. Next the curly brace references are put down. At this point, there is no text in the display. Text is added last and written over the cursor and other indicators for clarity.

The text is written one line at a time from top to bottom. Typically, an entire line is just dropped onto the canvas, but in order to do the sophisticated coloring desired in the CTE-TextDisplay class, an extra method is used. This method is the `draw-line-of-text` method and is where all of the colorization occurs. This procedure gets passed in a great deal of information including the vertical position of the line being drawn, the text to put down, and a swarm of increment values used to space the characters correctly. With this information, the `draw-line-of-text` command draws from left to right across the screen.

This left to right drawing motion is useful because it inherently solves the problem of how to handle comments. The default text color is set at the start of each line and drawing commences. But if at any point, the comment character | occurs, the color is reset to the comment text color and drawing continues. Thus, a line may start out being drawn in blue but then halfway through (corresponding to where the comment begins) be continued in green.

Next, the color is reset to the default for reserved words. The reserved words list is then brought up and iterated over. This list will contain either colors or lists of reserved words. When a color is encountered, the current color is set to be that color found. When lists of reserve words are found, each element in the list is compared with the line cur-

rently being drawn. If any of the reserved words are found on that line (surrounded by any of the valid characters mentioned in the design section and implemented in the `checkChar` function) those words are drawn ***over*** in the current set color. Thus, the entire line is drawn once and then in the cases where reserved words occur, those words are repainted again.

This list structure for reserved words has two side effects. The first is that a reserved word will ultimately be drawn according to its latest position in the `cte-reserved-words.curl` file in the case that it occurs more than one time. The second side effect is that the larger the number of reserved words, the slower the draw method will be, because the draw method must go through the entire list of reserved words once per line. The end result, however is fast enough for utility and looks good.

Chapter 6

Conclusion

6.1 The Curl Code Editor

All told, I am pleased with the final result of this thesis. The Curl Code Editor looks sharp. It is true to the vision of what I wanted it to be. This is not to say that there are not major differences between what was initially concieved of and the code base that now resides in the appendices of this document. However, the editor [which is not simply a result of isolated research and design but drew greatly on my personal experience coding in various environments] is pleasant to use and remarkably, really does make coding easier. All of the files herein were formatted with the editor. Also, the code became easier to write as the editor came on-line, especially with dynamic access to documentation, until finally I was editing the code for the editor in the editor itself.

6.2 The Curl Language

"Curl is a programming language." This summarizes most of my ***feelings*** about working with Curl. It gets the job done. Its good and bad points mostly balance. The ***feel*** of working with the language is that of a cross between LISP and Java. Curl has the prefix notation and list-based tendencies of LISP and the object-oriented web-centric trademarks of Java. Historically, these two languages have tended to rally support in opposing camps--that is to say that most people who like LISP, dislike Java, and vice versa, but Curl sits squarely in a moderate position between the two groups.

The Curl programming language has its complement of both good and bad points. I should prefix the discussion of these points with the statement that during the time I have worked with Curl, the language has been constantly changing and evolving. Even now, the most recent version of Curl is incompatible with the code in this document. Regardless,

some comments can be made about the intrinsic qualities of the language. The single biggest strength and weakness of Curl is that it runs in its own browser. Because it is intended to be used on the web, this is bad; it makes Curl code incompatible with what is already out there. On the other hand, the Curl browser allows the better aspects of the language to shine, including its platform-independence, access to its own run-time environment [a la LISP], and extensive documentation. These features are intrinsic to the language and bolster its utility greatly. In addition, the language achieves its ***gentle-slope*** philosophy in practice, and has nice language features including: first-class methods and variables, list structures, the `any` type, multiple inheritance, etc. Despite this, the syntaxis sometimes obscure and there remains no formal language specification. True to a work-in-progress, both the windowing interface and the overall class hierarchy are in constant flux. But in the end, Curl is a fully valid language for the future of web design, and it is my hope that the creation of this editor helps to advocate this position.

6.3 What I Learned

Aside from all the obvious things which are expounded upon in this document, there were a couple other things I gleaned from my thesis experience. The first is that an excessive amount of planning is just as bad as none at all. I found myself quite lost in some aspects of the design. I could not possibly know what to expect, because I forced myself to hold off on coding. In the end, this turned out to be the wrong approach. Secondly, I found that there is a great gap between language design theory and practice. Without going into detail, computer languages are like spoken languages, they evolve through use. Lastly, there is a great deal to be learned by reading source code.

6.4 Final Thoughts

Keep an open mind; adapt. I found that if I thought I knew where everything was going to go, I ended up lost and/or heading in the wrong direction. This was true in planning, designing, coding, managing time, et al.; revaluate; change. Listen.

Appendix A

Code

A.1 cte-constants.curl

```
|
| Curl Text Editor Files
| Jon Heiner
|
| Copyright (c) 1998
| Massachussetts Institute of Technology
| Laboratory for Computer Science
| All Rights Reserved.
|
| MIT MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY OF THIS
| SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
| IMPLIED WARRANTIES FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.
| MIT SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED AS A RESULT OF USING,
| MODIFYING OR DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
|=============================================================================

| File Description:
| This file constains in one place the values that are constant
| and necessary for the functioning of the curl text editor.

| Curl Text Editor Version
{define-constant cte-version:text="alpha"}

| name suffix information for the temporary file used to handle editing
| of files in the filesystem
{define-constant cte-kYesWeHaveNoFile:text="_!_YeSwEhAvEnOfIlEz2dAy ***
!_!"}

| name suffix information for the temporary file used to handle editing
| of files in the filesystem
{define-constant cte-kTempFileSuffix:text="~"}

| number of segments (of text) used by the textbuffer
| in order to maintain consistent file access.
{define-constant cte-kNumTBSegments:int=8}

| the default size of the segments used by the textbuffer when they are cre-
ated.
{define-constant cte-kTBSegmentSize:int=1024}

| the minimum size that segments can be before being flushed
{define-constant cte-kTBSegmentMin:int=128}

| the maximum size that segments can be before being flushed
{define-constant cte-kTBSegmentMax:int=2048}
```

```
| some index constants to be used w/i the segment data structure
{define-constant cte-kSegText:int=0}
{define-constant cte-kSegStart:int=1}
{define-constant cte-kSegEnd:int=2}
{define-constant cte-kSegSize:int=3}
{define-constant cte-kSegDirty:int=4}

| the default number of bits in the "infinite" CTEPoint reference
{define-constant cte-kPointBits:int=128}

| the size of the TextBuffer's bubble--our scratch pad
{define-constant cte-kBubbleSize:int=256}

| some constants used in the InputFilter for movement & searchs
{define-constant
  cte-kNewline:int=10
  cte-kReturn:int=13
  cte-kQuotes:int=34
  cte-kOpenCurly:int=123
  cte-kCommentChar:int=124
  cte-kCloseCurly:int=125
  cte-kEscape:int=129
  cte-kWordDelimiter:char=32 |space
  cte-kSpace:char=32} |space

| color constants
{define-constant
  cte-kCurlyFGColor:symbol='black
  cte-kCurlyBGColor:symbol='white}


{define-constant
  cte-kCursorFGColor:symbol='yellow
  cte-kCursorRimColor:symbol='black}

{define-constant
  cte-kGenericTextColor:symbol='blue
  cte-kCommentTextColor:symbol='green
  cte-kReserveWordTextColor:symbol='red}
```

A.2 cte-helpfile.curl

```
* Copyright (c) 1998  MIT LCS -- Jon Heiner *
[This file was written using the Curl Text Editor]

** Welcome to the Curl Text Editor
==================================

This editor is intended for use with Curl code. It
provides a number of helpful features to make working
with code fast, easy and efficient. This file is intended
```

```
to be opened in the Curl Text Editor. It provides an
overview of all the features the editor provides.

** Getting Help
===============

At anytime while you are working, you can bring up this file
by pressing C-x C-q.

** Movement
===========

The first thing to know is how to move around in a file.
Using the arrow keys, you can position the cursor. Also,
the standard Emacs movement key bindings are provided.

right:     C-f (Control & f together)   right-arrow-key
left :     C-b                          left-arrow-key
up   :     C-p                          up-arrow-key
down :     C-n                          down-arrow-key

Use the following key sequences to move through the
buffer more rapidly:

start-of-line: C-a
end-of-line:   C-e
start-of-file: C-/
end-of-file:   C-\

** Editing
==========

Use the following key sequences for editing:

delete-previous-character: backspace
delete-next-character:     C-h
delete-line:               C-k
toggle-capitals:           C-t
yank: (paste clipboard)    C-y

Also useful is the following key sequence which will repeat
the last command entered:
repeat:                    C-x .  (control-x, followed by period)

** Searching
============

To use the search function:

prompt-for-search-pattern:   C-s
search-forward-for-pattern:  C-]
search-backward-for-pattern: C-[

** Working with Files
```

```
======================

Use the following commands to save files, open new files, quit, etc.

save-file:  C-x C-s
open-file:  C-x C-f
quit:       C-x C-c

** Code Editing
===============

HILIGHTING: The Curl Text Editor is intended for editing Curl
code. Keywords are automatically hilighted. In order to change
the colors and reserved words, edit the cte-reservewords.curl
file or press C-x C-r

CLICKABLE KEY WORDS: The Curl Text Editor has direct access to
the Curl documentation database. If you are unsure how to use
a particular function, just click on that word. Or even more
useful, you can type out a sequence of commands that you want
to use or learn about, and by clicking on them, bring up the
correlative documentation either for tutorial purposes or as
reference while coding.

AUTOMATED TAB INDENTION: Curl code is much more readable if it
is neatly indented based on the scope of its curlies. By
ressing the TAB key, the line you are on will be correctly
indented based on the code above it. Repeatedly pressing TAB
will move you through a section of code, correctly indenting
(or "grinding") it. Further, by pressing C-z, you can grind
an entire file for readability.

** Changing Key Bindings
========================

all the conrols in this file can be rebound
for more comfortable or sophisticated editing
by the user. Multy-level commands such as C-x C-c
are perfectly fine. To edit the commands, chang
the cte-keysequences file or just press:  C-x C-k
```

A.3 cte-inputfilter.curl

```
| inputfilter.curl
|
| Curl Text Editor Files
| Jon Heiner
|
| Copyright (c) 1998
| Massachussetts Institute of Technology
| Laboratory for Computer Science
| All Rights Reserved.
```

```
|
| MIT MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY OF THIS
| SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
| IMPLIED WARRANTIES FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.
| MIT SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED AS A RESULT OF USING,
| MODIFYING OR DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
|=========================================================================

{include "cte-textbuffer.curl"}

{include "cte-keybase.curl"}
{include "cte-keysequences.curl"}

| File Description:
| This file contains the CTEInputFilter class definition.
|
| The input filter does exactly that: it filters input events from the
| user and then sends the appropriate commands to the text buffer and
| to the display.

{define-class CTEInputFilter{EventHandler}

|*** define class-fields ***|

   |the symbol that the input filter determines from key input
  protected mySymbol:symbol

   | internal Key Sequence List used to search the global list
  myKSList:list

   | this is how we access the internal representation of the text
  myTextBuffer:CTETextBuffer
  myFile:File

   | this is how we access the external way the text is viewed
  |myTextDisplay:CTETextDisplay
  |myTextDisplay:CodeText
  | nope...now the CTEInputFilter class is a parent of the display class
  | which handles the relationship to the buffer/memory.

|*** define class-methods ***|
  |{define {CTEInputFilter.init f:text t:CTETextDisplay}
    {define {CTEInputFilter.init f:text ...}
      {set self.myKSList {append gCTEKeySequences gCTEKeyBase}}
      |{output {append gCTEKeySequences gCTEKeyBase}}
      |{output "DEBUG: Init CTEInputFilter - pass "}
      }

  | override (interesting) EventHandler methods
    {define public {CTEInputFilter.key-press e:KeyEvent}:void
      |{self.myTextDisplay.insert-char e.key}
      |{output e.key " --> " {cast char e.key} "  (inputfilter)"}
      {self.searchKeySequences e.key}
      }
```

```
        {define protected {CTEInputFilter.insertChar key:char}:void
          |    {self.myTextBuffer.writeChar key}
          {output "DEBUG: CTEInputFilter.insertChar " key}
          }

      {define protected {CTEInputFilter.goLineUp}:void | towards beginning of
buffer
          |    {letrec p1:CTEPoint={self.myTextBuffer.getPoint}
            |    p2:CTEPoint={self.myTextBuffer.find cte-kReturn forward?=false
movePoint?=true}
            |    p3:CTEPoint={self.myTextBuffer.find cte-kReturn forward?=false
movePoint?=false}
            |    p4:CTEPoint={point- p1 p2}
            |      {if {point< p4 {point- p2 p3}}
              |  {self.myTextBuffer.setPoint {point+ p3 p4}}
              |  {self.myTextBuffer.setPoint {dec p2}}}}
          {output "DEBUG: CTEInputFilter.goLineUp"}
          }
        {define protected {CTEInputFilter.goLineDown}:void |towards end of buffer
          |    {letrec p1:CTEPoint={self.myTextBuffer.getPoint}
            |    p2:CTEPoint={self.myTextBuffer.find cte-kReturn forward?=true
movePoint?=true}
            |    p3:CTEPoint={self.myTextBuffer.find cte-kReturn forward?=true
movePoint?=false}
            |    p4:CTEPoint={point- p2 p1}
            |      {if {point< p4 {point- p3 p2}}
              |  {self.myTextBuffer.setPoint {point+ p3 p4}}
              |  {self.myTextBuffer.setPoint {inc p3}}}}
          {output "DEBUG: CTEInputFilter.goLineDown"}
          }
        {define protected {CTEInputFilter.goCharForward}:void
          |    {self.myTextBuffer.setPoint {inc self.myTextBuffer.getPoint}}
          {output "DEBUG: CTEInputFilter.goCharForward"}
          }
        {define protected {CTEInputFilter.goCharBackward}:void
          |    {self.myTextBuffer.setPoint {dec self.myTextBuffer.getPoint}}
          {output "DEBUG: CTEInputFilter.goCharBackward"}
          }
        {define protected {CTEInputFilter.gotoLineStart}:void
          {output "DEBUG: CTEInputFilter.gotolinestart"}
          }
        {define protected {CTEInputFilter.gotoLineEnd}:void
          {output "DEBUG: CTEInputFilter.gotolineend"}
          }
        {define protected {CTEInputFilter.gotoFileStart}:void
          {output "DEBUG: CTEInputFilter.gotofilestart"}
          }
        {define protected {CTEInputFilter.gotoFileEnd}:void
          {output "DEBUG: CTEInputFilter.gotofileend"}
          }
        {define protected {CTEInputFilter.gotoNextWord}:void
```

```
        |    {self.myTextBuffer.setPoint {inc {self.myTextBuffer.find cte-
kWordDelimiter}}}
        {output "DEBUG: CTEInputFilter.gotoNextWord"}
        }
      {define protected {CTEInputFilter.gotoPrevWord}:void
        |     {self.myTextBuffer.find cte-kWordDelimiter forward?=false move-
Point?=true}
        |    {self.myTextBuffer.setPoint
          |     {inc {self.myTextBuffer.find cte-kWordDelimiter forward?=false
movePoint?=false}}}
        {output "DEBUG: CTEInputFilter.gotoPrevWord"}
        }
      {define protected {CTEInputFilter.gotoNextCurly}:void
        |    {self.myTextBuffer.find cte-kOpenCurly}
        {output "DEBUG: CTEInputFilter.gotoNextCurly"}
        }
      {define protected {CTEInputFilter.gotoPrevCurly}:void
        |    {self.myTextBuffer.find cte-kCloseCurly}
        {output "DEBUG: CTEInputFilter.gotoPrevCurly"}
        }
      {define protected {CTEInputFilter.scrollPageForeward}:void | ** requires
get page size from myDisplay Buffer
        {output "DEBUG: CTEInputFilter.scrollPageForward"}
        }
      {define protected {CTEInputFilter.scrollPageBackward}:void
        {output "DEBUG: CTEInputFilter.scrollPageBackward"}
        }
      {define protected {CTEInputFilter.gotoEndOfWord}:void
        |    {self.myTextBuffer.setPoint {dec {self.myTextBuffer.find cte-
kWordDelimiter}}}
        {output "DEBUG: CTEInputFilter.gotoEndOfWord"}
        }
      {define protected {CTEInputFilter.gotoStartOfWord}:void
        |    {self.myTextBuffer.setPoint {inc {self.myTextBuffer.find cte-
kWordDelimiter forward?=false}}}
        {output "DEBUG: CTEInputFilter.gotoStartOfWord"}
        }
      {define protected {CTEInputFilter.gotoTopOfPage}:void | ** requires get
page size from myDisplay Buffer
        {output "DEBUG: CTEInputFilter.gotoTopOfPage"}
        }
      {define protected {CTEInputFilter.gotoBottomOfPage}:void
        {output "DEBUG: CTEInputFilter.gotoBottomOfPage"}
        }
      {define protected {CTEInputFilter.delPrevChar}:void
        |    {self.myTextBuffer.backspaceCharDelete}
        {output "DEBUG: CTEInputFilter.delPrevChar"}
        }
      {define protected {CTEInputFilter.delNextChar}:void
        |    {self.myTextBuffer.delete 1}
        {output "DEBUG: CTEInputFilter.delNextChar"}
        }
      {define protected {CTEInputFilter.delWord}:void
        {output "DEBUG: CTEInputFilter.delWord"}
```

```
  }
{define protected {CTEInputFilter.delLine}:void
  {output "DEBUG: CTEInputFilter.delLine"}
  }
{define protected {CTEInputFilter.delWithinCurlies}:void
  {output "DEBUG: CTEInputFilter.delWithinCurlies"}
  }
{define protected {CTEInputFilter.changeWord}:void
  {output "DEBUG: CTEInputFilter.changeWord"}
  }
{define protected {CTEInputFilter.changeLine}:void
  {output "DEBUG: CTEInputFilter.changeLine"}
  }
{define protected {CTEInputFilter.changeWithinCurlies}:void
  {output "DEBUG: CTEInputFilter.changeWithinCurlies"}
  }
{define protected {CTEInputFilter.changeChar}:void
  {output "DEBUG: CTEInputFilter.changeChar"}
  }
{define protected {CTEInputFilter.findPattern}:void
  {output "DEBUG: CTEInputFilter.findPattern"}
  }
{define protected {CTEInputFilter.nextPattern}:void
  {output "DEBUG: CTEInputFilter.nextPattern"}
  }
{define protected {CTEInputFilter.prevPattern}:void
  {output "DEBUG: CTEInputFilter.prevPattern"}
  }
{define protected {CTEInputFilter.repeat}:void
  {output "DEBUG: CTEInputFilter.repeat"}
  }
{define protected {CTEInputFilter.undo}:void
  {output "DEBUG: CTEInputFilter.undo"}
  }
{define protected {CTEInputFilter.beginMacro}:void
  {output "DEBUG: CTEInputFilter.beginMacro"}
  }
{define protected {CTEInputFilter.endMacro}:void
  {output "DEBUG: CTEInputFilter.endMacro"}
  }
{define protected {CTEInputFilter.doMacro}:void
  {output "DEBUG: CTEInputFilter.doMacro"}
  }
{define protected {CTEInputFilter.bindMacro}:void
  {output "DEBUG: CTEInputFilter.bindMacro"}
  }
{define protected {CTEInputFilter.toggleCapital}:void
  {output "DEBUG: CTEInputFilter.toggleCapital"}
  }
{define protected {CTEInputFilter.yank}:void
  {output "DEBUG: CTEInputFilter.yank"}
  }
{define protected {CTEInputFilter.toggleSpecialChars}:void
  {output "DEBUG: CTEInputFilter.toggleSpecialChars"}
```

```
        }
      {define protected {CTEInputFilter.fileSave}:void
        {output "DEBUG: CTEInputFilter.fileSave"}
        }
      {define protected {CTEInputFilter.fileSaveAs}:void
        {output "DEBUG: CTEInputFilter.fileSaveAs"}
        }
      {define protected {CTEInputFilter.fileOpen}:void
        {output "DEBUG: CTEInputFilter.fileOpen"}
        }
      {define protected {CTEInputFilter.quitEditor}:void
        {output "DEBUG: CTEInputFilter.quitEditor"}
        }

  |*** functions used to handle searches through the Key Sequences list struc-
ture ***
      {define protected {CTEInputFilter.doFunction f:symbol k:int}:void
        {output "DEBUG: CTEInputFilter.doFunction " f}
        }

      {define private {CTEInputFilter.searchKeySequences k:int}:void
        |{output "enter Search Key Sequences"}
        {let i:list=self.myKSList

        {while {not {eq? i '{}}}
            {if {= {self.translate {car {car i}}} k}
              {begin
               || we've found a match. if it is a symbol, we're done. o/w reas-
sign myKSList
                {if {eq? symbol {typeof {car {cdr {car i}}}}}
                  {begin
                    |{output "function found: " {car {cdr {car i}}}}
                    |the function is actually called by the child using:
                    |{self.doFunction {car {cdr {car i}}} k}
                    {set self.mySymbol {car {cdr {car i}}}}
                    {set self.myKSList {append gCTEKeySequences gCTEKeyBase}}
                    }
                  {begin
                    |{output "cdr car i: " {cdr {car i}}}
                    {set self.myKSList {cdr {car i}}}
                    {set self.mySymbol 'null}}}
                {return}}}
              {set i {cdr i}}
              }
            |{output "no match: " k}
            {set self.myKSList {append gCTEKeySequences gCTEKeyBase}}
            {set self.mySymbol 'null}
            }
          }

      {define private {CTEInputFilter.translate t:any}:int
          |sanity check
          {if {eq? {typeof t} text}
            {begin
```

```
            {cond
             {{= 1 {length t}} {return {cast int {aref t 0}}}}  | just one char-
acter
             {{= 3 {length t}}                                  | hopefully a C-
               || yes, this giant switch statement is ugly, but necessary.
               || it allows the user to enter C-? type expressions, which is a
nice feature.
               {cond
                 {{text-equal? "C-2" t} {return 0}}
                 {{text-equal? "C-2" t} {return 0}}
                 {{text-equal? "C-@" t} {return 0}}
                 {{text-equal? "C-`" t} {return 0}}
                 {{text-equal? "C-A" t} {return 1}}
                 {{text-equal? "C-a" t} {return 1}}
                 {{text-equal? "C-B" t} {return 2}}
                 {{text-equal? "C-b" t} {return 2}}
                 {{text-equal? "C-C" t} {return 3}}
                 {{text-equal? "C-c" t} {return 3}}
                 {{text-equal? "C-D" t} {return 4}}
                 {{text-equal? "C-d" t} {return 4}}
                 {{text-equal? "C-E" t} {return 5}}
                 {{text-equal? "C-e" t} {return 5}}
                 {{text-equal? "C-F" t} {return 6}}
                 {{text-equal? "C-f" t} {return 6}}
                 {{text-equal? "C-G" t} {return 7}}
                 {{text-equal? "C-g" t} {return 7}}
                 {{text-equal? "C-H" t} {return 8}}
                 {{text-equal? "C-h" t} {return 8}}
                 {{text-equal? "C-I" t} {return 9}}
                 {{text-equal? "C-i" t} {return 9}}
                 {{text-equal? "C-J" t} {return 10}}
                 {{text-equal? "C-j" t} {return 10}}
                 {{text-equal? "C-K" t} {return 11}}
                 {{text-equal? "C-k" t} {return 11}}
                 {{text-equal? "C-L" t} {return 12}}
                 {{text-equal? "C-l" t} {return 12}}
                 {{text-equal? "C-M" t} {return 13}}
                 {{text-equal? "C-m" t} {return 13}}
                 {{text-equal? "C-N" t} {return 14}}
                 {{text-equal? "C-n" t} {return 14}}
                 {{text-equal? "C-O" t} {return 15}}
                 {{text-equal? "C-o" t} {return 15}}
                 {{text-equal? "C-P" t} {return 16}}
                 {{text-equal? "C-p" t} {return 16}}
                 {{text-equal? "C-Q" t} {return 17}}
                 {{text-equal? "C-q" t} {return 17}}
                 {{text-equal? "C-R" t} {return 18}}
                 {{text-equal? "C-r" t} {return 18}}
                 {{text-equal? "C-S" t} {return 19}}
                 {{text-equal? "C-s" t} {return 19}}
                 {{text-equal? "C-T" t} {return 20}}
                 {{text-equal? "C-t" t} {return 20}}
                 {{text-equal? "C-U" t} {return 21}}
                 {{text-equal? "C-u" t} {return 21}}
```

```
    {{text-equal? "C-V" t} {return 22}}
    {{text-equal? "C-v" t} {return 22}}
    {{text-equal? "C-W" t} {return 23}}
    {{text-equal? "C-w" t} {return 23}}
    {{text-equal? "C-X" t} {return 24}}
    {{text-equal? "C-x" t} {return 24}}
    {{text-equal? "C-Y" t} {return 25}}
    {{text-equal? "C-y" t} {return 25}}
    {{text-equal? "C-Z" t} {return 26}}
    {{text-equal? "C-z" t} {return 26}}
    {{text-equal? "C-3" t} {return 27}}
    {{text-equal? "C-[" t} {return 27}}
    {{text-equal? "C-`" t} {return 27}}
      {{text-equal? "C-4" t} {return 28}}
      {{text-equal? "C-\\" t} {return 28}}
      {{text-equal? "C-\|" t} {return 28}}
      {{text-equal? "C-5" t} {return 29}}
      {{text-equal? "C-]" t} {return 29}}
      {{text-equal? "C-\}" t} {return 29}}
    {{text-equal? "C-6" t} {return 30}}
    {{text-equal? "C-^" t} {return 30}}
    {{text-equal? "C-~" t} {return 30}}
    {{text-equal? "C-/" t} {return 31}}
    {{text-equal? "C-7" t} {return 31}}
    {{text-equal? "C-_" t} {return 31}}
    {{text-equal? "C-!" t} {return 33}}
    {{text-equal? "C-'" t} {return 34}}
    {{text-equal? "C-#" t} {return 35}}
    {{text-equal? "C-$" t} {return 36}}
    {{text-equal? "C-%" t} {return 37}}
    {{text-equal? "C-&" t} {return 38}}
    {{text-equal? "C-’" t} {return 39}}
    {{text-equal? "C-(" t} {return 40}}
    {{text-equal? "C-)" t} {return 41}}
    {{text-equal? "C-*" t} {return 42}}
    {{text-equal? "C-+" t} {return 43}}
    {{text-equal? "C-," t} {return 44}}
    {{text-equal? "C--" t} {return 45}}
    {{text-equal? "C-." t} {return 46}}
    {{text-equal? "C-0" t} {return 48}}
    {{text-equal? "C-1" t} {return 49}}
    {{text-equal? "C-9" t} {return 57}}
    {{text-equal? "C-:" t} {return 58}}
    {{text-equal? "C-;" t} {return 59}}
    {{text-equal? "C-<" t} {return 60}}
    {{text-equal? "C-=" t} {return 61}}
    {{text-equal? "C->" t} {return 62}}
    {{text-equal? "C-?" t} {return 63}}
    {{text-equal? "C-8" t} {return 127}}
    }}
{{text-equal? "left-arrow" t} {return 134}}
{{text-equal? "up-arrow" t} {return 135}}
{{text-equal? "right-arrow" t} {return 136}}
{{text-equal? "down-arrow" t} {return 137}}
```

```
            }}}
        {return -1}
        }

}
```

A.4 cte-keybase.curl

```
| Curl Text Editor Key Base

| Global Variable: gCTEKeyBase defines a list of all possible base keys
| in list form. Symbols preceded with "C-" mean w/ the Control Key.
{define-variable gCTEKeyBase:list}

{set gCTEKeyBase
  '{ | begin a list for the default functions
    {"C-2"  cte-value-0}
    {"C-a"  cte-value-1}
    {"C-b"  cte-value-2}
    {"C-c"  cte-value-3}
    {"C-d"  cte-value-4}
    {"C-e"  cte-value-5}
    {"C-f"  cte-value-6}
    {"C-g"  cte-value-7}
    {""  cte-value-8}
    {""  cte-value-9}
    {"C-j" cte-value-10}
    {"C-k"  cte-value-11}
    {"C-l"  cte-value-12}
    {"
"  cte-value-13}
    {"C-n"  cte-value-14}
    {"C-o"  cte-value-15}
    {"C-p"  cte-value-16}
    {"C-q"  cte-value-17}
    {"C-r"  cte-value-18}
    {"C-s"  cte-value-19}
    {"C-t"  cte-value-20}
    {"C-u"  cte-value-21}
    {"C-v"  cte-value-22}
    {"C-w"  cte-value-23}
    {"C-x"  cte-value-24}
    {"C-y"  cte-value-25}
    {"C-z"  cte-value-26}
    {""  cte-value-27}
    {"C-4"  cte-value-28}
    {"C-5"  cte-value-29}
    {"C-6"  cte-value-30}
    {"C-7"  cte-value-31}
    {" " cte-value-32}
    {"!" cte-value-33}
```

```
{"'" cte-value-34}
{"#" cte-value-35}
{"$" cte-value-36}
{"%" cte-value-37}
{"&" cte-value-38}
{"'" cte-value-39}
{"(" cte-value-40}
{")" cte-value-41}
{"*" cte-value-42}
{"+" cte-value-43}
{"," cte-value-44}
{"-" cte-value-45}
{"." cte-value-46}
{"/" cte-value-47}
{"0" cte-value-48}
{"1" cte-value-49}
{"2" cte-value-50}
{"3" cte-value-51}
{"4" cte-value-52}
{"5" cte-value-53}
{"6" cte-value-54}
{"7" cte-value-55}
{"8" cte-value-56}
{"9" cte-value-57}
{":" cte-value-58}
{";" cte-value-59}
{"<" cte-value-60}
{"=" cte-value-61}
{">" cte-value-62}
{"?" cte-value-63}
{"@" cte-value-64}
{"A" cte-value-65}
{"B" cte-value-66}
{"C" cte-value-67}
{"D" cte-value-68}
{"E" cte-value-69}
{"F" cte-value-70}
{"G" cte-value-71}
{"H" cte-value-72}
{"I" cte-value-73}
{"J" cte-value-74}
{"K" cte-value-75}
{"L" cte-value-76}
{"M" cte-value-77}
{"N" cte-value-78}
{"O" cte-value-79}
{"P" cte-value-80}
{"Q" cte-value-81}
{"R" cte-value-82}
{"S" cte-value-83}
{"T" cte-value-84}
{"U" cte-value-85}
{"V" cte-value-86}
{"W" cte-value-87}
```

```
{"X" cte-value-88}
{"Y" cte-value-89}
{"Z" cte-value-90}
{"[" cte-value-91}
{"\\" cte-value-92}
{"]" cte-value-93}
{"^" cte-value-94}
{"_" cte-value-95}
{"`" cte-value-96}
{"a" cte-value-97}
{"b" cte-value-98}
{"c" cte-value-99}
{"d" cte-value-100}
{"e" cte-value-101}
{"f" cte-value-102}
{"g" cte-value-103}
{"h" cte-value-104}
{"i" cte-value-105}
{"j" cte-value-106}
{"k" cte-value-107}
{"l" cte-value-108}
{"m" cte-value-109}
{"n" cte-value-110}
{"o" cte-value-111}
{"p" cte-value-112}
{"q" cte-value-113}
{"r" cte-value-114}
{"s" cte-value-115}
{"t" cte-value-116}
{"u" cte-value-117}
{"v" cte-value-118}
{"w" cte-value-119}
{"x" cte-value-120}
{"y" cte-value-121}
{"z" cte-value-122}
{"`" cte-value-123}
  {"\|" cte-value-124}
  {"\}" cte-value-125}
{"~" cte-value-126}
{"C-8" cte-value-127}
{"left-arrow" cte-value-134}
{"up-arrow" cte-value-135}
{"right-arrow" cte-value-136}
{"down-arrow" cte-value-137}
}}
```

A.5 cte-keysequences.curl

```
| Curl Text Editor Key Sequences

| Global Variable: gCTEKeySequences defines a list of possible key sequences
| in list form. Symbols preceded with "C-" mean w/ the Control Key.
```

```
{define-variable gCTEKeySequences:list}

{set gCTEKeySequences

'{  | begin a new list
    {"C-x" {"C-k" cte-edit-key-sequences}
      {"C-r" cte-edit-reserve-words}
      {"C-f" cte-file-open}
      {"C-c" cte-quit-editor}
      {"C-q" cte-help}
      {"C-s" cte-file-save}
      {"."   cte-repeat}
      {"C-z" fubartest}}
    {"C-/" cte-goto-file-start}
    {"C-\\" cte-goto-file-end}
    {"C-a" cte-goto-line-start}
    {"C-b" cte-go-char-backward}
    {"C-d" cte-del-next-char}
    {"C-e" cte-goto-line-end}
    {"C-f" cte-go-char-forward}
    {"C-h" cte-del-previous-char}
    {"C-k" cte-del-line}
    {"C-n" cte-go-line-down}
    {"C-p" cte-go-line-up}
    {"C-s" cte-find-pattern}
    {"C-t" cte-toggle-capital}
    {"C-y" cte-yank}
    {"C-z" cte-indent-all}
    {"C-]" cte-next-pattern}
    {"C-[" cte-prev-pattern}
    |{"C-*" cte-reload-environment}
    {""  cte-indent}
    {"left-arrow" cte-go-char-backward}
    {"up-arrow" cte-go-line-up}
    {"right-arrow" cte-go-char-forward}
    {"down-arrow" cte-go-line-down}
    | and insert those characters we wish to display
    {"
"  cte-insert-char}
    {" " cte-insert-char}
    {"!" cte-insert-char}
    {"'" cte-insert-char}
    {"#" cte-insert-char}
    {"$" cte-insert-char}
    {"%" cte-insert-char}
    {"&" cte-insert-char}
    {"'" cte-insert-char}
    {"(" cte-insert-char}
    {")" cte-insert-char}
    {"*" cte-insert-char}
    {"+" cte-insert-char}
    {"," cte-insert-char}
    {"-" cte-insert-char}
    {"." cte-insert-char}
```

```
{"/" cte-insert-char}
{"0" cte-insert-char}
{"1" cte-insert-char}
{"2" cte-insert-char}
{"3" cte-insert-char}
{"4" cte-insert-char}
{"5" cte-insert-char}
{"6" cte-insert-char}
{"7" cte-insert-char}
{"8" cte-insert-char}
{"9" cte-insert-char}
{":" cte-insert-char}
{";" cte-insert-char}
{"<" cte-insert-char}
{"=" cte-insert-char}
{">" cte-insert-char}
{"?" cte-insert-char}
{"@" cte-insert-char}
{"A" cte-insert-char}
{"B" cte-insert-char}
{"C" cte-insert-char}
{"D" cte-insert-char}
{"E" cte-insert-char}
{"F" cte-insert-char}
{"G" cte-insert-char}
{"H" cte-insert-char}
{"I" cte-insert-char}
{"J" cte-insert-char}
{"K" cte-insert-char}
{"L" cte-insert-char}
{"M" cte-insert-char}
{"N" cte-insert-char}
{"O" cte-insert-char}
{"P" cte-insert-char}
{"Q" cte-insert-char}
{"R" cte-insert-char}
{"S" cte-insert-char}
{"T" cte-insert-char}
{"U" cte-insert-char}
{"V" cte-insert-char}
{"W" cte-insert-char}
{"X" cte-insert-char}
{"Y" cte-insert-char}
{"Z" cte-insert-char}
{"[" cte-insert-char}
{"\\" cte-insert-char}
{"]" cte-insert-char}
{"^" cte-insert-char}
{"_" cte-insert-char}
{"`" cte-insert-char}
{"a" cte-insert-char}
{"b" cte-insert-char}
{"c" cte-insert-char}
{"d" cte-insert-char}
```

```
    {"e" cte-insert-char}
    {"f" cte-insert-char}
    {"g" cte-insert-char}
    {"h" cte-insert-char}
    {"i" cte-insert-char}
    {"j" cte-insert-char}
    {"k" cte-insert-char}
    {"l" cte-insert-char}
    {"m" cte-insert-char}
    {"n" cte-insert-char}
    {"o" cte-insert-char}
    {"p" cte-insert-char}
    {"q" cte-insert-char}
    {"r" cte-insert-char}
    {"s" cte-insert-char}
    {"t" cte-insert-char}
    {"u" cte-insert-char}
    {"v" cte-insert-char}
    {"w" cte-insert-char}
    {"x" cte-insert-char}
    {"y" cte-insert-char}
    {"z" cte-insert-char}
    {"`" cte-insert-char}
      {"\|" cte-insert-char}
      {"\}" cte-insert-char}
    {"~" cte-insert-char}
    }
  } | end of file
```

A.6 cte-point.curl

```
| point.curl
|
| Curl Text Editor Files
| Jon Heiner
|
| Copyright (c) 1998
| Massachussetts Institute of Technology
| Laboratory for Computer Science
| All Rights Reserved.
|
| MIT MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY OF THIS
| SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
| IMPLIED WARRANTIES FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.
| MIT SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED AS A RESULT OF USING,
| MODIFYING OR DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
|=========================================================================

| File Description:
| This file contains the CTEPoint class definition.
|
| The CTEPoint is basically a number from zero to positive infinity.
```

```
| It represents location/position in a textbuffer.
| Although we would like to represent positive infinity, for practical
| reasons, the highest number is an cte-kPointBits bit binary value of
| all 1's.

| for now...just implementing with a basic int

{define-class CTEPoint {}
  private myBits:int

  {define {CTEPoint.init val:int=0}
    {if {< val 0}
      {set self.myBits 0}
      {set self.myBits val}}
    }

  {define public {CTEPoint.getValue}:CTEPoint
    {return self}
    }

  {define public {CTEPoint.setValue val:CTEPoint}:CTEPoint
    {if {< val.myBits 0}
      {set self.myBits 0}
      {set self.myBits val.myBits}}
    {return self}
    }
  {define public {inc p:CTEPoint}:void
    {set p.myBits {+ p.myBits 1}}
    }

  {define public {dec p:CTEPoint}:void
    {set p.myBits {- p.myBits 1}}
    }

  {define public {point+ p1:CTEPoint p2:CTEPoint}:CTEPoint
    {return {new CTEPoint val={+ p1.myBits p2.myBits}}}
    }

  {define public {point- p1:CTEPoint p2:CTEPoint}:CTEPoint
    {return {new CTEPoint val={- p1.myBits p2.myBits}}}
    }

  {define public {point-eq? p1:CTEPoint p2:CTEPoint}:int
    {return {= p1.myBits p2.myBits}}
    }

  {define public {point< p1:CTEPoint p2:CTEPoint}:bool
    {return {< p1.myBits p2.myBits}}
    }

  {define public {point> p1:CTEPoint p2:CTEPoint}:bool
    {return {> p1.myBits p2.myBits}}
    }
```

```
}
```

A.7 cte-reservewords.curl

```
| Curl Text Editor
| Reserved Words List

| the proper form for this list is alternating colors and lists of double-
quoted words.
| it is ok to have several lists in a row, or several colors in a row.
| the CTE will use the most recent color and go through the lists in order.
| if no color is indicated, then the default color for reserved words will be
used

{define-variable gCTEReserveWords:list}

{set gCTEReserveWords
  '{ | begin list of reserved word
    grey
    {
      "Jon Heiner"
      "MIT"
      "LCS"
      "Copyright"
      }
    red
    {
      "!="
      "%"
      "*"
      "+"
      "-"
      "/"
      "<"
      "<="
      ">"
      ">="
      "Buffer"
      "Button"
      "CheckButton"
      "Circle"
      "ClickBox"
      "DragContainer"
      "DragImage"
      "Dragee"
      "DropTarget"
      "EventHandler"
      "Exception"
      "FileBox"
      "FileInputPort"
      "FileOutputPort"
      "Frame"
```

```
"HashTable"
"Hbox"
"IndentedBox"
"InputPort"
"Line"
"ListBox"
"MenuAction"
"MenuBar"
"OutputPort"
"PopupMenu"
"Port"
"RadioButton"
"RadioButtons"
"Rectangle"
"ScrollBox"
"Selectee"
"StringBuffer"
"SubMenu"
"Table"
"TextField"
"Vbox"
"View"
"\\="
"abs"
"anchor"
"and"
"any"
"append"
"append!"
"apply"
"apply-method"
"arccosine"
"arcsine"
"arctangent"
"arctangent2"
"aref"
"array"
"assq"
"big"
"bit"
"bit-and"
"bit-or"
"bit-sll"
"bit-sra"
"bit-srl"
"bit-xor"
"block"
"bold"
"br"
"break"
"cadr"
"car"
"cast"
"catch"
```

```
"cddr"
"cdr"
"center"
"char"
"choose"
"choose-randomly"
"choose-sequentially"
"code"
"cond"
"cons"
"cons*"
"continue"
"copy-list"
"cosine"
"define"
"define-class"
"define-constant"
"define-form"
"define-macro"
"define-variable"
"delq!"
"description"
"docref"
"documentstyle"
"dotimes"
"enumerate"
"eq?"
"eqn"
"equal?"
"error"
"eval"
"example"
"exponential"
"file-write-date"
"finally"
"float"
"for"
"format"
"get-buffer"
"get-buffer-of-size"
"get-string-buffer"
"get-string-buffer-of-size"
"hrule"
"if"
"image"
"import"
"include"
"input-var"
"int"
"int16"
"int8"
"invoke-method"
"isa"
"italic"
```

```
"itemize"
"keyword-supplied?"
"lambda"
"last"
"lastcdr"
"let"
"letrec"
"list"
"list-length"
"loop"
"make-list"
"map"
"map!"
"match"
"match-prefix"
"max"
"memq"
"min"
"neq?"
"new"
"nobreak"
"not"
"nth"
"nthrest"
"or"
"output"
"paragraph"
"popup-dialogue"
"popup-graphic"
"popup-text-query"
"power"
"print"
"proc"
"quote"
"random"
"release-buffer"
"release-string-buffer"
"require"
"rest-as-list"
"rest-as-vector"
"return"
"reverse"
"reverse!"
"s"
"section"
"set"
"sine"
"sleep"
"small"
"sort-list"
"sort-list!"
"stderr"
"stdout"
"sublist"
```

```
  "subsection"
  "subsubsection"
  "subtext"
  "symbol"
  "tagged-block"
  "tagged-loop"
  "tangent"
  "text"
  "text->symbol"
  "text-append"
  "text-compare"
  "text-equal?"
  "text-fill"
  "text-hash"
  "text-pos"
  "text-replace"
  "text-rpos"
  "throw"
  "tiny"
  "try"
  "typeof"
  "unless"
  "until"
  "value"
  "vector"
  "vector->list"
  "verbatim"
  "void"
  "void?"
  "vrule"
  "walk"
  "when"
  "while"
  "write"
  }
black  | for base types
{
  "int"
  "int8"
  "int16"
  "char"
  }
{
  "float"
  "bit"
  "any"
  "void"
  "text"
  "vector"
  }
}
```

A.8 cte-textbuffer.curl

```
| textbuffer.curl
|
| Curl Text Editor Files
| Jon Heiner
|
| Copyright (c) 1998
| Massachussetts Institute of Technology
| Laboratory for Computer Science
| All Rights Reserved.
|
| MIT MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY OF THIS
| SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
| IMPLIED WARRANTIES FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.
| MIT SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED AS A RESULT OF USING,
| MODIFYING OR DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
|=============================================================================

{include "cte-constants.curl"}
{include "cte-point.curl"}

| File Description:
| This file contains the CTETextBuffer class definition.
|
| The text buffer is a datatype abstraction which simulates
| a consistent, persistent, rapidly-accessable, infinite-length
| array of chars.  the accessor methods are extremely straight-
| forward and somewhat spartan.  The way in which these methods
| are used should be obvious, however they are explicitly documented.

{define-class CTETextBuffer {}

  ||### define class-fields ###||
  myHasAFile?:int  || a flag which when zero means we are a scratch buffer
  myPoint:CTEPoint || a number from 0 to positive infinity which
  || represents the current location in the buffer
  myBubble:text    || scratch pad to handle keyboard input.
  myBubbleSize:int || how much text is in the bubble.
  mySegments:list
  || there are a max of kNumTBSegments they rotate via a Random replacement
  || strategy.  each vector is comprised of
  || {data:text start:any end:any size:int dirty:bool}
  || in which text is size cte-kTBSegmentMax,
  || but  with cte-kTBSegmentSize text in it initially.
  || *** go to end of file for my BIG DISCUSSION ON SEGMENTS ***

  myFileIP:InputPort  || the output and input ports to which this
  myFileOP:OutputPort || buffer is associated.  Note that this file is
  || updated periodically, so it should be a temp (name~) file.
  || for saving, flush all segments and then copy the file name~ -> name

 ||### define class-methods ###||
```

```
   || PUBLIC METHODS/ACCESSORS
     {define {CTETextBuffer.init f:text=cte-kYesWeHaveNoFile}
       {set self.myPoint {new CTEPoint}}
       {set self.myBubble {new text cte-kBubbleSize}}
       {set self.myBubbleSize 0}

       || 1.) do something about the filePorts
       {if {text-equal? f cte-kYesWeHaveNoFile}                    | if no file
is passed in...
         {set self.myHasAFile? 0}
         {let file:File={xresolve-filename text}                  | ...if 'file'
does not exist
             {set self.myFileIP {file.read-open}}                  | then we get
a big fat error.
             {set self.myFileOP {file.write-open}}                 | this is an
error in file.read-open
             {set self.myHasAFile? 1}}}                          | we have a file

       || 2.) create the segments data structure ( mySegments:{list vector int
int} )
       {set self.mySegments {cons {new vector cte-kNumTBSegments}
                  {cons -1
                  {cons 1 '{}}}}}

       || 3.) if we have a file, then get the first cte-kTBSegmentSize
       ||     characters s from the file and put them in a new segment
       {if self.myHasAFile?
         {begin {set {aref {car self.mySegments} cte-kSegText}       | allocate
text array
                 {new text cte-kTBSegmentMax}}
                {set {aref {car self.mySegments} cte-kSegStart} 0}   | this segment
starts at the beginning of 'file'

                 {self.myFileIP.Seek 0 'start}                       | go to start
of file and read the first
                {set {aref {car self.mySegments} cte-kSegEnd}        | cte-kTBSeg-
mentSize characters (if we can)
                {self.myFileIP.ReadText                          | set the Segment End
variable to how many
                 {aref {car self.mySegments} cte-kSegText}    | characters are actu-
ally read
                    cte-kTBSegmentSize}}

                {set {aref {car self.mySegments} cte-kSegSize}       | set the size
to the end point from above
                 {aref {car self.mySegments} cte-kSegEnd}}
                {set {aref {car self.mySegments} cte-kSegDirty} false}| set dirty
to false, we haven't changed anything yet
                 }
          {begin {output "ERROR: Curl Text Editors (alpha) must be associated with
files"}
                 {exit}}
         }
       }
```

```
    {define public {CTETextBuffer.setPoint n:CTEPoint}:CTEPoint
      {self.popBubble}                                                    | "pops"
bubble
      {return {self.myPoint.setValue {n.getValue}}}                       | memory
leak?
      }

    {define public {CTETextBuffer.getPoint}:CTEPoint
      {return {self.myPoint.getValue}}
      }

    {define public {CTETextBuffer.writeChar l:char}:void
      {self.writeChar2Bubble l}
      }

    {define public {CTETextBuffer.writeText t:text}:int
      {let len:int={length t}
        {self.writeText2Bubble t {length t}}
        {return len}}
      }

    {define public {CTETextBuffer.writeTextWLen t:text len:int}:void
      {self.writeText2Bubble t len}
      }

    {define public {CTETextBuffer.read n:int}:text
      || __tricky__
      || 1.)  Due to usage patterns, it is likely that a read will
      || go straight to the segments, so first pop the bubble.
      {self.popBubble}
      || 2.) if we can get the text from just one segment, do so...
      || 3.) otherwise, it is easier to flush all the segments, and
      || 4.) read directly from ~ file
      }

    {define public {CTETextBuffer.delete n:any}:void
      || same as read...backspace delete is provided for speed.
      }

    {define public {CTETextBuffer.backspaceCharDelete}:void
      {if {> 0 self.myBubbleSize}
        {set self.myBubbleSize {- self.myBubbleSize 1}}
        || otherwise, we need to go to segment/filel
        }
      }


    || Since there are no regexps, we're really kind of stuck here.
    || So for now, just take a plain text string.
    || "pops bubble" if movePoint? is true.  returns the point.
    {define public {CTETextBuffer.find pattern:text forward?:bool=true move-
Point?:bool=false}:CTEPoint
      | not implemented...holding for regexps consideration
      }
```

```
  || PRIVATE METHODS/SEGMENT & BUBBLE HANDLING
    {define private {CTETextBuffer.popBubble}:void
      || if there's nothing in the bubble, do nothing; else...
      {if {> self.myBubbleSize 0}
        || write bubble's text to active segment
        {self.write2ActiveSegment self.myBubble self.myBubbleSize}
        || and reset the Bubble's Size to zero
        {set self.myBubbleSize 0}
        }
      }
    || optimize writeChar2Bubble by moving it into writeChar
    {define private {CTETextBuffer.writeChar2Bubble aKey:char}:void
      {let bSize:int={set self.myBubbleSize {+ self.myBubbleSize 1}}
        || write the character to the bubble
        {set {aref self.myBubble bSize} aKey}
        {if {>= bSize cte-kBubbleSize}  || if we've filled up the bubble, pop
it.
            {self.popBubble}}}
      }
    || optimize writeText2Bubble by moving it into writeTextWLen
    {define private {CTETextBuffer.writeText2Bubble t:text len:int}:void
      {if {> len cte-kBubbleSize}  || if you're just writing LOTS of text, then
        {begin {self.popBubble}  || pop the bubble and
               {self.write2ActiveSegment t len} || write straight to the segment
                }
        {let bSize:int={set self.myBubbleSize {+ self.myBubbleSize len}}
           {if {>= bSize cte-kBubbleSize} || if this fills or surpasses the bubble
but fits in it,
                {begin {self.popBubble}  || THEN pop bubble first...and write the
text to empty bubble
                   {move-text t 0 self.myBubble 0 len}}
                {move-text t 0 self.myBubble {- bSize len} len}}}} | careful here
on the locations
       }

   |  {define private {CTETextBuffer.flushSegment index:int}
      |    {let startPos:int=     | the segment's START Point in the file~
        | endPos:int=     | the segment's END Point in the file~
        | change:int=    | the (new size of segment) - (endPos - startPos), pos-
sibly negative
        | data:text=   | the segment's text
        |     {self.myFileIP.seek endPos 'start}            | set read pointer
to after segment
        |     {self.myFileOP.seek {+ endPos change} 'start}  | set write
pointer to "past" end of segment
        |     {self.myFileIP.CopyOut self.myFileOP {- {self.myFileIP.Con-
tentLength} endPos}}   | shift text
        |     {self.myFileOP.seek startPos 'start}           | go to start of
segment
        |     {self.myfileOP.writeText data}                | insert the edit-
ted segment's text
        |     }
       |    }
```

```
    {define private {CTETextBuffer.write2ActiveSegment t:text len:int}
      }
    }

  || *** method documentation may be a bit out of date...

  {doc setPoint
    The stPoint method moves the point (which maintains
    location within the buffer) to the location {param n} number
    of characters into the buffer.
    {enumerate {paragraph arguments: {param n} is a number from 0 to positive
infinity}
      {paragraph effects: changes the point location}
      {paragraph boundary conditions: {param n}<0: moves point to start
           of the buffer;  {param n}>buffer size:  moves point to end
           of the buffer.}
      {paragraph returns: the actual point location which is set}
      }}
  {doc getPoint
    The getPoint method returns the current locaton in the text buffer.
    {enumerate {paragraph effects: none, simple accessor}
      {paragraph returns: location in text buffer}}}
  {doc writeChar writeText writeTextWLen
    The write method inserts the text t (of length len, optional argument)
    at the current point in the textbuffer.
    {enumerate: {paragraph arguments: t is a valid array of chars.  l (letter)
is a char.
           if len is used, it MUST be the size of t, it is NOT checked.}
       {paragraph effects: adds the text t (char l) to the current buffer.}
       {paragraph boundary conditions: empty arrays are ignored.  any valid
array of chars
           will work.}
       {paragraph returns: an int that is the length of the text added to the
buffer (or void).}
       }}
  {doc read
    The read method reads {param n} characters from the buffer and
    returns them as a text.
    {enumerate {paragraph arguments: {param n} is an integer which is the maxi-
mum number
           of characters to get back from the buffer, beginning
           at the current point.}
      {paragraph effects: none, simple accessor}
      {paragraph boundary conditions: if {param n} causes us to go beyond the
           last character in the file, then only those characters
           in the buffer are returned, and the text is not length {param n}.}
      {paragraph returns: text (array of chars) which corresponds to at most
           the {param n} characters following the point.}
      }}
  {doc delete
    The delete method deletes {param n} characters from the buffer beginning at
the point.
    {enumerate {paragraph arguments: {param n} the maximum number of characters
to delete}
```

```
    {paragraph effects: deletes a section of the file beginning at the point.}
    {paragraph boundary conditions: if {param n} causes us to go beyond the
         last character in the file, then the characters up to
         the end of the file are deleted.}
    }}
```

[code size reduced for readability]

```
|| *** BIG DISCUSSION ON SEGMENTS ***

||Briefly, the way the TextBuffer works internally is that text/characters written to the
||buffer go initially to the "Bubble."  This is a small cache which fills up quickly and
||then "pops."  The TextBuffer has a single bubble which is associated with the currently
||"active" Segment at the point in that segment where editing is happening.  Thus, moving
||around while editing will pop the bubble.  So will switching among segments (which happens
||generally only when you move around while editing.)
||
||There are a number of segments which are simply "samples" from the file in which the user
||is working.  The TextBuffer starts off by slurping up the first N characters in the file
||and putting them in one segment. If you move to some portion of the file which is not
||currently in memory (in one of the segments), the TextBuffer will slurp up that portion
||of the file and put it in a _different_ segment which DOES NOT overlap any of the current
||segments.  There is a finite total number of segments, however, so periodically one of the
||segments is thrown out to make space for a new segment to be created.  This happens by
||random selection.  We could implement a LRU (Least Recently Used) replacement strategy,
||but the overhead is not worthwhile.  [Think about it, it's likely less efficient in practice.]
||In this way, the TextBuffer is able to edit an infinitely large file with efficiency,
||because edits are generally clustered.  The segments represent this clustering.
||
||...ON THE mySegments DATA STRUCTURE
||Simply put, mySegments has the type:  {list vector int int}          (list of anys)
||where the vector has type:            {vector text int int int int}  (array of anys)
||specifics are as follows:
||
||                  .-------------------------------------------------------------------------.
||.----------------. | .-------------------------------. .--------------------. .-------------. |
|| mySegments:list | | | A.) array: 1.) text           | | B.) active:int     | | C.)  segs:int | |
||o================o | |            2.) start:int      | | <0 --> none active | | # of segments | |
||                  | |            3.) end:int        | o====================o o===============o |
||                  | |            4.) size:int       |                                         |
||                  | |            5.) dirty:bool     |                                         |
||                  | o===============================o                                         |
||                  o=========================================================================o
||
||active: gives the index of the currently active segment.  -1 if none are active.
||segs:   give the number of structures in the array (the # of segments).
||text:   an array of characters from the file being editing.
||start:  where we began reading from the file when 'text' was created.
||end:    where we ended reading from the file when 'text' was created.
||size:   the size of the text as a consequence of edits.
||dirty:  whether we have edited text in this segment, or just read.
||
||eof
```

A.9 cte-textdisplay.curl

```
| cte-textdisplay.curl
|
| Curl Text Editor Files
| Jon Heiner
|
| Copyright (c) 1998
| Massachussetts Institute of Technology
| Laboratory for Computer Science
| All Rights Reserved.
|
| MIT MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY OF THIS
```

```
   | SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
   | IMPLIED WARRANTIES FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.
   | MIT SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED AS A RESULT OF USING,
   | MODIFYING OR DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
   |=========================================================================

   {include "cte-inputfilter.curl"}
   {include "cte-reservewords.curl"}

   | File Description:
   | this is the curl text editor visual class
   | it handles all the code visualization, pretty formatting, etc.
   | currently, the internals are being handled through the methods
   | in CodeText which are sufficient for handling arrays of text.
   | The CTETextDisplay class inherits from the CTEInputFilterClass which han-
dles
   | all the input coming in from the user. Inherited functionality
   | is mirrored in all the classes and can be accessed via the super.foo
   | functions. Currently, this is only used sparingly, but is provided
   | to make the class more useful/portable.

   {define-class CTETextDisplay {CTEInputFilter CodeText}
     file:File
     file-modified:int | have we modified the file
     cursorx:int       | cursor's x position from left [0,end]
     cursory:int       | cursor's y position from top  [1,max]
     curlymatching:bit | for curly matching, see draw method
     clipboard:text    | for yanked text
     pattern:text      | for searching
     function:symbol   | last function
     lastkey:int       | last function

     {define public {init f:text ...}
       {invoke-method CodeText 'init self "Press C-h for help..." {rest-as-is}}
       {invoke-method CTEInputFilter 'init self "Press C-h for help..." {rest-
as-is}}  |this works?
       {self.open-file f}
       | some control variables...
       {set self.file-modified false}
       {set self.pattern "pattern"}
       {set self.function 'null}
       {set self.lastkey 97} | set it to an a
       {self.gotoFileStart}
       }

     {define public {CTETextDisplay.key-press e:KeyEvent}:void
       {super.key-press e}
       {self.doFunction self.mySymbol e.key}
       {self.invalidate}

       | useful debug stuff
       |{output e.key}
       |{output "current " self.current " cursorx " self.cursorx " cursory "
 self.cursory}
```

```
      |{output e.key " --> " {cast char e.key}}

      |{output "current char: " {aref self.txt {self.get-current}}}
      |{output "get-current: " {self.get-current}}
      |{output "chars-till-prev-newline: " {self.chars-till-prev-newline}}
      |{output "chars-till-next-newline: " {self.chars-till-next-newline}}
      }

    {define protected {CTETextDisplay.doFunction f:symbol k:int}:void
      | Here, we would have preferred to not use a switch statement.
      | We would like to have used first-class procedures such that they
      | are passed as values and later evaluated, or even better to pass
      | symbols that can then be cast into the correct procedures, but the Curl
      | semantic does not support this. Consequently, we here take symbols
      | and "convert" them into procedures...
      {cond
        {{eq? f 'null}                      {return}}
        {{eq? f 'cte-insert-char}           {self.insertChar {cast char k}}}
        {{eq? f 'cte-go-line-up}           {self.goLineUp}} | towards beginning
of buffer
        {{eq? f 'cte-go-line-down}          {self.goLineDown}} |towards end of
buffer
        {{eq? f 'cte-go-char-forward}       {self.goCharForward}}
        {{eq? f 'cte-go-char-backward}      {self.goCharBackward}}
        {{eq? f 'cte-goto-next-word}       {self.gotoNextWord}}
        {{eq? f 'cte-goto-prev-word}       {self.gotoPrevWord}}
        {{eq? f 'cte-goto-next-curly}      {self.gotoNextCurly}}
        {{eq? f 'cte-goto-prev-curly}      {self.gotoPrevCurly}}
        {{eq? f 'cte-goto-file-start}      {self.gotoFileStart}}
        {{eq? f 'cte-goto-file-end}        {self.gotoFileEnd}}
        |{{eq? f 'cte-scroll-page-forward} {self.scrollPageForeward}} | **
requires get page size from myDisplay Buffer
        |{{eq? f 'cte-scroll-page-backward} {self.scrollPageBackward}} | **
requires get page size from myDisplay Buffer
        {{eq? f 'cte-goto-line-start}      {self.gotoLineStart}}
        {{eq? f 'cte-goto-line-end}        {self.gotoLineEnd}}
        {{eq? f 'cte-goto-end-of-word}     {self.gotoEndOfWord}}
        {{eq? f 'cte-goto-start-of-word}   {self.gotoStartOfWord}}
        |{{eq? f 'cte-go-to-top-of-page}    {self.gotoTopOfPage}} | ** requires
get page size from myDisplay Buffer
        {{eq? f 'cte-go-to-bottom-of-page} {self.gotoBottomOfPage}} | **
requires get page size from myDisplay Buffer
        {{eq? f 'cte-del-previous-char}     {self.delPrevChar}}
        {{eq? f 'cte-del-next-char}         {self.delNextChar}}
        {{eq? f 'cte-del-word}              {self.delWord}}
        {{eq? f 'cte-del-line}              {self.delLine}}
        {{eq? f 'cte-del-within-curlies}    {self.delWithinCurlies}}
        {{eq? f 'cte-change-word}           {self.changeWord}}
        {{eq? f 'cte-change-line}           {self.changeLine}}
        {{eq? f 'cte-change-within-curlies}{self.changeWithinCurlies}}
        {{eq? f 'cte-change-char}           {self.changeChar}}
        {{eq? f 'cte-find-pattern}          {self.findPattern}}
        {{eq? f 'cte-next-pattern}          {self.nextPattern}}
        {{eq? f 'cte-prev-pattern}          {self.prevPattern}}
```

```
        {{eq? f 'cte-repeat}                {self.repeat}}
        {{eq? f 'cte-undo}                  {self.undo}}
        {{eq? f 'cte-begin-macro}           {self.beginMacro}}
        {{eq? f 'cte-end-macro}             {self.endMacro}}
        {{eq? f 'cte-do-macro}              {self.doMacro}}
        {{eq? f 'cte-bind-macro}            {self.bindMacro}}
        {{eq? f 'cte-toggle-capital}        {self.toggleCapital}}
        {{eq? f 'cte-yank}                  {self.yank}}
        {{eq? f 'cte-indent}                {self.indent}}
        {{eq? f 'cte-indent-all}            {self.indentAll}}
        {{eq? f 'cte-toggle-special-chars} {self.toggleSpecialChars}}
        {{eq? f 'cte-file-save}             {self.fileSave}}
        {{eq? f 'cte-file-save-as}          {self.fileSaveAs}}
        {{eq? f 'cte-file-open}             {self.fileOpen}}
        {{eq? f 'cte-quit-editor}           {self.quitEditor}}
        {{eq? f 'cte-edit-key-sequences}    {self.editKeySequences}}
        {{eq? f 'cte-edit-reserve-words}    {self.editReserveWords}}
        {{eq? f 'cte-help}                  {self.help}}
        {{eq? f 'cte-reload-environment}    {self.reloadEnvironment}}
        }
      {if {neq? f 'cte-repeat}
        {begin
          {set self.function f}
          {set self.lastkey k}}}
      }

   | cursor movement functions
     | always call the cursor movement functions AFTER the actual editing has
been done
     | also: {CTETextDisplay.chars-till-prev-newline}:int
     |       {CTETextDisplay.chars-till-next-newline}:int

      {define private {CTETextDisplay.moveCursorRight}:void
       {if {= {cast int {aref self.txt {- {self.get-current} 1}}} cte-kNewline}
| if we have a return...follow it.
         {begin
           {set self.cursorx 0}
           {set self.cursory {+ self.cursory 1}}
           }
         {begin
           {set self.cursorx {+ self.cursorx 1}}
           }}}
     {define private {CTETextDisplay.moveCursorLeft}:void
       | if cursorx is below 0, then wrap around
       {if {> self.cursorx 0}
         {begin
           {set self.cursorx {- self.cursorx 1}}
           }
         {begin
           {set self.cursory {- self.cursory 1}}
           {set self.cursorx {self.chars-till-prev-newline}}
           }}}

   | basic movement functions
```

```
  {define protected {CTETextDisplay.goCharForward}:void
    {self.goR}
    |{super.goCharForward}
    }

  {define private {CTETextDisplay.goR i:int=1}:void
    {while {> i 0}
      {set i {- i 1}}
      {if {< {self.get-current} self.length}
        {begin
          {self.set-current {+ {self.get-current} 1}}
          {self.moveCursorRight}
          }}}
    }
  {define protected {CTETextDisplay.goCharBackward}:void
    {self.goL}
    |{super.goCharBackward}
    }
  {define protected {CTETextDisplay.goL i:int=1}:void
    {while {> i 0}
      {set i {- i 1}}
      {if {> {self.get-current} 0}
        {begin
          {self.set-current {- {self.get-current} 1}}
          {self.moveCursorLeft}
          }}}
    }
  {define protected {CTETextDisplay.goLineDown}:void
    {let d:int={self.chars-till-prev-newline}                    | where we
are from the previous newline
      {if {= {cast int {aref self.txt {self.get-current}}} cte-kNewline}     |
if we are on a newline
        {begin
          {self.goR}
| then step forward newline
          {if {= {cast int {aref self.txt {self.get-current}}} cte-kNewline}
| if we have another newline
            {return}                                                        |
then we're done
            | otherwise...move forward the correct number of characters
            {let l:int={+ {self.chars-till-next-newline} 1}  | how many char-
acters in this line
              {if {< d l}                                    | if this line is
long enough
                {self.goR i=d}                               | then just go to
the place we were
                {self.goR i=l}                                 | otherwise, go
to the end of the line
                }}}}
        | second case is that we were not sitting on a newline
        {let e:int={self.chars-till-next-newline}      |how far to end of this
line
          {self.goR i={+ e 1}}                         |move position to that
newline
```

```
            {set e {+ {self.chars-till-next-newline} 1}} |how many characters in
the next line
             {if {> e d}                                  | if the next line is
long enough
               {self.goR i={+ d 1}}                       | then just go to the
place we were
               {self.goR i=e}                             | otherwise, go to the
end of the line
               }}}
          }
        |{super.goLineDown}
        }

      {define protected {CTETextDisplay.goLineUp}:void
        {let d:int={self.chars-till-prev-newline}                    | get the
length of the current line
          {if {< d 1}                                                | if our
line is just a newline
            {begin
              {self.goL}                                             | then go
left one
              {self.goL i={self.chars-till-prev-newline}}            | and to
beginning of line, works even for i=0
              }
            {begin                                                   | if we're
not on a newline
              {self.goL i={+ d 1}}                                   | goto prev
newline
              {let e:int={self.chars-till-prev-newline}              | figure
length of previous line
                {self.goL i={max 0 {- e d}}}                         | go back
if prev line is longer
                }}
            }}
        |{super.goLineUp}
        }

      {define protected {CTETextDisplay.gotoLineStart}:void
        {self.goL i={self.chars-till-prev-newline}}
        }

      {define protected {CTETextDisplay.gotoLineEnd}:void
        {self.goR i={self.chars-till-next-newline}}
        }

    | basic editing
      {define protected {CTETextDisplay.insertChar k:char}:void
        | bad hack, handles returns (perhaps only on this screwy keyboard)
        | makes things "work."   can be switched on keyboard type.
        {if {= k cte-kReturn} {set k cte-kNewline}}

          {self.insert-char k}
        {self.moveCursorRight}

          {set self.file-modified true}
```

```
    }
  {define protected {CTETextDisplay.delPrevChar}:void
    | delete the character BEFORE the current position
    {self.delete-char}
    {self.moveCursorLeft}
    {set self.file-modified true}
    }
  {define protected {CTETextDisplay.delNextChar}:void
    | delete the character AFTER the current position
    {self.set-current {+ 1 {self.get-current}}}
    {self.delete-char}
    {set self.file-modified true}
    }

| more complex editing
  {define protected {CTETextDisplay.delLine}:void
    {self.goL}
    {let d:int={self.chars-till-next-newline}
      {self.goR}
      {if {< d 1} {set d 1}}

     {if {eq? self.function 'cte-del-line}       | handle subsequent delete
lines
        {set self.clipboard {text-append self.clipboard {subtext self.txt
{self.get-current} d}}}
        {set self.clipboard {subtext self.txt {self.get-current} d}}}
      {while {> d 0}
        {set d {- d 1}}
        {self.delNextChar}}
      }
    {set self.file-modified true}
    }

  {define protected {CTETextDisplay.yank}:void
    {output "clipboard: " self.clipboard}
    |{super.yank}

   {let l:int={length self.clipboard}
      k:int=0
      {while {< k l}
        {self.insertChar {cast int {aref self.clipboard k}}}
        {set k {+ k 1}}
        }}
    {set self.file-modified true}
    }
  {define protected {CTETextDisplay.repeat}:void
    {self.doFunction self.function self.lastkey}
    }
  {define protected {CTETextDisplay.toggleCapital}:void
    {let c:int={aref self.txt {self.get-current}}
      {if {and {> c 64} {< c 91}}       | we have a capital letter
        {begin
          {self.set-current {+ {self.get-current} 1}}
          {self.delete-char}           | delete the current letter
```

```
            {self.insertChar {+ c 32}}  |and insert its correlative capital
            }
           {if {and {> c 96} {< c 123}}  | we have a lower-case letter
            {begin
              {self.set-current {+ {self.get-current} 1}}
              {self.delete-char}
              {self.insertChar {- c 32}}
              }
            {self.goR}
            }}}}

  | tab indention
    {define protected {CTETextDisplay.indent}:void
      {self.gotoLineStart}
      {if {= self.cursory 1}
       {begin                      | special case for if we're on the first line
         {while {= {aref self.txt {self.get-current}} cte-kSpace}
           {self.delNextChar}}     | just delete leading white space
         {self.goLineDown}
         }
        {begin                     | but if we're NOT on the first line
          {let numSpaces:int=0
            numCurlies:int=0

           {self.goLineUp}          | go to start of previous line
            {self.gotoLineStart}
            | count the number of leading spaces
            {while {= {aref self.txt {self.get-current}} cte-kSpace}
              {set numSpaces {+ numSpaces 1}}
              {self.goR}
              }
            | count the number of unbalanced open curlies
            {let c:int={aref self.txt {self.get-current}}
              {while {!= c cte-kNewline}
                {if {= c cte-kCommentChar}          | ignore anything after
comments
                  {begin
                    {self.gotoLineEnd}
                    {self.goR}
                    {break}}}
                {if {= c cte-kOpenCurly}
                  {set numCurlies {+ numCurlies 1}}
                  {if {= c cte-kCloseCurly}
                    {set numCurlies {- numCurlies 1}}}}
                {self.goR}
                {set c {aref self.txt {self.get-current}}}
                }
              }
            | and delete white space
            {self.goR}
            {while {= {aref self.txt {self.get-current}} cte-kSpace}
              {self.delNextChar}}

              | useful debugging
```

```
            |{output "======="}
            |{output "DEBUG: numSpaces " numSpaces}
            |{output "DEBUG: numCurlies " numCurlies}
            | calculate the new offset
            {set numSpaces {+ numSpaces {* numCurlies 2}}}
            |{output "DEBUG: offset " numSpaces}
            | insert this many spaces
            |{self.goR}                | at the start of current line
            {while {> numSpaces 0}
              {self.insert-char cte-kSpace}
              {set numSpaces {- numSpaces 1}}}

              {self.goLineDown}       | go to next line
            {self.gotoLineStart}
            }}
        }}
    {define protected {CTETextDisplay.indentAll}:void
      | this is a very straightforward algorithm
      {let c:int=0  | curly count
        d:int=0  | line lenght holder
        k:int=0  | position holder

       {self.gotoFileStart}                          | first, goto start of file

       {while {< {self.get-current} self.length}    | for whole file
          {self.gotoLineStart}                       | goto start of each line
          | and delete leading spaces
          {while {= {aref self.txt {self.get-current}} cte-kSpace}
            {self.delNextChar}}

        {let i:int=c       | put in 2 spaces for each open curly in the count c
           {while {> i 0}
             {self.insert-char cte-kSpace}
             {self.insert-char cte-kSpace}
             {set i {- i 1}}}}

        {set d {+ 1 {self.chars-till-next-newline}}} | find this line's length

        {while {> d 0}                | count number of open curlies on this
line
           {set d {- d 1}}
           {set k {self.get-current}}
           {if {= cte-kCloseCurly {aref self.txt k}}
             {set c {- c 1}}
             {if {= cte-kOpenCurly {aref self.txt k}}
               {set c {+ c 1}}}}
           {self.goR}
           }

        {self.goLineDown}
         }}}

   | file movement
     {define protected {CTETextDisplay.gotoFileStart}:void
```

```
      {self.set-current 0}
      {set self.cursorx 0}
      {set self.cursory 1}
      }
    {define protected {CTETextDisplay.gotoFileEnd}:void
      | could be more efficient...but this is correct
      {while {< {self.get-current} self.length}
        {self.goR}}
      }

  | searching
    | note: {match pattern:text str:text start:int=0 len:int=-1}:int

    {define protected {CTETextDisplay.findPattern}:void
      | get a new pattern
      {let f:text={popup-text-query "Search: "}
        {if {void? f} {HBell}
          {begin
            {set self.pattern f}
            {self.nextPattern}}}}
      }
    {define protected {CTETextDisplay.nextPattern}:void
      | search forward for the current pattern
      {let p:int={match self.pattern self.txt start={+ {self.get-current} 1}
len={- self.length {self.get-current}}}
        {if {< p 0} {HBell} | nothing found
          {self.goR i={- p {self.get-current}}}}}
      }
    {define protected {CTETextDisplay.prevPattern}:void
      | search backwards for the current pattern
      {let p:int={match self.pattern self.txt start=0 len={self.get-current}}
        q:int
        {if {< p 0} {begin {HBell} {return}} | nothing found
          {begin
            {while {>= p 0}
              {set q p}
              {set p {match self.pattern self.txt start={+ p 1} len={-
{self.get-current} p}}}}
            {self.gotoFileStart}
            {self.goR i=q}
            }
          }}}

  | quitting and file stuff

    {define protected {CTETextDisplay.quitEditor}:void
      {if {and self.file-modified
          {eq? 'ok {popup-dialogue "Save File?" ok="Save" cancel="Don't Save"
reset=""}}}
        {self.write-file}}
      {if {eq? 'ok {popup-dialogue "Really Quit?" ok="Yes" cancel="No"
reset=""}}
        {exit}}  || this shouldn't really EXIT...just close window.
      }
```

```
    {define protected {CTETextDisplay.fileSave}:void
      {self.write-file}
      {set self.file-modified false}
      {output "File Saved."}
      }

    {define private {CTETextDisplay.write-file}:void
      {let op:OutputPort={self.file.write-open}
        {op.Seek 0 'start}
        {op.WriteText {self.get-text}}}
      }

    {define protected {CTETextDisplay.fileOpen}:void
      || save current file?
      {if {and self.file-modified
               {eq? {popup-dialogue "Save Current File?" ok="Save" cancel="No"
reset=""} 'ok}}
        {self.write-file}}
      || Open File
      {let f:text={popup-text-query "Open File: "}
        {if {void? f} {HBell}
          {self.open-file f}}}
      }

    {define private {CTETextDisplay.open-file f:text}:void
      {let u:File
        {try {set u {xresolve-filename f}}
          {catch {e:Exception} void}}
        {if {void? u} {begin {HBell} {return}}}
        {set self.file u}
        {letrec ip:InputPort={self.file.read-open}
          t:text={new text {ip.ContentLength}}
          {ip.Seek 0 'start}
          {ip.ReadText t {ip.ContentLength}}
          {self.set-text t}
          {self.gotoFileStart}
          {set self.file-modified 1}}}
      }

    {define protected {CTETextDisplay.editKeySequences}:void
      |{output "DEBUG: CTETextDisplay.editKeySequences"}
      || save current file?
      {if {and self.file-modified
               {eq? {popup-dialogue "Save Current File?" ok="Save" cancel="No"
reset=""} 'ok}}
        {self.write-file}}
      || Open File
      {let f:text="cte-keysequences.curl"
        {if {void? f} {HBell}
          {self.open-file f}}}
      }
    {define protected {CTETextDisplay.editReserveWords}:void
      || save current file?
```

```
        {if {and self.file-modified
                 {eq? {popup-dialogue "Save Current File?" ok="Save" cancel="No"
reset=""} 'ok}}
          {self.write-file}}
        || Open File
        {let f:text="cte-reservewords.curl"
          {if {void? f} {HBell}
            {self.open-file f}}}
        }
      {define protected {CTETextDisplay.help}:void
        || save current file?
        {if {and self.file-modified
                 {eq? {popup-dialogue "Save Current File?" ok="Save" cancel="No"
reset=""} 'ok}}
          {self.write-file}}
        || Open File
        {let f:text="cte-helpfile.curl"
          {if {void? f} {HBell}
            {self.open-file f}}}
        }
      {define protected {CTETextDisplay.reloadEnvironment}:void
        |{output "DEBUG: CTETextDisplay.reloadEnvironment"}
        {let u:File
          {try {set u {xresolve-filename "cte.curl"}}
              {catch {e:Exception} void}}
          {if {void? u} {begin {HBell} {return}}}
          {try {load u curl-env}
            {catch {e:Exception}}}
          }}

    | other functions

      {define public {pointer-press e:PointerEvent}:void
        {request-key-focus self}
        {if {not {isa Text {get-hit e}}} {return}}
        {let r:Text={cast Text {get-hit e}}
          x:int
          y:int
          {set x y {transform-coordinates self e.x e.y r}}
          {goto-word-doc {r.pick-word x y}}}}

      {define public {CTETextDisplay.draw gc:GraphicContext}:void
        {let font:FFont=self.font
          font-spacing:int={+ font.ascent font.descent font.letting}
          m:text
          m-length:int
          o:int
          y:int
          line:int=1
          r:FRect={gc.get-clip-rect}

          {set m self.txt}
          {set m-length self.length}
          {set y font.ascent}
```

```
        {gc.set-font font}

          | optimize based on clip rect

          | NOTE: font method variables
        | space   spacing between words
        | ascent  max height above baseline
        | descent max descent below baseline
        | letting extra vertical spacing between lines

        || ***************** draw the cursor
        |helpful debug: {output "ascent " font.ascent " descent " font.descent
" letting " font.letting}

        {gc.draw-rect {* self.cursorx font.space}
          {- {* self.cursory font-spacing} font.ascent}
          {* {+ self.cursorx 1} font.space}
          {* self.cursory font-spacing}
          cte-kCursorFGColor cte-kCursorRimColor 1}

        | ************** draw the curly brace references if necessary
        | put a box around the current curly and the corresponding
        | curly with a connecting line

        {if {or {= cte-kCloseCurly {aref self.txt {self.get-current}}}
            {= cte-kOpenCurly {aref self.txt {self.get-current}}}}
          {begin
            |{output "DEBUG: curly matching"}

             { let x:int=self.cursorx
              y:int=self.cursory
              c:int=1
              mx:int
              my:int
              save-cursorx:int=self.cursorx
              save-cursory:int=self.cursory
              save-current:int={self.get-current}

            | find the matching curly
              {if {= cte-kCloseCurly {aref self.txt {self.get-current}}}
                | we have a close curly
                {begin
                  {while {> c 0}
                    {self.goL}
                    {let k:int={self.get-current}
                      {if {= cte-kOpenCurly {aref self.txt k}}
                        {set c {- c 1}}
                        {if {= cte-kCloseCurly {aref self.txt k}}
                          {set c {+ c 1}}}}
                      {if {< k 1} {break}} | no match
                      }}}
                | we have an open curly
                {let k:int={self.get-current}
                  e:int={- self.length 1}
```

```
                {while {> c 0}
                  {self.goR}
                  {set k {self.get-current}}
                  {if {= cte-kCloseCurly {aref self.txt k}}
                    {set c {- c 1}}
                    {if {= cte-kOpenCurly {aref self.txt k}}
                      {set c {+ c 1}}}}
                  {if {> k e} {break}} | no match
                  }}}

              {set mx self.cursorx}
              {set my self.cursory}

            | draw bounding boxes and connecting lines
              {gc.draw-rect {* x font.space}
                {- {* y font-spacing} font.ascent}
                {* {+ x 1} font.space}
                {* y font-spacing}
                cte-kCurlyBGColor cte-kCurlyFGColor 1}
              {gc.draw-line {* {+ x 0.5} font.space}
                {- {* y font-spacing} font.ascent}
                {* {+ mx 0.5} font.space}
                {- {* my font-spacing} font.ascent}
                cte-kCurlyFGColor 1}
              {gc.draw-rect {* mx font.space}
                {- {* my font-spacing} font.ascent}
                {* {+ mx 1} font.space}
                {* my font-spacing}
                cte-kCurlyBGColor cte-kCurlyFGColor 1}
              | restore state
              {set self.cursorx save-cursorx}
              {set self.cursory save-cursory}
              {self.set-current save-current}
              }
            {set self.curlymatching false}
            }
          }

        | ************** draw the text
        {while {and {< o m-length} {<= {- y font.ascent} r.bottom}}
          {let next-o:int
            {set next-o {self.find-newline m o m-length}}
            {if {>= {+ y font.descent 1} r.top}
                    {begin
                       {if {= {aref m o} #\newline}
                        {set o {+ o 1}}}
                      {self.draw-line-of-text gc r y m o next-o m-length font.space}}}
            {set line {+ line 1}}
            {set y {+ y font-spacing}}
            {set o next-o}}}
        }}

    {define private {CTETextDisplay.draw-line-of-text gc:GraphicContext r:FRect
y:int m:text o:int next-o:int m-length:int xinc:int}:void
```

```
       | draw method. draw everything in blue, but change the color to green if
there's a comment
       | then, redraw over that in red any of the reserved words in the file cte-
reservewords.
       {gc.set-color cte-kGenericTextColor}
       {let x:int
         save-o:int=o
          c:char
          comment:char={cast char 124}
         j:list=gCTEReserveWords
         i:list  | iteration list for the reserved words
         w:text

             |whitespace characters
         reserved:bool=false | are we INSIDE a reserved word
          newline:char={cast char 10}  | used to END a line
          tab:char={cast char 9}
          return:char={cast char 13}
          space:char={cast char 32}

         {while {and {< o m-length}
                {<= {- x xinc} r.right}}
           |{gc.draw-text 0 y m o {- next-o o}}
           {set c {aref m o}}
           {if {eq? c comment} {gc.set-color cte-kCommentTextColor}}
           {if {eq? c newline} {break}}
           {gc.draw-char x y c}
           {set x {+ x xinc}}
           {set o {+ o 1}}
           }

        {gc.set-color cte-kReserveWordTextColor}                | set the color
for redrawing

         | now, for each of the reserved words, redraw in color if it is there
         | note: {match pattern:text str:text start:int=0 len:int=-1}:int

         {while {not {eq? j '{}}}
           {if {eq? symbol {typeof {car j}}}
             {gc.set-color {car j}}
             {begin
               {set i {car j}}
               {while {not {eq? i '{}}}                                  | for
all of the reserved words in the list
                 {set w {car i}}                                        | where
w = each reserved word
                 {set o save-o}                                          | ie:
o is the beginning of the line
                {set x {match w m start=o len={- next-o o}}}            | check
if w is on this line and assign x=pos
                 {while {> x 0}                                          | x<0
means it wasn't there, but if it is...
                   {if {and {self.checkChar {aref m {- x 1}}} {self.checkChar
{aref m {+ x {length w}}}}}}
```

```
                    {gc.draw-text {* {- x save-o} xinc} y w 0 {length w}}}  |
draw over IF it's surrounded correctly
                   {set o {+ x {length w}}}                                  |
trickiest thing is keeping track of o and save-o
                   {set x {match w m start=o len={- next-o o}}}
                   }
                 {set i {cdr i}}}}}
           {set j {cdr j}}}
         }}

      {define private {CTETextDisplay.checkChar c:char}:bool
        {let k:int = {cast int c}
          {if {or {= k 61}  | =
              {= k 123} | {
                {= k 125} | }
              {= k 58}  | :
              {= k 10}  | linefeed C-j
              {= k 13}  | CR       C-m
              {= k 9}   | tab
              {= k 32}  | space
              {= k 34}  | "
              {= k 124} | |
              }
            {return true}
            {return false}
            }}
        }

      {define private {CTETextDisplay.chars-till-prev-newline}:int
        || return the number of characters between the current
        || position and the newline before the current position
        {let o:int={self.get-current}
           m:text=self.txt
           |m-length:int=self.length

          |{set o {- o 1}}
          {while {> o 0}
            {set o {- o 1}}
            {if {eq? {aref m o} #\newline}
              {return {- {- {self.get-current} o} 1}}} | handles regular lines,
buffered with a newline at start
            }
          {return {- {self.get-current} o}} | handles first line of buffer
          }}

      {define private {CTETextDisplay.chars-till-next-newline}:int
        || return the number of characters between the current
        || position and the newline after the current position
        {let o:int={self.get-current}
           m:text=self.txt
           m-length:int=self.length

          {set o {+ o 1}}
          {while {< o m-length}
```

```
        {if {eq? {aref m o} #\newline}
              {break}}
        {set o {+ o 1}}}
      {return {- {- o {self.get-current}} 1}}
      }}

  }
```

A.10 run-double-windows.curl

```
{include "cte.curl"}

{letrec f2:text="cte-helpfile.curl"
  f1:text="samplefile.curl"
   c1:CTETextDisplay={new CTETextDisplay f1}
   c2:CTETextDisplay={new CTETextDisplay f2}
  s1:ScrollBox={ScrollBox {new WordPickFrame c1 word-picker=goto-word-doc}}
  s2:ScrollBox={ScrollBox {new WordPickFrame c2 word-picker=goto-word-doc}}

   v:View={new View
    {spaced-vbox border-width=2 border-style='black
      s1 s2}
        {browser-view}
        curl-window-type="source file"
        width=800
        height=800}
  {v.show}
  {request-key-focus c1}
  }
```

A.11 run-with-scrollbar.curl

```
{include "cte.curl"}

{letrec f:text="cte-helpfile.curl"
   c:CTETextDisplay={new CTETextDisplay f}
  s:ScrollBox={ScrollBox {new WordPickFrame c word-picker=goto-word-doc}}

   v:View={new View
    s
        {browser-view}
        curl-window-type="source file"
        width=700
        height=500}
  {v.show}
  {request-key-focus c}
  }
```

A.12 run.curl

```
{include "cte.curl"}

{letrec f:text="samplefile.curl"
        c:CTETextDisplay={new CTETextDisplay f}
        v:View={new View c
                    {browser-view}
                    curl-window-type="source file"
                    width=900
                    height=1000}
  {v.show}
  {request-key-focus c}
  }
```

A.13 samplefile.curl

```
* Copyright (c) 1998  MIT LCS -- Jon Heiner *


`1234567890-=qwertyuiop[]\asdfghjkl;'zxcvbnm,./ (no return)
~!@#$%^&*()_+qwertyuiop{}|ASDFGHJKL:"ZXCVBNM<>? (no return)

(now enter the following characters with the control key down:)
`1234567890-=qwertyuiop[]\asdfghjkl;'z(skip the x!!!)cvbnm,./
~!@#$%^&*()_+QWERTYUIOP{}|ASDFGHJKL:"Z(agsain...skip)CVBNM<>?

(now enter the following characters)
esc
backspace
tab
return
space

(now enter)
C-X
C-x

|=================================================================
{define {fubar}     | our sample function
  {let x:int=1      | test for repetition on int int and int
    {if {> x 0}      | and test for and and for for
      {output "hello, world"} | this prints stuff
      }}}

{{curly braces} can {be {a {problem
without a {good}} indication of
where} {{matching} pairs}} {begin
and where} they end.}}

| abs anchor != % Buffer View output define define output

!= % * + - / < <= > >= Buffer Button CheckButton Circle ClickBox
DragContainer DragImage Dragee DropTarget EventHandler Exception
FileBox FileInputPort FileOutputPort Frame HashTable Hbox IndentedBox
```

```
InputPort Line ListBox MenuAction MenuBar OutputPort PopupMenu Port
RadioButton RadioButtons Rectangle ScrollBox Selectee StringBuffer
SubMenu Table TextField Vbox View – abs anchor and any append
append! apply apply-method arccosine arcsine arctangent arctangent2
aref array assq big bit bit-and bit-or bit-sll bit-sra bit-srl
bit-xor block bold br break cadr car cast catch cddr cdr center
char choose choose-randomly choose-sequentially code cond cons
cons* continue copy-list cosine define define-class define-constant
define-form define-macro define-variable delq! description docref
documentstyle dotimes enumerate eq? eqn equal? error eval example
exponential file-write-date finally float for format get-buffer
get-buffer-of-size get-string-buffer get-string-buffer-of-size
hrule if image import include input-var int int16 int8 invoke-method
isa italic itemize keyword-supplied? lambda last lastcdr let letrec
list list-length loop make-list map map! match match-prefix max memq
min neq? new nobreak not nth nthrest or output paragraph popup-dialogue
popup-graphic popup-text-query power print proc quote random
release-buffer release-string-buffer require rest-as-list
rest-as-vector return reverse reverse! s section set sine sleep
small sort-list sort-list! stderr stdout sublist subsection
subsubsection subtext symbol tagged-block tagged-loop tangent
text text->symbol text-append text-compare text-equal? text-fill
text-hash text-pos text-replace text-rpos throw tiny try typeof
unless until value vector vector->list verbatim void void? vrule
walk when while write}
```

References

Texts:

Cornell, Gary & Horstmann, Cay S. (1997). Core Jave: second edition. USA: Sunsoft Press.

Glickstein, Bob. (1997). Writing GNU Emacs Extensions: Editor Cusomization and Creations with Lisp. Cambridge, MA: O'Reilly & Associates, Inc.

Stallman, Richard M. (July 1996). GNU Emacs Manual; 12th edition. version 19.3. Boston, MA: Free Software Foundation.

Ward, Stephen A & Halstead, Robert H. (1990). Computational Structures. Cambridge, MA: MIT Press.

Primary Sources:

Stallman, Richard M. (July 18, 1997.) Manhattan Sammy's Deli.

On-Line:

Byrd and Block Communications, Inc. (September 23, 1997) More Info - Flight of the Bumblebee. [WWW Document] URL http://www.earthsky.com/1997/esmi970923.html.

Carroll, Lewis. (1998). Alice's Adventures in Wonderland. [WWW Document] URL http://www.bibliomania.com/Fiction/Caroll/CompleteWorks/index.html.

Peterson, Ivars. (March 3, 1997). Flight of the BumbleBee. [WWW Document] URL http://www.maa.org/mathland/mathland_3_31.html.

Raymond, Eric S. & Steele Jr., Guy L. (July 24, 1996). The Hacker Jargon File. [WWW Document] (version 4.0.0). URL http://earthspace.net/jargon/jargon_toc.html.

Tenniel, Sir John. (1998). The Cheshire Cat. [WWW Image]. http://www.math.umn.edu/~rudnaya/books/alice23a.gif.

Zetie, Ken. The Strange Case Of The Bumble Bee Which Flew. [WWW Document] URL http://users.ox.ac.uk/~zetie/story/bees.txt.