Adding Syntax Parameter to Sweet.js macro library for JavaScript

A Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Vimal Kumar

May 2015

The Designated Project Committee Approves the Project Titled

Adding Syntax Parameter to Sweet.js macro library for JavaScript

by

Vimal Kumar

APPROVED FOR THE DEPARTMENTS OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

May 2015

Dr. Thomas Austin	Department of Computer Science
Dr. Chris Pollett	Department of Computer Science
Dr. Ronald Mak	Department of Computer Science

ABSTRACT

Adding Syntax Parameter to Sweet.js macro library for JavaScript

by Vimal Kumar

Macros have a long history in computing, Mozilla Sweet.js provides a way for developer to enrich the JavaScript by adding new syntax to the language through the use of macro. Sweet.js provide the possibility to define the hygienic macros inspired by Scheme. In this paper I present the implementation of the Syntax-Parameter feature to SweetJS library. Syntax parameter is a mechanism for rebinding a macro definition within the dynamic extent of a macro expansion. In implementation I defined the "syntaxparam" which define and bind the syntax parameter part of the compiler, "syntaxLocalValue" which pull the syntax parameter definition in the defined scope and "replaceSyntaxParamTransform" which transform to the identifier within the macro body using syntaxLocalValue.

ACKNOWLEDGMENTS

I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals. I would like to extend my sincere thanks to all of them.

I am highly indebted to Dr. Thomas Austin and Tim Disney, for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

TABLE OF CONTENTS

CHAPTER

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

The Basics

LaTeX can be viewed as a compiled programming language, in contrast to that nightmare known as Microsoft Word, which can be viewed as an interpreted language. So, to typeset a document in LaTeX, you create a text file that has the `.tex` extension. This file includes some special commands, known as macros. Then you compile your `.tex` file by running LaTeX, which produces your typeset document, as a pdf.

In this paper, a few basics are discussed. There are plenty of good online resource if you need help with more advanced topics.

1.1 Typesetting Text

To typeset text, you type whatever you want. Multiple spaces are ignored when typesetting, and the end of a line is treated as another space. Consequently, when you are typing, you can break lines anywhere, like here or here, since the lines are formatted automatically when you typeset the document. You start a new paragraph by leaving a blank line.

See how easy it is to start a new paragraph? A blank line does the trick.

1.2 Special Characters

Typesetting text is generally pretty easy. However, there are some special characters that will not be typeset as you might expect. In the remainder of this section we consider some of the most common of these special characters.

The backslash "\" is used as the "escape" character, meaning that whatever

follows a backslash is interpreted as a macro. For example, when `\LaTeX` is typeset, it looks like LaTeX, which is a lot different from LaTeX.

To get double quotes, use two single quotes. That is, the left double quote is ``, while the right double quote is ''. When you do it correctly, quoted text looks "like this." If you use the double quote key, you will always get right-quotes, which looks "like this," and is almost certainly not what you want.

A tilde "~" is used as a "tie," that is, a space is inserted, but no line break can occur. For example, you might type Dr. Stamp just to be sure that the line of text does not break between Dr. and Stamp, as it otherwise might.

The percent sign is used for comments—everything following a percent sign on a given line is ignored when you LaTeX your file. If you want a percent sign to appear in your document, use `\%`, which will give you this %.

The dollar sign also has special meaning, since it is used to start and end math formulas. To typeset a dollar sign, use `\$`, like this $.

To force LaTeX to insert a space, use a backslash followed by a space, that is, `\ `. You can put in multiple extra spaces if you want.

1.3 Fonts

To change fonts, enclose the text in curly brackets and give the appropriate font command. For example, to italicize text, *do this*, and to get boldface, **this is the ticket**. Another useful font is `this one`, which produces a typewriter-like font.

1.4 Math Basics

Math typesetting is a big, big, big topic—here we just cover some of the most basic issues. For more information, look online.

The dollar sign is used to start and end math formulas, like πr^2. If you want your formula displayed on a line by itself, then double dollar signs

$$d(X, Y) = \sum_{i,j} |x_{ij} - y_{ij}|$$

are your friend.

In math formulas, text gets typeset as math symbols. To insert regular text in a math formula, you can use the `\mbox` command. Here is an example of a displayed equation with text

$$\binom{n}{26} 26!\, 26^{n-26} < 26^n \approx 2^{4.7n} \text{ is a pretty formula.}$$

1.4.1 Numbered Equations

One of the most useful features of LaTeX is its symbolic cross-references. What this means is that you can give names to equations, figures, tables, citations, etc., and refer to those equations, figures, tables, citations, etc., by their names. Then when you LaTeX the document, the correct numbers will magically appear in place of the names. This is very convenient when you move things around or you insert or delete stuff. You should definitely use this feature as much as possible.

In this section, we discuss numbered equations, Then in Chapter 2 we consider other examples of symbolic cross-referencing. Again, this is a feature you should use, since it will save you a lot of time in the long run.

To typeset a displayed equation with a number, you can use `\begin{equation}`

to begin the equation, and `\end{equation}` to end the equation. You also want to provide a label for the equation. For example,

$$\sum_{i=1}^{26} m_i \left(n - \sum_{j=1}^{i} m_j \right) = n^2 - \sum_{i=1}^{26} m_i \sum_{j=1}^{i} m_j = \sum_{i=1}^{25} \sum_{j=i+1}^{26} m_i m_j \tag{1}$$

gives us a numbered equation.

Note that labels can be used for just about anything that is numbered. Generally, to refer to a label, we use `\ref{label}`, but for equations, you probably want to use `\eref{label}`, since that will put parenthesis around the number.

So, we can now refer to equation (1) anywhere in the paper. Better yet, if we move, insert, or delete text, the numbering will still be correct.

1.4.2 Last Word on Equations

Again, typesetting equations is a big topic and we have only given the most basic of basics here. Just remember that almost anything is possible, and there are plenty of good resources available.

1.5 Lists

Numbered lists are not difficult. Here is an example of a numbered list:

1. Let `score` $= \infty$
2. Construct an initial `putativeKey`
3. Parse the ciphertext using `putativeKey`
4. Compute `newScore` based on the resulting putative plaintext
5. If `newScore` $<$ `score` then let `key` $=$ `putativeKey` and `score` $=$ `newScore`

6. Modify `key` to obtain a new `putativeKey`

7. Goto 3

Bulleted lists are similar to numbered lists. Here is an example of a bulleted list:

- How can we determine an initial putative key?
- How can we systematically modify the key?
- Given a putative key, how can we compute a score?

1.6 Verbatim

To force LaTeX to typeset exactly what you type, you might want to use the `\verb` macro. Here is an example: `four score  and seven`. Note that everything between the "+" signs is typeset as it appears.

If you happen to have a lot of verbatim text, you can use the verbatim environment, like this:

```
Four score
and seven
years
ago.........
```

Generally, verbatim should be used sparingly, if at all. But, since verbatim text has already appeared several times in this document, I thought it should be explained.

CHAPTER 2

Symbolic Cross-References

This chapter has more information on symbolic cross-referencing. In addition, there are examples showing how to typeset a figure and a simple table.

2.1 Figures

Figure 1 has been typeset so that it occupies half the width of text on a page (`width=0.5`). Note the symbolic cross-reference, that is, you simple `\ref{fig:encrypted_virus}` to obtain the appropriate number for Figure 1.

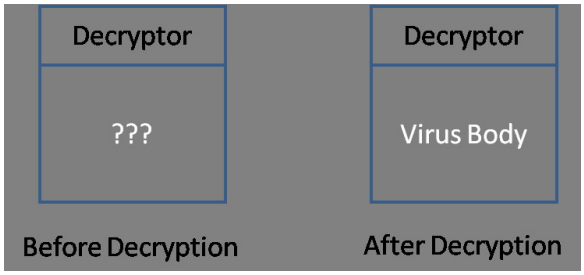

Figure 1: Encrypted virus before and after decryption.

If you move things around, insert or delete text, Figure 1 will still give you the correct number. This is a very handy feature.

2.2 Tables

Table 1 shows some interesting stuff. I hope you enjoy it.

2.3 Citations

References are cited using the `\cite` command. For example, see [?] for information on the Zodiac killer. You can include more than one reference in a single

Table 1: MysteryTwister Zodiac Challenge

Case	Description	Percentage
1F	Fast outer hill climb and English stats	13.00%
1S	Slow outer hill climb and English stats	4.00%
2F	Fast outer hill climb and Zodiac 408 stats	70.00%
2S	Slow outer hill climb and Zodiac 408 stats	84.00%

citation, like this [?, ?, ?, ?, ?, ?, ?, ?, ?] or this [?, ?].

2.4 The Last Word

There are a lot of other uses for symbolic cross-referencing. For example, this chapter title has a label so I can refer to Chapter 2 (as I just did). My recommendation is that you include a label with anything and everything that is numbered. If it has a number, there is a good chance that you will refer to it at some point.

Finally, note that you need to LaTeX a file at least twice (three times to be safe) without changing anything to be sure that the numbering is correct. Why is that? Good question. When you LaTeX your document, various auxiliary files are created. Most of these files contain information related to cross-references that appear in your `.tex` file (or files). The next time you LaTeX your document, the information from the previous run is read from these auxiliary files. Consequently, after making changes to any `.tex` file, you will need to LaTeX your document at least twice to be sure that the cross-reference information is updated in the resulting pdf. In fact, you should run LaTeX three consecutive times to be certain that cross-references are correct.

CHAPTER 3

The Files

For this thesis format, you LaTeX the file `thesis.tex`. But first, you need to make some modifications to `thesis.tex`. The title of your report, committee members, etc., are specified in `thesis.tex`. All of the things that you need to modify are indicated by comments beginning with five consecutive asterisks, so search for "*****" in `thesis.tex` and make the necessary changes.

You put the actual content of your report in the following files:

- `abs.tex` — abstract
- `ack.tex` — acknowledgements
- `chap1.tex`, `chap2.tex`, and so on — chapters
- `bib.tex` — references
- `appA.tex`, `appB.tex`, and so on — appendices (if any)