

Creating Unique Digital Signatures using CRC32 Standard

John Fulda, Kos Pharmaceuticals, Inc., Weston, FL

ABSTRACT

CRC32 is a standard algorithm used by commonly available software to produce an eight-character, hexadecimal number using all the bytes in a target file. This number, a “digital signature”, changes when any byte in the target file changes. The digital signature does not change when the file name or creation date changes.

Using this SAS program to calculate the CRC32 signature for all files in a folder enables the user to determine that the target file is identical in content to the original source file. Any file, regardless of file type, can be validated that no changes have taken place. This program is the core for a system that manages file integrity.

INTRODUCTION

This program was developed as an inexpensive, easy to implement solution to several real world problems:

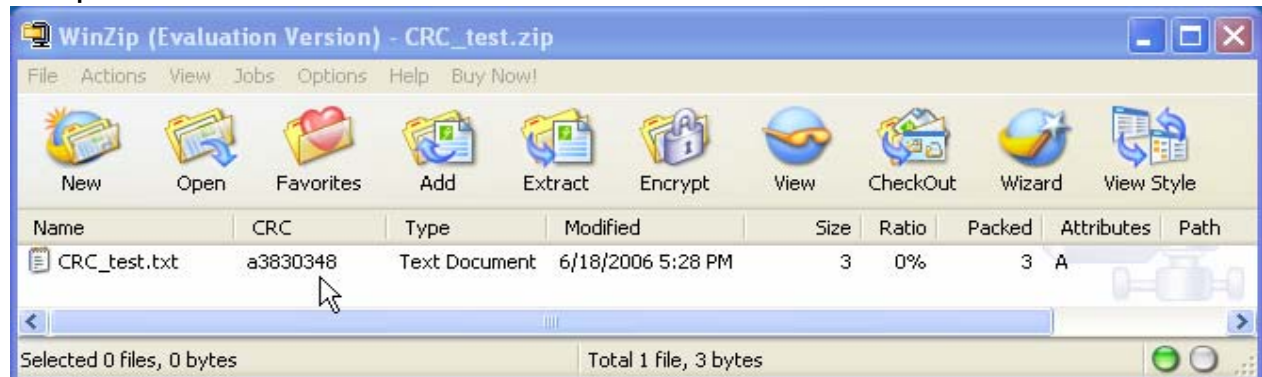
- First, when a SAS program is validated, how can I be sure that the copy of the SAS code that I am using is identical to the validated copy?
- Second, when I have multiple copies of SAS programs, which ones are the same? This becomes more of a challenge when several people share programs, and change the file names and the creation dates.
- Third, how can I be sure that two data files are identical, particularly when they are very large files which would take a long time to run a PROC COMPARE?
- Fourth, when files exist on multiple computers, how can they be compared without needed to move all the files onto the same computer?
- Fifth, how can a solution be found to all of the above that is auditable and creditable within the business world? Of particular concern to someone who works in the Pharmaceutical industry, how can I be assured that the solution implemented in SAS will be acceptable in documentation submitted to the FDA (Food and Drug Administration)?

SOLVING THESE FIVE PROBLEMS

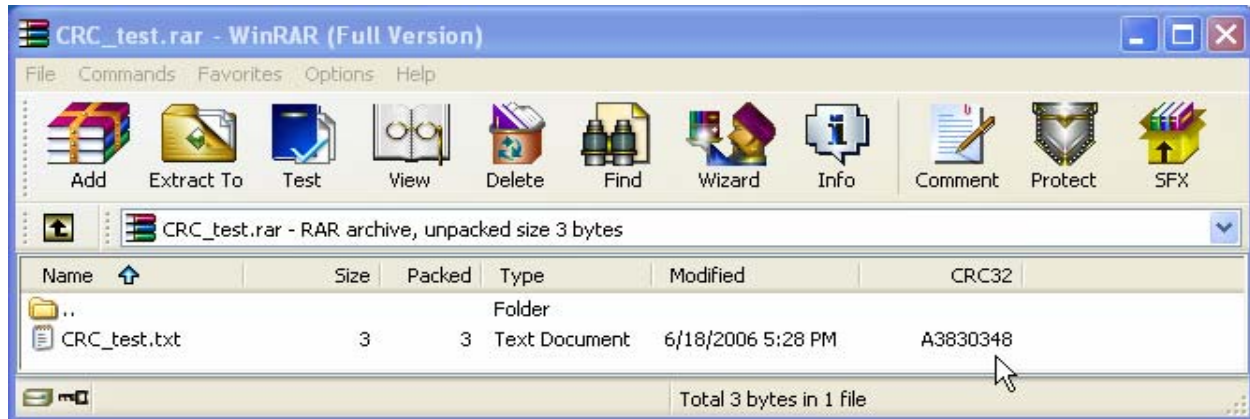
In trying not to reinvent the wheel, I investigated what software was already available, which was accepted as a fast and reliable method to calculate a unique digital signature. I discovered that the most commonly used software incorporate what is called a “Cyclic Redundancy Check” or “Cyclic Redundancy Checksum”, and is abbreviated as CRC-32. (or “CRC 32” or simply “CRC”) This algorithm, a set of rules for reading a file, and computing the results, has been adopted as an international standard by ISO (International Organization for Standardization: ISO 3309), ITU (International Telecommunications Union: ITU-T V.42), and the International Hydrographic Organization (IHO: S-57). Since CRC32 is a widely accepted standard, writing a SAS program which implements this standard will give the SAS program the credibility it needs to be accepted.

Of additional interest is the fact that CRC 32 is already available in such commonly used programs as WinZip, WinRAR, and Windows XP zip file browser. Screen prints are shown below of these programs. In some cases, the CRC display was hidden and not included in the default display. In that case it was necessary to set the program Options to show the CRC values. Note that the **cursor in the screen capture image points to an eight-character hexadecimal number** titled CRC, CRC-32, and CRC 32. Even though the display may be in upper or lower case, the CRC value for the test file, CRC_test.txt is the same for all three programs: **A3830348**.

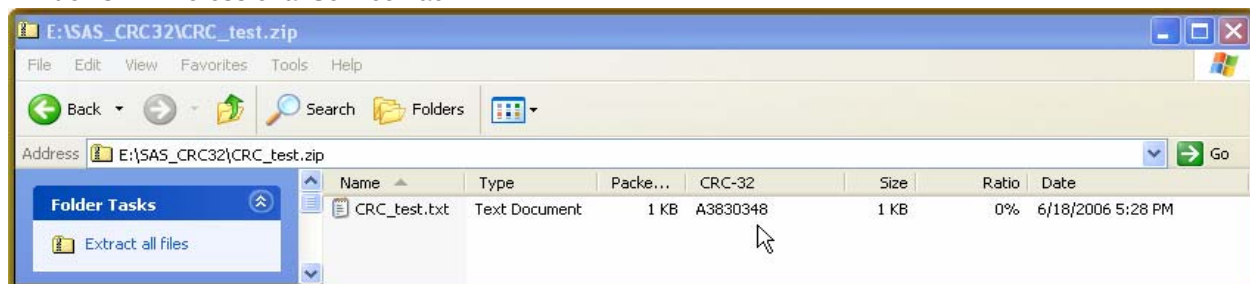
WinZip Version 10.0



WinRAR Version 3.10

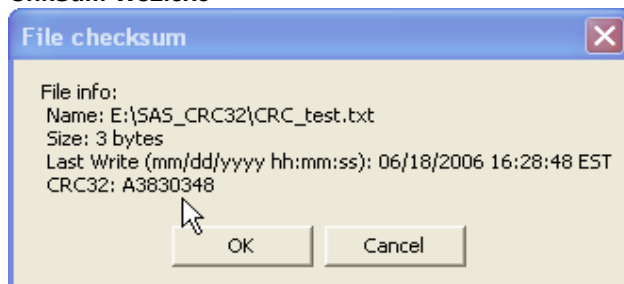


Windows XP Professional Service Pack 2



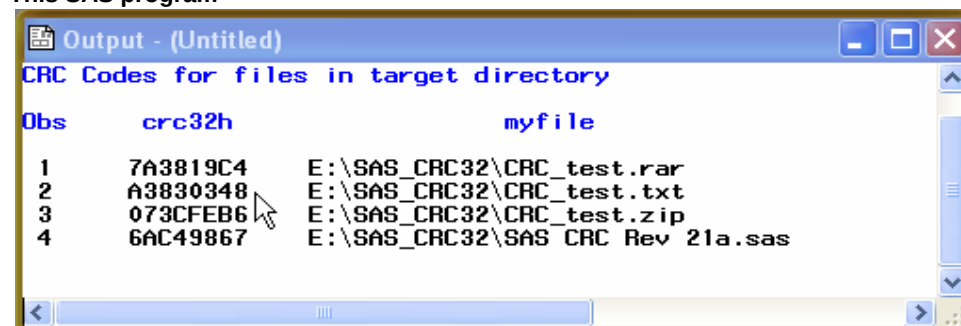
In exploring this feature further, I downloaded some shareware which also computed the CRC code. The results were the same digital signature:

ChkSum-W32.exe



Finally, after writing this SAS program, I compare the results to all of the above, and got the same answer for the same file CRC_test.txt: **A3830348.**

This SAS program



HOW DOES THE PROGRAM WORK?

Answer: quite well! I'm not a mathematician, so I will have to depend on the explanations I read. As I understand it, the CRC algorithm is a highly efficient way to read all the bytes in a file, and perform calculations on each and every byte in such a way that the change of even on bit out of millions and millions of bits will cause the CRC to arrive at a very different 32 bit number show as an 8 digit hexadecimal number. There are over 4 billion unique values (2^{32} , or 4,294,967,296 values). The odds of two different files having the exact same CRC code are about 4,000 times greater than of being struck by lightning. Twice.

There are different implementations of Cyclical Redundancy Checks. CRC 32 uses 32 bit numbers, and that is why there this standard is called CRC 32. The CRC uses what is called "polynomial modulo two arithmetic". There are a number of articles listed at the end of the article for those who have the background and interest to explore the inner workings of the algorithm further. I didn't need to know how it work; all that I needed is to know that it works.

In the 1980s, CRC-32 was adopted as a method to detect errors in data transmission. Since the source file and the destination file do not have to be on the same computers, all that is necessary is to run the CRC-32 check before sending a file, send the file, send the CRC-32 code, and then compute the CRC-32 value on the received file. One version of XMODEM file transmission protocol uses this technique, as well as ZMODEM and HDLC. The old IBM 8-inch floppy disk controller used a similar algorithm for error-detection, and since them most all personal computer disk controllers use this. A disk controller computes a CRC as it writes a disk sector, and then it appends that CRC to the data. When the data is read back, a new CRC is computed from the recovered data and compared to the original CRC. If the CRC values differ, an error has occurred and the operation is repeated.

WRITING THE PROGRAM

How do you write a program when you don't understand the algorithm? My goal was to implement CRC-32 in SAS, so that I could take the results and develop applications from this routine. In searching the internet, I found programs in FORTRAN, C, and Visual Basic, but not in SAS. These was only one SAS reference to CRC, and that was in a SAS discussion group, which did not give the SAS code. (Note: As I was finishing up this article, I did a fresh GOOGLE search, and discovered that there has been one article published with SAS code for CRC-32. I've listed it at the end of this article, under "Recommended Reading". I learned a lot about coding from working on this project, and am glad that I was not aware of this article until I was able to find my own solution .

Since I have a cookbook knowledge of Visual Basic, I tried several examples until I could find one I understood. Then I modified the Visual Basic program to print intermediate results. Using this as my working model, I then began writing the SAS program, frequently printing intermediate results that matched the Visual Basic results. I've added comment at the end of the SAS listing about my program logic. Some of the formats, such as **s370fib4**. I learned by reading the SAS manual and testing them. I dumped each value in binary, decimal, and hexadecimal until I got the desired results. This SAS code is not the most compact, or the most elegant. But it works. And it is given here as an example of how, through brute force, persistence, and trial-and-error it is possible to find solutions to very technical problems if you break them down into small parts, and solve each part one at a time.

SAS CODE

Following is the SAS program which computes the CRC-32 code for every file in the specified folder. Identification numbers to the right of the SAS code match the commentary which follows this listing.

```
* SAS CRC32 calculation;
* -----*;
*   This program uses logic based on the Visual Basic program found at:
*   http://www.vbaccelerator.com/home/VB/Code/Libraries/CRC32/article.asp
* -----*;
*-- Create table of 255 values one time;
*-- Results are global macro variables used in the CRC32 calculation;
* -----*;

options noxwait;
%let crc_setup_status=STARTUP;

%macro crc_setup;
  %global crc_setup_status;
  %if &crc_setup_status=EXECUTED
  %then %goto exit;

  data table;
    * results are macro variables computed in the program;
```

1

```

* no input files or output files need to be defined;
*--create all variables using hex values. Store as positive integers;
*--for easy of coding, this conversion can take four steps;

hex_100      =input('00000100'x,s370fib4.);
hex_100h     =put(hex_100,hex8.);
hex_2        =input('00000002'x,s370fib4.);
hex_2h       =put(hex_2,hex8.);
hex_FF       =input('000000FF'x,s370fib4.);
hex_FFh      =put(hex_FF,hex8.);
hex_FE       =input('FFFFFFFE'x,s370fib4.);
hex_FEI      =input(put(hex_FE,hex8.),hex8.);
hex_FEh      =put(hex_FE,hex8.);
hex_FEIh     =put(hex_FEI,hex8.);

dwPolynomial =input('EDB88320'x,s370fib4.);
dwPolynomialh=put(dwPolynomial,hex8.);
dwPolynomialI=input(put(dwPolynomial,hex8.),hex8.);
dwPolynomialIh=put(dwPolynomialI,hex8.);
hex_7f       =input('7FFFFFFF'X,S370FIB4.);
HEX_7Fh      =put(hex_7f,hex8.);
hex_7fi      =input(put(hex_7f,hex8.),hex8.);
HEX_7FIh     =put(hex_7fi,hex8.);
hex_00       =input('FFFFFF00'x,s370fib4.);
HEX_00h      =put(hex_00,hex8.);
hex_00i      =input(put(hex_00,hex8.),hex8.);
HEX_00ih     =put(hex_00i,hex8.);
hex167       =16777215;
hex167h      =put(hex167,hex8.);
hex167i      =input(put(hex167,hex8.),hex8.);
hex167ih     =put(hex167i,hex8.);

*--using the hex-variables defined above, now create 256
macro variables to be used later;
do i = 0 to 255;
    dwCrc = i;
    do J = 8 to 1 by -1;
        dwBand1 = band(dwCrc,1);
        if dwBand1 then
            do;
                dwCrcA = band(dwCrc,hex_FEI);
                dwCrcB = int(dwCrcA/hex_2);
                dwCrcC = band(dwCrcB,hex_7fi);
                dwCrcD = bxor(dwCrcC,dwPolynomialI);
                dwCrc = dwCrcD;
            end;
        else do;
            dwCrcA = band(dwCrc,hex_FEI);
            dwCrcB = int(dwCrcA/hex_2);
            dwCrcC = band(dwCrcB,hex_7fi);
            dwCrc = dwCrcC;
        end;
    end;
    dwCrc_hexval=put(dwCrc,hex8.);

*---covert from unsigned to signed number ---*;
c8      =input( put(dwCrc,hex8.),$hex8.);
dwCrc_sign =input(c8,s370fib4.);
call symput('look' || left(i),dwCrc);
output;
end;

run;
%let crc_setup_status=EXECUTED;
%exit:

```

2

3

4

```
%mend crc_setup;
```

```
%macro getcrc32(myFile);
```

```
  %crc_setup;
```

```
  filename raw "&myFile";  *--for each file...;
```

```
  data b;
```

```
    infile raw lrecl=1 recfm=f unbuffered;  *--Read each byte;
    retain crc32Result;
```

```
  *--define constants used in this calculation--*;
```

```
  hex_FFF      =input('FFFFFFFF'x,s370fib4.);
```

```
  hex_FFFI     =input(  put(hex_FFF,hex8.),hex8.);
```

```
  hex_FFFh     =put(hex_FFF,hex8.);
```

```
  hex_FFFFIh   =put(hex_FFFI,hex8.);
```

```
  hex_FF       =input('000000FF'x,s370fib4.);
```

```
  hex_FFh      =put(hex_FF,hex8.);
```

```
  hex_00       =input('FFFFFF00'x,s370fib4.);
```

```
  HEX_00h      =put(hex_00,hex8.);
```

```
  hex_00i      =input(  put(hex_00,hex8.),hex8.);
```

```
  HEX_00ih     =put(hex_00i,hex8.);
```

```
  hex_100      =input('00000100'x,s370fib4.);
```

```
  hex_100h     =put(hex_100,hex8.);
```

```
  hex167       =16777215;
```

```
  hex167h      =put(hex167,hex8.);
```

```
  hex167i      =input(  put(hex167,hex8.),hex8.);
```

```
  hex167ih     =put(hex167i,hex8.);
```

```
  if _n_=1 then crc32Result=hex_FFFI;  *-- set initial value--*;
```

```
  input icode pib1.;  *--get the next byte;
```

```
  iLookupa     = band(crc32Result,hex_FF);
```

```
  iLookup      = bxor(iLookupa,icode);
```

```
  crc32Resulta = band(crc32Result,hex_00i);
```

```
  crc32Resultb = int(crc32Resulta / hex_100);
```

```
  crc32Resultc = band(crc32Resultb,hex167i);
```

```
  crc32Result  = crc32Resultc;
```

```
  test2        =symget('look' || left(iLookup));
```

```
  crc32Result  =bxor(crc32Result,test2);
```

```
  crc32        =bnot(crc32Result);
```

```
  crc32h       =put(crc32,hex8.);
```

```
  call symput('crc32',crc32h);
```

```
  call symput('myFile',"&myfile");
```

```
  keep crc32h;
```

```
run;
```

```
*--Create a single record containing file name and CRC32 code;
```

```
data mycode;
```

```
  set b end=eof;
```

```
  attrib myfile length=$255;
```

```
  myfile="&myfile";
```

```
  if eof then output;
```

```
  keep myfile crc32h;
```

```
run;
```

```
*--Add this new file name and code to master file;
```

```
data master;
```

```
  set master mycode;
```

```
run;
```

```
%mend getcrc32;
```

5

6

7

```

*-----*;
*--Get a directory listing to file: /basic, /attribute (no) dir, /subdirectories;
*--Leave off the /s parameter if you do NOT want subdirectories included;

x 'dir "D:\SAS_CRC32" > D:\temp9999.txt /b /a-d /s';

filename dirout "D:\temp9999.txt";
*--Read file names from temporary file created through MS-DOS command;
data myFiles;
    infile dirout trunccover;
    attrib GetFile length=$255;
    input  GetFile $char255.;
run;

*-- delete the temporary dataset;
x 'del D:\temp9999.txt'; run;

*--initialize master before calling macros--*;
data master;
    if _n_ gt 999; *--create dataset with no observations;
run;

*--Run the CRC32 macro for each file in the list;
data myFiles;
    set myFiles;
    *if _n_ le 6; * Test on 5 files only;
    *run macro for each observation in myFiles;
    call execute('%getcrc32(||GetFile||)');
run;

*--Print the results;
title "CRC Codes for files in target directory"; run;
proc print data=master; run;

```

8

9

COMMENTARY ON THE SAS CODE

The comments listed below which reference the numbers in boxes above, next to the SAS code, are intended to help anyone trying to understand how this program works get a more detailed explanation of the program logic than is obvious from the SAS code or the program comments.

1. STRUCTURE

The program is setup as two macros. The first, **%CRC_SETUP**, calculates a look-up table used by the second macro. The macro variable **CRC_SETUP_STATUS** is used as a flag, so that when the macro is called multiple times in a program, the table-build macro is only run one time.

2. INPUT AND PUT STRUCTURE

Since SAS has different functions than Visual Basic, I had to use multiple SAS statements when one statement worked in Visual Basic. I was not such which functions to use, so when I was developing this program, I tried several way of coding the data. Variable length was significant, and I got different results when the length of the variables changed. Because of the way signed-integers and unsigned-integers differ, very large numbers caused unexpected problems. What I finally did (SAS code not shown) was write a macro to dump the binary, decimal, and hexadecimal values of each and every variable. That way, I was able to trace every change and transformation until I got the right combination of formats and functions. This was the result of trial, error, and more error until I got it right.

3. LOGICAL FUNCTIONS

The logical functions BAND, BXOR, BNOT, and INT were modeled from the Visual Basic program.

4. CALCULATED TABLE VALUES STORED AS MACRO VARIABLES

The subroutine creates 256 table values, which are stored as macro variables so that the second macro can make use of these calculated values.

5. COMPUTING THE CRC-32 VALUE

Macro **%GETCRC32(myfile)** reads the target file and computes the CRC-32 value for that file. There does not appear to be any processing penalty or advantage to setting **LRECL=1** in the INFILE statement. This setting makes it cleaner to understand that the program reads one byte at a time.

6. REFERENCE VALUES

Hex values used in calculations are defined up front. Since both macros share these values, another approach would have been to define these one-time as global macros. Whether or not this would significantly improve processing time is unknown. Variable **CRC32RESULT** is defined in several statements to make it easier to debug intermediate results.

7. SAVE FILE NAME AND CRC_32 VALUES

The name of the target dataset along with its CRC-32 code is stored in SAS dataset **MASTER**. This technique permits results to be saved for multiple datasets. This feature is quite helpful when running a job at nights which reads thousands of datasets!

8. OLD DOS COMMAND STILL WORKS WELL

This old Microsoft DOS command still works! The command is used to create a text file, which contains a list of all the files to be read. This file can be renamed and saved for archival purposes.

9. RUNNING THE SAS PROGRAM

This simple three-line SAS program is all that is needed to run the CRC-32 macros. The **CALL EXECUTE** statement reads each line from the text file of files to be read, and then calls the **%GETCRC32** macro. The rest is history.

BIG ENDIAN – LITTLE ENDIAN PROBLEM NOT SOLVED

In writing this program, I ran across a problem I had never heard about. There is a problem with writing any SAS program that manipulates bits and bytes, because different computer systems store data differently. The problem is referred to as the "big endian, little endian" problem (a reference to an incident involving eggs, in that classic book, Gulliver's Travel's).

The following is from the SAS help v8.2 on "Endian": The ordering of the bytes for the integer varies depending on the platform (operating environment) on which the integers were produced. The ordering of bytes differs between the "big endian" and the "little endian" platforms. These colloquial terms are used to describe byte ordering for IBM mainframes (big endian) and for Intel-based platforms (little endian). In the SAS System, the following platforms are considered **big endian**: IBM mainframe, **HP-UX**, **AIX**, **Solaris**, and **Macintosh**. The following platforms are considered **little endian**: **VAX/VMS**, **AXP/VMS**, **Digital UNIX**, **Intel ABI**, **OS/2**, and **Windows**.

This program was written and tested on a little endian Windows system. The SAS code needs to be modified and validated to run on a big endian system. The following SAS code will check to see what family of computer the program is running on. Since I only had a Windows system to test this program, I could not check it against a big endian system. If you intend to run this code on a big endian system, check it first!

```
* Determine byte order;
* Just a reminder - does not abort the program;
data _null_;
    endian="Unknown Computer";

    if put(input('1234'x,ib2.),hex4.)='1234'
    then
        do;
            endian='BIG    ';
            put "PROGRAM WILL NOT WORK ON THIS COMPUTER!";
            PUT "THIS IS A big Endian COMPUTER";
        end;
    else
        if put(input('1234'x,ib2.),hex4.)='3412'
        then endian='LITTLE';

        call symput('endian',endian);
run;
%put value of endian is &endian; run;
```

VALIDATION OF THE SAS CODE

Several test files, include E:\SAS_CRC32\CRC_test.txt were used during program development. (For your information, this test file is three bytes long, and contains the letters: **ABC**. You can recreate this file yourself, and then get the same CRC-32 value that I did: **A3830348**.

The best validation of this program was to run it against hundreds of real files, and compare the CRC-32 value as calculated by SAS to the value calculated by any of the other programs listed above. When I first did this, I found that certain very large CRC-32 values were incorrect because of the way negative numbers are stored. Once I corrected this problem, all values have been the same.

APPLICATIONS

Having worked in the Pharmaceutical industry for many years, I am very aware that documentation and validation of my work needs to meet the standards set by the Food and Drug Administration (FDA). Specifically, FDA News Part 11 Compliance Report Vol. 2 No. 23 page 4 states:

"To determine that the hand-signed document is the same as the electronic record, the agency requires that the exact filename of the e-record, its size in bytes, the date and time of its creation and an authentication value, such as a check sum or unique mathematical value, are included on the handwritten copy provided to the FDA."

Using the CRC-32 standard allow validated program code, data, and other files to comply with the requirements for an authentication value that is an auditable criteria to prove that the file being examined is identical to the historical file for which a CRC-32 code is provided.

This capability to quickly validate entire folders of programs and data opens the door to the development of a file inventory and management system. Using the techniques demonstrated in this SAS program, file names can be automatically captured from a folder, and select files (such as all files and only those file ending in .SAS) can be checked against values stored in a SAS database for changes.

CRC-32 gives a much higher level of protection against changes then file protection and read-only libraries. Depending upon file protection schemes and library system still leaves vulnerabilities to intentional and unintentional file corruption, and misidentification of files. Since CRC-32 is not impacted by file names or creation dates, and because the source file and the target file do not have to exist on the same computer, CRC-32 provides a quick, cost-effective, reliable solution to file management.

CONCLUSION

The SAS program is being shared, and placed in the public domain, in order to provide the SAS user community with another tool to increase the power and effectiveness of SAS as a solution to managing those "housekeeping" tasks which are so necessary for maintaining a professional work environment.

I hope that the story of how this program came about can serve as an example for SAS programmers who are faced with seemingly impossible technical challenges, that there can be creative, verifiable solutions which do not require mastery of the subject matter; only a practical understanding of what results are expected.

If anyone does develop a file management application based on this SAS code, I would appreciate it if they would publish the results in a user group paper such as this, and share their success with the rest of the SAS community. Such a publication would be more than enough payback for the time and effort spent in developing and documenting this program.

ACKNOWLEDGMENTS

This SAS program uses logic based on the Visual Basic program found at: <http://www.vbaccelerator.com/home/VB/Code/Libraries/CRC32/article.asp> , copyright © 2003 Steve McMahon steve@vbaccelerator.com. All rights reserved. Last Updated: 2 November 2003. This model program is 20 lines of Visual Basic code to create the lookup table, 11 lines to calculate the CRC value, and 5 lines to use the class. The site, [vbAccelerator.com](http://www.vbAccelerator.com), is devoted to providing free, advanced source code to Visual Basic programmers since June 1998 and now also to the .NET and C# community.

The **WinZip** program is available from WinZip Computing, Mansfield, CT 06268. www.winzip.com.

WinRar is available from RARLAB, Bremen, Germany, at: www.win-rar.com.

RECOMMENDED READING

An excellent description of the CRC-32 algorithm, and the various optimizations you can apply when implementing it (a fascinating and heavily mathematical process) is provided in Donald Knuth's classic programming books "The Art of Programming: , Volume 2: Seminumerical Algorithms, Section 4.

For a quick overview of this area, www.Wikipedia.org is an excellent source.

- For information on Donald Knuth, see: http://en.wikipedia.org/wiki/The_Art_of_Computer_Programming
- Big endian and little endian are discussed in: http://en.wikipedia.org/wiki/Big_endian
- The CRC code is discussed at: <http://en.wikipedia.org/wiki/Crc32>

Additional web resources on the CRC algorithm can be found at:

http://www2.rad.com/networks/1994/err_con/crc_how.htm and
<http://www.relisoft.com/Science/index.htm>

Three of the standards which address CRC coding include:

- [ITU-T V.42] International Telecommunications Union, Error-correcting Procedures for DCEs Using synchronous-to-Synchronous Conversion, ITU-T Recommendation V.42, 1994, Rev. 1.
- [ISO 3309] International Organization for Standardization, "Information Processing Systems--Data Communication High-Level Data Link Control Procedure--Frame Structure", ISO 3309, October 1984, 3rd Edition.
- [IHO S-57] is the electronic data transfer standard prepared by the International Hydrographic Organization (IHO) committee. The geodatabase model allows the user to import Electronic Navigational Charts (ENC) .000 files, which in turn provides the user with a highly accurate vector data set that is ideal as foundation data.

The **ChkSum-W32.exe** program referred to in this paper is a Windows based program that will calculate an authentication value (checksum or unique mathematical value) for a file. It is extensively documented. The program can be downloaded from: <http://www.homestead.com/checksum/>

After writing this paper, and doing some final searching, I stumbled on the following web site:

<http://www.diahome.org/NR/rdonlyres/58348DDD-DBDC-4DC0-8025-49E5F6B38A33/0/dij1961.pdf>

"The Use of Cyclic Redundancy Check (CRC-32) and Adler-32 Checksums for Source Code Verification" by Allan Glaser, Manager, Statistical Programming, Merck & Co., Blue Bell, PA as printed in Drug Information Journal, Vol. 37, pp. 147 – 154, © 2003. His article focuses on using digital signatures to manage source code. Well worth reading for another independent solution to the problems discussed in the article.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

John Fulda
Senior Manager, SAS Programming
Kos Pharmaceuticals, Inc
2100 North Commerce Parkway
Weston, FL 33326
Work Phone: 954-331-3816

jfulda@kospharm.com
johnfulda@hotmail.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.