Advanced Automatic Post Processing Of Output Data Using Text Processors

Matjaž SKRINAR
Ph. D.
Ass. Prof.
Fac. of Civ. Eng.
Smetanova 17
2000 Maribor
Slovenia
mcs@email.si

Born on 17. 1. 1964, Maribor, Slovenia.
Studies: 1983-1989, B.Sc., 1989-1992, M. Sc., 1992-1998, Ph. D., all at Univ. of Maribor
Member: IABSE , GAMM and several national associations
Research fellow at Univ. of Trieste, Trieste, Italy

Summary

The paper presents a seldom seen approach to the creation of a final report on an engineering analysis. The approach has been efficiently implemented in a computer package for geotechnical analysis, although it can be actually implemented in many similar engineering applications. The implementation of procedures briefly described in the paper namely offers a solution where the obtained results, accompanied with all essential equations, are automatically transmitted to a desktop publishing package without any user's intervention. Such a solution is primarily oriented towards the users who require not only representative forms of the results obtained, but also a comprehensive report created practically without any efforts. The paper studies the navigation procedures of the word processor WinWord - versions 6, 7 (known as 95), 97 and 2000, all under various Windows platforms (3.*, 95 & 95, 2000 and ME).

Keywords: Automatic report creation, text processor, Microsoft WinWord, Visual Basic

1. Introduction

With the extensive progress of computational power, and consequently the calculation speed, more attention has been devoted not only to the pure engineering analysis itself, but also to pre- and post-processing of data. While early computer packages provided the users only with numerical results from input data files which were usually manually prepared, modern packages offer user-friendly data input capabilities and also various graphical output representations. Although all graphical representations from the screen can be also exported and stored on the hard disk for further manipulations, the obtained numerical results are still frequently stored in files on disks as plain text files in ASCII character set without any symbols or equations. For example, letters from the Greek alphabet are thus replaced with a combination of characters that indicate the pronunciation of the considered character. Such output files with numerical results are usually afterwards manually retrieved by the user into a chosen text processor and also adequately manually formatted by inserting the proper characters or symbols. In this manner the definite form of the report is generated.

2. The Scope of Automatic Post Processing of Results Using Text Processors

There are two main reasons why automatic post processing of the obtained results using postprocessors should be implemented.

First, in engineering analysis the transparency of the obtained results is usually very important as it must allow the supervisors of the project to follow each step of the analysis and thus to detect eventual errors. While in the past the results were often limited to plain numerical output, the development of the computer power has provided graphical support for a better representation and understanding of the problem considered. While the errors in discretisation of the computational model can be easily spotted out from graphical presentations, numerical errors can remain undetected. This is especially true when the final result of the analysis performed is influenced by a large number of partial coefficients obtained from various formulas. An effective presentation of the derivation of the final result should thus present all essential steps accompanied not only with partial numerical results but also with all corresponding equations.

Secondly, such a history of the result development is also very important in the educational process where students must learn how to process from the first phase towards the final solution. As the computer generated report covers all computational steps, the students can concentrate on the important aspects, as for example proper selection of the computational model, and furthermore, to carefully study the influence of a single parameter change on the final results, thus achieving the so important engineering feeling for a selected problem. Using this approach students can easily study how to improve the results obtained by altering the utilised model, as they can use their computer programs not as the final solution providers but rather as tools that just perform sometimes monotonous and lengthy mathematical manipulations. This allows them to supervise and carefully scrutinise each step of their own analysis.

3. The Selection of the Software for Automatic Report Generation

While considering the idea of preparing a representative final report form, the question about the word processor was the basic question that had to be answered. Although many powerful word processors are available, the program WinWord (Microsoft) has become the rational choice. The decision was clearly supported by the fact that the earliest versions of the program used the programming language WordBasic in which macro commands are available. This allowed easy navigation of the text processor by setting the appropriate macro for a required action.

As WinWord can perform only basic mathematical operations (within tables only) it is not suitable for performing complex numerical analysis, and thus another software had to be selected to perform engineering analysis. Among many packages that allow programming in various languages, Visual Basic was selected, because it is a very widely used tool for many engineering applications. As both selected programs, WinWord and Visual Basic, are products of Microsoft Company, so the compatibility of both software is assured. The advantages of the selection of the two programs became even more advantageous when WinWord, version 97, was presented. In this version the WordBasic script language was replaced by Visual Basic script.

4. Creating of the Data Transfer Link between the two Programs

To enable an application in Visual Basic to crate the report, a link between the programs must be established first. This is achieved by the following commands

```
Set ww = CreateObject("WordBasic")
ww.appshow
ww.FileNew template_name$
```

Fig. 1 Establishing the link between the program and WinWord processor

The first command creates an object that will serve to the Visual Basic application as a pointer to the file in the text processor. The selected name *ww* is not prescribed and can be chosen arbitrarily.

The second command makes the WinWord application visible and restores the application window name to the Task List, so that the final user can simultaneously follow the creation of the report as it is generated. It actually starts the WinWord program, and if WinWord is already activated, the

command *appshow* has no effect. As the link between the two programs was denoted with *ww*, all commands for the navigation of the text processor must be preceded by this name.

The third command in Fig. 1 opens a new document in WinWord based on the template defined in string variable *template_name$*. This is especially convenient as it allows various users to create reports equipped with their logotypes in the header and/or footer of the final report or to create different types of analysis with different templates. The prepared templates must be stored in a subfolder where also the original WinWord template *normal.dot* and all other templates are stored (this location is different for various versions of Windows and can be obtained with the command Tools/Options/File locations in WinWord).

The last command can be replaced by the command *ww.filenewdefault* that creates a new document on the basis of *normal.dot* template.

5. The Report Front Page Set-up

The first page of the report should normally present all general information about the report: the name of the company or the person who performed the analysis, the title and subtitle of the project covered by the report, place and date of creation, etc.. Various lines should consequently have different styles. A typical example of line style manipulation is presented in Fig. 2.

```
ww.bold: ww.FontSize 16:ww.Font "Times New Roman"
ww.insert "Name of the company " + Chr$(13)
ww.FontSize 13
ww.insert "Engineer:" + name$ + Chr$(13)
```

Fig. 2 Creation of a part of the front page

Three commands are executed in the first line. The first one activates the bold font, the second command defines the size of the font, while the third selects the type of the font. The three commands can be executed in various orders.

While the first three commands just prepare the WinWord to write the required text, the second line actually writes (inserts) the text in the report file in the text processor. The text can be stored either in a string variable or it can be given between the double quotation marks - as shown in the last line in Fig. 2. To write the text into a new line, the end of the previous line of the text must be followed by the carriage return code - Chr$(13).

6. Creation of Figures, Storage and their Retrieval in the Report

Although all graphical output can be placed automatically into the report file, it is perhaps better for the final user to select the figures that he or she requires in the final report. This is due to the fact that the user of the application can select among many different types of graphical representations, and that the selected graphical representation can be further manipulated (for example by selecting plot ranges or scale factor). Consequently, all the produced figures are not interesting, so they do not have to be stored automatically in the report file.

The figure cannot be inserted into the report file directly (unlike the results stored in variables). It should be first stored on the hard disk into a separate output file by an appropriate command given by the user (for example by a double click on it). The figure is saved in the Windows Bitmap (*bmp*) standard, the only standard supported by Visual Basic so far. The figure can be afterwards easily inserted in the word processor file to complete the report file. The required commands are given in Figure 3.

```
SavePicture Picture1.Image, "File_name.bmp"
ww.InsertPicture CurDir$ +"File_name.bmp", "0"
```

Fig. 3. Commands for storage and retrieval of a program generated figure

The first command in Figure 3 does not navigate the WinWord processor, it just saves the figure plotted in Figure 1 (or an appropriate other number) on the disk into the current working directory under the name *File_name* with the extension *bmp*. If the figure should be stored into a different location as the current working subfolder, the whole path must be given in front of the file name.

The second command in Figure 3 retrieves the saved figure from the hard disk into the report. As the working directory for WinWord (obtained through commands Tools/Options/File locations in WinWord) is generally not the same as the folder where the application is stored, the whole path must be submitted to the text processor, followed by the name and extension of the graphical image.

7. Creation of Equations

The problem of creating the equation in the report was solved by storing all the required equations in separate files on the hard disk (usually in a separate subfolder) and then just by inserting them in the final report in front of the pertaining result. Actual manipulation of the equation (i.e. editing, cutting and pasting or writing numbers) was not planned from the start.

There are several reasons for this decision:

WinWord allows the usage of various mathematical equation editors. One of the most frequently used equation editors (besides the original Equation Editor, which is a part of WinWord), is MathType (which is sometimes offered when the Equation Editor is opened). For actual programming of the equation a detailed structure of the considered equation editor would be required, which is not always known. For example, the Equation Editor built in WinWord is only a truncated version of the MathType equations editor, which was sold separately. It is actually not a product of Microsoft, so it does not execute the macro command.

The second argument why the equations are not actually written but only inserted, is also strictly related to the selection of equation editor. Although it is expected that various equation editors should write the equations in a format readable to competitive editors, this is not generally true. Although they share the same origin, even MathType and WinWord 'truncated' version make the same equation look differently, especially when Greek alphabet or various brackets are presented. This usually is the case when WinWord tries to read an equation written by MathType editor. Vice versa is not so often true, and thus the considered application uses previously created equations (created with the Equation Editor) and just inserts them in front of the results.

The third fact that speaks against the creation of complete equations (with numerical values) is that although equations are long, the mathematical operations within are usually relatively simple (addition, subtraction, multiplication and division) and can be easily verified if the complete equation is presented in analytical form only.

Therefore the creation of an equation is reduced to the insertion of a WinWord file followed by the computed result (Figure 4)

```
ww.ChDefaultDir CurDir$ & "\equations\", 0
.....
InsertFile "equation_name.DOC", "", 0, 0: ww.editclear -1: ww.insert "=" + str$(result) + Chr$(13)
```

Fig. 4. Commands for insertion of an equation

If the equations are stored in a subfolder, entitled for example *equations*, and many equations have to be inserted, the insertion process can be accelerated by changing the default directory of WinWord to the folder where the equations are stored. The command is given in the first line of Figure 4.

The actual insertion of the equation usually follows the computation of the result using considered equation. It is achieved with *InsertFile* command, followed by the name of the file where the corresponding equation is stored. The insert command is followed by the *editclear* command that shifts the cursor one space to the left directly behind the equation, thus eliminating an empty space between the equation and the symbol of equality. The equality sign is followed by the obtained numerical result, converted in a string variable. As the result value in the presented case has no units, it is followed by the code for carriage return.

8. Symbols, Units, Subscripts and Superscripts

Various symbols or letters from other alphabets (especially Greek letters) are very frequently used in engineering. It is possible to insert symbols from all fonts installed.

Regardless if the result follows an equation or just a symbol, it can be accompanied by units. Units can be stored into separate WinWord files and retrieved in the report file just like equations, or, if they are relatively simple, they can be generated by the application. The final results and corresponding units can be also presented with underlined, bold or italics fonts, and also in greater size than the rest of the report, to clearly differ from the less important results.

```
ww.InsertSymbol "Symbol", , 113: ww.subscript: ww.insert "B": ww.subscript
ww.insert " = ": ww.bold: ww.FontSize 14: ww.underline
ww.insert str$(result): ww.superscript: ww.insert "o": ww.superscript: ww.insert Chr$(13)
```

Fig. 5. Commands for insertion of a symbol and manipulation of font properties

The first command in Figure 5 inserts the Greek letter θ, (defined with character number 113), from the Symbol font. The symbol has a subscript B and it is further followed by = sign. The result, stored in the variable *result* is converted into a string and written in bold with font size 14 and underlined. The degree sign is added at the end of the value indicating an angle.

9. Tables

Sometimes not just a single-valued result is obtained in the analysis or the results are given for a group of input values. In such cases the results are usually stored in tables. Prior to storing the results in a form of a table, the number of columns and rows must be known.

```
ww.TableInsertTable "", "11", Str$(nlayers% + 2), , , "0", "167"
ww.insert "Column name": ww.nextCell
```

Fig. 6. Creation of a table

The dimensions of the table must be given as string variables, as shown in the first line of Figure 6. This can be achieved directly by putting the value between the quotation marks if the dimension is constant and does not depend on the analysis. Thus the number of columns in Figure 6 is 11. If a dimension of the table can vary from example to example, it must be converted into a string variable. In Figure 6 the value of rows of the table is computed from variable *nlayers* that is increased for two (lines) to secure the space for the definitions of columns and units.

After the execution of the command in the first line of Figure 6 the cursor in the WinWord program waits in the first cell of the table, i.e. the first cell in the first row.

The first command in the second line in Figure 6 puts some text into the first cell, while the second command puts the cursor in the next cell which is the neighbouring cell in the first row. The process of filling in the table is thus accomplished from the first cell in the first rows until the last cell in the last row is reached as the command *nextCell* shifts the cursor also from the last cell of a row into the first cell of the underlying row.

While the cursor is positioned in the last cell of the table it is possible to adjust the table - to select the alignment and set the column width. The appropriate commands are given in Figure 7.

```
ww.TableSelectTable
ww.TableColumnWidth "", "0.38 cm", , , True, "0"
ww.centerpara
ww.charright 1
ww.LineDown 1
```

Fig. 7. Adjustment of a table

```
ww.editbookmark "a", 0, True
ww.charright 1
ww.LineDown 1
ww.editbookmark "b", 0, True
ww.EditGoTo "a"
ww.editbookmark "a", 0, , True
ww.TableSelectTable
```

```
ww.TableColumnWidth "", "0.38 cm", , , True, "0"
ww.centerpara
ww.EditGoTo "b"
ww.editbookmark "b", 0, , True
ww.charright 1
```

Fig. 8. Adjustment of a table for version 6 & 7

The first command in Figure 7 selects the complete table created while the second line forces the table to automatically adjust the width according to the actual values stored. It decreases the width of the selected cells as much as possible without changing the way how text wraps in the cells. The values are further centric aligned by the command *centerpara* (similar commands are *leftpara* and *rightpara*). The last two commands in Figure 7 first move the cursor outside the last row (the cursor remains in the row level, but outside of the table) and the last command finally moves the cursor below the table (a carriage return would result into a new row of the table).

The procedure in Figure 7 is valid for version 97, and later of WinWord processor. For versions 6 and 7 (95) the commands presented in Figure 8 must be implemented. If the routine presented in Figure 7 is applied with older versions of WinWord, the cursor sometimes remains in the last cell of the table.

10. Closure of Program and Termination of the Link between the Programs

After the analysis is completed, the results written into WinWord must also be stored and saved for printing. Although the user can simultaneously follow the creation of the final report and also request printing of the report before the engineering application has been closed, this option is not recommended as it sometimes causes errors, especially with older versions of WinWord.

Therefore, the application must save the report file prior to its closure, close the report file and also close WinWord. Figure 9 presents the necessary steps.

```
ww.filesaveas path$ & "name_of_report_file.doc"
ww.FileClose
ww.appclose
```

Fig. 9. Closure of report creation

In Figure 9, the first command saves the report file under the name, given between double quotation marks. The extension *doc* must be added to the name. To store the report file into a requested folder, the required path must be given in front of the file name. This command can be implemented also during the analysis to secure the partial saving of the file.

The second command in Figure 9 closes the report file. As the file was saved immediately prior to the closure, WinWord closes it without asking to save it.

The last command in Figure 9 closes the link between the application and WinWord by closing WinWord, even if it was already open prior to starting the engineering application. Any further attempt to write into the input file results in an error.

11. Conclusions

The paper describes some basic steps and techniques for creating an automatically generated output file by navigating the WinWord word processor. Some brief explanations and solutions to cover the differences among various versions 6, 7 (95), 97 and 2000 of the word processor are also presented.

A report created in such fashion is much more distinguishing than the standard output of the data as it can be equipped with all the main implemented equations and accompanied with the obtained results. To increase the quality of the output file corresponding templates with predefined headers and footers can be created separately (equipped with the logo of the user and other company data).

Using this approach it is also possible to offer the user to decide between short and full reports - a report with the main results only or the complete report where also the minor results are included. In

the case of the complete report the final user (or the reviser of the project) can track all the results obtained from the computational history which assures the transparency of the whole analysis. This is important not only from the aspect of qualitative representation of the results obtained, but also builds the confidence of the final user to the results obtained.

The objective of the paper is certainly not to explain every single command in detail, but to encourage the developers of engineering software to start implementing such techniques in their software that might make this kind of final report creation a future standard.

12. References

[1] Microsoft Corporation, “Microsoft Word Help Topics”, *Built-in help in WinWord*, 1983-2000