position and potetially create the appearance of compromising a NDA.

I still want to look at both borland (focused on their cost structure and agility as well as their windows product capability) and lotus. (cost structure, ability to sustain their agressive marketing, and probable product plans/strastegies)

I suggest we have jeffr do lotus and you do Borland. What think?

Mail-Flags: 0001
From mikemap Sat Jan 26 14:27:27 1991
To: chasst fredg peteh
Subject: Borland
Date: Sat Jan 26 14:27:24 1991

I am doing a competitive analysis of Borland for the Exec Retreat. I would like to have someone from each of your areas that specializes in Borland work with me. Who should it be?

Mail-Flags: 0001
From mikemap Mon Jan 28 07:45:01 1991
To: betsyd
Subject: Director task
Date: Mon Jan 28 07:45:00 1991

Noticed in all of the UR ladder info there were not time guidelines. Could you and company review and recommended minimum, average and max time for movement from each level to the next. Thanks.

Mail-Flags: 0001
From mikemap Mon Jan 28 07:45:08 1991
To: gerardba
Cc: chrisp
Subject: Win Word Macro Programming
Date: Mon Jan 28 07:45:05 1991

who is preparing a response / analysis of this?

>From nathanm Wed Jan 23 13:55:49 1991
To: adrianw chrism gerardba gregs peterj
Cc: billg bobatk darrylr edwardj gregw jeffr karenh markz mikemap tonyw
Subject: Win Word Macro Programming
Date: Wed Jan 23 14:52:11 PDT 1991

I've been working on a couple of huge memos recently and I decided I needed a command that Win Word didn't have. No problem, I thought (naievly) I'll roll up my sleeves and write a Word Macro. Well, after spending a lot of time on this, both experimentally and reading the technical reference I've reached some conclusions:

- The Word macro facility dramatically misses the mark in terms of power and funcitonality. For a wide range of things that should be easy to do, it is useless.

- In particular, it does not appear to have even the simplest concepts that

other sucessful text editor macros have had for years. Emacs, to name one, has an enormously better programming model (discussed below).

- I really hate to say "I told you so", but in point of fact I did bring this up a couple of years ago. I suggested a specific project for somebody to take a set of 3rd party emacs macros, and some Brief macros and either implement them in Opus, or at least make sure you could. This clearly was never done.

- Finally, I think that this has a lot of implications for our concept of "IDT"s and factoring apps into objects which can be manipulated by a macro language. Win Word appears to me to be an example of exactly the wrong way to do this - going through the motions of exposing features to a macro, and incurring the cost and complexity of doing this, but doing so in such a way that you totally miss the real power of programmibilty. All of the cost with little of the gain.

By now you may be thinking that this is just flame mail from some guy that got pissed at the product. Not so. I really love Win Word, and I suspect that I use it as much if not more on a daily basis than anybody on the To: or CC: line. I just want to see it be more useable.

Here is the original problem. I want to have references in a technical report. I could do this using the Word concept of footnotes, and just put the footnotes at the end of the document. The trouble is that I wanted to have ordinary footnotes at the bottom of the page as well. In effect I wanted to have two different clases of "footnote". Since I wanted to have my references at the end of the document I did not need to have both of them use the special formatting aspect of page bottom footnotes. I thought this would be easy.

My first assumption was that I would just copy the InsertFootnote command and edit it to make another, similar macro. I *assumed* that InsertFootnote would look something like this:

```
Prompt user for footnote-reference (auto-numbered or typed by user)
Format footnote-reference (using style in style guide)
Insert footnote-reference at cursor
Create a footnote buffer (place for user to type footnote contents)
Tell system that this buffer is linked to the reference in document
If (footnotes at bottom of page)
        Tell system this buffer must be on same page as reference
else if (footnotes at end of document)
        Tell system to put this buffer at end of document (in order)
Move cursor to buffer
Insert footnote-reference
Postion cursor in buffer
Let the user type
```

Each of the individual lines would probably be a special purpose function. Clearly there is some special functionality to deal with the formatting of footnotes when they are at the bottom of the page, how to spill them across to other pages etc. Nevertheless, the basic operations of setting up the user interface to footnotes is independent of this.

I guessed that Annotations would probably be very similar - there wolud be just a couple of differences between the InsertAnnotation macro and the InsertFootnote macro. Annotations are just another weird class of footnotes which are kept in a separate buffer which is not printed by default.

I was wrong. InsertFootnote is not implemented in the Word macro language - there is a trivial macro which just calls a built in function. This is a symptom that something is terribly wrong, because it SHOULD be able to be implemented via something much like the procedure above - there is no performance or other reason to not do this.

I pressed on and thought hey, its a bad sign that they didn't use it themselves, but it isn't by itself fatal. I'll just write the equivalent of the steps above myself. I immediately ran into a number of problems because none of the fundamental constructs seem to be exposed. Even if there is some nice way to solve my references problem that I overlooked, this is a terrible situation, hence this email.

I've written literally thousands of lines of Emacs macros, and in emacs you would do something almost exactly like the steps I outlined above. The reason that this is possible in Emacs and not in Word is that Word does not seem to expose the right granularity of funcitons, and does not seem to have been designed to program. The Emacs architecture is based on the concept of a buffer - a piece of text which can be manipulated with the full power of the editor. It has functions in the following categories:

1. Basic buffer functions to create and delete buffers, read a file into a buffer, save a buffer as a file, and invoke the editor on a buffer.

2. There are a set of functions between buffers (such as insert-buffer-at-cursor which puts the contents of one buffer into a specified place in another buffer). These are not many of these, but they are powerful.

3. Inside of a buffer there are a variety of simple built in editing functions for manipulating text. These functions can be bound to keys. (As an aside, normal keys and function keys are treated the same way - for example normal text entry keystrokes are handled by binding the InsertChar("a") function to the "a" key).

4. There are also a variety of complex functions for editing inside of a buffer. As much as possible, these are implimented in the Emacs macro language using primitive built in functions, but in some cases they need to impliment a built in function in Emacs itself (such as search-buffer-for-string. Note that replace isn't built in). Users can write their own macros, or can record a set of keystrokes).

This sounds very straightforward, but it is amazingly powerful. I have seen an entire email package - similar in functionality to Wzmail or WinMail - which was implimented entirely as an Emacs macro. This is not a weird stunt either - thousands of UNIX users in university use that mailer daily. I have personally written extensive programming language support (automatic code structuring etc) for Emacs - without there being ANY special support in Emacs for doing this.

The problem with Win Word, as I see it, is that we do not seem to support 1 or 2 - the buffer functions, nor do we seem to export the right granularity of built in functions (4) so that people can really do interesting things. That is what makes a system programmable - and also EASY to program. Despite the fact that Emacs uses lisp for its macro language syntax (which is abomnible), it is MUCH easier to program - somebody thought about programming it and provided a conceptual model. Macros in Win Word, by comparison, seem to be something that was added in order to "go through the motions" of adding a macro language.

Perhaps the necessary support is there, but I couldn't find it in the manual, the technical reference, or several 3rd party books on Win Word. (As an aside, another great thing about Emacs is that you can just browse the macro code that impliments the default commands see how they are implemented in order to write your own.)

Emacs is not unique in providing this - I just happen to know it best. Brief, and even the Z editor written HERE AT MICROSOFT has a better macro programming model than WinWord does. I am not an Emacs fanatic either - it is terrible at lots of things - but it is probably the most programmable editor around, and it had all of this years before WinWord ever started as a project (yes, even before Cashmere!).

Note that there are a number of red herrings which arose the last time I suggested we look at the Win Word macro programming model:

- This is NOT harder to do, and it is NOT a performance problem. There is really no technical excuse. It might be hard to retrofit now that we have done it another way (even that I doubt). I actually think that a clean internal programming architecture is substantially EASIER to develop than the ad hoc approach we appear to have.

- It has nothing to do with GUI. Emacs is character oriented, but this does not matter - we are talking about the fundamental programming architecture. There will have to be some special built in functions to support the Word formatting engine, but I actually think these are quite easy to specify.

- It is not a niche thing. It happens that the editors which have the best progammability features were written by programmers for programmers. This does not mean in any way that this functionality is tied to programmers, or to some niche market where people implement mailers inside their editor. In point of fact the model that I am suggesting is EASIER to program than what you already have, as well as being significantly more powerful. If you think macros are important, then do them right.

- Finally, I suspect that there are some ways that you could stand on your head and write a macro to solve my specific problem (I think I have a good idea how to do it, as a matter of fact). The point that I am trying to make is that the progamming model is not well thought out and isn't general enough to make this easy. Even if my specific case can be solved with some hack, that isn't the point - giving an easy and complete model for macro programming is.

I hope that we can do a better job in Pyramid. I think that there is a ton of benefit to be derived here just in terms of giving end users more power. There is also the prospect of getting 3rd parties to write really nice stylesheet/ macro add ons. Finally I think that there is an opportunity in making macro programming substantially easier for the end user by doing it visually (this is not something that Emacs or other editors do - yet). First however you have to make macro programming possible - which it really isn't today in a meaningful sence.

It is also essential that we think about this when exposing macros in other products, and in desiging our central macro language strategy.

Nathan