EBCDIC and ASCII, COBOL and IEEE

A Guide to Data Conversions

By Michael C. Mattias

September, 1999

COBOL DATA CONVERSIONS

One question commonly asked on COBOL-related Internet newsgroups goes something like this:

"We have some COBOL-produced data [on our mainframe]. We need to convert it to use with BASIC/C/C++/COBOL programs on a personal/Unix/Apple computer. Any ideas appreciated. Thanks in advance. "

With programs written in COBOL (**CO**mmon **B**usiness **O**riented **L**anguage) still in widespead use, especially on mainframe computers, but computing and data management increasingly performed at the user workstation level, the ability to share data between these two worlds assumes greater importance each day.

Taking this task as a whole, and given file transfer between computers, using media or communications software, is no longer the "black art" it once was, there are two challenges:

1. COBOL produces numeric data to COBOL specifications; while this is fine if the destination computer is using COBOL, C/C++/BASIC programs and most other software such as spreadsheets and database managers cannot read and write COBOL numeric data when stored in its native form.
2. If the COBOL data were produced on an IBM mainframe or AS/400, the data are stored in a different character set; and the data - or rather, as we'll see, portions of the data - must undergo a character set conversion.

The purpose of this document is to familiarize the reader with these two challenges and to suggest conversion strategies for some common scenarios.

COBOL DATA TYPES

While COBOL programs write and read common text using standard alphabetic and numeric characters, numeric datatypes are another matter. COBOL produces numeric data using its own set of rules; data which for the most part cannot be read "as is" by software written in BASIC and C/C++, nor imported "as is" into commercial spreadsheet and database software.

The rules for COBOL datatypes are covered in much greater detail in the author's text and graphics tutorial, "Understanding COBOL Data Types" (See Appendix).

In summary, COBOL data stored in files are described by a "File Description" ("FD") found in the COBOL program which reads or writes the data record; within the FD, the PICTURE and USAGE clauses define the size, maximum value and bit patterns used for each data item in the record.

COBOL data consist primarily of text data plus four basic numeric datatypes; these are shown in the sample File Description in Figure One.

```
FD THE-FILE-DESCRIPTION

LABEL RECORDS OMITTED.
```

```
01 THE-RECORD.

   05 TEXT-DATA            PIC X(30).

   05 DISPLAY-NUMERIC      PIC S9(04)V99.

   05 SIGNED-SEPARATE      PIC S9(6)       SIGN SEPARATE.

   05 COMP-NUMERIC         PIC S9(06)      USAGE COMP.

   05 COMP-3-NUMERIC       PIC 9(4)V9(2)   COMP-3.
```

FIGURE ONE: Typical COBOL File Description

Display Numeric. Each digit specified in the PICTURE clause occupies one byte of storage; each byte is stored as the character representation of a radix 10 digit (0-9). Signs may be explicit (SIGN IS SEPARATE), in which case a character containing a plus or minus sign is appended or prepended; or implicit, in which case the trailing (default) or leading digit has the sign encoded within it.

Computational. (USAGE IS COMP, COMP-4, COMP-5, COMP-X, or BINARY). The numeric value is expressed as a pure binary integer. The data require as many whole bytes of storage as may be required to store the maximum value based on the PICTURE clause; but if WORD storage is in effect, the data are stored in two, four or eight bytes of storage, whichever is sufficient to hold the maximum radix ten value. Negative values are expressed in twos complement form[1].

COBOL data with USAGE IS COMP is always stored with the least-significant bytes at the highest address (sometimes called "little endian"). COMP-4 data is stored with the most- and least-significant bytes and words reversed (sometimes called "Intel" format, or "big endian"). COMP-5 storage is "whatever the operating system does"; i.e., COMP-5 would be different on Unix boxes running on a Motorola chip and a PC running an Intel chip. COMP-4 and COMP-5 data are only used with word storage.

COMP data storage is highly dependent on the compiler manufacturer; the specifics above apply to Microfocus COBOL compilers and the IBM mainframe COBOL compiler.

COMP-X is primarily used as an alphanumeric data type in which otherwise unprintable constant data (e.g., printer control characters) can be entered by the programmer using hexadecimal notation; and is often subject to implementor (compiler developer) variations. COMP-X , when treated as numeric, is the same as COMP-5, except it is always unsigned, and always stored with the MSB at the highest address.

Binary-Coded-Decimal, or BCD. (USAGE IS PACKED-DECIMAL or COMP-3). One half byte of storage is used for each digit of the radix ten PICTURE clause; one half byte is added to hold a sign half-byte (nibble). When the total number of nibbles would be odd, the data are always right-justified within the next highest number of whole bytes. The slack nibbles are usually set to 'x0', but this is implementor-defined.

BCD is also the "packed" datatype format used on IBM AS/400-series computers.

Floating Point. (USAGE IS COMP-1 or COMP-2) These are IEEE single and double-precision floats. COMP-1 (single) occupies four bytes of storage; COMP-2 (double) occupies 8 bytes of storage. (Some implementor variations may be found).

Except for COMP-1 and COMP-2, all COBOL numeric datatypes listed above are stored with implicit decimal points. COBOL compilers generate code to multiply or divide by ten the requisite number of times based on the PICTURE clause in the FD.

There is one other kind of COBOL data type which may require conversion - the "edited numeric." Edited numeric datatypes are numeric values with non-numeric characters in them - for example, containing an explicit decimal point, or with leading spaces rather than leading zeroes. These can be converted as alphanumeric using PICTURE IS X(total size of all characters, including editing characters).

IEEE DATA TYPES

The Institute of Electrical and Electronics Engineers, Inc. (IEEE) (http://www.ieee.org) is an industry-based voluntary standards association which has defined bit patterns for various standard data types used in computing. The major types are:

Character. Text; also used for display numeric items, except any sign is always separate and explicit. These are not often used for numeric data, and require one byte of storage per character.

Binary Integers. (16, 32 and 64 bit) Pure binary integers; negative values are stored as the twos complement of the absolute value. The 16-bit are called "short", the 32-bit "long" and the 64-bit "quad" integers. IEEE are also defines unsigned versions of binary integer, often called the HALFWORD, WORD and DOUBLEWORD.

When binary integers are stored in "Intel" ("big-endian") format, they are stored with the bytes and words inverted; that is, the most significant byte is at a higher address than the least-significant byte, and the most significant word is at a higher address than the least significant word. (See COBOL COMP-4 and COMP-5 for detail).

Floating Point (single or double precision). These data types consist of 32 (single) or 64 (double) bits and contain a biased binary exponent, a sign bit and a binary mantissa.

OTHER DATA TYPES

Some popular software products (notably Microsoft Visual Basic) also use quasi-proprietary data types called "currency " and "variant"; some language products also offer BCD types. These are not part of the IEEE specifications.

CHARACTER SETS

A character set is the definition of what bit patterns are used to represent various printable characters (e.g., alphabetic letters, character digits and punctuation symbols) and control sequences (e.g., video and printer control codes). There are both single-width and double-width character sets; single-width character sets use seven or eight bits contained in one byte; double-width character sets use sixteen bits (two bytes).

Two popular single-width character sets are ASCII and EBCDIC. Double-width character sets include DBCS, the IBM double-byte-character-set; and Unicode, a character set used in the Windows? operating system.

In this paper we will consider only the single-width ASCII and EBCDIC character sets.

ASCII and EBCDIC Character Sets

The American Standard Code for Information Interchange, or ASCII (usually pronounced as'-key) character set is a single-width character set using seven bits to define 128 different characters. There is also an eight-bit version called "Extended ASCII " which defines the bit patterns for 256 different characters.

The Extended Binary-Coded Decimal Interchange Code, or EBCDIC (usually pronounced ip'-seh-dik) character set uses eight bits to define some 193 different characters. EBCDIC is not universal; that is, there are differences based on the national language.

IBM mainframe and AS/400 computers store data using the EBCDIC character set; Apple and IBM-compatible PCs, as well as Unix systems, store data using the ASCII character set.

All common numeric digits, alphabetic letters, and punctuation symbols are defined both in ASCII and EBCDIC; however, each character set defines unique characters; that is, there is not necessarily an EBCDIC representation of every ASCII character; nor is there an ASCII representation of each EBCDIC character.

Table I shows some common characters used in data conversion and the associated hex and decimal values using both the EBCDIC and ASCII character sets.

TABLE I: Character Sets and Data Representation

Character Item	ASCII-hex	ASCII-decimal	EBCDIC-hex	EBCDIC-decimal
Letter 'A'	x'41'	65	x'C1'	193
Letter 'a'	x'61'	97	x'81'	129
Digits '0' - '9'	x'30' -x'39'	48-57	x'F0' - x'F9'	240-249
Plus Sign '+'	x'2B'	43	x'4E'	78
Minus Sign '-'	x'2D'	45	x'60'	96
Decimal Point '.'	x'2E'	46	x'4B'	75

CONVERTING COBOL DATA

Specific actions are dependent on the source character set and the destination character set and data type. **The one requirement common to all conversions is that the COBOL file Description (FD) for the source data must be available.** The FD will be found in the COBOL source code of any program which reads or writes the data record or in a "copylib" included in that program.

Table II shows three common scenarios for COBOL data conversion:

TABLE II

Conversion Scenarios

Scenario	Source	Destination
One	EBCDIC-COBOL	ASCII-COBOL
Two	EBCDIC-COBOL	ASCII-IEEE
Three	ASCII-COBOL	ASCII-IEEE

Scenario One: You have COBOL data produced on a mainframe or AS/400. You want to get it to an IBM-PC or Unix computer in COBOL format to use with COBOL programs. NOTE: Several COBOL development systems offer packaged solutions to deal with this scenario.

Scenario Two: You have COBOL data produced on a mainframe or AS/400. You want to get it to an IBM-PC or Unix computer to use with BASIC or C/C++ programs,

Scenario Three: You have COBOL data produced on an IBM-PC or Unix computer. You want to use this data on the same system with BASIC or C/C++ programs.

In Scenarios One and Two, a character set conversion is required; in Scenarios Two and Three a datatype conversion is required.

CHARACTER SET CONVERSION

If you have an EBCDIC source and ASCII destination (Scenarios One and Two), you might think you can just convert each character of the input using the "EBCDIC to ASCII" feature of your communications software, or using a standard conversion utility. But if the source data contain binary, BCD or floating-point values, there is a problem. For example, assume the source data has this COBOL File Description:

```
FD THE-FILE-DESCRIPTION

LABEL RECORDS OMITTED

RECORD CONTAINS 4 CHARACTERS.

01 THE-RECORD.

     05 VALUE-1 PICTURE IS S9(4) USAGE COMP.

     05 VALUE-2 PICTURE IS S9(4) USAGE COMP.
```

If the two characters of VALUE-1 equal x'00C1' (193 decimal), an EBCDIC-ASCII converter will convert this to x'0041'. Why? Although the first byte of this two-byte data field, x'00', converts to x'00', the second byte, x'C1', is the EBCDIC value for the letter 'A' and converts to x'41', the value for the letter 'A' using the ASCII character set. The data are now corrupted: the (decimal) value 193 has "magically" changed to decimal value 65.

If we expand the file description a bit:

```
01 THE-RECORD.

     05 VALUE-1 PICTURE IS S9(4) USAGE COMP.

     05 VALUE-2 PICTURE IS S9(4) USAGE COMP.

     05 TEXT-1 PICTURE IS X(5).
```

If TEXT-1 equals x'4040404040' (five spaces using the EBCDIC character set), it needs to be converted to x'2020202020' (five spaces using the ASCII character set); but we still do not want to convert VALUE-1 and VALUE-2 from EBCDIC to ASCII.

We can perform this same analysis with USAGE IS COMP-3 (BCD), COMP-1 or COMP-2 (floating point), COMP-4, COMP-5 or COMP-X (other binary), and (non-separate) signed DISPLAY numeric data for VALUE-1 and VALUE-2 and arrive at the same general rule:

EBCDIC to ASCII conversions must be done on a field-by-field basis if the record contains binary, BCD, floating point or non-separate-signed display numeric data.

Our first reaction is that it would be a lot simpler if all, or none, of the data required EBCDIC-ASCII conversion. This first reaction is absolutely correct. All other things being equal, the best way to convert data is to convert the data on the EBCDIC system, changing all numeric fields to use:

PICTURE IS S9(m)V9(n) USAGE DISPLAY SIGN IS SEPARATE

'm' and 'n' represent the number of digits before and after the (implied) radix 10 decimal point. Given a PICTURE, USAGE and SIGN of S9(7)V99 SIGN SEPARATE , the value 12345.67 will be "001234567+". If the separate sign is specified as LEADING, the result is "+001234567".

When USAGE DISPLAY SIGN SEPARATE data are transferred from the EBCDIC to the ASCII system it can ALL be converted from EBCDIC to ASCII: either during the download, or after arrival on the destination system.

Unfortunately, all other things are not always equal. When it is not possible to expand the numeric data on the source machine to the USAGE and SIGN above, any EBCDIC to ASCII conversion will need to occur on a field-by-field basis. While some sophisticated communications software packages may allow the loading of a COBOL FD as a "download" parameter, many do not; consequently, the user will need to know the exact size, location, PICTURE and USAGE of every field in the record being converted.

DATA TYPE CONVERSIONS

Once the data have been converted to ASCII (following the rules for numeric fields), conversion to IEEE data types is straightforward, if tedious.

COBOL programs, of course, can read the data directly, provided the FD on the destination system matches the FD of the source system. (Scenario One).

However, BASIC/C/C++ programs (Scenarios Two and Three) must convert COBOL-produced numeric data to IEEE data types supported by that language product.

USAGE DISPLAY SIGN SEPARATE or unsigned DISPLAY numeric data can be read using intrinsic functions in BASIC or C/C++. In BASIC, the VAL function is used; in C/C++, scanf can be used; in either case, the result assigned to any datatype.

Signed DISPLAY numeric data, when the sign is not separate, must have the leading or trailing byte in which the sign is encoded converted to a digit and overall sign; then the value may be read as a USAGE DISPLAY type, multipled by minus one if negative, and the value assigned to any data type convenient.

The value of binary and BCD data can be calculated using the bit patterns, word/byte orientation and (for BCD) the sign nibble and assigned to any IEEE data type.

Floating point data may be cast as four-byte or eight-byte entities and directly read, either by explicit casting or the use of unions. Alternately, the user may compute the value of the four or eight byte entity using the supplied mantissa, biased exponent and sign bit. Regardless of conversion choice, rounding differences should be expected when converting floating-point data types.

As all COBOL numeric data uses implied decimal points, data must be multiplied or divided by the power of ten specified by any implicit or scaled decimals in the COBOL PICTURE clause, and by minus one if the data are negative.

Lessons From the School of Hard Knocks.

In Real Life, there are some facts about these data conversions for which you should be prepared:

- COBOL numeric data is often unsigned, even if the PICTURE clause calls for signed data. The IBM mainframe COBOL compiler provides options to allow "unsigned" data to be treated as "positive" at run time; unfortunately, this allows data to be stored in the same fashion.
- COBOL data, especially from older systems, may contain binary zero (x'00') in FILLER fields which are defined with an aphanumeric PICTURE such as PIC X(10). While mainframe systems are not terribly fussy about binary zero, Windows-based PCs often use x'00' as a string terminator character and may give unpredictable results.
- The difference between "word" and "byte" alignment for USAGE COMPUTATIONAL is not discernible from the COBOL file description; it is a function of the hardware, specific compiler and sometimes, compile-time options selected by the developer.

SUMMARY

COBOL-produced data can be made accessible to software written in other languages or running on other computers; however, the user must be aware of both character set and datatype conversion rules. Knowledge of both facets of the conversion is required for success.

APPENDIX

Bibliography and Reference

Microfocus COBOL Language Reference. Issue 14, April 1994. Copyright 1989-1994 Microfocus Ltd. The portions of the manual used refer to the ANSI COBOL standards X3.23-1985 and X3.23a-1989.

VS COBOL II Application Programming Language Reference (release 3.0, 3.1, 3.2). Seventh Edition, December 1990. Copyright 1987, 1990 International Business Machines Corporation.

"Understanding COBOL Data Types", copyright 1997 Michael C. Mattias, available at no charge for non-commercial use at selected sites on the World Wide Web or by contacting the author. This tutorial is available free for non-commercial use from the Flexus Corporation web site http://www.flexus.com/softwaredownload.html as COBDATA.ZIP.

Acknowledgments

Special thanks to Leonard J. Jowers, Birmingham AL; John E. Carter, Smyrna GA; Alistair Maclean, Royston, Herts, UK

Publication space on the World Wide Web courtesy Flexus Corporation, makers of 100% compiler independent tools for the professional COBOL programmer.

About the Author

Michael Mattias is a data processing consultant specializing in data management and ANSI and EDIFACT Electronic Data Interchange. He has written numerous articles for *BASICally Speaking,* a journal for BASIC-language programmers; and published several packages of instructional and utility software for COBOL and BASIC-language programmers. Based in Racine WI (between Milwaukee WI and Chicago IL), he may be contacted via e-mail at michael.mattias@gte.net

Copyright and Redistribution License