RATF Analysis using the RAT (Random Analysis Tool) and RD (Random Dump)

Synopsis:

The RAT Random Analysis Tool

```
Rat(C) Copyright Bill Buckels 2013. All Rats Reserved.
Random Analysis Tool to determine record length, field layout, view data,
dump data and obtain stats for DOS 3.3 Random Access Text Files (RATF's).
Usage:    rat filename -l(record length 1-256) (-options)
          rat filename configfilename (-options can be in config file)
Options: -n(number of records to list) -r(from record 0-32767) -r(to record)
         -b(byte offset field list) i.e. -b0,10,20
          note: byte offset can be used to select fields of interest only
         -a select all records (by default only active records are displayed)
-s -d -v slot drive volume i.e. -s6 -d1 -v254
command line record length and options are preceded by a dash as shown.
config file options are of the form l=(record length), etc. (name=value pairs)
additional config file option i.e.      h=FieldName,FieldName,FieldName
field header option when used with i.e. b=0,10,20
Output:   7 bit ascii to the screen, can be redirected: rat > tofile
```

The RAT is a command line program for the Aztec C65 DOS 3.3 "Unix-like" Shell written in Aztec C65 (CII Version). It is both a diagnostic tool for determining the contents of a DOS 3.3 Random Access Text File (RATF) and a file dump utility for RATF's.

The RAT's file dump comes in 3 flavors:

Diagnostic: Long List of Annotated Field Data Based on Record Length Only.
Export: Comma Separated Value (CSV) Format Based on a "field map". Suitable for importing into a DataBase.
Rescue: A Long List of each active text field in the RATF.

The RAT's options can be specified using the command line, a config file, or a combination of the two. When using a config file, only the file name of the RATF being read followed by the config file name needs to specified... all other information that can be entered on the command line can be in the config file, which uses the same single character commands as their command line equivalent.

RD Random Dump Hex Viewer

```
RD(random dump)(C) Copyright Bill Buckels 2013. All Rats Reserved.
Usage: rd options file1 [options] [file2] ... [> myfile.txt]
This program performs a dump in hex and ascii of the specified
file(s) to the standard output. Default is to the screen but can
also be redirected to a file, a printer, or through a filter program.
Options: r p w l o h t - options with values cannot be combined.
+  switch on - options without values can be combined i.e. +rpt
+r switch - view file as random access text file
+p switch - view file a page at a time - Q or ESC to exit
+w(value  - decimal) - display width 1-16 (default 16)
+l(value  - decimal) - record length 1-256 (off by default)
+o(value  - decimal) - view file beginning at offset (default is 0)
+h(value  - hexadecimal) - same as above
+t switch - view file as 7 bit values (strip hi-bits - all files)
-  switch off - options can be combined or separate i.e. -rp or -r -p
Over-rides: +s(slot 4 5 6 7) +d(drive 1 2) +v(volume 1-255)
```

RD is a command line program for the Aztec C65 DOS 3.3 "Unix-like" Shell written in Aztec C65 (CII Version).

RD has more features than a typical hex viewer including user defined screen width and has some of the same features as the RAT like record length, but is more general and provides a view of all DOS 3.3 file types, although it too works with RATF's.

Like the RAT, RD's output can be redirected to a disk file. But unlike the RAT, RD's output is purely diagnostic.

CHTYPE Command Line Utility to Change DOS 3.3 File Type

```
Usage:     chtype filename -tfiletype (-options)
Filetypes: -tT -tI -tA -tB -tR -tS -ta -tb (case sensitive)
Options:   slot drive volume i.e. -s6 -d1 -v254 (default)
Filetype and options are preceded by a dash as shown.
```

CHTYPE is included on the RANDOM33.dsk which is the DOS 3.3 distribution disk the RAT and RD and their demo files are on. It is also distributed separately. CHTYPE can be used to change a RATF from a text file to a type S file for DOS 3.3 applications written in other languages like C that expect sequential text files only.

Text File Storage Format in DOS 3.3 and Other Old Systems

So you want to work with RATF's? Or maybe "want" is too strong a word!

We'll talk about the RAT and RD shortly, but first if you are under the impression that there are 3 types of 7 bit text files, all sequential; one that lives on the Mac (or in ProDOS) with carriage returns at the end of each line, one that lives in Unix and Linux with linefeeds at the end of each line, and one that lives in Windows (and MS-DOS) with carriage returns and linefeeds at the end of each line, you are generally correct with some notable caveats.

Each system of the early days had its own text character set. For example, in 1963 while IBM worked with the ASCII standardization committee, they devised EBCDIC which extended BCDIC.

EBCDIC was described as [abridged] "An alleged character set ...in at least six mutually incompatible versions, all featuring such delights as non-contiguous letter sequences and the absence of several ASCII punctuation characters... (exactly which characters are absent varies according to which version of EBCDIC you're looking at) ...spurning the already established ASCII standard. ...IBM's own description of the EBCDIC variants and how to convert between them is still internally classified top-secret... a manifestation of purest evil."

The Commodore 64's PETSCII evolved from that early 1963 Ascii Standard and not the 1967 version, which most if not all other computer character sets based on ASCII use.

ASCII was the most commonly used character encoding on the World Wide Web until December 2007, when it was surpassed by UTF-8. By that time decades had passed and systems (including DOS 3.3 and ProDOS) had come and gone.

The Ascii Standard encoded only 7 bits (0-127), with the 8th bit added and used as a parity bit to detect transmission errors over serial communication lines. While the IBM-PC and MS-DOS had used the upper characters 128-255 to extend 8 bit Ascii for such uses as graphics characters for display, and allowed the storage of the higher characters in its text files, DOS 3.3's BASIC operating system did not.

And unlike later Apple Systems like ProDOS and on the Mac, or MS-DOS and the Unises of the day, DOS 3.3 did not simply store its low ascii as 7 bit low ascii text in its text files (although DOS 3.3 made its text appear like 7 bit text to a BASIC programmer).

The Text in DOS 3.3 text files is 7 bit Ascii with the parity bit set.

An Overview of the RATF format

The RATF (Random Text File) was even a neater idea than misrepresenting 7 bit Ascii by storing 7 bit text with the parity bit set!

* Extending the same format used for storing strings in sequential text files complete with carriage returns in fixed length records padded with Ascii 0's was another neat idea, because the resulting RATF extended DOS 3.3's BASIC "sandbox" beyond the realm of sequential storage without the need to open (and expose) the other DOS 3.3 file types!

* Allowing BASIC programmers to write past the end of records in a RATF was even a neater idea! Records without borders!

* Even neater yet was using byte offsets at random to put random text strings into random records and not bothering with fixed length fields.

* Not storing the number of records anywhere in the file gave the BASIC programmer of the day even more reason to ignore good programming practices and the freedom to just put any old text at all anywhere in a file at random. The data integrity problem was solved! For the BASIC programmer there was no data integrity needed!

* Not putting a header in the RATF format with field lengths and record lengths eliminated the need to have other file types in BASIC. Who needs other filetypes when you are programming in BASIC anyway!

* And not having a file length in bytes eliminated the drudgery of worrying about appending to the end of something. After all, DOS 3.3 really only had sectors linked to other sectors. What would append have meant?

* Not needing to have data in a file was cool and being able to leapfrog around empty records while writing data into others at random may have been the greatest departure from structured database programming of all.

Protecting the DOS 3.3 File System from BASIC Programmers

So as you can see by all the neat things that Apple Computer allowed BASIC programmers to do with text files, the use of other DOS 3.3 file types had little purpose (with the exception of Binary Image Files created with BSAVE and read into memory with BLOAD). The BASIC programmer (like the Aztec C65 programmer of the day) really needed only Text and Binary files. The DOS 3.3 system and the applications programmer's use of the other DOS 3.3 filetypes was secured for those more capable of doing damage!

In order to protect BASIC programmers from themselves, DOS 3.3 BASIC opens text files only and no other DOS 3.3 file types. A DOS 3.3 BASIC program will generate a FILE TYPE MISMATCH error when it tries to open any DOS 3.3 file type other than a text file!

But if you are not a BASIC programmer, it is only necessary to keep RATF's as text files to share with BASIC and other applications that expect a RATF to be a text file.

Changing RATF's to Type S Files

There is no other reason that a RATF cannot be a type S file. And if you want to protect your RATF from DOS 3.3 BASIC programmers, change your RATF to a type S and leave it that way. You can use the chtype utility that comes with the RAT (or some other means) to change it to a type S file.

While you are at it, you could build yourself some sorted indices (also type S to protect them from BASIC programmers) for key fields with record and field offsets from your RATF to avoid the overhead of sequential searches.

And while you are at it you could write yourself a packing utility that rebuilds your indices when done packing. And you could use an unprintable character (like Ascii 1) to mark deleted records in your type S database and deleted nodes in your type S indices, and consider converting these permanently to 7 bit plain text and porting them to ProDOS for faster file access.

While you are at it, you could also consider losing the carriage returns at the end of the fields and create a header format for storing types other than text.

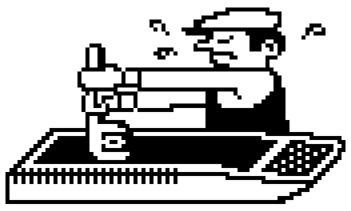
Record Length Analysis:

If you are running in an emulator like AppleWin, and have your speed set to fastest, drink your favorite beverage quickly... this will not likely take too long if the BASIC programmer who created the RATF has not succumbed to the inherent unstructured random laziness promoted by this file format's flexibility.

Record Lengths

The RAT and RD both let you enter hypothetical (random) record lengths between 1 and 256, and you get to see how it goes, while they dump the RATF to the screen (or you can redirect output to a disk file since these are shell programs).

The RANDOM33.dsk that the RAT, and RD (Random Dump) live-on contains a sample RATF called RATF (of course). RATF provides a predictable test suite that you can work with to try-out the RAT and RD yourself and follow along if you wish.



MKRAT BASIC program used to create RATF

```
 100  REM CREATE TEST DATA FOR RAT
 105  REM VALUES FOR FIELD DATA
 110 A$ = "FIRST"
 120 B$ = "SECOND"
 130 C$ = "THIRD"
 135  REM BYTE OFFSETS FOR FIELD DATA
 140 X$ = ",B0"
 150 Y$ = "WRITE RATF,B10"
 160 Z$ = "WRITE RATF,B20"
 170 D$ =  CHR$ (4)
 190  REM WRITE RECORDS INTERLEAVED WITH BLANK RECORDS
 200  PRINT D$;"OPEN RATF,L30"
 210  FOR J = 0 TO 10
 220 K = J * J
 230  PRINT D$;"WRITE RATF,R"; STR$ (K);X$
 250  PRINT A$
 260  PRINT D$;Y$
 270  PRINT B$
 280  PRINT D$;Z$
 290  PRINT C$
 300  NEXT J
 310  PRINT D$;"CLOSE"
 320  END
```

A Visual Chore

Determining the the Record Length of a RATF in DOS 3.3 is a visual chore. Unlike in ProDOS, which stores the file length in bytes in the Volume Directory or Subdirectory which is readable as a file, and which stores the number of records in the Auxiliary Type field (ProDOS GetFileInfo Call),

You can use a hex viewer like RD (Random Dump) to determine if some of the fields line-up (if you can find one like RD).

Example RD Output: Default Width - Record Length = 30 (+L30)

```
Command Line: RD +RT +L30 RATF > myfile.txt
```

```
00000 46 49 52 53 54 0D 00 00 00 00 53 45 43 4F 4E 44  FIRST.....SECOND
00010 0D 00 00 00 54 48 49 52 44 0D 00 00 00 00        ....THIRD.....

0001E 46 49 52 53 54 0D 00 00 00 00 53 45 43 4F 4E 44  FIRST.....SECOND
0002E 0D 00 00 00 54 48 49 52 44 0D 00 00 00 00        ....THIRD.....

0003C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0004C 00 00 00 00 00 00 00 00 00 00 00 00 00 00        ..............

0005A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0006A 00 00 00 00 00 00 00 00 00 00 00 00 00 00        ..............
```

Note how RD's view splits records in the RATF with blank lines and provides the starting offset for each record. This is a more useful view that you would expect from a hex viewer. Like any other hex viewer RD can also provide a standard view as well but the view above is somewhat more helpful for this visual chore. Note also that the hex display on the left is in low-ascii values. We even see the "real" Ascii values like carriage returns that appear as unprintable characters on the right. And the characters on the right are converted as well. This conversion of hex to low-Ascii is not RD's default view.

RD's default view of any file is the same as other hex viewers
like the one that is built into CiderPress; hex values are not converted to
low-ascii. But RD has more than one way to skin a RAT!

Sample RD Output: Width = 10 (+W10)

Command Line: RD +RT +L30 +W10 RATF > myfile.txt

```
00000 46 49 52 53 54 0D 00 00 00 00                    FIRST.....
0000A 53 45 43 4F 4E 44 0D 00 00 00                    SECOND....
00014 54 48 49 52 44 0D 00 00 00 00                    THIRD.....

0001E 46 49 52 53 54 0D 00 00 00 00                    FIRST.....
00028 53 45 43 4F 4E 44 0D 00 00 00                    SECOND....
00032 54 48 49 52 44 0D 00 00 00 00                    THIRD.....
```

Note how RD's view of the same data is still split into records with blank
lines in between. But now we have adjusted the screen to fit the record
bounds and since the sample data that I have used has nicely aligned fields
the display is very pretty indeed!

RD has lots of other features too, and you are encouraged to explore them.
Refer to the document titled RDUMP.txt that came with all this for more
detail about RD, or see RD's source to see how it does what it does, and try
it out yourself on some different files to see how that goes!

But a hex viewer knows its place (and RD definetly is a team player in a
game of RAT quest), but when it comes to catching RATF's it doesn't hurt to
think like a RAT! So now let's have a look at what the RAT does with a RATF
and a record length.

Command Line: RAT -L30 RATF

```
========Record 2
0  : FIRST
10 : SECOND
20 : THIRD
========Record 3
0  : FIRST
10 : SECOND
20 : THIRD

cont'd

========Record 51
0  : FIRST
10 : SECOND
20 : THIRD
========
52 records read (L = 30) 8 active.
partial record - 6 bytes...
```

Like RD, this works pretty well! And the RAT takes a somewhat different view
than a typical hex editor's display can provide; it breaks out only the
fields that it finds. You can see some additional information in the left
column beside the field data (above). Those are byte offsets in the records.

So it follows that if the text in the fields is recognizable and contains
some common content at the same relative position in the record, and if the
byte offsets line-up, you have likely found the RATF's file format, and
maybe you are even ready to extract the data using the RAT, and import it into
a real database. Or whatever your reason.

Now let's see what happens when we pick the wrong record size with the RAT!

Command Line: RAT -L20 RATF

```
========Record 2
0  : FIRST
10 : SECOND
========Record 3
0  : THIRD
10 : FIRST
========Record 4
0  : SECOND
10 : THIRD
========Record 8
0  : FIRST
10 : SECOND
========Record 9
0  : THIRD
========Record 15
10 : FIRST
========Record 16
0  : SECOND
10 : THIRD

etc...
```

As you can see above, if one iterates through a series of record lengths using the RAT, a decipherable pattern should occasionally emerge. This is also doable in the RD hex viewer, but the output is not formatted as clearly. You have probably noticed by now that by default, the rat skips records that are unused.

Using the Aztec C DOS 3.3 Shell's scripting capabilities, and redirection to a disk file you could if you wished, automate a partial dump by calling the RAT in the script repetitively with different record lengths to "zero-in" on the record size by reviewing your output afterwards and determining your "best fit".

Rescue Mode -L911 and Negative Mode --

If someone has used a RATF for random information and variable length fields at random, a hex editor (even RD) might be a little annoying or maybe even useless for zeroing-in on what you got!

Remember, not only is a RATF for random access, but also for storing random text strings with no notion of fields whatsoever. At some point it may be time to cut your losses and use the RAT to dump every bit of text in a RATF and call it a day.

The RAT actually has two settings for that (both unlisted in the RAT's usage) and both settings take only the arguments provided below...
all other switches are cancelled:

Rescue Mode -L911

Usage: RAT -L911 RATF > icu.txt

A long list of all the text in the RATF. For field level recovery.

Negative Mode --

Usage: RAT -- -L(record length) RATF > dreamsoffields.txt

Before setting the file length to 911, which is your last resort with the RAT, if you are convinced that you have a RATF that has valid string data that has been stored in records but in freeforms, or maybe several particular orders (some buggers are creative and will mix several record layouts in one file) you can use the "undocumented" -- switch.

This mode can be problematic because unlike Rescue Mode (-L911) you do not end-up with a long list. Instead you end-up with a comma separated value (CSV) list with a variable number of entries per line.

In the 1990's I programmed in a database language on the IBM-PC called Clarion which allowed "unions" in database files called groupovers (I think). This meant that several structures could co-exist in a single file. So all kidding aside if this is what you have, then the -- (negative mode) switch is your only option for a RATF export. You will need to split your records later manually... the RAT can't help you with that.

Soon we will look at the best case scenario! But now some features of the RAT and some other information that we have skipped-over:

Record Counts

The RAT gives you counts, based on the random record length you entered; active records, and number of records.

Tailing Bytes

There is one hitch of course, and since the DOS 3.3 file system is in sector length rather than byte length like ProDOS, or for that matter, modern filing systems, you can get a load of tailing bytes at the end of the records. So the RAT lets you enter a number of records, or first record and and last record so you are not spending humunguous amounts of time staring at your screen watching the bytes after the "real" data flows past.

Unused Records

A RATF can contain huge chunks of unused records as well, so you may need to make a number of short passes to zero-in on the right size (when the fields in the records line-up) then make a final pass.

If you redirect your output to a disk file, you will end-up with a 7 bit Ascii sequential file in ProDOS (Mac) format that you can even import into a MS-Windows data base like MS- Access if you wish.

Field Length Analysis:

A well formed RATF with consistent fields, that has mostly data in it, and terminates the data with carriage returns, and pads the data with NUL bytes, can be an easy chore to analyze... one can hope!

The RAT also allows you to specify a field map either on the command line or in a config file.

You will likely have varying degrees of success analyzing field sizes, since a RATF is just a primitive flat file capable of storing only text mixed with NUL bytes, whether in ProDOS or DOS 3.3.

On the other hand, common Data like First Name, Last Name, Address, City, Province, PostalCode, Phone etc. could be patterned in such a way that you are likely to succeed, and all the better if you already know what the RATF is all about. As I say, degrees of success can vary using RAT, but if the first fields line-up and the data makes sense, even if all fields aren't filled, the RAT can give you a pretty good idea what's in a RATF file.

Config Files and the RAT's Output - CSV (comma separated value) lists

RATF.CFG

```
# ---------------------------------------------
# RAT config file example
# same as the rat command command line switches...
# Additional Features can be added here too...
# the following two are the same as their basic
# counterparts as well.
# ---------------------------------------------

* L - record length
l=30

* B - byte offsets (fields)
b=0,10,20

* H - header field names, for delimited lists.
h=FirstName,MiddleName,LastName
```

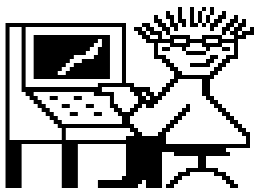
The command line to try this is as follows:

```
RAT RATF RATF.CFG
or
RAT RATF RATF.CFG > disk.file
```

Sample Output

```
FIRSTNAME,MIDDLENAME,LASTNAME
FIRST,SECOND,THIRD
FIRST,SECOND,THIRD
FIRST,SECOND,THIRD
FIRST,SECOND,THIRD
FIRST,SECOND,THIRD
FIRST,SECOND,THIRD
FIRST,SECOND,THIRD
FIRST,SECOND,THIRD
```

Last Word



Good luck using the RAT, RD (Random Dump), CHTYPE, and the other programs the RAT comes with.

Bill Buckels
bbuckels@escape.ca
Aug 2013