Notes on RichText() Class for Xbase^{++}

12/16/1999
Version 1.00

Author: Paul C. Laney
Internet: pclaney@compuserve.com

Original idea: Tom Marchione, Internet: tmarchione@compuserve.com

RIGHTS/RESTRICTIONS

2. As a Xbase^{++} user, you are granted the right to modify, compile and link the code for end-user application work, under the following conditions:

a. You do not redistribute the code without permission from the author.
b. All file headers are kept intact.
c. The source code and/or object modules are not offered for sale.
d. You do not port the code to another language, for other than personal use.
e. You do not offer a substantially competitive language product for sale or distribution.
f. You do not use the code in any kind of general purpose database management, reporting or computer language application.
g. You do not use the code in any type of auditing or emergency response application (NOTE: This restriction is based on the fact that the author has business interests in these markets. If you have an application in one of these markets, but would still like to use the RichText() class, please contact the author. Depending on the application and the market, it may be possible to make alternate arrangements).
h. You accept complete responsibility for the use of the code and any effects that it might have.
I. You understand that future versions of the code may not be offered for free (although you may continue to use this version for as long as you like).

CONTENTS OF VERSION 1.00

System Files:

README.RTF	This file
README.TXT	Plain text version of this file
ALASKRTF.PRG	Source code for RichText() class
RICHTEXT.CH	Header file for RichText() class
ADDON.PRG	Some extra functions which are used by the RichText()-class

Sample Files:

DEMRTF.PRG	Sample application to add to the TopDown-Demo
FLOWERS.DBF	Database file used by sample application

FLOWERS.DBT	Memo file associated with FLOWERS.DBF
VEGGIES.DBF	Database file used by sample application
RTFDEMO.RTF	Output file from sample application
MERGEIN.RTF	Input file for sample application mailmerge
MERGEOUT.RTF	Output file from sample application mailmerge

GETTING STARTED

If you're really curious about what this thing does, add the files to the Tddemo and compile and run the application Then open the output file (RTFDEMO.RTF and MERGEOUT.RTF) in an RTF-capable word processor. If you don't have time to compile and run the app, I've included unedited copies of the output files in the package.
Which files you have to add to the TopDown-Demo. Those files are:

demRtf.prg
alaskrtf.prg
addon.prg
richtext.ch

and to the DATA subdirectory:

flowers.dbf
flowers.dbt
veggies.dbf
rtfdemo.rtf
mergein.rtf
mergeout.rtf

Furthermore add the following lines to the bottom of your Project.xpj:

```
// Richtext
demRtf.prg
alaskrtf.prg
addon.prg
```

Last, but not least, add the following lines to demoMenu.prg before the HELP section:

```
********** RICHTEXT **********
oSubMenu := XbpMenu():New( oMenuBar ):Create()
oSubMenu:title := "~Richtext"

oSubmenu:AddItem( { "~RichText", {|| demRtf() } } )
oMenuBar:addItem( { oSubMenu, NIL } )
```

That’s all

NOTE: As indicated in the RTFDEMO.RTF file, this file includes support for an automated table of contents. However, to see it, you need to insert it. Consult your word processor documentation for instructions, if you have never generated an automated table of contents.

REQUIREMENTS

RichText() was written for use in Xbase^{++} applications, in combination with the TopDown library. Although it can also run without the TopDown-library (see WITHOUT TopDown). To use the class, you need to have a functional Xbase^{++} installation. Thanks to the Xbase^{++} pre-processor, you don't need to understand object-oriented programming to use the code. However, if you decide to fiddle with the source code of the underlying class, you will need to have a basic understanding of object-orientation.

NOTE: By changing one or two user interface functions (TdMsg() and TdYesNo()), it should be possible to use the class in pure DOS-based Xbase^{++} applications. You have to change it into:

```
FUNCTION tdYesNo( cTxt )
RETURN ( Alert( cTxt, { "Yes", "No" } ) == 1 )

FUNCTION TdMsg( cTxt )
   Alert( cTxt )
RETURN (NIL)
```

WITHOUT TopDown

The RichText-class can perfectly run without the TopDown-library. You only need to rewrite the functions tdYesNo() and tdMsg(). The DOS-based functions are shown above.

OVERVIEW

RichText() is a class written for Xbase^{++}, designed to export formatted data directly to Rich Text Format (RTF) files from within an application. RichText() allows you to leverage the power of common word processors as report formatting engines.

Version 1.0 of the class includes: (1) the source code, build and README files, (2) Xbase^{++} directives for xBASE-style command implementation, and (3) a small sample prg. Version 1.0 of the class is being offered at no charge. This version implements a small subset of the RTF specification, as taken from Microsoft documentation. RTF is a very detailed specification, and the intent of Version 1.0 is to provide easy access to the most basic and useful features of RTF, including attributes such as page setup, fonts, paragraph formatting, character formatting, headers & footers, tables, etc. To date, most of my testing has been done with Microsoft Word.

WHAT IS RTF?

RTF is an open Microsoft file specification for transferring formatted text between applications.

As such, it it's a great way to move formatted database information into word processors, in a **true** word processing format.

HOW CAN RTF HELP ME?

RTF is a great solution in at least three situations:

3. When there is a need to manipulate output in word processors, or integrate database information into other formatted documents.
4. When there is a requirement to build a custom report that must meet strict, detailed formatting guidelines.
5. When there is a need to publish a "turnkey" report that might include different formats, page orientations, fonts, etc. in a single document (my personal favorite…).

The RichText() class is a simple, low-overhead solution for any of these situations. You can produce one-of-a kind, free-form word-processing documents (for a good example, see the sample provided with the class). The class is inherently oriented toward paragraphs, bullets, hanging indents, page orientation, etc., not standard database reporting fare such as column headers, subtotals, etc. (although you can easily dump data into nicely-formatted tables). Moreover, the output files are **true** word-processing files, without extraneous spaces, carriage returns, and other "lint" that makes life difficult in a word processing context. If you have ever tried to pass off a flat text output file (or even some RTF files produced by third party report generators) as "word processor compatible", you'll know what I mean. Compatible? Yes. Usable? Hardly.

ESSENTIAL FEATURES OF RTF

For detailed information about RTF, consult the Microsoft specification, which is available on the Internet. It is also helpful to analyze the contents of RTF files generated from word processors. Although the xBASE layer of RichText() makes many of the technical details transparent, it will be helpful to clarify a few issues here.

Like many "open" file formats, RTF files are text files. RTF uses 7-bit ASCII characters (0-127) to represent formatting codes and data (higher-order ASCII characters must be represented in hexadecimal, which is handled transparently by RichText()). The two essential formatting characters in RTF are "\" (backslash) and "{}" (curly braces). All formatting commands are preceded by a backslash, and all "groups" are enclosed in curly braces. Groups delineate portions of a document to which combinations of formatting information may apply. Groups can be nested; the RTF file itself is considered a group, so all RTF files have at least one pair of curly braces, which encloses the entire contents of the file.

All RTF files have an RTF header (which is to be distinguished from the headers and footers that you may place in an RTF document). The RTF header appears at the top of the file and contains information about fonts, colors, styles, etc. For example, the font table within the RTF header contains a list of all of the fonts that are used in the document. To switch fonts later the document, an index into this font table must be used. A similar system is used for colors. The RichText()

class creates a basic RTF header for you.

An application that reads RTF files is called an "RTF viewer", which, for our purposes, is usually a word processor. For the most part, data items that you write to an RTF file appear within RTF viewers in the sequence that you write them. However certain RTF commands implement alternate "destinations", such as headers and footers. RTF also supports the "sections" that are found in Microsoft Word, with section-specific layouts and formats.

FINAL NOTES

As I use the class more and more, it has become obvious to me that the class is more useful than I had originally imagined. As a result, I have spent a lot more time developing it than I had originally imagined… <g>. In deference to the costs of adding new features, and improving the usability of existing ones, I probably will not be able to distribute future versions of the class for free. However, the cost should be insignificant relative to the benefits that you can offer to your end users, and your improved ability to give them exactly what they want for output.

Some additional features that are either planned for the future, or that are partially implemented already, include.

Improved Control Over Table Output
Improved Control Over Paragraph Formatting
Improved Application-Level Components
Colors
Lines and graphic objects
Pictures
Styles
Language Support
Footnotes/Endnotes
Positioned Objects and Frames
Index Entries
Bookmarks
Anything else in the RTF spec that seems useful…

If all goes well, Version 2.0 will effectively become a complete database publishing tool, and we will all become custom documentation gurus to our end users ...<g>. In addition, I may develop a companion class for use with Delphi (so far, in my limited use of Delphi, I have not come across a class that generates comprehensive rich text output, although Delphi does provide an RTF-aware control).

I greatly appreciate the feedback that I have received to date. If you have any ideas or suggestions, I'd love to hear them.

Have fun!

Paul C. Laney

VERSION 1.00 FEATURE ADDITIONS

1. Page Number Support

   You can add a simple page number via the **INSERT PAGENUMBER** command. Typically, this is most useful when defined within a header or footer.

6. File Checking

   When defining the RTF, the **WARNOVERWRITE** clause will cause the class to generate a warning message if an existing file will be over-written. In addition, you can use the **GETFILENAME** and **FORCEPATH** clauses to allow the user to specify a file name, as well as to force the file to be written to a specific path.

7. Table of Contents Support

   You can generate an automated table of contents. You can mark any paragraph for inclusion in a table of contents at a particular level by specifying the **TOCLEVEL** clause. Refer to the sample application for a simple example.

VERSION 0.91 FEATURE ADDITIONS

Merge Away!

I added a crude subsystem and demonstration of how one could exploit RTF files for defining report formats and merging database information into them. With this system, you can embed Xbase^{++} expressions throughout the "primary" merge file (which can be any old RTF), and then run a relatively tiny Xbase^{++} program to merge your database info into it. It's very much like a standard "mail-merge" that you might do from a word processor, but with the sophistication and flexibility of Xbase^{++}.

For a simple, silly example, see files MERGEIN.RTF and MERGEOUT.RTF. MERGEIN.RTF is a "primary" merge file, and MERGEOUT.RTF is the actual file that is produced by the example program. Notice that MERGEIN.RTF contains a number of delimited Xbase^{++} expressions. Most of these are intended to return text, but I also threw in a couple examples to demonstrate how you might control the record pointer from the merge file itself. In other words, any valid, macro-compilable Xbase^{++} expression is fair game. Of course, any functions that you call in the primary merge file must be linked into your program (if this doesn't ring a bell, see the Xbase^{++} documentation regarding "REQUEST").

The wonderful advantage of this approach is that you can effectively use a word processor (!) to design a report format (for certain types of applications). If you need to tweak the look of the output, or even change it drastically, you just load up MS-Word and make the change to the primary

file -- no clumsy report writer, and no code changes. And, of course, you get all of the power of a **true**, word-processor based report.

The downside of the system, as it currently exists, is that it is very slow. My algorithm is not particularly efficient, and there are inherent speed limitations. Not only does it have to macro compile every Xbase^{++} expression that it encounters, it does it **repeatedly** for each record. Arrrgh! In addition, I have not yet included an error handler to trap bad Xbase^{++} expressions, though this would be simple to do.

The system seems to work OK with the example, but be warned that I have not tested it with anything else yet. I imagine that it would be easy to make it self-destruct in its current incarnation...