*Def*DOC Overview

A TEX-inspired, Lisp-based document processing system

by Rahul Jain

$Id : overview.tex, v1.22004/02/1420 : 26 : 22rjainExp$

0.1 Notes

This is a work-in-progress. Please submit any corrections, recommendations, or criticisms to Rahul Jain <rjain@common-lisp.net>.

0.1.1 Trademarks

0.1.2 Copyrights

Chapter 1

General Concepts

Def Doc is the result of my experience with both TeX and Lisp. The conceptual foundations of TeX, specifically the ability to define macros to simplify repetitive formatting and the beautiful final product, are immensely useful and desirable.

1.1 TeX

Unfortunately, the syntax, the overall document model, even after being extended by LaTeX, and the memory model often keep the ideals of TeX unrealizable in practice.

1.1.1 Syntax

For example, the lack of abstraction in the syntax prevents most people, including me, from really understanding how to write macros that do non-trivial processing of their contents. Furthermore, the design of TeX as primarily a document markup language makes writing code for these macros very confusing and aesthetically unpleasing, in the opinions of many.

1.1.2 Document Model

The document model of TeX is very primitive. It is simply a nested sequence of vboxes and hboxes, which gets the job done very well, but some ability to define specialized types of these boxes instead of simply adding properties to them would be desirable. Instead of using a macro system to extend the document model, we could have the option of using a type system for the cases where it is more suitable.

1.1.3 Memory Model

The memory model used by TeX is essentially a large, statically-allocated set of arrays for each of the various data structures. For linked lists, there is a reference counter and the list cells have type tags to indicate what exact kind of list they are part of. The sizes of the arrays is fixed at compile-time, so if your document is too large or complex for the sizes fixed in your executable, you must modify the source and increase the defined constants. This process can become very cumbersome for a novice computer user who just wants to be able to publish a book.

1.2 Lisp to the rescue

For each of these problems, Lisp has a solution.

1.2.1 Syntax

Lisp was originally designed for symbolic processing. Syntax manipulation is the most commonly used aspect of Lisp's symbolic processing capabilities currently. The `defmacro` form provided by Common Lisp allows one to define new language forms which have full access to the symbolic information of the code within their bodies. Within the macro definition, the entire facilities of the Common Lisp language may be used, including any other functions or macros you have defined. This model is extremely powerful, and many advanced techniques for using it are explained in Paul Graham's book, *On Lisp* [Gra93].

1.2.2 Document Model

The document model *Def* Doc uses is based on the Common Lisp Object System (CLOS). Every document element is an object with a type that determines how that part of the final output document is created. This conversion function is a multimethod, which means that the methods of it can dispatch on either the document element's type or that of the output format or both, and then reuse the implementation of the methods that specialize on the superclasses of the types on which that method specializes. For further information about using CLOS, a good general introduction is Sonya Keene's book, *Object-Oriented Programming in* Common Lisp [Kee89].

1.2.3 Memory Model

The Lisp memory model requires dynamic management of the storage heap. There are no required behaviors as far as allowing unreferenced objects to not waste memory, but all implementations support some form of garbage collection. Therefore, there is no need to implement any type of memory management in *Def* Doc, since we can assume that the host implementation does a suitable job for the user who chose it.

Chapter 2

Syntax

The *Def*Doc syntax will be defined in terms of a modified Lisp reader. The tilde (~) character will be used as a "dispatch macro character" to define special *Def*Doc syntactic forms. The backslash (\) character will be a single escape character, which causes the reader to interpret the immediately following character as though it were a normal text character.

2.1 Code vs. Text

The syntax will be defined such that the reader will know when a specific form is code or text, so that TeX's problem of unsightly escaping in code will not be an issue.

Chapter 3

Document Model

Like TeX, the basic document elements are boxes of elements which are arranged in a vertical sequence or a horizontal one. TeX calls these vboxes and hboxes, respectively.

Bibliography

[Gra93] Paul Graham. *On Lisp*. Prentice Hall, 1993. Available at `http://www.paulgraham.com/onlisptext.html`.

[Kee89] Sonya E. Keene. *Object-Oriented Programming in* Common Lisp. Addison-Wesley, 1989.