The COBOL programming language, in 3(or more) pages.

By Marc Hage

Introduction

This is an introduction paper on the COBOL programming language. It is far from a complete discussion on the subject.
I will start out by giving a brief history of COBOL.
Second I will show the layout of a COBOL program. This because it is something very characteristic to COBOL. It is almost a characteristic, or 'feature', of the language.
Third I will give some coding examples to show to what extent the language differs from the 'C-like' languages that are taught to 1st year students. This 'showing of differences' is not exhaustive due to the '3 page indication' guideline for this report.
Fourth I will tell you what I think of COBOL and what it is useful for.

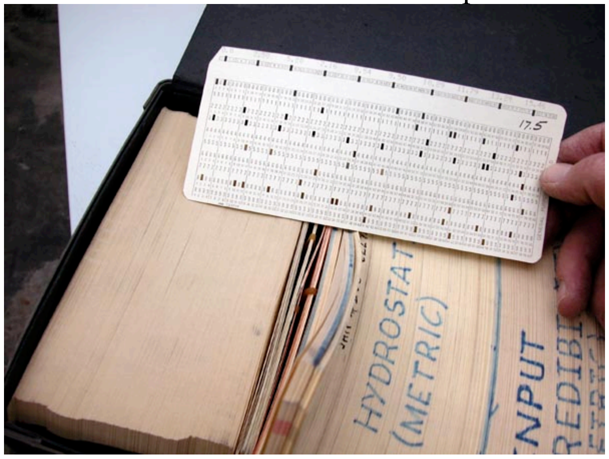
History of COBOL

In 1952, Grace Murray Hopper began a 'journey' that would eventually lead to the language we know as COBOL. She began by developing a series of programming languages that became more and more like natural language. The language used phrases to express the operations of business data processing. FLOWMATIC was the result of this evolutionary journey.
Through the 1950's, other computing leaders were also working through the challenge of creating a practical business language. IBM had produced a language named COMMERCIAL TRANSLATOR.
In 1959, an industry-wide team was assembled to formulate a common business programming language. The Conference on Data System Languages (CODASYL) led by Joe Wegstein of National Bureau of Standards (now National Institute of Standards and Technology) developed a new language, and created the first standardized business computer programming language.
COBOL (Common Business Oriented Language) was developed under supervision of the U.S. Department of Defense in cooperation with computer manufactures, users and universities. The initial specifications for COBOL were presented in a report of the executive committee of CODASYL committee in April of 1960. It was designed to be a business problem oriented, machine independent and capable of continuous change and development.
Since 1960,COBOL has undergone considerable updates and improvements. It has emerged as the leading data processing language in the business world. The standard language specification has three levels low, middle and high so that standard COBOL can be implemented on computers of varying sizes.

Despite the attempts at standardization, variations in COBOL implementations continue to exist. Most deviations or "extensions" are intended to take advantage of hardware or environmental features which were not defined in the standard definition.
(source: http://www.legacyj.com/cobol/cobol_history.html)

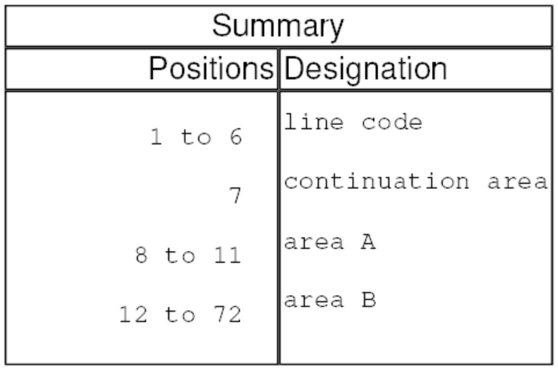
COBOL Layout Basics

COBOL program writing started in the days of punch-cards so it adheres to strict code-formatting guidelines. Those punch cards had rows and character positions (spaces). The cards had a maximum width of 80 spaces.



(image source: http://www.staff.ncl.ac.uk/roger.broughton/iomedia/pc.htm)

Essentially, the first 6 spaces are ignored by the compiler and are usually used by the programmer for line numbers. These numbers are not the same as those in BASIC where the line number is used as part of the logic (e.g. GOTO 280, sending the logic to line 280). The 7th space is called the continuation area and can contain only '*', '/' or '–'. These characters have special meaning which I will not discuss any further.
The 8th to 11th space is called area A. Besides an area A there is also an area B. These areas prescribe certain key-words and uses of those areas that a COBOL programmer must adhere to. I will, again, not discuss them any further.



Summary	
Positions	Designation
1 to 6	line code
7	continuation area
8 to 11	area A
12 to 72	area B

(image source: http://cobol.404i.com/)

Characteristics of COBOL

I will start out by giving HELLO WORLD (sorry Natalia):

```
000010 IDENTIFICATION DIVISION.
000020 PROGRAM-ID.   HELLO-WORLD-PROG.
000030 AUTHOR.       MARC HAGE.
000040*The standard Hello world program
000050
000060 ENVIRONMENT DIVISION.
000070
000080 DATA DIVISION.
000090 WORKING-STORAGE SECTION.
000100 01 TEXT-OUT    PIC X(12) VALUE 'Hello World!'.
000110
000120 PROCEDURE DIVISION.
000130 MAIN-PARAGRAPH.
000140        DISPLAY TEXT-OUT
000150        STOP RUN.
```

Four Divisions

COBOL program code is divided into four basic divisions:

- The IDENTIFICATION division is used to identify the program and the programmer.
- The ENVIRONMENT division describes the hardware being used in the program.
- The DATA division describes files, data and constants used through out the program.
- The PROCEDURE division describes what the program does via commands given.

I will not elaborate on the IDENTIFICATION and ENVIRONMENT divisions and skip right ahead to the DATA division.
The DATA division is where memory space in the computer is allocated data and identifiers being used throughout the program. The division contains two important sub divisions, called SECTIONs: the FILE SECTION and the WORKING-STORAGE section.
The FILE SECTION is used to define data that will be read from or written to during execution. Data consists of ‘records’ that can, in turn, consist of ‘levels’.
The file section for a customer-data record may look like this:

```
000400 DATA DIVISION.
000410 FILE SECTION.
000420
000430 FD INPUT-FILE.
000440 01 CUSTOMER-DATA.
000450       03 NAME       PIC X(12).
000460       03 ADDRESS.
000470             05 HOUSE-NUMBER   PIC 99.
000480             05 STREET         PIC X(19).
000490             05 CITY           PIC X(13).
```

```
000500       03 CUST-NUMBER PIC 9(6).
```

The WORKING-STORAGE SECTION contains data that is used as temporary memory during program execution. Effectively, this is where, for example, an identifier is defined that will hold the result of a calculation. E.g.:

```
000500 DATA DIVISION.
000510 WORKING-STORAGE SECTION.
000520
000530 01 RECORD-COUNTER    PIC 9(5).
```

The procedure division is where the logic of the program actually found. Here is where the various commands are written.
COBOL is a modular language, in that a program is usually broken up into units described as paragraphs.

```
000900 PROCEDURE DIVISION.
  000910 CONTROL-PARAGRAPH.
  000920       PERFORM READ-DATA-FILE
  000930       PERFORM CALULATE-PRICES
  000940       PERFORM PRINT-PRICE-REPORT
  000950       STOP RUN.
```

The PERFORM statement is used to 'call' other paragraphs (or procedure) to do each task. These paragraphs would appear in the same coding and are part of the same program.

Command and Logic in COBOL

In this paragraph I will list some of the more common command and logic that a programmer can use in COBOL. For a complete discussion I recommend reading "cursus COBOL 85" by Chris Verkoulen or "A SIMPLIFIED GUIDE TO STRUCTURED COBOL PROGRAMMING" by Daniel McCracken. (now that we have our disclaimer, let continue…)

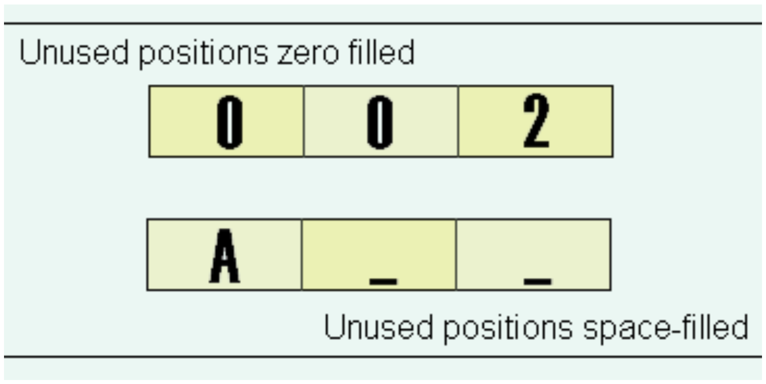
ACCEPT and DISPLAY

These are used to accept data, e.g. key-board input, and display data, e.g. console or printer.

MOVE

This simply tells the computer to place a certain item of data to a specified identifier. Care must be taken when moving data. In other languages, such as C, when to assign a value to XYZ (defined at the beginning of the program as an integer for example), then that's it. In COBOL, you have to be sure that the value you are moving to XYZ item is not bigger than the defined size.

Conversely, moving data that is smaller than the definition has certain effects. If 2 was moved to item XYZ above, then the number 2 is written to the right-most position and the leading positions are zero-filled (see figure below). Likewise, moving "A" to ABC above, the letter "A" would be written to the left-most position with the trialing positions being space-filled.



(image source: http://cobol.404i.com/)

PERFORM

The PERFORM verb is one of the most important in COBOL (alongside MOVE). It can be used to call a paragraph (or subroutine if you will). The PERFORM verb can also form the bases of a repetitive loop (or sub-routine) until a certain condition has been met. For Example:

```
     :
000290 PROCEDURE DIVISION.
000300 XYZ-PARAGRAPH.
000310       PERFORM COUNT-PROCESS UNTIL W-COUNTER > 10
000320       STOP RUN.
001000
002000 COUNT-PROCESS.
002010    COMPUTE W-COUNTER = W-COUNTER + 1
002020    DISPLAY 'Number of loops is ' W-COUNTER.
```

In the above code, COUNT-PROCESS is executed until the value of W-COUNT has reached 11.
The format for an Out-of-Line PERFORM is:

```
PERFORM [paragraph-name] UNTIL [condition]
```

An In-Line PERFORM, rather than execute a paragraph (aka procedure), allows for the repeated execution of a series of commands. The format for an In-Line PERFORM is:

```
PERFORM UNTIL
{action}...
END-PERFORM
```

This type of PEFORM tests the condition before the following statements are allowed to proceed. Using WITH TEST can be used to define when the test is done:

```
:
 000290 PROCEDURE DIVISION.
 000300 XYZ-PARAGRAPH.
 000305       MOVE ZERO TO W-COUNTER
 000310       PERFORM WITH TEST AFTER UNTIL W-COUNTER > 10
 000320          COMPUTE W-COUNTER = W-COUNTER + 1
 000330          DISPLAY 'This is loop number: ' W-COUNTER
 000340       END-PERFORM
 000350       DISPLAY 'Counter is now equal to: ' W-COUNTER
 000360       STOP RUN.
```

If you wanted to loop a desired number of times you could use TIMES.
The format is:

```
PERFORM {identifier or literal} TIMES
{action}...
END-PERFORM
```

IF THEN ELSE
This basic logic structure is known to us all. One thing that can be said is that it requires THEN and END-IF to be set.
The format is:

```
IF {identifier-1} {condition} {identifier-2 or literal} ...
THEN {statements}
[ELSE {statements}]
END-IF
```

EVALUATE
If there are a large number of conditional alternatives it would be convenient to use some switch-like statements known from C. COBOL offers programmers the EVALUATE verb. The format is:

```
           { identifier-1 }           { identifier-2 }
           { literal-1    }           { literal-2    }
EVALUATE   { expression-1 }   ALSO    { expression-2 }
           { TRUE         }           { TRUE         }
           { FALSE        }           { FALSE        }

     WHEN  {statement-1}...
     WHEN OTHER  {statement-2}...

END-EVALUATE
```

To give an instructive example:

```
      EVALUATE W-NUM
              WHEN 1 MOVE 10 TO NEW-DATA
                 DISPLAY 'NEW-DATA IS 10'
              WHEN 2 MOVE 20 TO NEW-DATA
                 DISPLAY 'NEW-DATA IS 20'
              WHEN 3 MOVE 30 TO NEW-DATA
                 DISPLAY 'NEW-DATA IS 30'
              WHEN OTHER MOVE ZERO TO NEW-DATA
                 DISPLAY 'NEW-DATA IS 0'
        END-EVALUATE
```

Arithmetic

COBOL (of course) has features for doing arithmetic. The interesting thing to note is that the syntax, like so many other COBOL statements, consists mainly of English (US) words. I will sum up the formats for the 4 basic arithmetic operations:

```
ADD {identifier-1 or literal}... TO {identifier-2 or literal}...
      [GIVING {identifier-3}]
      [NOT] [ON SIZE ERROR {statements}]
 [END-ADD]

SUBTRACT {identifier-1 or literal}... FROM {identifier-2 or literal}...
      [GIVING {identifier-3}]
      [NOT] [ON SIZE ERROR {statements}]
 [END-SUBTRACT]

MULTIPLY {identifier-1 or literal}... BY {identifier-2 or literal}...
      [GIVING {identifier-3}][ROUNDED]
      [NOT] [ON SIZE ERROR {statements}]
 [END-MULTIPLY]

DIVIDE {identifier-1 or literal} BY {identifier-2 or literal}...
      GIVING {identifier-3} [ROUNDED] [REMAINDER {identifier-4}]
      [NOT] [ON SIZE ERROR {statements}]
 [END-DIVIDE]
```

In the format specifications above the parts from GIVING till ON SIZE ERROR are optional.

COBOL has numerous other operations that are very important to it. For example STRING and file related operations. But it would go beyond the scope of this text to discus those here. A more comprehensive introduction to COBOL, which also severed as guideline and source for this text, can be found at http://cobol.404i.com.

My opinion of COBOL

For me COBOL programming proved to be fairly hard since I only have experience with C like languages and a little bit with Basic. The reason why it is hard for me is the fact that it is 'all words and little symbols'. That may sound confusing but for example a for() loop, know from 1st and 2nd year courses in Java, looks much more direct and simple than PERFORM THING-TO-DO UNTIL W-COUNTER > SOME-VALUE, in my humble student opinion. You have to keep in mind that the syntax of those languages may have seemed to be as difficult at first as COBOL syntax. Consequently and to be short: I found it a little bit hard.
The fact of the mater remains that COBOL is still very much alive today. Large companies, e.g. insurances companies or banks use COBOL programs to large extend: ABN-AMRO, Achmea, De Belastingdienst, etc.
Nowadays the discussion about wetter or not COBOL will be replaced by one of the dozens of other languages that seem to be more adequate (according to their advocates) has lessened in strength. More and more effort is being put in linking up COBOL code with other programming environments or languages. For example: COBOL and .NET or COBOL and Java.
What a COBOL administrator working at a large Dutch insurance company told me was that they let the COBOL programs do the (batch) processing of transactions and provide a user interface via Java programs. This is sometimes even taken to further extend: provide an interface (implemented as a web interface for example) to a COBOL-program interface (implemented in Java for example) which in turn controls the actual COBOL written software. This scheme is also used by ABN-AMRO (for online banking).
An event that stressed the difference between theory and practice in programming languages was (of course) the 'Year 2K problem'. I assume you know all about it but the thing I had mist was that it where COBOL programs, relics from the 1960's to 1980's that where causing the 'problem'.
Nowadays the interest in COBOL has not caught on with the new generation of programmer in the Netherlands. More and more COBOL programming is being outsourced to countries such as India and China because the demand for maintenance and construction of new COBOL software has all but diminished.
So you could conclude to "get with the program and start COBOL-krassen (Dutch)" but there are those that give other insights, such as the following taken from the Wikipedia page dealing with COBOL:

"Criticism

Critics have argued that COBOL's syntax serves mainly to increase the size of programs, at the expense of developing the thinking process needed for software development. In his letter to an editor in 1975 titled "How do we tell truths that might hurt?", Computer scientist Edsger Dijkstra remarked that "The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offence"(from Selected Writings on Computing: A Personal Perspective)."
(source: http://en.wikipedia.org/wiki/COBOL#Criticism)