Introduction au développement sous COBOL/AS400

par Faisel Chabli (Mon site web) (Blog)

Date de publication : 24 février 2011

Dernière mise à jour :

Cet article a pour objectif de vous introduire dans le développement de programmes COBOL sous AS400 (Technologie d'IBM). La difficulté résidant dans l'apprentissage d'un tel langage est dans la disposition d'un serveur AS400, chose s'avérant inabordable sauf si l'on est des employés d'une structure en disposant ou lorsqu'on est étudiant dans un établissement offrant des cours sur cette technologie. Néanmoins, il y a des sites proposant leurs serveurs avec certaines restrictions pour y développer directement ses applications.

I - Introduction....3
II - Configuration de l'environnement de travail....3
2.1 - Présentation....3
2.2 - Architecture de l'environnement de travail....3
2.2.1 - Description de l'environnement utilisé....3
2.2.2 - Création des différents objets prérequis....3
III - Architecture d'un programme COBOL sous AS/400....3
3.1 - La mise en page....3
3.2 - Structure d'un programme....4
3.3 - Les éléments du langage....4
3.3.1 - Les Mots réservés ou Mots-clés....4
3.3.2 - Les Mots-utilisateur....5
3.3.3 - Les Constantes....5
3.3.4 - Les Opérateurs....5
IV - Quelques notions importantes....5
4.1 - La déclaration de variables....5
4.2 - Les instructions de base....6
4.2.1 - DISPLAY (Afficher)....6
4.2.2 - ACCEPT (Saisir)....6
4.2.3 - MOVE (instruction d'affectation)....6
4.3 - Les instructions de calcul....7
4.3.1 - ADDITION....7
4.3.2 - SOUSTRACTION....7
4.3.3 - MULTIPLICATION....7
4.3.4 - DIVISION....8
4.3.5 - COMPUTE....8
4.3.6 - MODULO....8
4.4 - Les instructions de fin....9
4.4.1 - L'instruction de fin de programme : STOP RUN....9
4.4.2 - L'en-tête de fin de programme : END PROGRAM nom-programme....9
4.4.3 - L'instruction de sortie de sous-programme : EXIT PROGRAM....9
4.4.4 - Les clauses de fin d'instruction : END-instruction[]....9
4.5 - Les variables structurées....9
4.6 - Les tableaux....10
4.6.1 - La déclaration en COBOL....10
4.6.2 - L'accès à un élément du tableau....11
4.6.3 - L'initialisation d'un tableau....11
4.6.3.1 - Initialisation statique à une valeur identique pour toutes les cases....11
4.6.3.2 - Initialisation statique avec des valeurs différentes : clause REDEFINES....12
4.6.3.3 - Initialisation dynamique....12
4.7 - Les instructions de manipulation des chaînes....12
V - Exemple de programme simple....13
5.1 - Création du programme....13
5.2 - Codage du programme....13
5.3 - Compilation et exécution du programme....13
VI - Exemple de programme plus avancé....13
6.1 - Création du programme....13
6.2 - Codage du programme....13
6.3 - Compilation et exécution du programme....13
VII - Conclusion....13
VIII - Remerciements....13

I - Introduction

Cet article met le point sur les éléments de base dans le développement COBOl sous AS400. Certains exemples sont fournis et vous permetteront d'appréhender le concept général du langage.

II - Configuration de l'environnement de travail

2.1 - Présentation

wikipédia : COBOL est un langage de programmation de troisième génération créé en 1959. Son nom est l'acronyme de **CO**mmon **B**usiness **O**riented **L**anguage qui révèle sa vocation originelle : être un langage commun pour la programmation d'applications de gestion.

2.2 - Architecture de l'environnement de travail

2.2.1 - Description de l'environnement utilisé

Les exemples qui seront abordés ultérieurement ont été créés et testés sur un serveur AS400 (V5R4).

2.2.2 - Création des différents objets prérequis

On aura à créer uen bibliothèque de travail et deux fichiers sources : l'un pour les programmes COBOL et l'autre pour les DDS (Fichiers physqiue et logique).

Les commandes à utiliser :

- CRTLIB : Pour créer une bibliothèque. Exemple : **CRTLIB LIB(TESTCOBOL) TYPE(*TEST) TEXT('Ma bibliothèque de test')**
- CRTSRCPF : Pour crér un 1er fichier source pour les programmes COBOL. Exemple : **CRTSRCPF FILE(TESTCOBOL/QCBLILESRC) TEXT('Fichier source de mes PGMs COBOL')**
- CRTSRCPF : Pour crér un 2ème fichier source pour les DDS (PF et LF). Exemple : **CRTSRCPF FILE(TESTCOBOL/QDDSSRC) TEXT('Fichier source de mes DDS PF et LF')**

III - Architecture d'un programme COBOL sous AS/400

3.1 - La mise en page

Contrairement à la plupart des autres langages, développer en COBOL nécessite un respect rigoureux de la mise en page de vos programmes.

La colonne 7 peut contenir :

- une ***** : Pour une ligne de commentaire.
- un **tiret** : Pour marquer la coupure d'un mot réservé (instruction, etc.), d'une variable, d'un littéral

Le reste du programme ne peut être saisi qu'entre les colonnes 8 et 72. On peut distinguer les deux marges suivantes :

- La marge **A** (col. 8) : Pour le début des entêtes de division, de section, de fin de programme, les noms de paragraphes, les noms de paragraphes, les indicateurs de niveau tels FD ..., les nombres de niveau 77 et 01.
- La marge **B** (col. 12) : Contient le reste du programme.

3.2 - Structure d'un programme

Tout programme COBOL comporte quatre divisions. Dans la norme COBOL-85, seule la première division étant obligatoire.

- **IDENTIFICATION DIVISION.** : Contient le nom du programme (obigatoire), le nom de l'auteur du programme, date de création du programme, ...
- **ENVIRONMENT DIVISION.** : Contient des informations sur l'environnement (matériel et logiciel) dans lequel le programme s'exécute.
- **DATA DIVISION.** : Contient la description des données qui sont traitées par le programme (variables, fichiers, paramètres, ...).
- **PROCEDURE DIVISION.** : Contient la description des traitements à effectuer par le programme.

Exemple de programme affichant **Bonjour tout le monde !** :

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. TESTPGM01.
      * Programme TESTPGM01 Version 1.00 :
       AUTHOR. Faisel Chabli.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-AS400.
       OBJECT-COMPUTER. IBM-AS400.
       DATA DIVISION.
       PROCEDURE DIVISION.
       DEBUT-PROGRAMME.
      *****************
      *
            DISPLAY "Bonjour tout le monde !".

       FIN-PROGRAMME.
      ***************
            EXIT PROGRAM.
```

Les différents instructions de ce petit programme seront expliquées ultérieurement dans cet article.

3.3 - Les éléments du langage

Les mots sont formés à partir des caractères alphanumériques sans dépasser 30 caractères.

Ces mots sont classés en 4 catégories comme suit :

- Les **Mots réservés** ou **Mots-clés**
- Les **Mots-utilisateur**
- Les **Constantes**
- Les **Opérateurs**

3.3.1 - Les Mots réservés ou Mots-clés

Ces mots ont une signification précise pour le compilateur (ex. **IDENTIFICATION**, **STOP**, **DISPLAY**, **ACCEPT** ...)

3.3.2 - Les Mots-utilisateur

Ces mots sont créés par l'utilisateur et servent à identifier des objets du programme; on distingue :

- Les **Noms de données** : Ce sont les variables utilisées dans le programme (dans la *PRCOEDURE DIVISION*) et déclarées dans le *WORKING-STORAGE*.
- Les **Noms-condition** : : Précisent les valeurs que peuvent prendre une donnée
- Les **noms de paragraphe** et **Noms de section** en *PROCEDURE DIVISION* : Servent à identifier une séquence d'instructions.

3.3.3 - Les Constantes

On distingue :

- Les **Constantes figuratives** : Certaines valeurs spécifiques ont des noms spéciaux et qui peuvent être attribuées aux variables (exemples : *ZERO, ZEROS, SPACE, SPACES, HIGH-VALUE, LOW-VALUE* ...
- Les **Constantes littérales** : Composées des *Numériques* (Chiffres de 0 à 9, signe + ou -, point décimal) et *Non-numériques* (Tous les caractères écrits entre " ")

3.3.4 - Les Opérateurs

On distingue :

- Les **opérateurs arithmétiques** : + - * / **
- Les **opérateurs relationnels** : >, >=, *EQUAL TO* , *GREATER THAN* ...
- Les **opérateurs logiques** : *AND, OR, NOT*

IV - Quelques notions importantes

4.1 - La déclaration de variables

Définition de la variable en *DATA DIVISION*, *WORKING-STORAGE SECTION* ; 3 possibilités selon qu'on a affaire à une donnée élémentaire (*niveau 77 ou 01*), une donnée structurée (*niveau 01*), un nom-condition (*niveau 88*).

Exemple :

```
    77 FIRSTNAME PIC XXXXXXXXXX.
    77 LASTNAME PIC X(10).
01 AGE  PIC 99 .
77 VARIABLE  PIC A(20)  VALUE "A TEST".
77 QTE    PIC 9(5) VALUE ZERO  USAGE BINARY.
77 PI  PIC 9V9(4) VALUE 3.1416 USAGE COMP.
77 JUSTESPACE PIC X(80)  VALUE SPACES.
77  TRAIT  PIC X(80) VALUE ALL "-".
```

Les données alphabétiques (PIC A) et alphanumériques (PIC X) sont normalement alignées à gauche, alors que les données numériques (PIC 9) sont alignées à droite.

4.2 - Les instructions de base

4.2.1 - DISPLAY (Afficher)

Cette instruction permet l'affichage de littéraux et/ou du contenu de nom-données. 2 formats sont disponibles : le 2ème format permet le positionnement à l'écran mais il ne permet l'affichage que d'une donnée à la fois.

FORMAT 1 : DISPLAY donnée1 donnée2 ... [WITH NO ADVANCING]

FORMAT 2 : non implémenté sur tous les compilateurs.

Exemple :

- DISPLAY "Hello " LASTNAME
- DISPLAY "Entrez votre nom : " NO ADVANCING
- DISPLAY "Azerty" AT 0514 WITH HIGHLIGHT
- DISPLAY "Menu du jour" AT LINE ligne COL colonne WITH BLANK SCREEN

4.2.2 - ACCEPT (Saisir)

Permet d'introduire des données provenant généralement du clavier dans une variable utilisateur.

FORMAT 1 : **ACCEPT MAVARIABLE**

FORMAT 2 : Permet le transfert des date et heure-système dans une variable utilisateur :

ACCEPT nom-donnee FROM

- **DATE** : variable d'image 9(6) : date système au format AAMMJJ
- **DAY** : variable d'image 9(5) : date système au format AAJJJ (JJJ = n° jour dans l'année)
- **DAY-OF-WEEK** : variable d'image 9 : n° du jour dans la semaine (1=lundi, ..., 7=dimanche)
- **TIME** : varibale d'image 9(8) : HHMMSSCC (Heures, minutes, secondes, centièmes)

4.2.3 - MOVE (instruction d'affectation)

La clause **VALUE** déjà vue permet d'affecter une valeur initiale (initialisation) à une variable lors de la déclaration en *WORKING-STORAGE SECTION*; mais pour affecter une valeur à une donnée en cours de programme il faut utiliser l'instruction **MOVE**.

Exemple :

```
MOVE MAVARIABLE1 TO MAVARIABLE2 [MAVARIABLE3 ...]
```

*Pour que l'instruction **MOVE** fonctionne correctement, il faut que l'élément émetteur et l'élément récepteur soient de types compatibles ; attention aux troncatures quand les images respectives ne sont pas de même longueur ; voici quelques exemples d'affectation.*

4.3 - Les instructions de calcul

⚠ *Dans ces instructions de calcul (à part le **MODULO**), les données doivent toujours être des données numériques, et non **pas des données d'édition** (voir plus loin), sauf après GIVING.*

4.3.1 - ADDITION

On peut procéder des manières suivantes :

a. ADD MAVAR1 MAVAR2 ... TO MAVAR-N

Exemple :

```
ADD X  Y  TO  Z
```

Comme résultat : Z = Z + X +Y

b. ADD MAVAR1 MAVAR2 ... GIVING MAVAR-N

Exemple :

```
ADD X Y GIVING Z
```

Comme résultat : Z = X +Y

4.3.2 - SOUSTRACTION

On peut procéder des manières suivantes :

a. SUBTRACT MAVAR1 MAVAR2 ... FROM MAVAR-N

Exemple :

```
SUBTRACT X Y 10 FROM Z
```

b. SUBTRACT MAVAR1 FROM MAVAR2 GIVING MAVAR-N

Exemple :

```
SUBTRACT X FROM Z GIVING Y
```

Comme résultat : Y = Z - X

4.3.3 - MULTIPLICATION

On peut procéder des manières suivantes :

a. MULTIPLY MAVAR1 BY MAVAR-N

Exemple :

```
MULTIPLY MAVAR1 BY MAVAR2
```

b. MULTIPLY MAVAR1 BY MAVAR2 GIVING MAVAR3

Exemple :

```
MULTIPLY X BY Y GIVING Z
```

Comme résultat : Z = X * Y

4.3.4 - DIVISION

On peut procéder des manières suivantes :

a. DIVIDE MAVAR1 BY MAVAR2

Exemple :

```
DIVIDE X BY Y
```

Comme résultat : Y = X / Y

b. DIVIDE VAR1 BY VAR2 GIVING VAR3

Exemple :

```
DIVIDE X BY Y GIVING Z
```

Comme résultat : Z = X / Y

c. DIVIDE VAR1 INTO VAR2

Exemple :

```
DIVIDE X INTO Y
```

Comme résultat : Y = Y / X

*Les 4 instructions de calcul peuvent être terminées par la clause **ROUNDED** (arrondi).*

4.3.5 - COMPUTE

le COMPUTE est utilisé comme suit : COMPUTE VAR1 [ROUNDED] = expression mahématique

Exemple :

```
COMPUTE VAR1 = (VAR2 * VAR3)/VAR4
```

4.3.6 - MODULO

L'instruction **MOD** peut être obtenue directement en COBOL grâce à l'instruction *DIVIDE* suivi de la clause *REMAINDER* (*reste*).

Exemple :

```
VAR1 VAR2 MOD VAR3
```

ça revient à faire :

```
DIVIDE VAR2 BY VAR3 GIVING VARX REMAINDER VAR1
```

La variable **VARX** doit être déclarée même s'elle ne sert à rien dans ce cas.

4.4 - Les instructions de fin

4.4.1 - L'instruction de fin de programme : STOP RUN

Cette instruction obligatoire provoque l'arrêt de l'exécution du programme et rend le contrôle au système d'exploitation; il n'est pas interdit d'utiliser plusieurs instructions **STOP RUN** dans un même programme, mais cette pratique est **fortement déconseillée**.

4.4.2 - L'en-tête de fin de programme : END PROGRAM nom-programme

Cette en-tête (ce n'est pas une instruction) facultative sert à délimiter la fin du programme identifié dans le paragraphe *PROGRAM-ID. nom-prog*. Cet en-tête sert principalement à délimiter des sous-programmes imbriqués.

4.4.3 - L'instruction de sortie de sous-programme : EXIT PROGRAM

Cette instruction indique la fin logique d'un sous-programme ou procédure appelée; elle rend le contrôle au programme appelant.

4.4.4 - Les clauses de fin d'instruction : END-instruction[]

La plupart des instructions du COBOL 85 sont munies d'une clause facultative de fin d'instruction.

Exemple :

```
READ fichier   AT END  MOVE 1 TO fin-fich
DISPLAY "Fichier vide"
END-READ
```

Dans les anciennes versions COBOL, les clauses de fin d'instruction n'existaient pas; il était de règle de terminer chaque instruction par un point. Avec la version 85, il est possible d'écrire toute *PROCEDURE DIVISION* sans utiliser le point (sauf cas obligatoires : STOP RUN., marques de §, ...). Nous utiliserons une syntaxe intermédiaire, commune, consistant à terminer les instructions par un point, sauf si elles constituent un bloc d'instructions incluses entre un *nom-instruction* et *END-nom-instruction*.

4.5 - Les variables structurées

En algorythmique, on peut déclarer une variable strcuturée comme suit :

```
Var
 VEHICULE :  struct
   NUM_IMMAT : struct
      NUM1 : entier
      NUM2 : chaîne(3)
```

```
      DEPT : entier
      fin-struct
   MARQUE : chaine(30)
   MODELE : chaine(30)
   DATE_ACHAT : struct
      JOUR : chaine(2)
      MOIS : chaine(2)
      AN : chaine(2)
      fin-struct
   fin-struct
```

En COBOL, on peut traduire cette variable comme suit :

```
01 VEHICULE.
 02 NUM-IMMAT.
  03 NUM1  PIC 9(4).
  03 NUM2 PIC X(3).
  03 DEPT PIC 9(3).
 02 MARQUE  PIC X(30).
 02 MODELE PIC X(30).
 02 DATE-ACHAT.
  03 JOUR  PIC XX.
  03 FILLER  PIC X VALUE "/".
  03 MOIS  PIC XX.
  03 FILLER  PIC X VALUE "/".
  03 AN  PIC XX.
```

Les nombres-niveaux doivent être compris entre 01 et 49 ; l'incrément d'un sous-niveau peut être supérieur à 1. Exemple :

```
01 TRUC.
    05 MACHIN.
       10 BIDULE. etc.
```

Le nom de donnée **FILLER** permet de définir, dans une donnée structurée, une donnée élémentaire à laquelle il est prévu de ne pas avoir accès; évite d'avoir à imaginer un nom de variable à laquelle on n'aura pas à faire référence dans le programme (*intéressant pour l'édition*).

La structure principale est toujours considérée comme étant de type alphanumérique, même si les variables élémentaires qui la constituent sont de type numérique !

4.6 - Les tableaux

4.6.1 - La déclaration en COBOL

Exemple1 : En Algorythmique :

```
TPRIX : tab[1:12] de réel
```

En COBOL :

```
01 T-PRIX.
     02 PRIX OCCURS 12 PIC 9(8)V99.
```

Exemple2 : En Algorythmique :

```
TREPRES : tab[1:30] de struct
                                NOM : chaine[20]
```

```
                                    SAL : réel
                                    fin-struct
```

En COBOL :

```
01 T-REPRES.
     02 NOM OCCURS 30 PIC X(20).
     02 SAL  OCCURS 30 PIC 9(6)V99.
```

Exemple3 : En Algorythmique :

```
TCA : tab[1:30] [1:12] de entier

tableau à 2 dim.: CA mensuel par n° représentant
```

En COBOL :

```
01 T-CA.
     02 T-REPRES OCCURS 30.
          03 CA OCCURS 12 PIC 9(6).
```

La clause OCCURS ne peut pas figurer au niveau 01. Le 1er élément a toujours l'indice 1 (contrairement au langage C)

4.6.2 - L'accès à un élément du tableau

On accède à un élément via son indice dans le tableau.

Exemple1 : Tableau à une dimension :

```
PRIX (3) ,   NOM (14) ou NOM OF T-REP (14),   SAL (8)
```

Exemple2 : Tableau à plusieurs dimensions :

```
CA (28, 1) = C.A. du 28ème représentant en janvier
```

4.6.3 - L'initialisation d'un tableau

L'initialisation est possible de 3 manières différentes :

4.6.3.1 - Initialisation statique à une valeur identique pour toutes les cases

Exemple :

```
01 T.
 02 T-REPRES OCCURS 30.
  03 NOM PIC X(20) VALUE SPACES.
  03 SAL PIC 9(6)V99  VALUE0.
```

4.6.3.2 - Initialisation statique avec des valeurs différentes : clause REDEFINES

Exemple :

```
01 JOURS-MOIS VALUE "312831303130313130313031"
01 REDEFINESJOURS-MOIS.
  02 NB-JOURS OCCURS12 PIC 99.
```

Résultat : NB-JOURS (I) vaudra 31, NB-JOURS (2) vaudra28, ... NB-JOURS (12) vaudra 31.

4.6.3.3 - Initialisation dynamique

En utilisant desi nstructions de saisie ou d'affectation dans la *PROCEDURE-DIVISION*. Exemple :

```
PERFORM VARYING I FROM 1 BY 1 UNTIL I>30
  DISPLAY"Entrez le nom du représentantn°"I":"NO ADVANCING
  ACCEPT NOM (I)
  DISPLAY"Entrez son salaire de base :"NO ADVANCING
  ACCEPT SAL (I)
END-PERFORM.
```

Exemple avec tableau à 2 dimensions (Tableau des CA par représentation et par mois) :

```
PERFORM VARYING I FROM 1 BY 1 UNTIL I>30
  DISPLAY"Représentant n°"I
  PERFORMVARYING J FROM 1 BY 1 UNTIL J>12
    DISPLAY"CA du mois n°"J":"NO ADVANCING
    ACCEPTCA (I, J)
  END-PERFORM
END-PERFORM.
```

4.7 - Les instructions de manipulation des chaînes

La puissance de définition des données en COBOL(tailles prédéfinies des PIC combinées aux données structurées) fait que nous aurons rarement à utiliser ces instructions. Parmi les plus utiles :

INSPECT ch TALLYING nb FOR ALL car.

Exemple : si *ch* contient "ABRACADABRA" et *car* contient "A", alors *nb* recevra 5.

INSPECT ch REPLACING ALL car1 BY car2.

Exemple : si *ch* contient "ELECTRICITE", *car1* contient "E" et *car2* contient "I" alors *var* sera remplacé par "ILICTRICITI".

STRING ch1 ch2 ... INTO ch3 : concaténation de chaînes.

Exemple : **STRING ch1 "-" ch2 INTO ch3**. si *ch1* contient "Azerty" et *ch2* contient "Uiop", alors *ch3* recevra "Azerty-Uiop".

UNSTRING ch1 DELIMITED BY [ALL] ch2 INTO ch3 ch4 ....

Exemple : **UNSTRING "Ces 3 mots" DELIMITED BY ALL SPACE INTO ch1 ch2 ch3**. *ch1* contiendra "Ces", *ch2* contiendra "3"et *ch3* contiendra "mots".

V - Exemple de programme simple

5.1 - Création du programme

5.2 - Codage du programme

5.3 - Compilation et exécution du programme

VI - Exemple de programme plus avancé

6.1 - Création du programme

6.2 - Codage du programme

6.3 - Compilation et exécution du programme

VII - Conclusion

VIII - Remerciements

Cette petite introduction au développement COBOL sous AS400 permet de découvrir la puissance et la faiblesse de ce langage ancien et qui tient toujours la route grâce aux grands projets tournant sur une telle technologie.

Cet article sera assujeti à d'éventuelles mises à jour dans le temps.