Developing Database Publishing Systems Using TEX

Jeffrey McArthur
ATLIS Publishing Services
jmcarth@atlis.com

Abstract

Many directories and publications are created and maintained using "off the shelf" desktop publishing systems. Producing a publication may take months of hard work. Some publications can be converted to a database publishing system which is capable of producing a finished book in a matter of hours. This paper looks at some of the issues involved in developing a database publishing system that uses TEX as the typesetting system.

The Key Features of a Content Management System

Content management systems store information in a database. The list below enumerates some of the benefits of developing a system to maintain information using TEX as the typesetting engine:

1. Consistency: The typesetting is regular and predictable. Books that are typeset using desktop publishing are often broken into sections. Each section is done by a different person. Each person has control over the layout of their section. This can result in subtle differences between sections of a publication. A content management system using TEX removes the possibility of undesirable differences between sections.
2. Timeliness: Document preparation no longer takes days or months. Pushing a key can generate an up-to-date publication reflecting the state of the data in the system. A complete publication can be finished in a matter of hours using TEX. Information constantly changes and last minute changes will automatically be incorporated in the final output with no additional effort. The predictable nature of publication generation allows for tighter scheduling of production.
3. Indexing: Automatic generation and extraction of index information are part of the typesetting process. A content management system using TEX as the typesetting engine provides the capability to create indexes that would be impossible to do using desktop publishing in a timely manner.
4. Repeatability: The ability to generate a book over and over again with different data is a key feature of content management systems. Multiple books can be quickly created using different subsets of the information contained in the system.

This paper focuses on using TEX as the back end or typesetting engine in a content management system. The choice of TEX as the typesetting system provides many benefits but also affects the development and implementation.

Limitations of Desktop Publishing Systems

Desktop publishing systems are designed around the WYSIWYG paradigm. It is easy to create great looking pages but there are no integrity checks on the data. Without validation on the input data it is possible to have dates like "February 31", or to have a two letter state abbreviation like "23". Errors of this type are amazingly common. Spelling and grammar checking will not detect errors of this type. In a desktop publishing system the information is just textual data. If a person changes his email address, all the documents in the system must be searched and a replacement done to each and every instance. In content management systems each person is an entity. All calls to the person are by reference. If the email address of the person changes, then all references to the person are automatically updated.

One of our clients took three months with up to fifteen people to generate the index to one of their books. That index can be generated in a matter of hours with a content management system developed for them.

Desktop publishing systems are not designed for publishing large quantities of data quickly. TEX works hand in hand with the content management system to quickly and accurately generate printed or electronic documentation.

Most desktop publishing systems provide tools to import database information. The WYSIWYG paradigm means that the user is prompted to "flow" the data into the system. Style sheets provide only limited capabilities over club and widow control. Import capabilities are usually limited to importing a single table, query, or view of the data. Content management systems do not have this limitation.

Database Development Must Work Hand-in-hand with TEX

All successful content management systems must keep the goal of producing the output documents in mind during all phases of development. TEX as a typesetting engine places demands on the database design.

Sorting. It is possible to sort in TEX, but the macros to do this are complex and difficult to use. Database systems are designed for sorting and each system should be used for its strengths. Thus, all sorting should be done by the database system and not in TEX.

Accents. Many database systems in the United States are not able to accurately store accented data. Some database systems translate accented characters to some other form internally. This can cause problems when the data is output for TEX to typeset. The database system must be configured so that it is easy for the user to enter accented data, and be able to get that data back out into a file that can be typeset by TEX.

Publication order. Normalized databases have an implicit order. This seldom matches the order that is desired in the printed or electronic output. TEX can rearrange data. It is easy to exceed the capacity of TEX by trying to store large quantities of data internally so that they can be rearranged on output. The database should be designed to allow quick generation of the data in an order which closely resembles the final output. This makes the job of typesetting the data with TEX easier.

Line lengths. One limitation of TEX is that the input file must be broken into lines. Most implementations of TEX only allow one to two thousand characters on a line. This is a serious limitation with database publishing. The database system must separate the data into lines that are less than the input-line limit of TEX.

Special characters. Publishing data from a database places some requirements on the macros developed in TEX. A common requirement is that all printable characters must be usable. If the user can enter a character, they want that character to output in the finished document. If the user enters a backslash, the resulting output needs to print the backslash. Although this causes difficulties with TEX, there are several methods for solving the problem. One solution is to have the database filter the data for input into TEX. This is quite slow since it requires the database system to check every character to see if it needs to be "escaped". Another approach is to change the way TEX reads the data by using verbatim macros such as are commonly used for listings. The composition file generated by the database system does not need to be editable by a human since it is only a temporary file used to transfer data from the content management system into TEX.

This approach allows many of the control characters normally used by TEX to have their `\catcode` changed to that of a letter. Macro packages used to typeset databases often end with the following sequence:

```
\catcode`\$=\other
\catcode`\%=\other
\catcode`\&=\other
\catcode`\#=\other
\catcode`\_=\other
\catcode`\^=\other
\catcode`\{=\other
\catcode`\}=\other
```

The lines above allow most characters to be printed. The tilde character, — —, however, is a special case. Internet URLs often contain a — —. The problem with typesetting internet URLs is that the — — character should be typeset as ∼ and not as — —. Using a —∼— instead of a — — is more consistent with the way the information looks when keyed or when viewed in a browser. The simplest solution to changing the typesetting of — — is to do the following:

```
\def\Tilde{\hbox{$\sim$}}
\catcode`\~=\active
\let~=\Tilde
```

Changing the `\catcode` of —— and —— means that grouping cannot be done using them. This is usually not a problem. The parameter text rules of TEX provide a mechanism to specify the boundaries of the input parameters.

Typesetting `\` is a bit more challenging. One solution is to change the escape character to one that is non-printable. One choice is character 255. Very few typefaces define a glyph for character 255. On the PC, character 255 prints as a space. The character is non-printable, so it is difficult to give an

example how to use it. Below is a set of macros that use character 255 as the escape character. In the example below, character 255 has been replaced by the sequence `M-^?` so that the macros are readable.

```
\chardef\other=12
\def\QuoteBS{
    \begingroup
    \catcode`\\=\other
    \EndQuoteBS}
\begingroup
\catcode`\M-^?=0
\catcode`\\=\other
M-^?gdefM-^?EndQuoteBS#1\End{
    M-^?line{M-^?hfil***#1***M-^?hfil}
    M-^?endgroup}
M-^?endgroup
```

There is a problem with macros like the one above: they are difficult to read and maintain. Few texts, other than ones on TEX, use a lot of \ characters. A more rational approach is to restrict the input of data into the content management system to disallow entering a \ except in those few fields that actually require it. This simplifies writing the macros, makes the macros much more readable, and more maintainable.

Index generation. TEX has the ability to create auxiliary output files that list the page number where the `write` occurred. TEX can extract information that can be used to form an index. We have used two different methods of creating extracted indexes. One method is for TEX to be responsible for outputting a file suitable for composition with TEX. That is, TEX creates a file that can be run through TEX. The file is sorted prior to running through TEX, using a simple sort program. One major disadvantage to this method is that the resulting code is very difficult to read because many characters have their `\catcode` changed. Another difficulty is in sorting the resulting index. Care must be taken to allow the file to be sorted properly. It is usually easier for the database program to export a "sort key" into the data stream for TEX to pass on to the extracted index file than to try and create the "sort key" in TEX. This is particularly true if the index contains entries like 3M which need to sort under M (Minnesota Mining and Manufacturing). Accented characters are a problem. Care must be taken to allow accented characters to pass through TEX without any change.

A second approach is for TEX to only output the "record id" and page number. The auxiliary file is read into a database program and a new composition file is generated with the page number data. This method has a couple of advantages:

1. The database provides the tools to sort the information.
2. Indexes where an entry may occur under many headings are much easier to handle.

Extraction from the Database

The information in the database or content management system must be exported for use by TEX. For simple database projects it is possible to use standard database export utilities. Typesetting directly from the quote-delimited format can be done by making the double quote character active to start the input, and then using a form of tail-recursion to call macros for the next field.

For each field there are three macros. One to start the input, one to store or typeset the data, and one to finish the field and call the macro to start the next field. If the database contained only three fields: first name, last name, and phone extension, then an exported quote-delimited file would look like this:

```
"Jeffrey","McArthur","4253"
"Jeannine","McArthur","1234"
"Bilbo","Baggins","5678"
```

The following macro would typeset the data, moving the first name after the last name and setting the name left-justified on the line and the extension right-justified on the line.

```
\chardef\other=12

% These assume " is active
\def\BegFirstName{
    \begingroup
    \catcode`\"=\other
    \MidFirstName}

\def\BegLastName{
    \begingroup
    \catcode`\"=\other
    \MidLastName}

\def\BegExtension{
    \begingroup
    \catcode`\"=\other
    \MidExtension}

% These assume " is other.
% They pull in the parameter
% and store or set it
\def\MidFirstName#1","{
    \gdef\ValFN{#1}
```

```
    \EndFirstName}

\def\MidLastName#1","{
    \gdef\ValLN{#1}
    \EndLastName}

\def\MidExtension#1"{
    \line{
        \ValLN,
        \ValFN
        \hfil
        #1
    }
    \EndExtension}

% these assume " is other
% they reset the catcode and
% call the next field
\def\EndFirstName{\endgroup\BegLastName}
\def\EndLastName{\endgroup\BegExtension}
\def\EndExtension{\endgroup}

\catcode`\"=\active
\let"=\BegFirstName
```

There are several problems with typesetting quote delimited files. TEX can only work with files that do not exceed its input buffer line length, usually one to two thousand characters. Databases with large record structures can generate lines longer than this. If the database adds a new field to the table, then the output file would be a different structure and TEX would no longer typeset the file properly.

Generation of composition files. The approach we have taken to preparing the data for use with TEX is to write a program that generates a composition file. The program processes each record in the database and tags and outputs each field. This gives full control over the order of the data and can provide any special processing required.

We have used a descriptive tagging scheme which allows the same extracted data to be used for more than one purpose. The data file can be composed with different sets of macros producing different output. Using the same data file for multiple outputs has various advantages. For instance, the generation of a composition file can be a time-consuming process. TEX can process the data two to ten times faster than the data file can be generated. For example it can take up to four hours to extract a sixty megabyte composition file; TEX can compose that file into about twelve-hundred pages in less than half an hour.[1]

SGML. Content management systems that use SGML encoding can be typeset using TEX. Valid SGML document instances are stored as memo fields in relational databases. The document instances are extracted into composition order and "stitched together" using TEX macros. With proper care TEX can directly typeset the SGML document instances. This requires that all the tags follow consistent casing scheme, or preferably a change to the SGML declaration to make the tags case-sensitive. If no output processing or filtering on the SGML is required, the composition file can be quickly generated. This is one of the few cases we have encountered where the composition file can be generated faster than TEX can compose it.

SGML tables. SGML tables using the SoftQuad table model are computationally expensive to do in TEX. Some of the SGML tables we have typeset extend beyond four pages. Our implementation of macros in TEX to typeset SoftQuad SGML tables uses `\halign`. This has proven to be very memory intensive. Large tables require a version of TEX that can use more than 20 Meg of RAM. TEX normally processes pages quickly, but on slower hardware[2] TEX can take up to ten minutes to process a single table. During that time no pages are output. Once the table processing has finished, TEX resumes quickly outputting pages.

Entity and table validation. During our implementation of SGML content management systems we ran into some problems with tables. A valid SGML document instance can have an invalid table. That is, the table can define three columns and actually have four or more columns of data. The SGML parser is not designed to catch this type of problem. Typesetting SGML document instances that have malformed tables causes TEX to generate an error message. The content management system hides most of the details from the end users. Error messages generated by TEX are helpful to those well versed in TEX, but to an average user they are total gibberish. To avoid this problem we developed a small program using a variant of Lex[3] that compares the number of column definitions with the number of actual columns. We further enhanced the program to process the ampersand character `&` and make

[1] Using a Pentium 166 running on a Novell 4.11 network with Paradox tables.

[2] 486DX33 with 32 Meg of RAM.

[3] TPLexYacc 3.01, this version of Lex emits Pascal code instead of C.

sure that it was only used as the start of an entity and that all entities were followed by semicolons. Making & active and using \csname and \endcsname allowed TEX to process the SGML entities. The macros below show how to do this:

```
\catcode`\&=\active
\def&#1;{\csname ENTITY#1\endcsname}
```

Push-button Book Generation

Any time the user wants to generate a book, or see what a book will look like, we can push a button and have the finished book in a few hours. We have used TEX to generate completely turn-key database publishing systems in which the user does not need to know anything about TEX.

The content management systems we have developed provide the user with the ability to select what sections or documents to print. We have also implemented systems where the user has some limited capabilities to change how the book is typeset. For instance, we have developed systems where the user can select the number of columns, the font sets to use, the point sizes to use, and the overall page size. The user cannot enter arbitrary combinations since the selection is limited to combinations that would work.

Test and Fit

Push-button generation means that the user does not need to make any decision on the fly about how the program is to typeset the book.

However, the measuring capabilities of TEX do allow for runtime calculations. For example, in a common directory style the phone numbers should fit into a right-hand column, with the text on the left. If the phone numbers follow a regular pattern, as they do in the United States and Canada, then the width of the phone number column can be calculated at run time. This is done by setting a test phone number in an hbox and taking its width. The advantage of this method is that the fonts can be changed and TEX will recalculate the column widths.

Test and fit can also be used in another phone number situation. Although phone numbers in the United States and Canada follow a 3-3-4 pattern, e.g. 301-578-4200, other countries do not follow that pattern. TEX can measure a foreign phone number to see if it will fit. This is accomplished by placing the phone number into an hbox. The width of the hbox can then be compared to the space allowed on the line for phone numbers. If the phone number is larger than the allowed space, the data can be re-typeset using a smaller or condensed font. This capability should be used judiciously or the pages will be aesthetically displeasing.

Widow and Club Control

TEX provides two penalties \clubpenalty and \widowpenalty that control how paragraphs break. The paragraph-breaking capability of TEX can be used for more than just simple lines. One common rule is to "leave and carry two" entities. That is, if the book is a list of people, then the requirement is that there are always at least two people prior to and following a column break. The information for a person may actually take several output lines. If each person is eventually placed in a \line and the list of people is typeset as a paragraph, then setting \clubpenalty=10000 and \widowpenalty=10000 means that TEX will always "leave and carry two" people.

One of the limitations of TEX is that its paragraph-breaking algorithm does not provide mechanisms for doing things like "leave and carry three" or higher. In cases like this the database extraction program must provide the information to TEX on where to allow it to break. One way to do this is to output each entity as a vbox. If TEX is in vertical mode then a series of vboxes cannot break if there is no glue between them. Let the database extraction program count the number of records and insert the glue and penalty commands at the allowed breakpoints. Another option is for the database program to only output the number of records and let TEX count the records as it processes them add glue at the appropriate points. So even though TEX does not provide the capability to "leave and carry three" this can easily be done in a content management system by having the database extraction program work with TEX.

Suppression and Selection

The macro capabilities of TEX provide powerful selection capabilities allowing TEX to act as a filter. For example, one application we developed typeset a large directory of phone and fax numbers. The sorting was complicated and the extraction from the database took several hours. We were able to produce two different books from the same composition file. The first book listed all the entries. The second book listed only those entities that had fax numbers. Because the books were generated from the same data file it was guaranteed that the order of the records in the books could not change.

TEX can also filter out visually redundant information. Normalization is the process of removing redundant information from a database. Extracting the information from a database will denormalize the data. Consider the following simple example. The database contains three tables: a department table containing the id number of the department and the name of the department, a person table containing the name and phone number of the person and a link table that connects the department and person tables. The SQL statement to select all the records from the tables for composition could look something like this:

```
SELECT DISTINCT
    D0.DeptId,
    D0.Department,
    D2.PersId,
    D2.First,
    D2.Last,
    D2.PersPhone
FROM
    "Department.DB" D0,
    "Link.db" D1,
    "Person.DB" D2
WHERE
    (D1.DeptId = D0.DeptId) AND
    (D2.PersId = D1.PersId)
ORDER BY
    D0.Department,
    D2.Last,
    D2.First
```

When the resulting answer table is output using descriptive tags the result would be a file looking like this:

```
\StartRecord
\DeptId{2}
\Department{Medlars}
\PersId{3}
\FirstName{Bilbo}
\LastName{Baggins}
\PersPhone{5678}
\EndRecord
\StartRecord
\DeptId{2}
\Department{Medlars}
\PersId{2}
\FirstName{Jeannine}
\LastName{McArthur}
\PersPhone{1234}
\EndRecord
\StartRecord
\DeptId{1}
\Department{Programming}
\PersId{1}
\FirstName{Jeffrey}
\LastName{McArthur}
\PersPhone{4200}
\EndRecord
```

All the fields are output into the composition file with a start and end tag for each record. It is usually undesirable to reprint duplicate information. Using combinations of `\def`, `\let`, and `\ifx`, TEX can filter out the duplicate information. Doing this type of filtering, or deduping, in TEX simplifies the generation of the composition file.

TEX can also do simple rearrangement of the data. Below is a set of simple macros that demonstrate these capabilities.

```
\let\StartRecord=\empty
\let\LastDept=\empty

\def\DeptId#1{\def\ValDeptId{#1}}
\def\Department#1{\def\ValDepartment{#1}}
\def\PersId#1{\def\ValPersId{#1}}
\def\FirstName#1{\def\ValFirstName{#1}}
\def\LastName#1{\def\ValLastName{#1}}
\def\PersPhone#1{\def\ValPersPhone{#1}}

\def\EndRecord{
    \ifx\LastDept\ValDeptId\else
        \let\LastDept=\ValDeptId
        \medbreak
        \line{\bf\ValDepartment\hfil}
    \fi
    \line{\ValLastName,
        \ValFirstName
        \hfil
        \ValPersPhone}
}
```

TEX could also be used to select which entries to typeset. The same data file could be used to generate a complete directory, and a separate directory for each department. For the complete directory TEX would set each record. For the department records TEX could test the department id.

Various Output Options

TEX generates DVI files. None of our end users were interested in DVI files. Using tools like DVIPS, and recently pdfTEX we have created PostScript and PDF files for our clients, as well as finished camera ready pages and film. We have used TEX to generate questionnaires to be programmatically faxed.

TEX generated faxes. The process of using TEX to fax questionnaires is outlined below:

1. The application generates a composition file.
2. The application executes TEX which composes the file and generates a DVI file. TEX generates an auxiliary file listing the account number, starting page
   and ending page of each questionnaire.
3. The application reads in the auxiliary file.
4. The application executes DVIPS to extract each questionnaire using the starting and ending page number from the auxiliary file. The name of the resulting PostScript file is based on the account number.
5. Acrobat Distiller converts the PostScript File into a PDF file and embeds all the needed graphics and fonts.
6. A second application watches the output directory from Acrobat Distiller. The file name, which is based on an account number, is used to look up the fax number. Using OLE, Acrobat Exchange is executed and the PDF file is opened. Using DDE and WinFax, the fax number is set. Using OLE, Acrobat Exchange is told to print the PDF to the Fax.

Caveats

Developing macros for database publishing requires a full understanding of all of the capabilities of TEX. Macro packages like LATEX are designed for authoring and they were not designed for database publishing. The macro writer must understand how to write output routines, use `\mark`, and even the list capabilities of TEX as described in Appendix D of *The TEXbook*. Compared to desktop publishing systems this is a very high threshold. Finding and training people to write macros of this complexity is a difficult task.

All the logic to typeset the pages must be programmed into TEX. The rules for when to break, when to kern, and so on must be incorporated into the macros. This means that the output pages are very regular. One often-heard complaint is "can't you change the typesetting in just this one case?" TEX is not WYSIWYG. This means that the user of a turn-key system that uses TEX as its typesetting engine may not allow any visual fine-tuning of the pages.

Although TEX is very robust, many DVI translation programs have problems with complex pages. We have seen numerous implementations fail trying to process large complex pages. Many DVI translation programs are "fussy" about the types of graphics they will use. Integration of "off-the-shelf" versions of TEX is relatively easy. Integration of "off-the-shelf" DVI translation programs has been problematic.

Most implementations of TEX and the related DVIware for the PC use environment variables. The tools assume that there is only one implementation of TEX being used. Configuration management is much more difficult than it should be.

The most serious disadvantage to using TEX as the back end system is the difficulty finding people with the aptitude to learn and understand how to write complex macros. It is relatively easy to find someone to use a desktop publishing system. Finding someone who can write the macros for a content management system is a lot harder. We are always looking for qualified people. The lack of qualified people is part of the overall shortage in the computer industry.

Conclusion

TEX has proven to be a valuable tool in developing content management systems, but it cannot be quickly adopted. It takes time and commitment to find, hire, and train the people needed to develop the macros. There are disadvantages to using TEX as the typesetting engine, but the disadvantages are compensated for by the quality and speed of the resulting system.