SPECS for COBOL FINAL  
(see pgs. 587–601 of Course PDF dated 9/13/20)

**PROGRAMMERS / ANALYSTS: (Nancy Slocum, Pomela Dominguez)**  
Code COBOL  
Document code  
Analyze maintenance requests  
Support apps (including bug fixes)

**TESTER: (Julius Broschk)**  
Create test data  
Create error messages for invalid data  
Create testing plan & lead discussion with team on how testing will go  
Create JCL to debug & run the app  
Run/debug & analyze test results  
Back-up COBOL maintenance & support apps, including bug fixes

**TECH LEAD: (JoJo Zawawi)**  
Project planning, milestones, schedule & lead meeting/discussions, etc.  
Set up source management on GIT  
Interpret specs & lead team in app design

Create list of all resources necessary for creating/debugging app? programs, JCL, copybooks, data files, etc.  
Write high-level specs for programmer/analysts such as I/O design, etc.

Create app development standards:

Programming & code reviews  
Overall development approach & project plan  
Documentation standards (keep it simple)

Back-up COBOL programming  
Buy coffee & pizza for the team

Jon recommends that we:

1. Re-watch the Report Generation video:  
<https://community.ibm.com/community/user/ibmz-and-linuxone/viewdocument/simplified-cobol-report-writing>

2. Flowchart out the:

(a) JCL Run Job

(b) Application (in terms of modules & paragraphs)

(c) Don’t forget the “Link-Edit Requirements and the Types of Subprogram Calls” (see pg. 451 of the course materials).

3. Understand every para, statement & variable, and how each relates to the specs.  
Document the programs as you code:  
— Program-level documentation — what the program does;  
— Paragraph-level documentation — basically, pseudo-code; and  
— Specific inline comments, as needed, to clarify complex logic.

4. Develop / document / test from the top-down:

(a) Obtain existing (working) boilerplate code from programs we’ve already coded:  
ID & ENVIRONMENT DIVISION  
DATA DIVISION FDs  
PROCEDURE DIVISION  
— OPEN / READ / WRITE / CLOSE  
— Housekeeping  
— Table Loading  
— Data Validation / Error Handling Routines

(b) Document & create stub paragraphs

(c) Compile / Link / COBUCLD stub paragraphs

(d) Add one scenario at a time — Compile / Link / COBUCLD that & only that scenario

(e) Test with data that exercises to the specs & beyond  
— Force S0C7s, Divide by Zero, Arithmetic Overflow, etc.

5. Jon reminds us that he’s more interested in our COBOL skills than in the business results / values.

6. ALL WE TURN IN ARE THE PROGRAM CODE & THE REPORT.  
We are supposed to keep every resource we create & stash it in a GIT repository for safekeeping & resume building. I’m (JoJo) supposed to set up the GIT, and I’ll do that next.

**GENERAL INFO**

This pgm reads a sorted QSAM data file “PARTSUPP”.

Each record in PARTSUPPcontains info re a single part, its suppliers, suppliers’ addresses, and purchase orders.  
Supplier Addresses are in a table inside the record.  
Purchase Order records are in a table inside the record.

PARTSUPP record layout — see page 5. Can be copied & pasted if you go via a plain-text program first. But it’s in a copybook.

The pgm edits the PARTSUPP input file, using a series of subroutines (one for each 05-level group in the PARTSUPP copybook).

Errors found in the input record are written to an error file.

Good records are written to 4 separate QSAM files (passed to downstream for processing):

1. PARTS — **parts** information

2. SUPPLIERS — info on **suppliers** for a given **part**

3. ADDRESS — details 3 different **addresses** for each **supplier**

4. PURCHASE — contains data from 3 **purchase orders** cut to a **supplier** for a given **part**

A CONTROL-BREAK report is written out for each of the good records that pass the edit criteria.

DISPLAY statements can be used for one-off purposes (i.e., if the pgm should ABEND due to bad numeric data in the input file).

**SPEC**

1. Sort the PARTSUPP file on PARTNO.

2. Execute Parts/Suppliers pgm (reading PARTSUPP) & split data into:

(a) 4 Output Data Files:

PARTS (“PARTEDIT”) parts information

SUPPLIERS (“SUPPEDIT”) info on suppliers for a given part

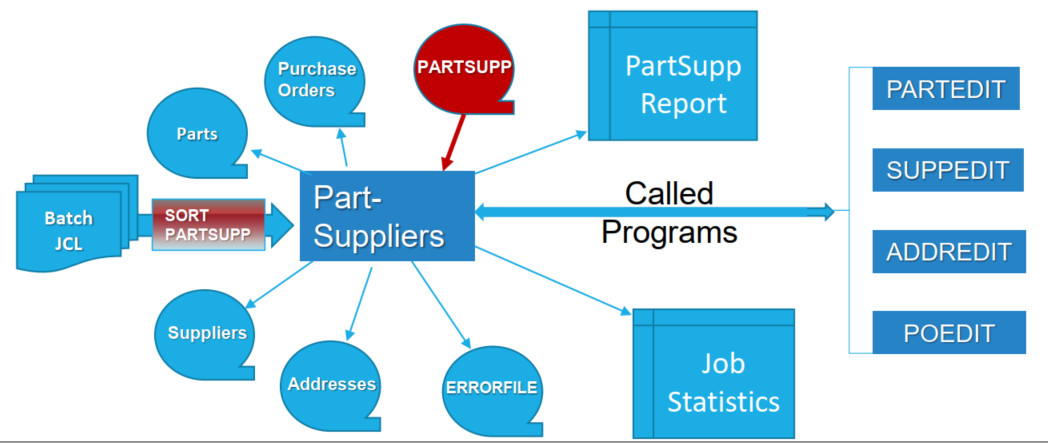
ADDRESS (“ADDREDIT”) details 3 different addresses for each supplier

PURCHASE (“POEDIT”) contains data from 3 purchase orders cut to a supplier for a given part (POEDIT)

(b) Error File

(c) Report File

(d) SYSOUT (for DISPLAY of pgm stats & potential problems such as ABENDS, etc.)



**3. EDITS for Parts / Supplier / Addresses / Purchase Order data:**

**05 PARTS**

PART-NUMBER Required field — can’t be blank. MUST be unique.

PART-NAME Required field — can’t be blank.

VEHICLE-MAKE Required field — can’t be blank. Must be one of the listed 88-level fields.

VEHICLE-MODEL Required field — can’t be blank.

VEHICLE-YEAR Required field — can’t be blank. Must be between 1990 and 2019.

WEEKS-LEAD-TIME Must be numeric. Must be between 1 and 4.

**05 SUPPLIERS**

SUPPLIER-CODE Required field — can’t be blank. Does NOT have to be unique.

SUPPLIER-TYPE Required field — can’t be blank. Must be one of the listed 88-level fields.If SUPPLIER-TYPE is SUBCONTRACTOR, SUPPLIER-RATING must be Highest Quality.

SUPPLIER-NAME Required field — can’t be blank. Does NOT have to be unique.

SUPPLIER-PERF Required field — can’t be blank.Jon says “A PIC 9 cannot be zeros. But what is common is tocheck for “IS NUMERIC” and if true then check for IS > ZERO.See slides ~210.”

SUPPLIER-RATING Must be one of the listed 88-level fields.

SUPPLIER-STATUS Must be one of the listed 88-level fields.

SUPPLIER-ACT-DATE Optional, but if a value exists in the field, ensure date is valid.

**05 SUPP-ADDRESS** (3 occurrences but I think the “occurrences” has been nixed, since it was ADDRESS-1, -2, -3 in case someone had an apartment # or something)

ADDRESS-1 Required field — can’t be blank.

CITY Required field — can’t be blank.

ADDR-STATE Required field — can’t be blank.ZIP-CODE&ADDR-STATE must match in the STATEZIP file.

ZIP-CODE Required field — can’t be blank. ZIP-CODE&ADDR-STATE must match in the STATEZIP file.  
NOTE: In the input file, ZIP-CODE is 10 bytes; in the output file, it’s PIC X(05). Jon says the first 5 bytes are all that matter for this application.

ADDRESS-TYPE Must be one of the listed 88-level fields.

**05 PURCHASE-ORDER** (3 occurrences)

PO-NUMBER Required field — can’t be blank.

BUYER-CODE Required field — can’t be blank.

ORDER-DATE Required field — can’t be blank.Must be a valid date.

QUANTITY Required field — can’t be blank. Must be between 0 and 999,999. IF QUANTITY is > 0, UNIT-PRICE must be > 0.

UNIT-PRICE IF QUANTITY is > 0, UNIT-PRICE must be > 0.

DELIVERY-DATE Optional, but if there is data, must be a valid date & date must be later than ORDER-DATE.

01 PART-SUPP-ADDR-PO.

05 PARTS.

**\* PART-NUMBER CODES HAVE TO BE UNIQUE**

10 PART-NUMBER PIC X(23) VALUE SPACES.

10 PART-NAME PIC X(14) VALUE SPACES.

10 SPEC-NUMBER PIC X(07) VALUE SPACES.

10 GOVT-COMML-CODE PIC X(01) VALUE SPACES.

10 BLUEPRINT-NUMBER PIC X(10) VALUE SPACES.

10 UNIT-OF-MEASURE PIC X(03) VALUE SPACES.

10 WEEKS-LEAD-TIME PIC 9(03) VALUE 0.

10 VEHICLE-MAKE PIC X(03) VALUE SPACES.

88 CHRYSLER VALUE 'CHR'.

88 FORD VALUE 'FOR'.

88 GM VALUE 'GM'.

88 VOLKSWAGEN VALUE 'VW'.

88 TOYOTA VALUE 'TOY'.

88 JAGUAR VALUE 'JAG'.

88 PEUGEOT VALUE 'PEU'.

88 BMW VALUE 'BMW'.

10 VEHICLE-MODEL PIC X(10) VALUE SPACES.

10 VEHICLE-YEAR PIC X(04) VALUE '0000'.

10 FILLER PIC X(14) VALUE SPACES.

05 SUPPLIERS.

**\* SUPPLIER-CODE & SUPPLIER-NAME DO NOT HAVE TO BE UNIQUE**

10 SUPPLIER-CODE PIC X(10) VALUE SPACES.

10 SUPPLIER-TYPE PIC X(01) VALUE SPACES.

88 SUBCONTRACTOR VALUE 'S'.

88 DISTRIBUTOR VALUE 'D'.

88 MANUFACTURER VALUE 'M'.

88 IMPORTER VALUE 'I'.

10 SUPPLIER-NAME PIC X(15) VALUE SPACES.

10 SUPLIER-PERF PIC 9(03) VALUE 0.

10 SUPPLIER-RATING PIC X(01) VALUE SPACES.

88 HIGHEST-QUALITYVALUE '3'.

88 AVERAGE-QUALITY VALUE '2'.

88 LOWEST-QUALITY VALUE '1'.

10 SUPPLIER-STATUS PIC X(01) VALUE SPACES.

88 GOVT-COMM VALUE '1'.

88 GOVT-ONLY VALUE '2'.

88 COMMERCIAL-ONLY VALUE '3'.

**\* SUPPLIER-ACT-DATE used to be PIC 9(08); Jon agrees should be PIC X.**

10 SUPPLIER-ACT-DATE PIC X(08) VALUE 0.

05 SUPP-ADDRESS OCCURS 3 TIMES INDEXED BY ADDR-IDX.

10 ADDRESS-TYPE PIC X(01) VALUE SPACES.

88 ORDER-ADDRESS VALUE '1'.

88 SCHED-ADDRESS VALUE '2'.

88 REMIT-ADDRESS VALUE '3'.

10 ADDRESS-1 PIC X(15) VALUE SPACES.

**\* IF ADDRESS-2 AND/OR ADDRESS-3 ARE EMPTY, DON’T PRINT**

10 ADDRESS-2 PIC X(15) VALUE SPACES.

10 ADDRESS-3 PIC X(15) VALUE SPACES.

10 CITY PIC X(15) VALUE SPACES.

10 ADDR-STATE PIC X(02) VALUE SPACES.

**\* In input file, ZIP-CODE is 10 bytes; in output file, it’s PIC X(05).**

**\* Jon says the first 5 bytes are all that matter for this application.**

10 ZIP-CODE PIC 9(10) VALUE 0.

05 PURCHASE-ORDER OCCURS 3 TIMES INDEXED BY PO-IDX.

10 PO-NUMBER PIC X(06) VALUE SPACES.

10 BUYER-CODE PIC X(03) VALUE SPACES.

10 QUANTITY PIC S9(7) VALUE +0.

10 UNIT-PRICE PIC S9(7)V99 VALUE +0.

10 ORDER-DATE PIC 9(08) VALUE 0.

10 DELIVERY-DATE PIC 9(08) VALUE 0.

ADDRESS STATE / ZIP CODE EDITS

Create a one-dimensional COBOL table out of the below STATEZIP file (also available in DDS0001.LEARN.STATE.ADDRESS.ZIP).

Address records must have a valid State Abbreviation / Zip Code combination.Example:  
The ZIP code for AK must be between 99501 and 99950.  
The ZIP code for DC must be between 20001–20039, 20042–20599 & 20799–20799.

NOTE: There are several states with multiple valid ZIP code entries (thus, multiple rows in the file). Regarding this, Jon asks: would it be better to handle with a Table SEARCH, or a PERFORM VARYING operation?

Alaska AK 99501 99950

Alabama AL 35004 36925

Arkansas AR 71601 72959

Arkansas AR 75502 75502

Arizona AZ 85001 86556

California CA 90001 96162

Colorado CO 80001 81658

Connecticut CT 06001 06389

Connecticut CT 06401 06928

Dist of Colum DC 20001 20039

Dist of Colum DC 20042 20599

Dist of Colum DC 20799 20799

Delaware DE 19701 19980

Florida FL 32004 34997

Georgia GA 30001 31999

Georga GA 39901 39901

Hawaii HI 96701 96898

Iowa IA 50001 52809

Iowa IA 68119 68120

Idaho ID 83201 83876

Illinoi IL 60001 62999

Indiana IN 46001 47997

Kansas KS 66002 67954

Kentucky KY 40003 42788

Louisiana LA 70001 71232

Louisiana LA 71234 71497

Massachusetts MA 01001 02791

Massachusetts MA 05501 05544

Maryland MD 20331 20331

Maryland MD 20335 20797

Maryland MD 20812 21930

Maine ME 03901 03992

Michigan MI 48001 49971

Minnesota MN 55001 56763

kc96 DataMO MO 63001 65899

Mississippi MS 38601 39776

Mississippi MS 71233 71233

Montana MT 59001 59937

North Carolina NC 27006 28909

North Dakota ND 58001 58856

Nebraska NE 68001 68118

Nebraska NE 68122 69367

New Hampshire NH 03031 03897

New Jersey NJ 07001 08989

New Mexico NM 87001 88441

Nevada NV 88901 89883

New York NY 06390 06390

New York NY 10001 14975

Ohio OH 43001 45999

Oklahoma OK 73001 73199

Oklahoma OK 73401 74966

Oregon OR 97001 97920

Pennsylvania PA 15001 19640

Rhode Island RI 02801 02940

South Carolina SC 29001 29948

South Dakota SD 57001 57799

Tennessee TN 37010 38589

Texas TX 7330 73301

Texas TX 75001 75501

Texas TX 7550 79999

Texas TX 88510 88589

Utah UT 84001 84784

Virginia VA 20040 20041

Virginia VA 20040 20167

Virginia VA 20042 20042

Virginia VA 22001 24658

Vermont VT 50001 05495

Vermont VT 05601 05907

Washington WA 98001 99403

Wisconsin WI 53001 54990

West Virginia WV 24701 26886

Wyoming WY 82001 83128

EDITED RECORDS — DISPOSITION

If the record passes all tests in the subprogram, then:

1. The subroutine should send back a return code of zero.

2. The PARTSUPP (calling program) should:

(a) MOVE the record data to the associated output file:  
 PARTS (“PARTEDIT”)  
 SUPPLIERS (“SUPPEDIT”)  
 ADDRESSES (“ADDREDIT”)  
 PURCHASE ORDERS (“POEDIT”)

(b) Write the output file;

(c) Write the existing PARTSUPP file.

If record has *any* errors, then:

1. The subroutine sends back a Return Code of 8.

2. The PARTSUPP driver program writes the record to the Error File & does NOT write the record to the “good record” output file(s).

3. “When calling subroutines, stop calling subsequent subroutines after 4 errors.”  
If there are < 4 errors:

(a) The bad record must include error messages in the output file.

(b) This means that the output file record layout needs additional space for error message(s) that accompany the invalid data fields in the record.

4. “When calling subroutines, stop calling subsequent subroutines after 4 errors.”  
If there are >3 errors:

(a) The bad record is written to the output file with a single message that the record is completely invalid.

CALLS TO 'CEEDAYS'

DATE HANDLING ROUTINES

Use your own shop’s date-handling utility, or IBM’s Language Environment (“LE”) program “CEEDAYS”. Example (this is Jon’s DDS0001.LEARN.COBOL CDAT2 program):

ID DIVISION.

PROGRAM-ID. CDAT2.

DATA DIVISION.

WORKING-STORAGE SECTION.

01 W-INPUT-DATE-INT PIC 9(9) COMP.

01 W-PICSTR-IN.

10 W-PICSTR-LTH-IN PIC S9(4) COMP VALUE 8.

10 W-PICSTR-STR-IN PIC X(8) value 'YYYYMMDD'.

01 W-DATE-IN-CEE.

10 W-DATE-IN-LTH-CEE PIC S9(4) COMP VALUE 8.

10 W-DATE-IN-STR-CEE PIC X(8).

01 FC.

10 FC-SEV PIC S9(4) COMP.

10 FC-MSG PIC S9(4) COMP.

10 FC-CTW PIC X.

10 FC-FAC PIC X(3).

10 FC-ISI PIC S9(8) COMP.

PROCEDURE DIVISION.

A000-MAINLINE.

MOVE 19990228 TO W-DATE-IN-STR-CEE

CALL 'CEEDAYS' USING W-DATE-IN-CEE

W-PICSTR-IN, W-INPUT-DATE-INT, FC

IF FC-SEV NOT = ZERO

DISPLAY 'BAD DATE'

ELSE

DISPLAY 'GOOD DATE'

END-IF

MOVE 19990229 TO W-DATE-IN-STR-CEE

CALL 'CEEDAYS' USING W-DATE-IN-CEE

W-PICSTR-IN, W-INPUT-DATE-INT, FC

IF FC-SEV NOT = ZERO

DISPLAY 'BAD DATE'

ELSE

DISPLAY 'GOOD DATE'

END-IF.

GOBACK.

OUTPUT FILE RECORD LAYOUTS  
(from Jon’s DDS0001.COBOL.COPYLIB)

Write the output files using these record layouts.

Data in the “good records” from the PARTSUPP input file will be moved into these records field-for-field.

Jon says that we may notice that the size and/or types of these output file record & field layouts are not the same as the PARTSUPP**input** file. Our choice to either modify the record layout(s) or program around the discrepancy.

PURCHRDS.cpy:

01 PURCHASE-ORDER.

05 PO-NUMBER PIC X(06) VALUE SPACES.

05 BUYER-CODE PIC X(03) VALUE SPACES.

05 QUANTITY PIC S9(8) COMP VALUE ZEROS.

05 UNIT-PRICE PIC S9(7)V99 COMP-3 VALUE ZEROS.

05 ORDER-DATE PIC X(08) VALUE SPACES.

05 DELIVERY-DATE PIC X(08) VALUE SPACES.

ADRESSES.cpy:

01 SUPP-ADDRESS.

05 ADDRESS-TYPE PIC X(01) VALUE SPACES.

88 ORDER-ADDRESS VALUE '1'.

88 SCHED-ADDRESS VALUE '2'.

88 REMIT-ADDRESS VALUE '3'.

05 ADDRESS-1 PIC X(15) VALUE SPACES.

05 ADDRESS-2 PIC X(15) VALUE SPACES.

05 ADDRESS-3 PIC X(15) VALUE SPACES.

05 CITY PIC X(15) VALUE SPACES.

05 ADDR-STATE PIC X(02) VALUE SPACES.

**\* In input file, ZIP-CODE is 10 bytes; in output file, it’s PIC X(05).**

**\* Jon says the first 5 bytes are all that matter for this application.**

05 ZIP-CODE PIC X(05) VALUE SPACES.

SUPLIERS.cpy:  
[IMPORTANT NOTE: Ensure that the copybook itself has the revised definition for SUPPLIER-CODE. The below has the correct definition.]

01 SUPPLIERS.

05 SUPPLIER-CODE PIC X(10) VALUE SPACES.

05 SUPPLIER-TYPE PIC X(01) VALUE SPACES.

88 SUBCONTRACTOR VALUE 'S'.

88 DISTRIBUTOR VALUE 'D'.

88 MANUFACTURER VALUE 'M'.

88 IMPORTER VALUE 'I'.

05 SUPPLIER-NAME PIC X(15) VALUE SPACES.

**\* Jon says “A PIC 9 cannot be zeros. But what is common is to**

**\* check for “IS NUMERIC” and if true then check for IS > ZERO.**

**\* See slides ~210.”**

05 SUPPLIER-PERF PIC 9(03) COMP VALUE ZEROS.

05 SUPPLIER-RATING PIC X(01) VALUE SPACES.

88 HIGHEST-QUALITY VALUE '3'.

88 AVERAGE-QUALITY VALUE '2'.

88 LOWEST-QUALITY VALUE '1'.

05 SUPPLIER-STATUS PIC X(01) VALUE SPACES.

88 GOVT-COMM VALUE '1'.

88 GOVT-ONLY VALUE '2'.

88 COMMERCIAL-ONLY VALUE '3'.

05 SUPPLIER-ACT-DATE PIC X(08) VALUE SPACES.

PARTS.cpy:  
[IMPORTANT NOTE: Ensure that the copybook itself has the revised definitions for VEHICLE-MODEL and BLUEPRINT-NUMBER. The below has the correct definitions.]

01 PARTS.

05 PART-NUMBER PIC X(23) VALUE SPACES.

05 PART-NAME PIC X(14) VALUE SPACES.

05 SPEC-NUMBER PIC X(07) VALUE SPACES.

05 GOVT-COMML-CODE PIC X(01) VALUE SPACES.

05 BLUEPRINT-NUMBER PIC X(10) VALUE SPACES.

05 UNIT-OF-MEASURE PIC X(03) VALUE SPACES.

05 WEEKS-LEAD-TIME PIC S9(04) COMP VALUE ZEROS.

05 VEHICLE-MAKE PIC X(03) VALUE SPACES.

88 CHRYSLER VALUE 'CHR'.

88 FORD VALUE 'FOR'.

88 GM VALUE 'GM '.

88 VOLKSWAGON VALUE 'VW '.

88 TOYOTA VALUE 'TOY'.

88 JAGUAR VALUE 'JAG'.

88 PEUGEOT VALUE 'PEU'.

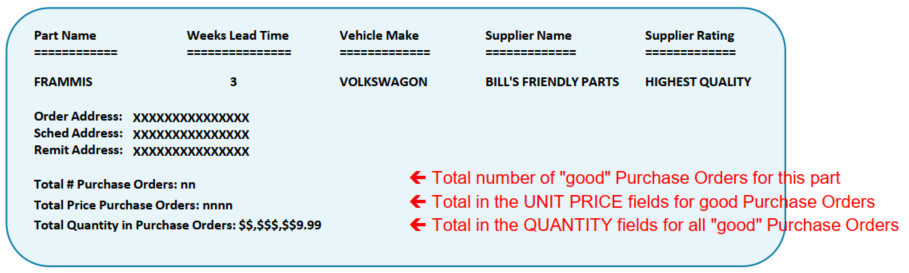
88 BMW VALUE 'BMW'.

05 VEHICLE-MODEL PIC X(10) VALUE SPACES.

05 VEHICLE-YEAR PIC X(04) VALUE '0000'.

REPORT

The file will be sorted on PARTNO.  
The report should be produced as follows, breaking on PARTNO:



Jon says that each separate ADDRESS from the table gets written out as a single record (thus, no “OCCURS” clause).

Jon says that one question to be addressed is “where does the data for this report come from?” Since the Part/Supplier program will be creating the “good record” output file, the options are:

1. Store the records in an internal COBOL table, in the Part/Supplier driver program; or

2. Write & run another program that creates the above report later in the same batch job stream. We would need another output data set created in the JCL for this.

[end]