M.I.T. Laboratory for Computer Science

Request for Comments No. 226
June 8, 1982

# SWAN -- A New Text Formatter

by Wayne C. Gramlich

## 1. Introduction

A project has been started to write a new text formatter in CLU. The name of this new formatter is SWAN. A swan is a beautiful elegant bird. It is hoped that SWAN will be a beautiful elegant text formatter. SWAN will be a batch oriented text formatter that is in many respects similar to Scribe. However, no attempt will be made to be 100% upward compatible with Scribe.

## 2. Goals

The initial goals of this formatter are fairly modest.

- SWAN should have good documentation. In particular, it should have a good reference manual and complete documentation of all the different document types. Eventually, it should also have a tutorial and a programmer's manual.

- SWAN should run on both TOPS-20 and VAX Unix.

- The bare-bones version of SWAN should be running by mid July 1982.

- SWAN should run substantially faster than the current version of Scribe. Admittedly the current version of Scribe is a very old and slow version.

- SWAN should eventually implement most of the features to be found in Scribe. These features include bibliographies, indexes, tables of contents, multiple output devices, cross-referencing, separate document compilation, and others.

- SWAN should eventually support some features not properly supported in the current

version of Scribe. These features include widow and orphan elimination, hyphenation, multi-column output, mathematical formatting, kerning, ligatures, and others.

It is important that some goals not be included in SWAN in order to keep the scope of the project reasonable.

~ SWAN will not be 100% upward compatible with Scribe. Where it is deemed appropriate SWAN will incompatible. The only compatibility goal is that someone should be able to edit a Scribe file into a SWAN file with only a very modest amount of effort. If people do not wish to expend this effort they are welcome to continue using Scribe.

~ SWAN will not try to produce output that is as pretty or as sophisticated as TEX output. The primary reason for this is that it is desired that SWAN run quickly. If people really want the features of TEX, they should use TEX.

~ SWAN will not directly provide a programmable language interface (i.e. macros). If SWAN does not provide a desired feature that a user desperately needs, that user will be encouraged to add code directly to SWAN in the language that SWAN is written in. However, it is not desired that casual users add code to SWAN.

Each of these goals will be discussed further in the paragraphs to follow.

Good user documentation is perhaps the single most important feature of a text formatter. A competent user with good user documentation can usually cope with a poor text formatter. Conversely, a good formatter with poor user documentation can be very difficult to use. Of course, SWAN is expected to be a good formatter with good user documentation. There are four clearly identifiable documents that SWAN should have. The first two are substantially more important than the second two.

1. The *SWAN Reference Manual* should be the definitive definition of what the user can and can not do in SWAN. It should be possible for the user to read the *SWAN Reference Manual* and decide what a sequence of SWAN commands will accomplish without resorting to experimentation (as is frequently necessary with Scribe.)

2. The *SWAN Database Manual* should be a manual that describes all the document types and environments that are available to a user. This manual should be published regularly and on a per site basis. The fact that Scribe does not have this manual is a big draw-back to Scribe. Hopefully, the production of the manual can be automated by a program that examines the SWAN database.

3. The *SWAN Tutorial* should provide the means for a naive user to learn how to use SWAN.

4. The *SWAN Programmer's Manual* should provide a useful overall picture for the person who is going to add a feature to SWAN. This manual should contain overall strategies, organization, and useful data structures. This manual should not contain information that will rapidly become out of date. Portions of this document should form the basis for the eventual *SWAN Programmer's Manual.* It is very important that the *SWAN Programmer's Manual* be kept separate from the *SWAN Reference Manual.* This is because the average user should not be encouraged to add features to SWAN.

CLU has been chosen as the implementation language for SWAN. This permits SWAN to run under both TOPS-20 and VAX Unix. Eventually, SWAN should be able to run under VAX VMS and on a Motorola 68000. SWAN could run under TOPS-10 and ITS, if someone wanted to write the appropriate CLU run-time system. Thus, SWAN could be run on all machines at LCS with the exception of PDP-11's. The small address space problem of the PDP-11's will probably preclude SWAN from ever running on a PDP-11.

One of the goals of SWAN is that a bare-bones version be running by mid-July. The reason for this goal is two fold. First, the VAX's that LCS is receiving do not come with a decent text formatter. Thus, the moment SWAN becomes available for the VAX's there will be an immediate user community. Second, initial performance deficiencies can be detected and corrected before the end of the summer. A third reason is that morale is better when the program is doing something.

Another goal of SWAN is that it be efficient. A better way of stating this goal is that SWAN should not be stupidly inefficient. Scribe has one glaring area where it is stupidly inefficient. Both the database files and auxiliary file are stored in ascii. This makes it easy to read and debug these files; however, it takes a long time read in and parse these files. SWAN will reduce this overhead by storing the database and auxiliary files in machine readable form. The machine readable database files will be automatically generated from human readable files. The auxiliary file will have a companion program that prints it in human readable form. An additional way that SWAN should be able to improve performance over Scribe is that it will probably treat the input file more at the word level than at the character level.

One of the eventual goals of SWAN is that it will essentially supersede Scribe. SWAN will not attempt to do this by being 100% upward compatible with Scribe. Instead, SWAN will attempt to do this by providing most of the Scribe features in a fashion that is not too different from the way that the feature is currently provided in Scribe. Some Scribe-like features should be completely

superseded by better features in SWAN. It is hoped that table generation will be one such area. The additional speed of SWAN over Scribe should lure people into expending the effort required to switch from Scribe to SWAN.

There are some features that are either unimplemented (mathematical text formatting, ligatures, kerning), improperly implemented (footnotes, widow elimination), or improperly documented (multi-column output, hyphenation) in Scribe. SWAN will eventually attempt to make up for these deficiencies. The one area where SWAN will attempt to perform significantly better is in the area of page makeup.

Probably one of the most controversial areas of SWAN will be in the area of programmability. Most of the other text formatters that are around implement some sort of macro package that permits the user to program the formatter. Even Scribe implements a Turing equivalent macro package despite the fact that the implementor did not think that macros are a good idea for formatters. This suggests that some form of programmability should be provided in SWAN. SWAN will attempt to avoid complicated macros. The primary reason for this avoidance is that macros are difficult to write, difficult to debug, difficult to read, and tend to run slowly. Instead, the approach to programmability that SWAN will take is to permit sophisticated users to directly add CLU code to SWAN. CLU tends to be much easier to write, debug, and read. Also, CLU code should run substantially faster than macros. Another advantage of providing programmability at the CLU level is that it will tend to discourage user's from programmability unless they absolutely have to.

## 3. Initial Implementation Ideas

This section contains some initial ideas about the implementation of SWAN. Material in this section is not presented in any particular order. Some of the material may not be particularly comprehensible either.

### 3.1. The SWAN Database

The database is one area where SWAN can be substantially better than Scribe. Upon examining the Scribe database, one notices that there is a large amount of repetition for each database entry. For each different device in a document definition almost all entries are replicated with only minor deviations. Also, many of the document types replicate the definitions of a large number of

environments. This suggests that the database can be substantially condensed. All environments should be specified in a device independent fashion. Of course, not all environments are device independent. Device dependencies should be specified as differences from the device independent specification on a per device basis. The same idea can be applied to environment specification across multiple document types. All environments can be specified once in a document type independent fashion. Again, not all environments are device independent. So, only differences between the global environment and document type dependent environments need to be stored with a document type definition. Thus, the way that the SWAN database should be organized is as 1) a single global set environment definitions and 2) a set of document types that contain only the differences to the global definitions.

Database Manual] This new database organization dovetails nicely with the concept of the *S*WAN *Database Manual.* Currently, there is nothing to prevent someone from writing a program that examines the Scribe database and produces a "*Scribe Database Manual*". However, the resulting document would be rather bulky and not substantially better than just looking directly at the Scribe database. However, with the new database organization, the *S*WAN *Database Manual* becomes substantially easier to generate and use. The *S*WAN *Database Manual* is broken into three distinct sections. The first section alphabetically lists all of the available environments (for all document types.) The second section alphabetically lists all of the document types, what environments are supported by the document type, and environment differences. The third section alphabetically lists all of the devices supported by a site and the characteristics of each device. Substantial portions of this manual can be automatically produced by a program that examines the SWAN database.

Database stored in machine readable form] As was mentioned earlier in this document SWAN should store the database in machine readable form. The machine readable form should automatically be generated from the human readable form. CLU implements a couple of operations for reading and writing arbitrary CLU objects to and from files preserving any sharing. GCREAD (for Garbage Collector Read) and GCWRITE (for Garbage Collector Write) are the names of two operations. These operations should make processing of the SWAN database very easy and efficient.

### 3.2. Devices

The initial output device for SWAN will be the Alto. This means that it will not be necessary to be constantly running upstairs to the Dover to examine SWAN output. The next output device for SWAN will be a press file driver for the Dover. Eventually device drivers for the line-printer, BBN Bitgraph, XGP, and Diablo should be produced.

Internally, SWAN should probably have a private representation of a page. This internal representation will be general enough to support each of the devices listed above. The internal representation should support the ability to place any arbitrary character in any arbitrary location. Also, the internal representation should support the ability to draw lines (and maybe ellipses). The device driver will have the responsibility of producing one page of output from one of these internal pages. The simpler device drivers will not have to keep any state information between each page generation. However, a complicated device driver (like the press file driver) will probably have to keep state information between pages. In this situation it would be nice if CLU had coroutines. Instead, the implementors of such complicated device drivers will have to make do with manually saving and restoring the relevant information in an object. Hopefully, only the XGP driver and the press file driver will fit into this complicated device driver category.

Frequently, a user will want to produce a rough draft of a document on a device that is different from the device that the document is finally intended for. Typical reasons for this are 1) the final device may be temporarily broken, 2) the final device may be heavily used and the draft device lightly used, and 3) the draft device may be substantially faster than the final device. In this situation, the user really wants the output that comes out on the draft device to look as close as possible to the output that will eventually come out of the final device. Scribe supports multiple devices, but Scribe does not support this idea of producing output destined for one device on another device. In order for SWAN to make a stab at trying to do this, it will be necessary for SWAN to use two different document types in the production of a document. This should not be too difficult provided it is built into SWAN from the start.

### 3.3. Word Oriented vs. Character Oriented

Scribe is basically a character oriented system. As a case in point, Scribe is perfectly willing to print some arbitrary control character in the input file in some user specified font. In order to do this, Scribe must be willing to process the entire file on a character by character basis. This can be very time consuming. SWAN will instead immediately process the input file into words and then deal in terms of words. In some SWAN environments, spaces, tabs, new-lines are significant. In these environments, they will be treated as SWAN "words". By dealing in terms of words, SWAN should be able to easily support dictionaries, spelling correction, word cross-referencing, and hyphenation.

Frequently, the user needs to access symbols that are not ascii characters. The solution adopted by MIT for Scribe will be used to solve the problem. Each symbol will be assigned a printing name such as ALPHA and CIRCLEDOT. Every place the user needs the symbol, the user will type the text name (along with some syntax to SWAN to indicate that the word is a symbol and not a word.) For example in Scribe "@Alpha[]" will print the greek letter alpha.

In Scribe there is exactly one reserved ascii character (namely the at-sign "@"). This was probably an over reaction to the fact that Scribe was an immediate successor to PUB and PUB had 20 or 30 special characters, each of which interacted with one another in various strange and wondrous ways. SWAN will not be constrained to use only one reserved ascii character. However, definite restraint will be used in deciding to use more than one reserved ascii character. One idea is to reserve the character '#' for non-ascii symbols (e.g. "#Alpha" and "#CircleDot"). Another idea is to have braces ("{}") for brief environments such as a math environment where all variable names are italicized (e.g. "{alpha + beta}"). A single consistent quoting convention will be used if more than one reserved character is used (e.g. "@#, @{, and @}").

### 3.4. Special Environments

Sometimes text formatters are asked to format material that is not really text. Programs, tables, and pictures are examples of such material. It is usually very painful for the user to format this kind of material. SWAN's approach to solving this problem is to provide special purpose environments tailored to the type of material being formatted.

Frequently, material at LCS contains sections of program text. For example, in CLU it is

standard practice that keywords are to printed in bold-face, comments are to be justified and printed in italics, and indentation is important. In Scribe, this can be done, but the resulting input text to Scribe is barely recognizable as program text. In SWAN, it should be possible to directly point to the actual working program text without modification and include it into a document. The program environment should be general enough to support a variety of different language formats.

Scribe has the ability to produce tables of information. Scribe provides this ability by permitting the user to have very explicit control over each field in a table (such as flush left, flush right, center, and tab to next field.) Again, the resulting input text to Scribe is barely recognizable as a table of information. The SWAN approach to this is to have an environment that can specifically deal with rows and columns of data. The user should have such high level commands as 1) center all data in this column, 2) align all decimal points in a column of numbers, and 3) interpret numbers in "E format" but print them in standard scientific notation (e.g. "3.7E-17" would produce "$3.7\times10^{-17}$", where the -17 should actually be in a smaller font.) Further, the user should have the ability to specify lines of varying thickness around the data to make it look like a data table. Again, it would be desirable to have SWAN be able to directly reference the actual data file that contains the data.

Sometimes the user would like to draw simple pictures containing boxes and lines. Editors like TED (and probably EMACS) provide a mode whereby the user may draw crude boxes and lines in ascii. These figures look pretty crude if they are included in the final document. However, it should not be too difficult to produce an environment that can interpret the ascii lines and boxes and produce real lines and boxes in the final document. Further bells and whistles should include 1) the ability to center text in box, 2) draw arrowheads, 3) use circles and ellipses instead of boxes, 4) use rounded corners on lines instead of square corners, and 5) draw dots on intersecting lines and loops on non-intersecting lines for electrical circuits. There will be a temptation to put too much capability into such a picture environment. It will always be possible for a user to use a DRAW-like package to produce the more complicated figures for a document. The general rule of thumb should be to minimize the amount of effort required to produce a picture. A feature should be added to the picture environment only if it will permit the user to produce a picture in less time that it would take in a DRAW-like package. It should also be explained to the user that it is frequently much easier to draw the picture by hand than to try to use either a DRAW-like package of a SWAN style picture environment.

Many people at LCS include non-trivial math in their documents. A straight text formatter is basically designed to produce one-dimensional text instead of the two-dimensional material required for mathematics. Such a math environment should be able to handle 1) summation signs, 2) integral signs, 3) division, 4) italicized variables, 5) greek variables, 6) boldfaced vector variables, and 7) matrices. A lot of work should go into the user interface of this environment to make it easy for a user to produce an arbitrarily complex mathematical formula. It would be nice if a user could somehow take the output of a system like MACSYMA and directly include it into a document.

Other environments would include a chemistry environment, a physics environment, and a SWAN documentation environment. The chemistry and physics environments would directly understand chemical molecules and physics material. The SWAN documentation environment would be used exclusively for producing SWAN manuals. It would have the ability to print a sample input to SWAN and the resulting SWAN output. There are undoubtably other environments that sophisticated users would like to construct. These environments would be written by the sophisticated user in CLU and directly added to SWAN.

### 3.5. Miscellaneous

It would be very nice if the clusters out of which SWAN is constructed had enough generality to permit other people to come along and use pieces to produce other applications. However, having a general interface to SWAN clusters is not a very high priority. This is because it is more important to get SWAN running than it is to provide useful packages to help other people in their projects.

One important feature of SWAN will be its ability to produce documents on a chapter by chapter basis. One problem with Scribe is that it stores absolute page numbers in the auxiliary file. Thus, when a chapter grows by one page it is necessary to reprocess all following page chapters to get the page numbers right again. This can easily be corrected by storing page numbers relative to the beginning of the chapter.

The M.I.T. thesis format should be treated as an example of the sorts of things that SWAN page layout should be capable of performing. An M.I.T. thesis has absolutely every page numbered starting with the title page numbered page one, followed by dedications and acknowledgments, followed by the tables of contents, tables, and figures, followed by chapter one. Another example is the M.I.T. thesis margin requirement. Absolutely no text is permitted within one inch of any edge

of the page. The only thing permitted within this no-mans-land is the page numbers. Since SWAN will be used for M.I.T. theses, it must be able this stringent page layout requirements.

## 4. Summary

This RFC has described the initial goals and implementation ideas for SWAN, a new text formatter.