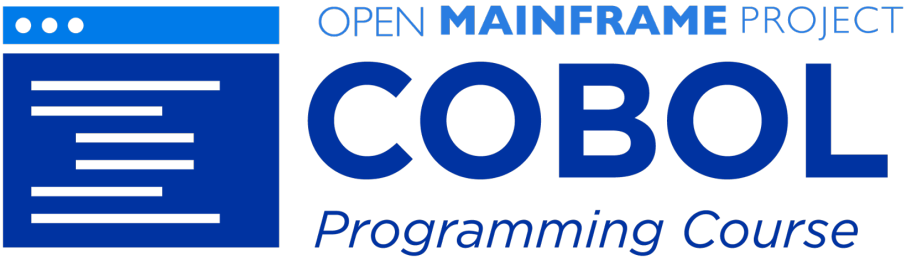


Cours de Programmation COBOL 1

Démarrage

Version 2.3.0

Copyright

Preface

Abstract

Un langage de programmation informatique a été conçu spécifiquement pour les affaires le COBOL (Common Business-Orineted Language). Aujourd'hui COBOL reste toujours aussi pertinent, manipulant 3 000 milliards de dollars dans le commerce chaque jour.

Cette publication s'adresse aux débutants qui cherchent à coomprendre les bases de la programmation COBOL. Il décrit comment travailler avec COBOL en utilisant des outils modernes, notamment Visual Studio Code avec les extensions d'édition Zowe et Z Open. Il décrit comment écrire, tester, exécuter et déboguer des programmes COBOL

Auteurs

Michael Bauer is a development leader for the Open Mainframe value stream at Broadcom and is a squad lead for the Zowe open source initiative. Zowe, a popular framework of modern interfaces for z/OS, opens the mainframe to DevOps tools and practices. Mike leads the Command Line Interface (CLI) squad, which created and recently spun-off the successful Zowe Explorer extension for Visual Studio Code. A frequent speaker and blogger, Mike runs interactive workshops around the world for those interested in incorporating mainframe in their enterprise DevOps initiatives.

Ahmed Eid is a computer engineering student from Egypt. He was a mentee for the Open Mainframe Project 2021 Summer Mentorship under the COBOL Programming Course, helping to improve the content of the course.

Zeibura Kathau is a technical writer for the Mainframe DevOps value stream at Broadcom. He works on the open-source projects Che4z and Code4z, which are IDE extension packages for mainframe developers. He has 8 years of experience in the Information Technology field.

Makenzie Manna is an IBM Redbooks Project Leader in the United States. She has 3 years of experience in the Computer Science Software Development field. She holds a Master's degree in Computer Science Software Development from Marist College. Her areas of expertise include mathematics, IBM Z and cloud computing.

Paul Newton is a Consulting IT Specialist in the United States. He has 40 years of experience in the Information Technology field. He holds a degree in Information Systems from the University of Arizona. His areas of expertise include IBM Z, z/OS, and LinuxONE. He has written extensively on implementation of z/OS based technology.

Jonathan Sayles is a technical educator at IBM, where he conducts presentations, seminars and training courses, as well as producing educational materials.

His more than 40 years in the IT education and computer industries encompass work within both academic and corporate development organizations. He has also been engaged as a software developer/designer/consultant, educator, and author, with a focus on relational database, IDE, and object technologies. In addition to authoring/publishing 16 books, Jon has written and published more than 150 articles in technical journals, and served as technical editor for several IT magazines. He is also co-author of IBM Redbook publications Transitioning: Informix 4GL to Enterprise Generation Language (EGL), SG24-6673 and z/OS Traditional Application Maintenance and Support, SG24-7868.

Hartanto Ario Widjaya is a computer science student from Singapore Management University. He was a mentee for the Open Mainframe Project 2021 Summer Mentorship under the COBOL Programming Course, helping to improve the content of the course with various additions and assisting new learners to incorporate COBOL as a part of their tech toolkit.

William Yates is a Software engineer working for IBM UK. For the majority of his career he has working on the CICS TS product mainly as a software tester and now as Test Architect. He has delivered technical content for many Redbooks, video courses and at conferences around the world. He is also one of the leaders of the Galasa project, building an open source integration test framework for hybrid cloud applications available at https://galasa.dev.

Remerciements

Special thanks to the following people for participating in the residency to shape the content in this publication.

- Dr. Tak Auyeung, Professor, American River College
- Jeffrey Bisti, Z Ecosystem Architect, IBM
- Ilicena Elliott, IT Specialist II, Employment Development Department
- Martin Keen, Technical Content Services, IBM
- Sudharsana Srinivasan, z Influencer Ecosystem Program Coordinator, IBM
- Suzy Wong, Information Technology Specialist, DMV

- 

Left-to-right: Ilicena, Suzy, Makenzie, Martin, Paul, and Tak

Contents

Partie 1 - Pour commencer **12**

1 Pourquoi COBOL ? **12**
1.1 Qu'est-ce que le COBOL ? 12
1.2 Comment le COBOL est-il utilisé aujourd'hui ? 12
1.3 Pourquoi-je devrais-je m'intéresser au COBOL ? 13

2 VSCode avec Zowe Explorer **15**
2.1 Introduction à Zowe Explorer 15
2.2 Utilisation de Zowe Explorer 15
2.3 Profils dans Zowe Explorer 16
2.3.1 Identifiants sécurisés 16
2.3.2 Création d'un nouveau profil 17
2.3.3 Modification des profils 20
2.3.4 Suppression des profils 23
2.4 Sommaire 26

3 VSCode avec Z Open Editor **27**
3.1 Introduction à l'éditeur IBM Z Open Editor 27
3.1.1 Qu'est-ce qu'IBM Z Open Editor ? 27
3.1.2 Le role du Language Server Protocol 28
3.1.3 Installation d'IBM Z Open Editor pour VS Code 28
3.1.4 Extension des variables 29
3.1.5 Mise en évidence de la syntaxe 30
3.2 Navigation dans le code 31
3.2.1 Vue générale (outline) 31
3.2.2 Vue Breadcrumb (fil d'Ariane) 32
3.2.3 Aller à la déclaration / référence 33
3.3 Complétion de code 34
3.3.1 Complétion des mots réservés COBOL 34
3.3.2 Complétion des API CICS, MQ, DB2 35
3.4 Refactorisation du Code 35
3.4.1 Renommer des variables 35
3.4.2 Gestion des erreurs 36
3.5 Sommaire 37

4 VS Code avec le package d'extension Open-Source Code4z **38**
4.1 Qu'est-ce que Code4z ? 38
4.2 Extensions de fichiers connues 39
4.3 Mise en évidence et coloration de la syntaxe 39
4.4 Vérification de la syntaxe et de la sémantique 39
4.5 Navigation du code 39
4.5.1 Aller à la définition 39
4.5.2 Trouver toutes les références 40

4.6 Prise en charge du cahier 40
4.7 Saisie automatique 41
4.8 Sommaire 41

5 Zowe CLI et plug-ins Zowe CLI 42
5.1 Qu'est-ce que Zowe CLI ? 42
5.2 Utilisation interactive de la CLI Zowe 43
5.2.1 Installation de l'interface de ligne de commande Zowe . . 43
5.2.2 Aide interactive 44
5.2.3 Profils Zowe 45
5.2.4 Interaction avec les ensembles de données z/OS 46
5.2.5 Interaction avec les travaux z/OS 47
5.3 Automatisation des tâches à l'aide de Zowe CLI 47
5.3.1 Soumission automatisée des tâches 47
5.3.2 Utilisation d'autres langages de programmation et intégration continue 50
5.3.3 Exemples supplémentaires 52
5.4 Le monde des outils open source modernes 52
5.5 Sommaire 52

6 Installation de VSCode et extensions 53
6.1 Installer les prérequis 53
6.1.1 Installer node.js 53
6.1.2 Installer le SDK Java 54
6.2 Installer VSCode 55
6.3 Installer les extensions VSCode 56
6.3.1 Explorateur Zowe 57
6.3.2 Éditeur ouvert IBM Z 58
6.3.3 Code4z 58
6.4 Sommaire 59

7 Installation de Zowe CLI et des plug-ins 60
7.1 Installer les prérequis - Node.js 60
7.2 Installer Zowe CLI 60
7.2.1 Installer à partir du registre public npm 60
7.2.2 Installer à partir du package groupé 61
7.3 Installer les plug-ins Zowe CLI 62
7.3.1 Installer à partir du registre public npm 63
7.3.2 Installer à partir du package groupé 63
7.4 Sommaire 63

Partie 2 - Apprentissage du COBOL 64

8 COBOL de base 64
8.1 Caractéristiques COBOL 64
8.1.1 Entreprise COBOL 65

8.1.2 Objectifs du chapitre 65
8.2 Que doit savoir un programmeur COBOL novice pour être un programmeur COBOL expérimenté ? 65
8.2.1 Quelles sont les règles de codage et le format de référence ? 65
8.2.2 Quelle est la structure de COBOL ? 67
8.2.3 Que sont les mots réservés COBOL ? 67
8.2.4 Qu'est-ce qu'une instruction COBOL ? 67
8.2.5 Quelle est la signification d'un terminateur de portée ? . . 67
8.2.6 Qu'est-ce qu'une phrase COBOL ? 68
8.2.7 Qu'est-ce qu'un paragraphe COBOL ? 68
8.2.8 Qu'est-ce qu'une section COBOL ? 68
8.2.9 Comment exécuter un programme COBOL sous z/OS ? . 68
8.3 Divisions COBOL 69
8.3.1 Structure des divisions COBOL 69
8.3.2 Quelles sont les quatre divisions du COBOL ? 69
8.4 PROCEDURE DIVISION expliquée 70
8.5 Information additionnelle 71
8.5.1 Manuels professionnels 71
8.5.2 En savoir plus sur les récents progrès de COBOL 71
8.6 Labo 72
8.7 Lab - Zowe CLI et automatisation 85
8.7.1 Zowe CLI - Utilisation interactive 85
8.7.2 Zowe CLI - Utilisation par programmation 91

9 Data division **95**
9.1 Variables / Éléments de données 95
9.1.1 Restrictions de nom de variable / élément de données et types de données 96
9.2 Clause PICTURE 96
9.2.1 Symboles de clause PIC et types de données 97
9.2.2 Codage des noms de variables COBOL / éléments de données 97
9.2.3 Représentation sous forme de chaîne de caractères de la clause PICTURE 97
9.3 Littéraux 98
9.3.1 Constantes figuratives 98
9.3.2 Relations de données 98
9.3.3 Niveaux de données 99
9.4 MOVE et COMPUTE 100
9.5 Labo 102

10 Traitement des tableaux **105**
10.1 Définir une table 105
10.2 Référence à un item de la table 106
10.2.1 Indice 106
10.2.2 Indexation 106
10.3 Chargement d'une table avec des donnees 107

10.3.1 Charger une table dynamiquement . . . 108
10.3.2 REDEFINES une valeur codee en dur . . . 108
10.3.3 INITIALISER une table . . . 108
10.3.4 Attribution de valeurs à l'aide de la clause VALUE . . . 109
10.4 Tableaux de longueur variable . . . 109
10.5 Recherche dans une table . . . 110
10.5.1 Recherche en série . . . 111
10.5.2 Recherche binaire . . . 111

11 La gestion des fichiers 113
11.1 Code COBOL utilise pour le traitement sequentiel des fichiers . . . 114
11.1.1 Entrees et sorties COBOL . . . 114
11.1.2 Paragraphe FILE-CONTROL . . . 114
11.1.3 Source de donnees externe COBOL . . . 115
11.1.4 Ensembles de donnees, enregistrements et champs . . . 116
11.1.5 Blocs . . . 116
11.1.6 Clause ASSIGN . . . 116
11.2 PROCEDURE DIVISION traitement sequentiel des fichiers . . . 117
11.2.1 Ouvrir l'entree et la sortie pour la lecture et l'ecriture . . . 118
11.2.2 Fermer l'entree et la sortie . . . 118
11.3 Techniques de programmation COBOL pour lire et ecrire des enregistrements de maniere sequentielle . . . 119
11.3.1 Execution du paragraphe READ-NEXT-RECORD . . . 120
11.3.2 Paragraphe READ-RECORD . . . 121
11.3.3 Paragraphe WRITE-RECORD . . . 121
11.3.4 Traitement iteratif du paragraphe READ-NEXT-RECORD 121
11.4 Labo . . . 122

12 Structure du programme 126
12.1 Styles de programmation . . . 127
12.1.1 Qu'est-ce que la programmation structuree . . . 127
12.1.2 Qu'est-ce que la programmation orientee objet . . . 127
12.1.3 Style de programmation COBOL . . . 128
12.2 Structure de la Division des procedures . . . 128
12.2.1 Controle du programme et flux a travers un programme de base . . . 128
12.2.2 Instructions d'execution en ligne et hors ligne . . . 129
12.2.3 Utilisation de performs pour coder une boucle . . . 130
12.2.4 Apprendre un mauvais comportement a l'aide du mot-cle GO TO . . . 130
12.3 Les paragraphes sous forme de blocs de code . . . 132
12.3.1 Concevoir le contenu d'un paragraphe . . . 133
12.3.2 Ordre et nommage des paragraphes . . . 133
12.4 Contrôle de programme avec paragraphes . . . 135
12.4.1 EFFECTUER LES TEMPS . . . 135
12.4.2 PERFORM THROUGH . . . 136

12.4.3 PERFORM UNTIL . . . 136
12.4.4 PERFORM VARYING . . . 137
12.5 Utilisation de sous-programmes . . . 138
12.5.1 Spécification du programme cible . . . 139
12.5.2 Spécification des variables du programme . . . 139
12.5.3 Spécification de la valeur de retour . . . 140
12.6 Utilisation de cahiers . . . 140
12.7 Sommaire . . . 141
12.8 Laboratoire . . . 141

13 Fichier de sortie **143**
13.1 Examen du processus de sortie d'écriture COBOL . . . 143
13.1.1 DIVISION ENVIRONNEMENT . . . 143
13.2 FILE DESCRIPTOR . . . 144
13.2.1 FILLER . . . 144
13.3 En-têtes de rapport et de colonne . . . 145
13.3.1 HEADER-2 . . . 147
13.4 PROCEDURE DIVISION . . . 147
13.4.1 MOVE sentences . . . 148
13.4.2 PRINT-REC FROM phrases . . . 148
13.5 Labo . . . 148

14 Expressions conditionnelles **150**
14.1 Logique booléenne, opérateurs, opérandes et identifiants . . . 150
14.1.1 Expressions conditionnelles et opérateurs COBOL . . . 151
14.1.2 Exemples d'expressions conditionnelles utilisant des opérateurs booléens . . . 152
14.2 Expression conditionnelle mots réservés et terminologie . . . 153
14.2.1 IF, EVALUATE, PERFORM and SEARCH . . . 153
14.2.2 États conditionnels . . . 153
14.2.3 Noms conditionnels . . . 153
14.3 Opérateurs conditionnels . . . 155
14.4 Expressions conditionnelles . . . 155
14.4.1 Instructions IF ELSE (THEN) . . . 155
14.4.2 Instructions EVALUATE . . . 156
14.5 Conditions . . . 157
14.5.1 Conditions relationnelles . . . 157
14.5.2 Conditions de classe . . . 158
14.5.3 Conditions de signe . . . 158
14.6 Laboratoire . . . 158

15 Expressions arithmétiques **161**
15.1 Qu'est-ce qu'une expression arithmétique ? . . . 161
15.1.1 Opérateurs arithmétiques . . . 162
15.1.2 Instructions arithmétiques . . . 162
15.2 Règles de priorité des expressions arithmétiques . . . 163

15.2.1 Parenthèses 163
15.3 Limitations des expressions arithmétiques 164
15.4 Opérandes d'instructions arithmétiques 164
15.4.1 Taille des opérandes 164
15.5 Exemples d'instructions arithmétiques COBOL 165
15.6 Labo 167

16 Types de données **170**
16.1 Représentation des données 170
16.1.1 Représentation de la valeur numérique 170
16.1.2 Représentation de textes 171
16.2 COBOL DISPLAY versus COMPUTATIONAL 172
16.3 Labo 172

17 Fonctions intrinsèques **174**
17.1 Qu'est-ce qu'une fonction intrinsèque ? 175
17.1.1 Syntaxe de fonction intrinsèque 175
17.1.2 Catégories de fonctions intrinsèques 175
17.2 Fonctions intrinsèques dans Enterprise COBOL for z/OS V6.3 . 176
17.2.1 Exemple mathématique 176
17.2.2 Exemple statistique 176
17.2.3 Exemple de date/heure 177
17.2.4 Exemple financier 177
17.2.5 Exemple de gestion de caractères 178
17.3 Utilisation de fonctions intrinsèques avec des modificateurs de référence 178
17.4 Labo 178

18 Traitement ABEND **180**
18.1 Pourquoi un ABEND se produit-il ? 180
18.2 Types d'ABEND fréquents 180
18.2.1 S001 - Différence de longueur d'enregistrement/taille de bloc 181
18.2.2 S013 - Paramètres DCB en conflit 181
18.2.3 S0C1 - Instruction invalide 182
18.2.4 S0C4 - Exception de protection de stockage 182
18.2.5 S0C7 - Exception de données 182
18.2.6 S0CB - Division par zéro 183
18.2.7 S222/S322 - Délai d'attente / Travail annulé 183
18.2.8 S806 - Module non trouvé 183
18.2.9 B37/D37/E37 - Espace de l'ensemble de données ou de l'index PDS dépassé 184
18.3 Meilleures pratiques pour éviter les ABEND 184
18.4 Routines ABEND 185

Partie 1 - Pour commencer

1 Pourquoi COBOL ?

Ce chapitre présente COBOL, en particulier en référence à son utilisation dans les systèmes d'entreprise.

- **Qu'est-ce que COBOL ?**
- **Commenter le COBOL est-il utilisé aujourd'hui ?**
- **Pourquoi comprendre-je m'intéresse au COBOL ?**

1.1 Qu'est-ce que le COBOL ?

Un langage de programmation informatique a été spécialement concu pour les entreprises, le Common Business-Oriented Language, COBOL. COBOL a transformé et soutenu les entreprises à l'échelle mondiale depuis son invention en 1959. COBOL est responsable du fonctionnement quotidien efficace, fiable, sécurisé et invisible de l'économie mondiale. La logique quotidienne utilisée pour traiter nos données les plus critiques est souvent mise en oeuvre à l'aide de COBOL. nb
De nombreux programmes COBOL ont des décennies d'améliorations qui incluent la logique métier, les performances, le paradigme de la programmation et les interfaces entre les programmes d'application et les moniteurstransactionnels, les sources de données et Internet.

Plusieurs centaines de langages de programmation ont été développés au cours des 60 dernières années dans le but de transformer le paysage des technologies de l'information. Certains de ces langages, tels que C, C++, Java et JavaScript, ont en effet transformé le paysage informatique en constante expansion. Cependant, COBOL continue de se distinguer des autres langages de programmation en raison de sa capacité inhérente à gérer de grandes quantités de données critiques stockées dans les plus gros serveurs tels que le mainframe IBM Z.

Continuellement mis à jour pour intégrer des paradigmes de programmation et des pratiques exemplaires modernisées et éprouvées, COBOL restera un langage de programmation essentiel dans un avenir prévisible. Apprendre le COBOL vous permet de lire et de comprendre le fonctionnement quotidien des systèmes critiques. La connaissance et la maitrise du COBOL sont une compétence requise pour etre un « développeur complet » dans les grandes entreprises.

1.2 Comment le COBOL est-il utilisé aujourd'hui ?

COBOL est partout. Vous avez probablement utilisé une application écrite en COBOL aujourd'hui. Par exemple, les statistiques suivantes :

- Environ 95% des opérations sur les Distributeurs automatiques utilisent du code COBOL.

- COBOL alimente 80% des transactions “personnelles”.
- Chaque jour, les systèmes COBOL gérent 3 000 milliards de dollars d’activitrés commerciales.

Quelle est l’ampleur eu déploiement du COBOL? Voici quelques chiffres étonnants:

- Chaque jour, il y a 200 fois plus de transactions COBOL exécutées qu’il n’y a de recherches Google.
- Il y a plus de 220 milliards de lignes de programmes COBOL en cours d’exécution aujourd’hui, ce qui équivaut à environ 80% du code utilisé dans le monde.
- 1 500 000 000 de nouvelles lignes de COBOL sont écrites chaque année.

1.3 Pourquoi-je devrais-je m’intéresser au COBOL ?

Le langage de programmation COBOL, l’optimisation du compilateur COBOL et les performances d’exécution du COBOL ont plus de 50 ans d’avancées technologiques qui contribuent à la fondation de l’économie mondiale. La logique métier de base de nombreuses grandes entreprises depuis des décennies d’amélioration et d’optimisation des règles métier sont intégrées dans les programmes COBOL.

Le fait est que quoi que vous lisiez ou entendiez à propos de COBOL, soyez très sceptique. Si vous avez l’opportunité de travailler directement avec une personne impliquée dans l’écriture ou la maintenance d’une application critique écrite en COBOL, vous en apprendrez davantage sur le fonctionnement de l’activité principale. Les chefs d’entreprise, les analystes commerciaux et les décideurs vont et viennent. La somme de toutes les bonnes décisions commerciales se trouve fréquemment dans les décennies de changements mis en oeuvre dans les programmes COBOL. La réponse à « Comment fonctionne cette entreprise ? » peuvt etre souvent trouvée dans ses programmes COBOL.

Ajoutez ce qui suit à votre connaissance du COBOL. C’est un mythe absolu que vous devez avoir au moins 50 ans pour être efficace en COBOL. COBOL est extrêmement facile à apprendre et à comprendre. L’une des nombreuses raisons pour lesquelles les institutions financières aiment le COBOL, est le fait qu’il n’est pas nécessaire d’etre un programmeur pour lire et comprendre la logique des programmes. Ceci est important car le code de métier ses applications critiques est soumis à un audit. Les auditeurs ne sont pas des programmeurs. Cependant, ils ont la responsabilité de s’assurer que les états financiers de l’entreprise sont présentés fidèlement. C’est le traitement COBOL qui entraine fréquemment les mises à jour du grand livre et les états financiers ultérieurs.

Maintenant, un enseignement tiré du monde réel. Un commentaire récemment fait dans un journal économique bien connu a été cité comme disant : « COBOL est un langage informatique utilisé dans les affaires et la finance. Il a été concu

pour la première fois en 1959 et est assez ancien et lent. » Une personne très expérimentée en technologie d'entreprise sait que la seule partie vraie de cette dernière phrase est que COBOL a été concu pour la première fois en 1959.

Ce n'est pas un secret pour personne que de nombreuses banques exploitent toujours des millions de lignes COBOL sur des mainframes. Ils voudraient probablement les remplacer à un moment ou un autre. Alors pourquoi ne l'ont-ils pas fait ? La plupart des banques existantes depuis assez longtemps ressentent encore la douleur de la crise des logiciels des années 1960. Après avoir dépensé énormément d'argent et de temps pour développer leurs systèmes informatiques, ils se sont finalement retrouvés avec un système central COBOL entièrement fonctionnel, bien testé et stable.

En parlant avec des personnes qui ont travaillé sur de tels systèmes, ils ont aujourd'hui des frontaux et des wrappers Java qui ajoutent des fonctionnalités ou des interfaces plus modernes, ils exécutent l'application sur des serveurs répliqués virtualisés, mais en fin de compte, tout passe par cette logique à coeur unique. Et cette logique fondamentale est rarement touchée ou modifiée, sauf si nécessité.

Du point de vue de l'ingénierie logicielle, cela a meme du sens. Les réécritures sont toujours plus chères que prévu, et prennent toujours plus de temps que prévu (OK, probablement pas toujours. Mais souvent.). Ne modifiez jamais un système en cours d'exécution, etc., sauf pour de très bonnes raisons techniques et commerciales.

2 VSCode avec Zowe Explorer

Zowe Explorer est une extension open source de VS Code qui permet aux développeurs et aux administrateurs système d'interagir avec les mainframes z/OS.

- **Introduction à Zowe Explorer**
- **Utiliser Zowe Explorer**
- **Profils dans Zowe Explorer**
  - **Identifiants sécurisés**
  - **Création d'un nouveau profil**
  - **Modification des profils**
  - **Suppression de profils**
- **Sommaire**

2.1 Introduction à Zowe Explorer

L'extension Zowe Explorer modernise la facon dont les développeurs et les administrateurs du système interagissent avec les mainframes z/OS. Travailler avec des ensembles de données et des fichiers USS à partir de VS Code peut être plus pratique que d'utiliser un émulateur 3270. L'extension offre les fonctionnalités suivantes:

- Créer, modifier, renommer, copier et télécharger des ensembles de données directement sur un mainframe z/OS.
- Créez, modifier, renommer et télécharger des fichiers USS directement sur un mainframe z/OS.
- Processus simplifié pour accéder aux ensembles de données, aux fichiers USS et aux Jobs.
- Interaction facile avec plusieurs systèmes z/OS

L'explorateur Zowe peut être installé dans VS Code en recherchant "Zowe Explorer" sur le marché la "Extension Marketplace" dans VS Code et en choisissant installer. Pour voir des instructions plus détaillées sur l'installation de cette extension, reportez-vous à "Installation de VSCode et des extensions".

2.2 Utilisation de Zowe Explorer

Zowe Explorer vous permet de travailler avec des ensembles de données, des fichiers Unix System Service (USS) et des tâches.

Zowe Explorer propose les fonctions suivantes :

Ensembles de données - Datasets

- Rechercher des ensembles de données correspondant aux filtres souhaités et afficher leur contenu
- Télécharger, modifier et télécharger des membres PDS existants

-
- Créer et supprimer à la fois des ensembles de données et des membres d'ensembles de données
- Interagir avec des ensembles de données de plusieurs systèmes simultanément
- Renommer les ensembles de données
- Copier des ensembles de données
- Soumettre un JCL à partir d'un membre de l'ensemble de données choisi

Fichiers USS

- Afficher plusieurs fichiers Unix System Services (USS) simultanément
- Télécharger, modifier et télécharger des fichiers USS existants
- Créer et supprimer des fichiers et des répertoires USS
- Interagir avec les fichiers USS de plusieurs systèmes simultanément
- Renommer les fichiers USS

Jobs

- Visualiser plusieurs jobs simultanément
- Télécharger le contenu du spool
- Interagir avec les jobs de plusieurs systèmes simultanément

Pour plus d'informations sur Zowe Explorer et les différents cas d'utilisation, visitez la place de marché marketplace

2.3 Profils dans Zowe Explorer

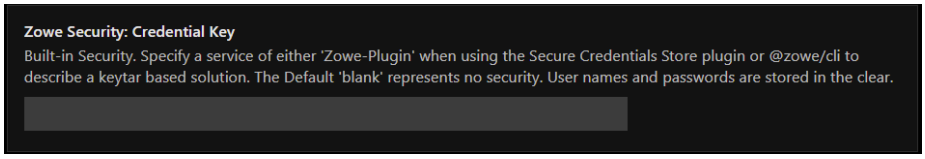
Les profils servent de point de contact pour Zowe Explorer et le Mainframe. Les profils contiennent l'URL des services API aulquels ous souhaitez vous connecter et vos informations d'identification. L'information de profil principale dont vous avez besoin pour Zowe Explorer sont la connexion z/OSMF. Si vous avez installé Zowe Explorer, vous pouvez suivre les étapes de cette section pour vous connecter au mainframe.

2.3.1 Identifiants sécurisés

Zowe Explorer dispose d'une zone de stockage d'informations d'identification sécurisé intégré. Cela vous permet de crypter les informations d'identification enregistrées dans votre machine et, par conséquent, d'effectuer votre connexion au Mainframe.

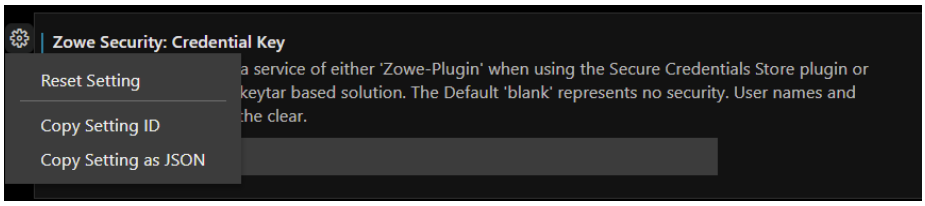
Pour activer cette fonctionnalité, faites ceci:

1. Cliquez sur **Gear Icon** en bas à gauche et sélectionnez **Settings**
2. Cliquez sur **User Settings** > **Extensions** > **Zowe Exporer Settings** Recherchez le champ **Zowe Security : Credential Key**



3. Tapez **Zowe-Plugin** dans la zone de texte. Cela déclenchera l'accès à la zone de stockage d'informations d'identification.

Alternativement, pour activer cette fonctionnalité en modifiant settings.json, survolez gear et cliquez sur "Copy Setting at JSON". Vous pouvez ensuite coller cette donnée dans settings.json et mettre à jour la valeur dans Zowe Plugin.

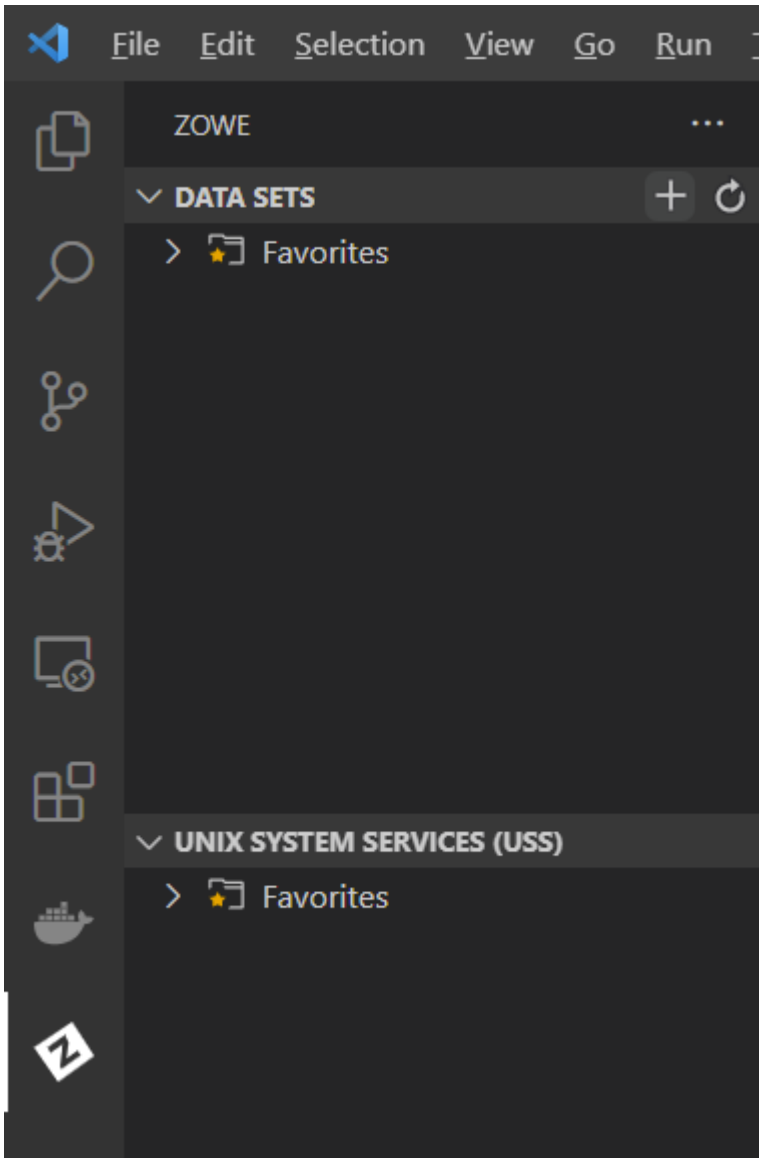


Remarque : si vous utilisez Zowe CLI et que vous avez installé le plugin Secure-Credential-Store, les étapes pour l'activer seront toujours les mêmes.

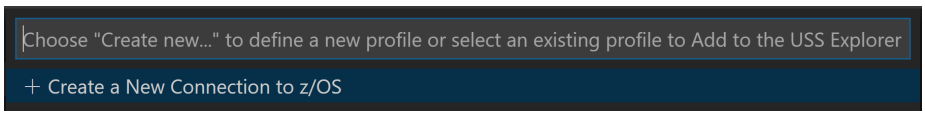
2.3.2 Création d'un nouveau profil

Suivez ces étapes :

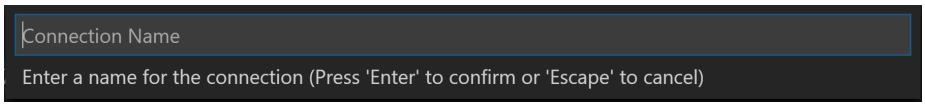
1. Accédez à l'arborescence Zowe Explorer sur le coté droit et recherchez le signe +.



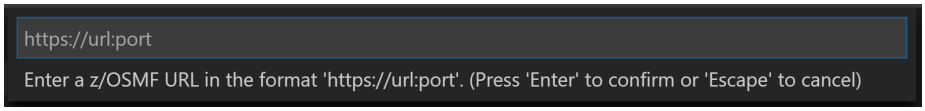
2. Cliquez sur le signe +. Une boite de dialogue apparaitra et vous demandera si vous souhaitez “Créer une nouvelle connexion à z/OS”.
3. Appuyez sur Entrée ou cliquez sur cette sélection.



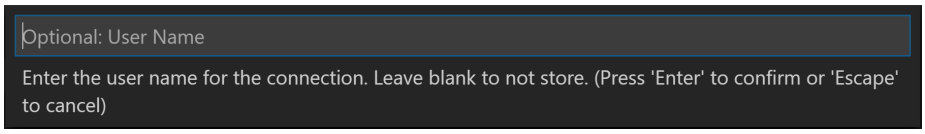
4. Saisir un Profile name dans le champ “Connection Name”.



5.Saisir l'URL et le port que vous avez recus par e-mail lors de votre inscription au cours COBOL. Les informations de connexion dont vous avez besoin ont le titre "IP address for VSCode extension".

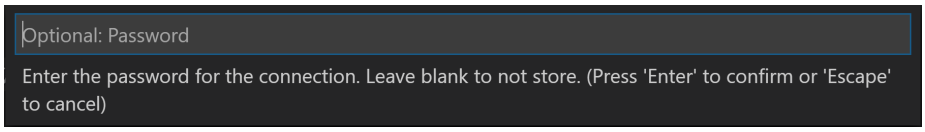


6. Entrez votre nom d'utilisateur. Celui-ci est également inclus dans l'e-mail.



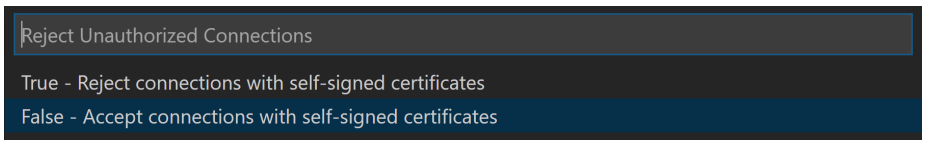
Remarque : vous pouvez laisser ce champ vide si vous ne souhaitez pas enregistrer vos informations d'identification sur votre machine. Votre nom d'utilisateur vous sera demandé une fois que vous commencerez à utiliser Zowe Explorer.

7. Entrez votre mot de passe.

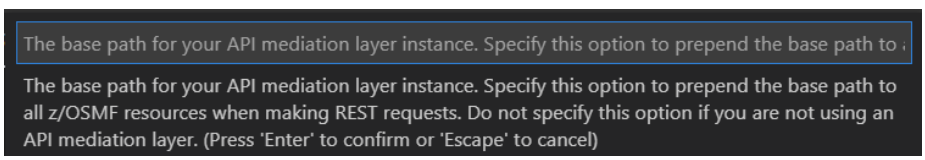


Remarque : vous pouvez laisser ce champ vide si vous ne souhaitez pas enregistrer vos informations d'identification sur votre machine. Votre nom d'utilisateur vous sera demandé une fois que vous commencerez à utiliser Zowe Explorer.

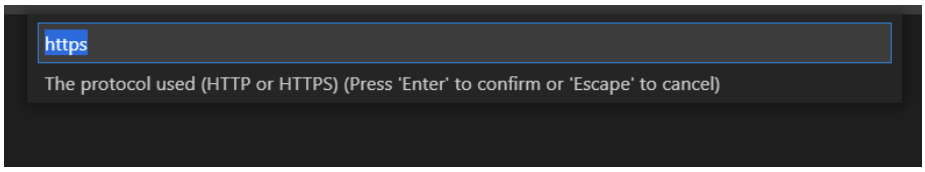
8. Sélectionnez True/False si vous souhaitez accepter ou rejeter les certificats auto-signés. Pour ce cours, veuillez sélectionner false.



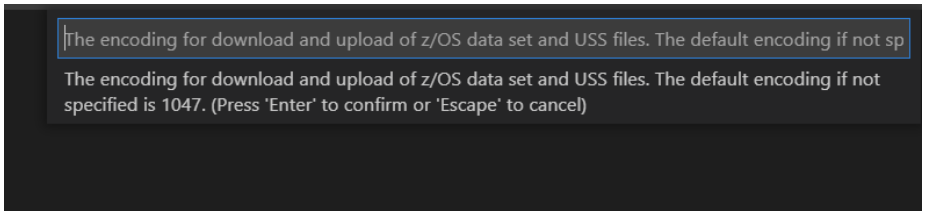
9. Appuyez sur **Enter** pour le reste des invites pour choisir les valeurs par défaut



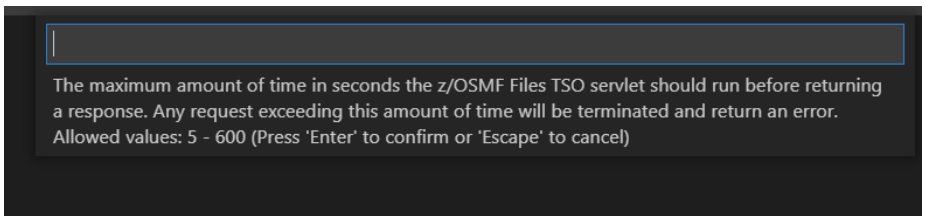
10. Saisir le protocol utilisé pour cette connexion. Pour ce cours conserver la valeur par defaut (“HTTPS”) et appuyer sur **Enter** pour continuer.



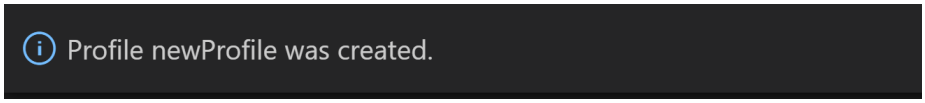
11. Saisir le mode d’encodage utilisépour le téléchargement et le chargement des data sets z/OS et des fichiers USS. Pour ce cours laisser cette zone vide et appuyer sur **Enter** pour continuer.



12. Saisir le temps d’attente maximum pour une réponse.Pour ce cours laisser cette zone vide et appuyer sur **Enter** pour continuer. to continue



Si vous réussissez, vous recevrez ce message d’information :



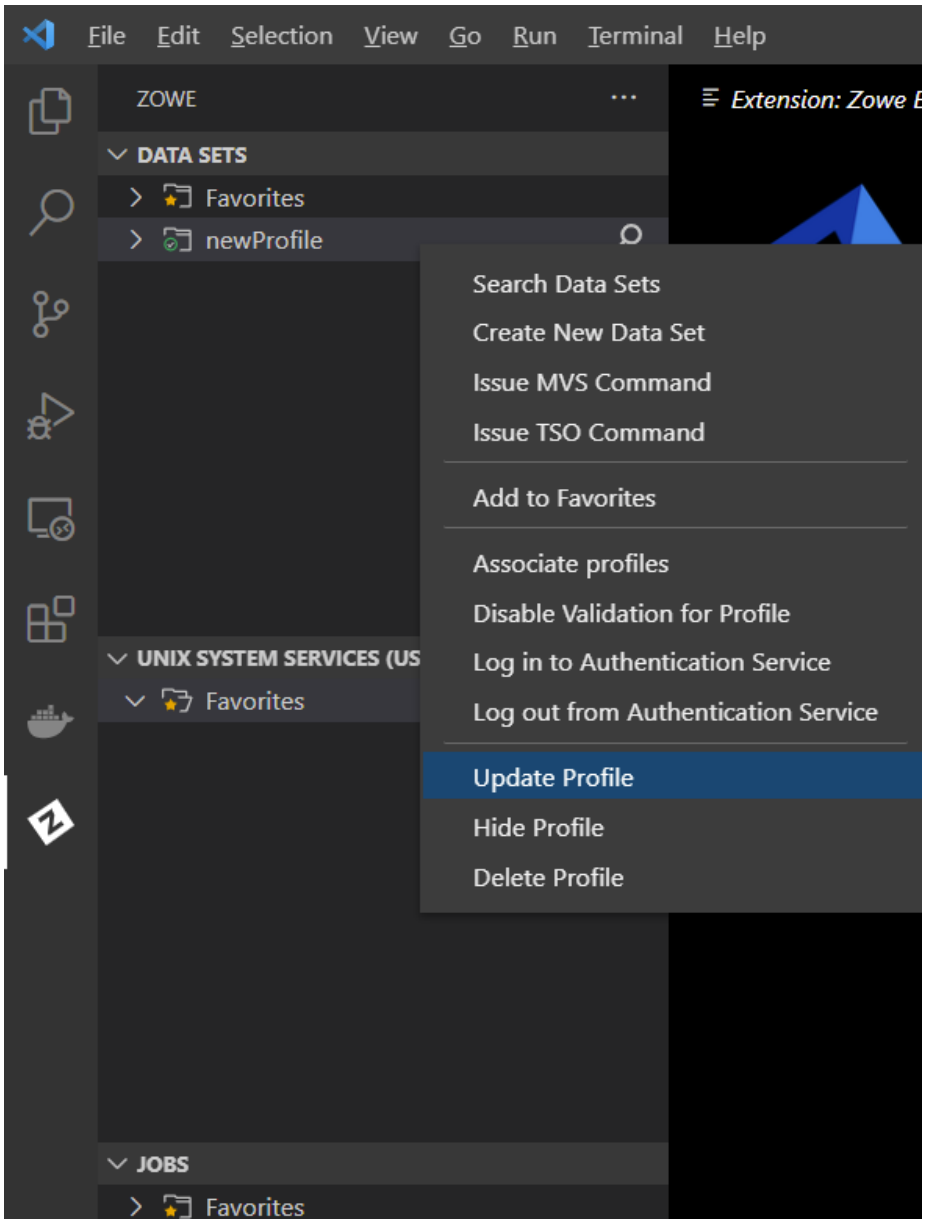
2.3.3 Modification des profils

La version Zowe Explorer v1.5.0 a introduit l’édition de profil. Cela vous permet de réviser les informations de votre profil existant et de continuer à utiliser Zowe Explorer.

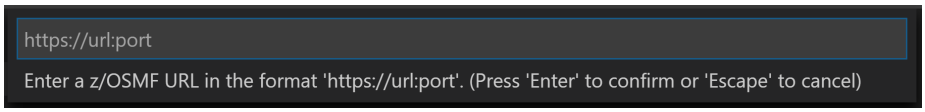
Suivez ces étapes :

1. Ajoutez votre profil à l’un des arbres Zowe Explorer.
2. Faites un clic droit sur le profil que vous souhaitez modifier
3. dans la liste déroulante, choisissez **Update Profile**. Une boite de dialogue s’ouvre et affiche les informations actuelles de votre profil, que vous pouvez

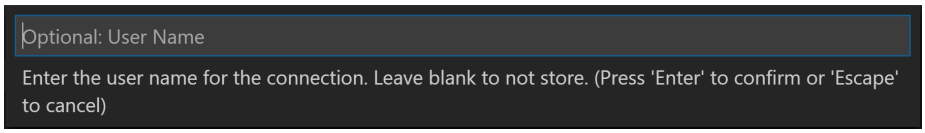
modifier au besoin.



3. Modifiez les informations d'URL si les modifications sont nécessaires, ou Enter pour confirmer que les informations sont toujours corrigées.

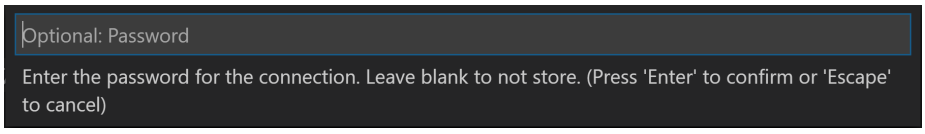


4. Modifiez votre nom d'utilisateur.



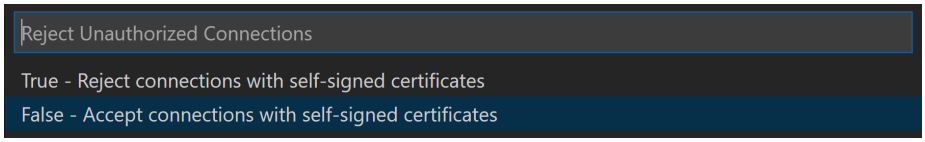
Remarque : vous pouvez laisser ce champ vide si vous ne souhaitez pas enregistrer vos informations d'identification sur votre machine. Votre nom d'utilisateur vous sera demandé une fois que vous commencerez à utiliser Zowe Explorer.

5. Modifiez votre mot de passe.

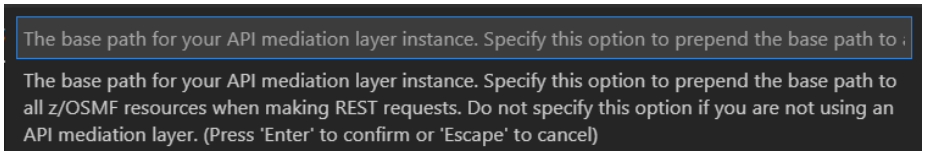


Remarque : vous pouvez laisser ce champ vide si vous ne souhaitez pas enregistrer vos informations d'identification sur votre machine. Votre nom d'utilisateur vous sera demandé une fois que vous commencerez à utiliser Zowe Explorer.

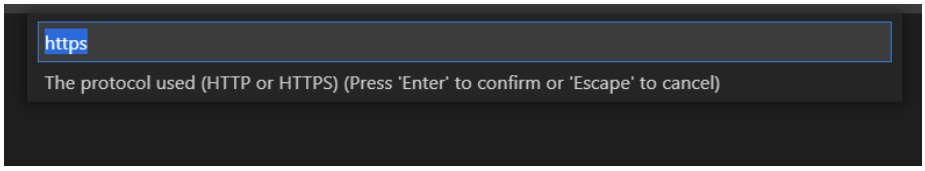
6. Modifiez vos connexions autorisées



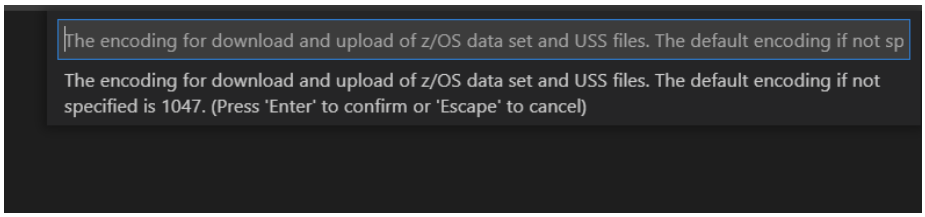
7. Modifiez votre chemin de base d'API



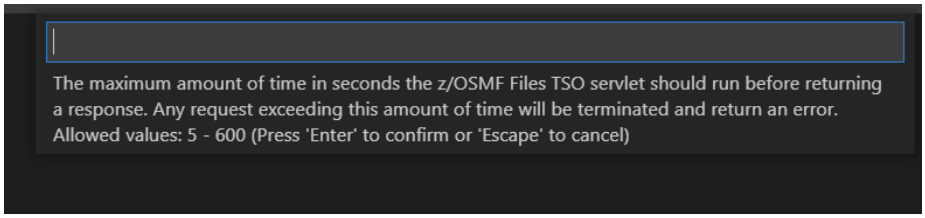
8. Modifier le protocole de connexion



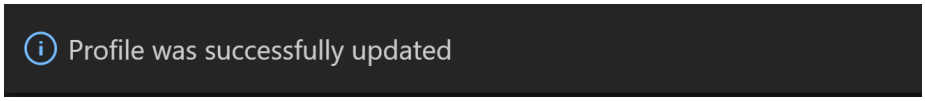
9. Modifier l'encodage



10. Modifier le délai maximum de réponse



Si vous réussissez, un message d'information apparaitra :



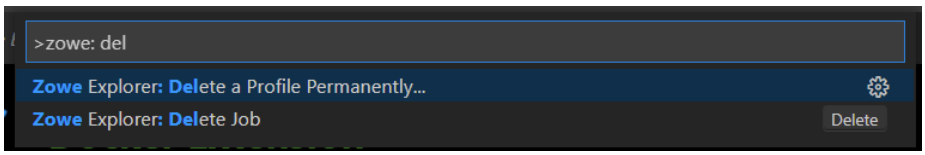
2.3.4 Suppression des profils

La version Zowe Explorer v1.5.0 a permis de supprimer des profils. Cela vous permet de supprimer définitivement les profils indésirables et de nettoyer vos fichiers. Vous pouvez supprimer des profils à l'aide de la palette de commandes ou dans l'arborescence.

Suivez ces étapes :

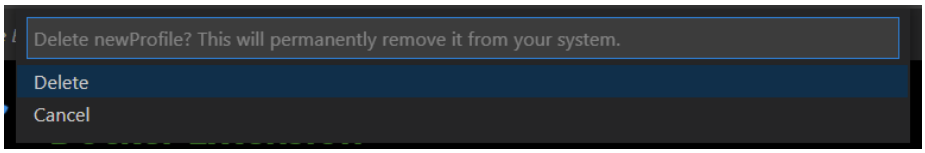
Palette de commandes :

1. Appuyez sur **CTRL+MAJ+P** ou cliquez sur **View** > **Command Palette** pour ouvrir la palette de commandes
2. Tapez "Zowe : Delete". Cette commande permet de supprimer définitivement un profil.

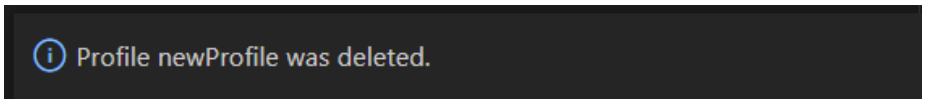


3. Sélectionnez le profil que vous souhaitez supprimer.

4. Confirmez que vous souhaitez supprimer votre profil.

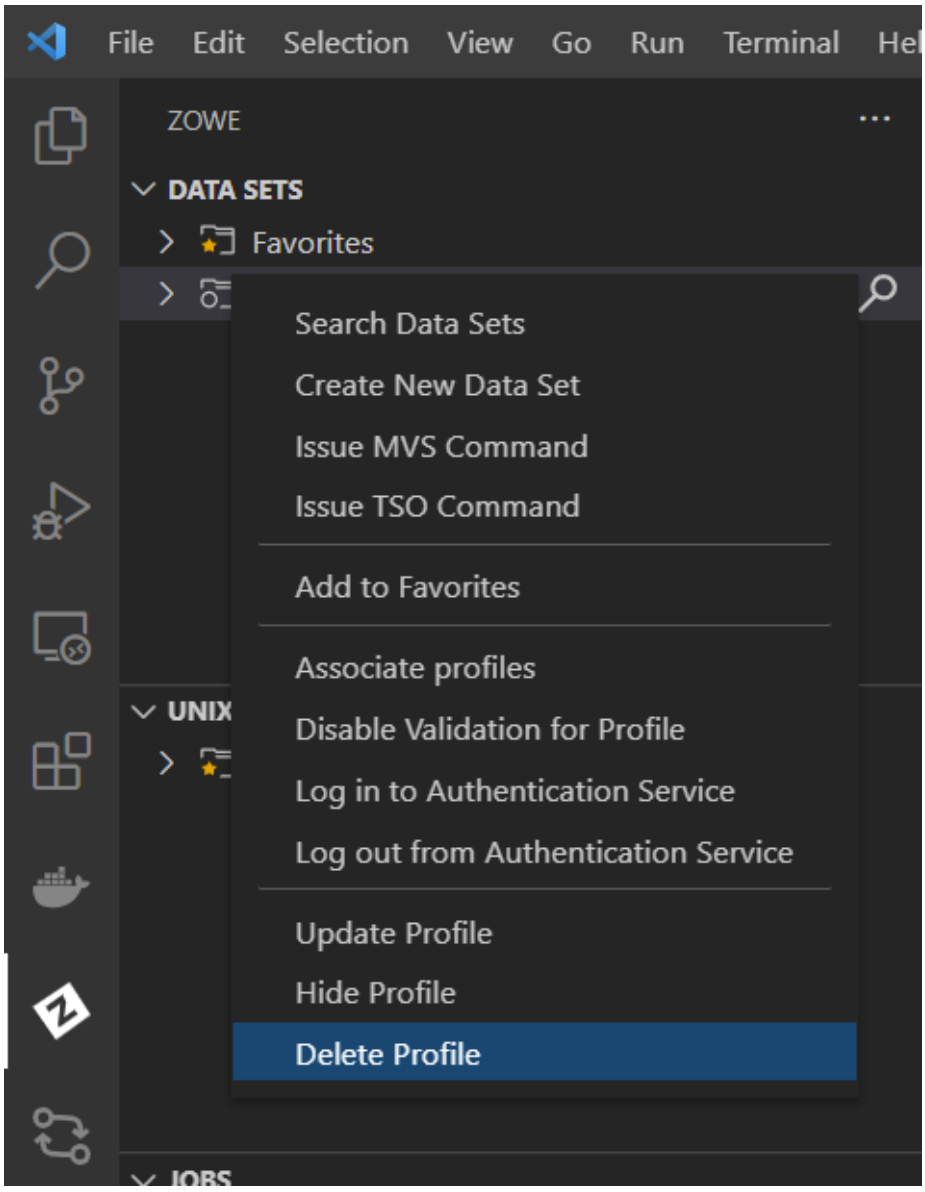


Une fois confirmé, le message suivant s'affiche :

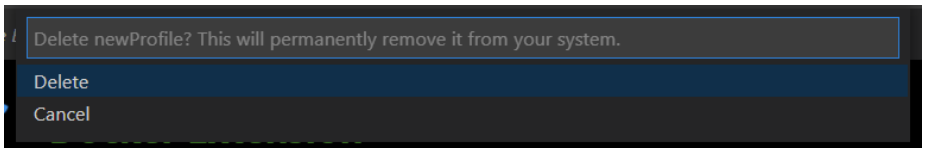


Arbre de l'explorateur Zowe :

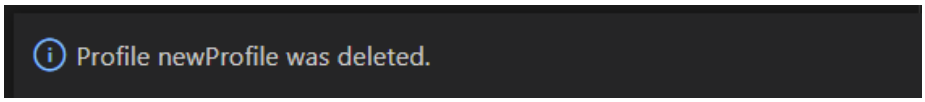
1. Cliquez avec le bouton droit sur le profil et sélectionné **Delete Profile**.



2. Confirmez que vous souhaitez supprimer votre profil.



3. Une fois confirmé, le message suivant s'affiche :



2.4 Sommaire

Dans cette section, vous avez appris les fonctionnalités de base de l'extension Zowe Explorer et comment créer et utiliser les profils « zosmf » compatibles avec Zowe.

3 VSCode avec Z Open Editor

Dans ce chapitre, nous expliquons comment utiliser l'extension IBM Z Open Editor pour VSCode et comment son utilisation peut vous aider à développer le code source COBOL dans un environnement riche en fonctionnalités.

- **Introduction à l'éditeur ouvert IBM Z**
  - **Qu'est-ce qu'IBM Z Open Editor ?**
  - **Le role du Language Server Protocol**
  - **Installation d'IBM Z Open Editor pour VS Code**
- **Édition de base**
  - **Extensions de fichiers connues**
  - **Marges**
  - **Variable d'extension**
  - **Mise en évidence de Syntaxe**
- **Navigation dans le code**
  - **Vue générale**
  - **Vue Breadcrumb**
  - **Aller à la declaration / reference**
- **Complétion de code**
  - **Complétion de mots réservés COBOL**
  - **Cométion de variable**
  - **completion CICS, MQ, Db2 API"
- **Refactoring du Code**
  - **Renommer les variables**
  - **Gestion des erreurs**
- **Résumé

3.1 Introduction à l'éditeur IBM Z Open Editor

Cette section présente IBM Z Open Editor.

3.1.1 Qu'est-ce qu'IBM Z Open Editor ?

IBM Z Open Editor est une extension gratuite pour Visual Studio Code (VSCode) qui prend en charge les langages COBOL, PL/I et JCL. Parallèlement à cette prise en charge linguistique, il fournit également une assistance basée sur le contenu pour les applications qui appellent les API CICS, MQ, IMS et DB2. Le code source n'a même pas besoin de résider sur z/OS, il peut se trouver dans un référentiel de code source, localisé dans un fichier ou sur z/OS. Bien que ce cours se concentre sur COBOL en tant que langage source, la plupart des fonctions que nous abordons s'appliquent également au PL/I et au JCL.

3.1.2 Le role du Language Server Protocol

Les environnements de développement intégrés doivent toujours fournir une plateforme riche pour tous les langages de programmation pris en charge, cependant, la prolifération de ces langages et la vitesse à laquelle les nouveaux éditeurs arrivent sur le marché rend le maintien au goût du jour difficile. Chaque éditeur devrait fournir un plugin spécifique pour chaque langue qu'il souhaite prendre en charge, et par conséquent rendrait la prise en charge d'un langage différent d'un éditeur à l'autre.

Microsoft a concu le Language Server Protocol (LSP) pour agir en tant que description commune de la manière dont des fonctionnalités telles que l'auto complétion doivent etre implémentées pour un langage spécifique. Les langages qui ont implémenté un protocole LSP peuvent donc être utilisés dans n'importe quel éditeur qui supporte LSP. De nombreuses entreprises et la communauté Open Source ont collaboré afin de fournir des serveurs LSP pour un éventail de langages différents.

Le LSP définit six fonctionnalités générales qui doivent être implémentées pour qu'un Protocole Serveur de Langage soit conforme au LSP. Ces fonctionnalités incluent la complétion de code, le survol, le positionnement direct, les symboles de l'espace de travail, la recherche de références et les diagnostics. IBM Z Open Editor fournit des serveurs conformes à LSP pour les langages Enterprise COBOL et Enterprise PL/I for z/OS. En plus d'être conformes, ils offrent également des fonctionnalités supplémentaires dont nous discuterons plus loin.

Remarque : Plus d'informations sur les LSP sont disponibles sur : https://langserver.org

3.1.3 Installation d'IBM Z Open Editor pour VS Code

IBM Z Open Editor peut être installé dans le code VS en recherchant "IBM Z Open Editor" dans la place de marché des extensions VSCode et en choisissant install. Une fois installé, l'éditeur par défaut sera activé pour fournir une expérience d'édition riche pour COBOL, PL/I et JCL. Il n'est pas nécessaire d'utiliser un éditeur spécifique pour ces langues. Pour voir des instructions plus détaillées sur l'installation de cette extension, reportez-vous à "Installation of VSCode and extensions". 3.2 Edition de base Dans le reste de ce chapitre nous utiliserons le programmes example CBL0001 sample pour démontrer la richesse d'une expérience d'édition de code COBOL avec VSCode. Donc démarrons VSCode, insa=tallons IBM Z Open Editor, si ce n'est pas déja fait, ouvrons CBL0001, and commencons. correspondance avec un un langage connu. For COBOL les associations suivantes sont utilisées: are used: - *.COBOL*

- *.CBL*
- *.COB*
  3.2.1 Extensions de fichier reconnues Afin que VSCode puisse utiliser les fonctionnalité de IBM Z Open Editor, il a besoin de savoir que nous

souhaitons éditer un fichier COBOL. VSCode accomplit ceci en comparant l'endroit et le nom du fichier ouvert à une liste d'extensions pour mise en

- *.COBCOPY*
- *.COPYBOOK*
- *.COPIE*

Ceux-ci sont appliqués à la fois aux fichiers locaux et aux fichiers contenus dans un ensemble de données partitionnées ou un PDS sur l'ordinateur central, que vous pouvez considérer pour plus de simplicité comme un dossier. Ainsi, un PDS appelé :

```
Z99994.COBOL
```

Ou un fichier sur le système de fichiers local appelé :

```
PROGA1.COBOL
```

```
Sera supposé être du code COBOL. Ces informations sont stockées dans le fichier
```

```
### Marges
```

```
La première chose que vous remarquerez lors de l'édition du code source COBOL es
```

3.1.4 Extension des variables

Tout en parcourant CBL0001, tapez « CTRL + G » pour aller vers une ligne de code spécifique. Une petite boite de dialogue s'ouvrira vous demandant la ligne à laquelle vous souhaitez accéder, appuyez « 87 » et s'appuyer sur la touche Enter. VSCode mettra en surbrillance cette ligne de code et vous y mènera directement, comme le montre la figure 1.

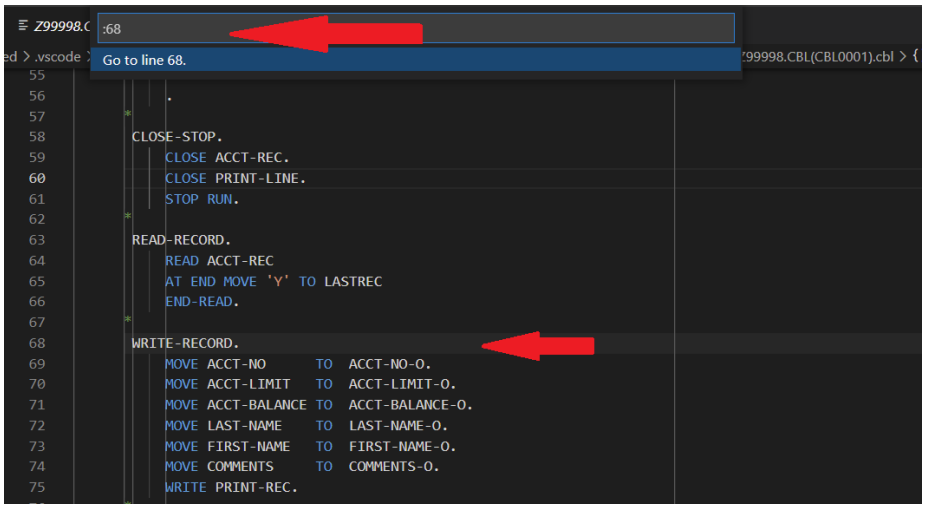


Figure 1. Navigation vers une ligne de code spécifique

Si vous passez votre souris sur le champ 'ACCT-NO-O, une fenêtre contextuelle apparaitra mettra en évidence la déclaration de cette variable a la figure 2.

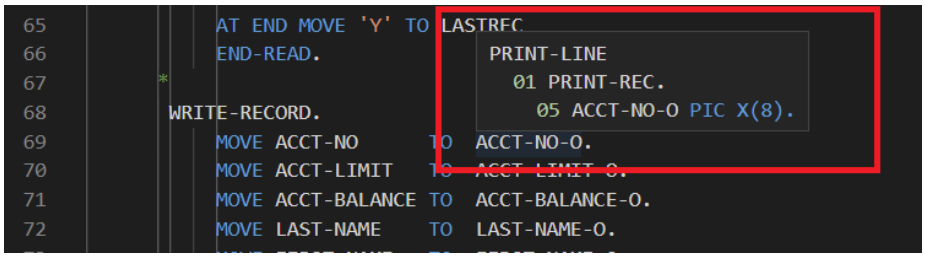


Figure 2. Afficher la déclaration de la variable

Étant donné que ce champ est une variable de niveau 05 imbriquée dans une variable de niveau 01, la fenêtre contextuelle affiche la déclaration du champ en tant que variable de huit octets avec Picture, le nom de la structure parente et la définition de fichier où elle se trouve. Si vous maintenez la touche CMD/Ctrl tout en survolant le champ, la fenêtre contextuelle contiendra en plus la ligne de code où la variable est définie ainsi que les trois lignes de code suivantes. Ces fonctionnalités peuvent être extrêmement utiles lors de l'analyse du code existant.

3.1.5 Mise en évidence de la syntaxe

Le code COBOL que vous êtes en train d'éditer sera également mis en évidence pour vous aider à comprendre les différents éléments du langage COBOL. Selon le thème de couleur que vous avez sélectionné dans VSCode, les commentaires, les mots réservés COBOL, les littéraux et les variables seront colorés différemment,

ce qui vous permettra de repérer les problèmes de syntaxe évidents dès le début avant même de soumettre le code à une la compilation.

3.2 Navigation dans le code

Bien que les exemples de code que nous utilisons dans cette section soient assez petits, le code que vous avez appris peut comporter des centaines ou des milliers de lignes. Pouvoir comprendre la structure générale du code source et pouvoir s'y retrouver sans se perdre est un gros avantage lors de l'édition de COBOL. Heureusement, il existe quelques fonctionnalités intéressantes pour vous aider, dont nous parlerons ensuite.

3.2.1 Vue générale (outline)

Dans la barre latérale de l'explorateur de VSCode, il y a une vue d'ensemble qui sera remplie chaque fois que vous éditez COBOL. Cette vue contient un apercu exapansible ou réductible de chaque division, structure de données et paragraphe de votre code. Cela vous permet de visualiser facilement la structure du code source. En cliquant sur un paragraphe, une division ou un élément de données particulier déplacera simultanément l'éditeur en affichant cette section du code et la mettra en surbrillance, comme illustré à la figure 3. Cela permet de se positionner sur une partie spécifique du code très facilement.

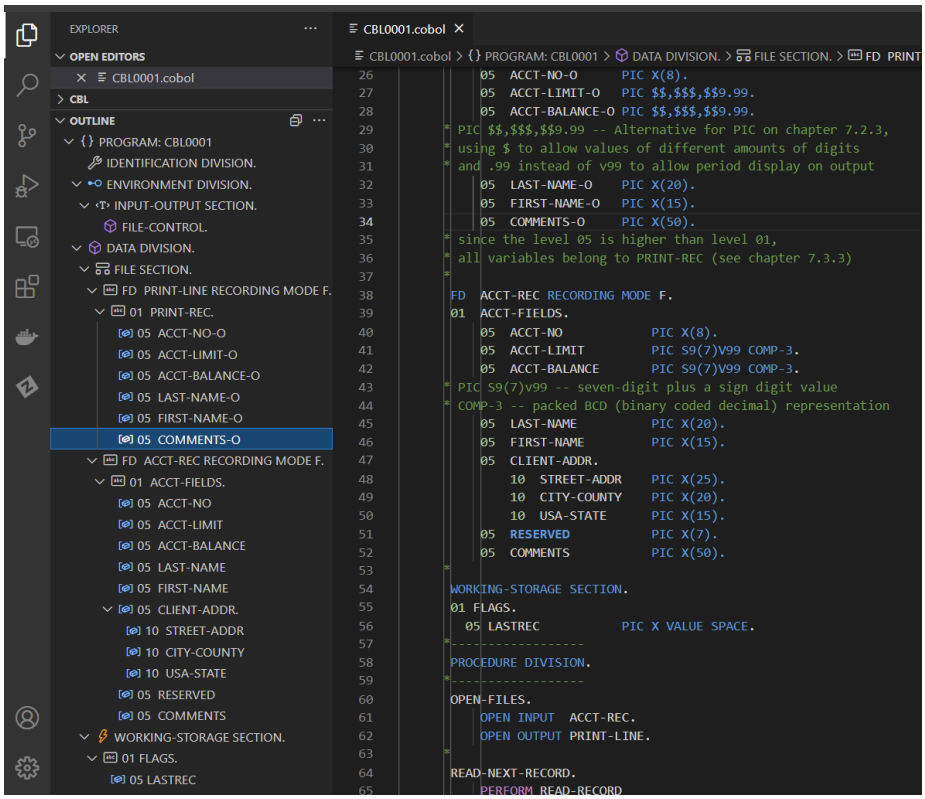


Figure 3. Outline dans VSCode

3.2.2 Vue Breadcrumb (fil d'Ariane)

De même, le Breadcrumb en haut de l'éditeur peut indiquer où se trouve la ligne de code actuelle dans la structure du code source COBOL. Au fur et à mesure que vous parcourez le code source dans l'éditeur, le Breadcrumb met à jour automatiquement en reflétant où vous vous trouvez dans la structure du programme et vous fournit un mécanisme pour passer à un élément différent du code. Encore une fois, si vous ouvrez CBL0001 dans VSCode et passez à la ligne 50, cette ligne est une déclaration du champ USA-STATE au sein de la structure ACCT-FIELDS, dans la FILE-SECTION de la DATA-DIVISION. En haut de l'éditeur, le Breadcrumb affiche les informations présentées dans la figure 4.

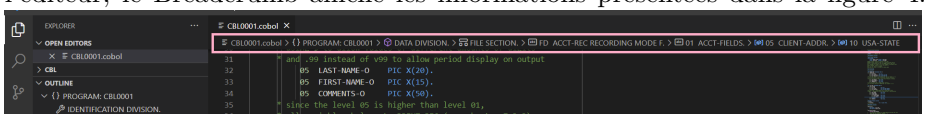


Figure 4. Breadcrumb dans VSCode

Cliquer sur l'un des éléments du Breadcrumb mettra en évidence cet élément du code dans l'éditeur, visualisant rapidement l'emplacement de cet élément dans

le code. Il affichera également une vue du code dans une fenêtre contextuelle, illustrée à la figure 5. , similaire à la vue d'ensemble discutée précédemment.

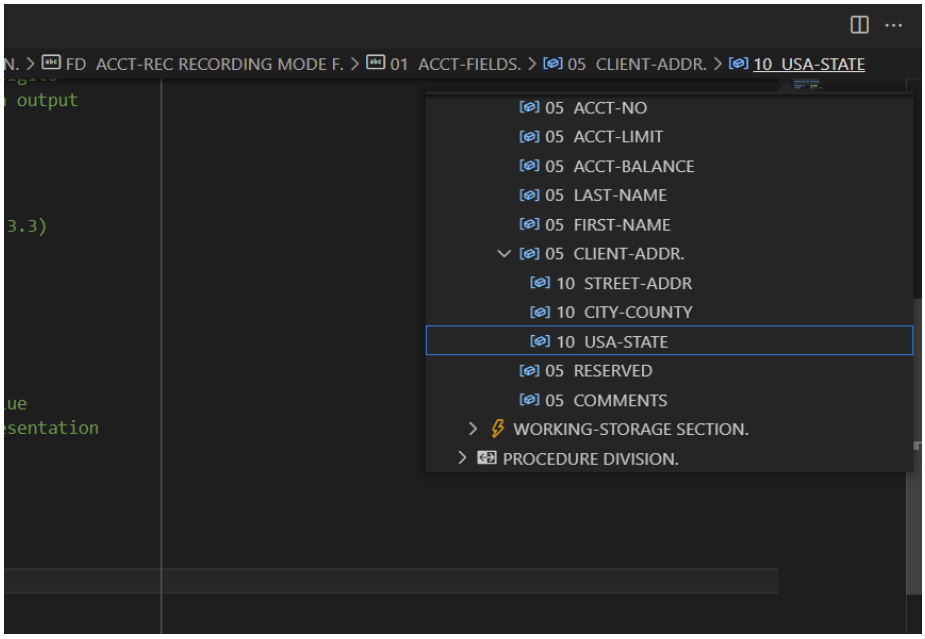


Figure 5. Vue contextuelle du code via la sélection du Breadcrumb

3.2.3 Aller à la déclaration / référence

En parcourant le code, vous rencontrez des instructions COBOL PERFORM ou des références de variables. Souvent, vous devrez naviguer jusqu'à la définition de ce paragraphe ou de cette variable pour suivre l'exécution du code. À la ligne 64 de CBL0001, nous voyons un ensemble d'instructions PERFORM. Placez le curseur dans le nom, READ-RECORD, sur la ligne 65, cliquer sur le bouton droit et sélectionnez **Go to Definition** . L'éditeur navigue alors jusqu'au paragraphe READ-RECORD à la ligne 82. Au lieu du clic droit, la même fonction peut être atteinte en utilisant la touche de fonction F12.

"Go to References" fait l'inverse de cette opération et vous permet de naviguer depuis la définition d'un paragraphe ou d'une variable vers tous les endroits qui, dans l'application, référencent ce paragraphe ou cette variable. Pour vous en rendre compte, naviguez à la ligne 82 de CBL0001, qui est à nouveau la déclaration du paragraphe READ-RECORD. Pour voir tous les endroits où ce paragraphe est appelé, faites un clic droit et sélectionné **Go to References** , ou appuyez sur la combinaison de touches **MAJ+F12** . Cela invoquera une nouvelle boîte de dialogue contextuelle qui affiche toutes les références à ce paragraphe dans le code, tel qu'illustré à la figure 6.

Remarque : Si **MAJ+F12** ne fonctionne pas pour votre machine, vous pourrez

peut-etre utiliser la combinaison de touches **Fn+F12 à la place.**

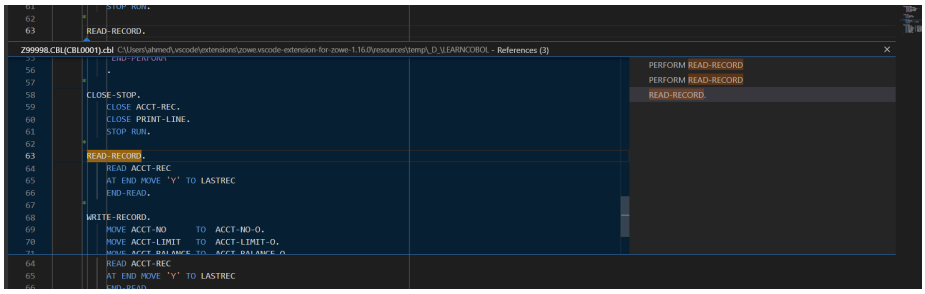


Figure 6. Recherche de références à paragraphe/variable dans VSCode

3.3 Complétion de code

La complétion de code n'est pas exactement un nouveau concept dans la plupart des IDE. Par exemple, l'éditeur Eclipse fourni depuis longtemps l'auto-complétion pour les API Java. La meme combinaison de touches, **CTRL+ESPACE** , déclenche cette fonction d'auto-complétion pendant que vous codez et peut être utilisée pour vous guider à travers la syntaxe COBOL et les appels CICS, IMS API.

3.3.1 Complétion des mots réservés COBOL

Lorsque vous taper un mot réserverve COBOL, vous pouvez utiliser « CTRL+ESPACE » et IBM Z Open Editor présentera, dans une fenêtre contextuelle, une liste de mots réservés COBOL possibles que vous pouvez essayer d'utiliser. L'utilisation des touches du curseur ou de la souris vous permet de sélectionner le mot-clé correct / recommandé en appuyant sur Enter pour le sélecteur et le reste du mot réservé sera entré pour vous, c'est-à-dire complété automatiquement ! [2] ### Complétion de Variable La meme technique peut être appliquée à la complétion de variable. Cela peut être particulièrement utile lorsque vous référencez une variable qui existe plusieurs fois dans différentes structures. Dans ces cas, la saisie semi-automatique peut vous aider à identifier la variable que vous souhaitez utiliser. Par exemple, créez une nouvelle ligne dans le paragraphe WRITE-RECORD. Sur la nouvelle ligne, entrez le code « MOVE ACCT-BA », puis appuyez sur **CTRL+ESPACE** pour appeler la saisie semi-automatique du code. Vous devriez voir une fenêtre contextuelle similaire à celle illustrée à la figure 7 ci-dessous.

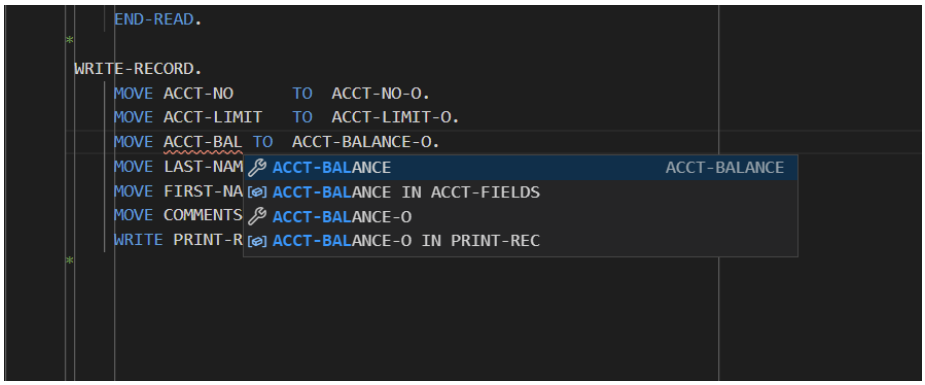


Figure 7. Auto-complétion dans VSCode

Vous pouvez voir que non seulement la variable ACCT-BALANCE est préssentie comme candidate potentielle, mais qu'elle présente également ACCT_BALANCE IN ACCT-FIELDS.

3.3.2 Complétion des API CICS, MQ, DB2

La saisie semi-automatique des variables s'étend également aux API CICS et DB2, appelées instructions EXEC CICS et EXEC SQL. Bien que la programmation COBOL pour DB2 et CICS ne soit pas un objectif principal ici, notez que si vous vous retrouvez à programmer pour l'une de ces API, cette fonctionnalité est disponible.

3.4 Refactorisation du Code

Travailler avec le code source se limite rarement à la création initiale, au cours du cycle de vie d'un programme, il sera modifié et retravaillé, nous appelons souvent ce travail refactoring. Cette section explore le renommage des variables et la gestion des erreurs.

3.4.1 Renommer des variables

Lors de la maintenance du code existant, vous devrez peut-etre refactoriser les noms de variables ou les noms de paragraphe. Faire cela manuellement peut être un processus pénible, car vous devez probablement mettre à jour à la fois la déclaration d'origine et toutes les références dans le code source. Heureusement, il existe une fonction pour cela, travaillons sur un exemple. De retour dans CBL0001, appuyez sur **CTRL+G** pour afficher la fonction aller à la ligne et aller à la ligne 40. Il s'agit de la déclaration de la variable ACCT-NO. Faites un clic droit sur la variable et sélectionnez "**Find All References**". De là, nous pouvons voir qu'en dehors de la déclaration, la variable est également référencée à la ligne 88. Donc, si nous renommons la variable, nous devrons probablement également mettre à jour cette référence. Pour effectuer le changement de nom, assurez-vous

que le curseur est sur la variable, puis appuyez sur **SHIFT/Fn+F2** . Cela fera apparaître une petite fenêtre contextuelle vous demandant de fournir un nouveau nom de variable, comme illustré à la Figure 8. Entrez ACCT−NO−TEST et appuyez sur **enter** .

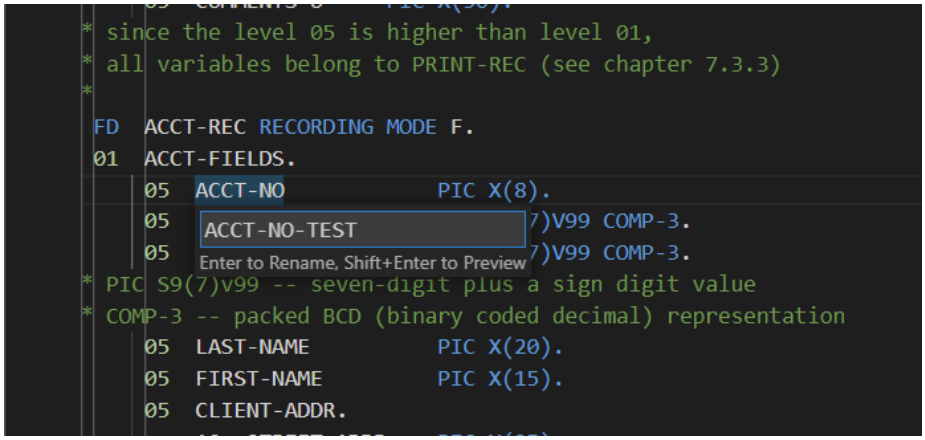


Figure 8. Renommer des variables

Vous remarquerez que la déclaration de la variable et la référence à la ligne 88 ont été mises à jour avec la nouvelle valeur. Comme indiqué précédemment, le même processus fonctionne également pour les noms de paragraphe. Par exemple, allez-vers et refactorisez le nom du paragraphe READ-RECORD pour qu'il soit READ-NEW-RECORD.

3.4.2 Gestion des erreurs

Le plug-in IBM Z Open Editor fournit également un niveau de vérification de la syntaxe du code source local. Bien qu'il ne soit pas aussi complet que le compilateur, il s'agit d'une méthode permettant d'identifier rapidement les erreurs de base dans votre code avant de le soumettre à la compilation. Pour le montrer, créons une erreur et voyons ensuite comment l'éditeur nous la montre. Tout d'abord, ouvrez la vue des problèmes en sélectionnant **View** puis **Problems** dans le menu de l'éditeur. La vue des problèmes doit s'ouvrir en bas de la fenêtre, comme illustré à la figure 9.

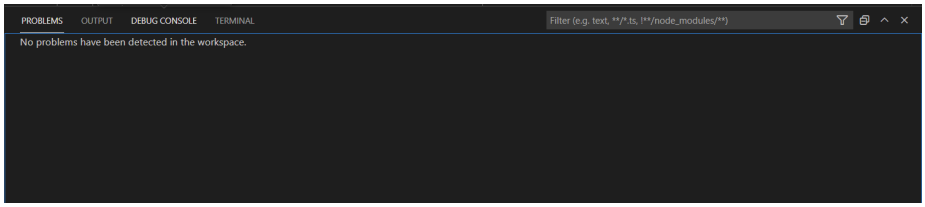


Figure 9. Vue Problems

Maintenant, nous devons introduire une erreur dans le code. Après la ligne 87,

ajoutez la ligne:

MOVE ACCT-NO TO ACCT-NO-NO.

Notez que cette ligne identifie de manière incorrecte la deuxième variable, qui n'existe pas. Une fois ligne saisie, vous remarquerez que la variable invalide a été soulignée en rouge pour la mettre en évidence en tant qu'erreur. De plus, la vue des problèmes a détectée une nouvelle erreur. Cliquer sur l'erreur mettra en surbrillance la ligne de code fautive dans l'éditeur, illustré à la Figure 10. , vous permettant de visualiser directement l'erreur.

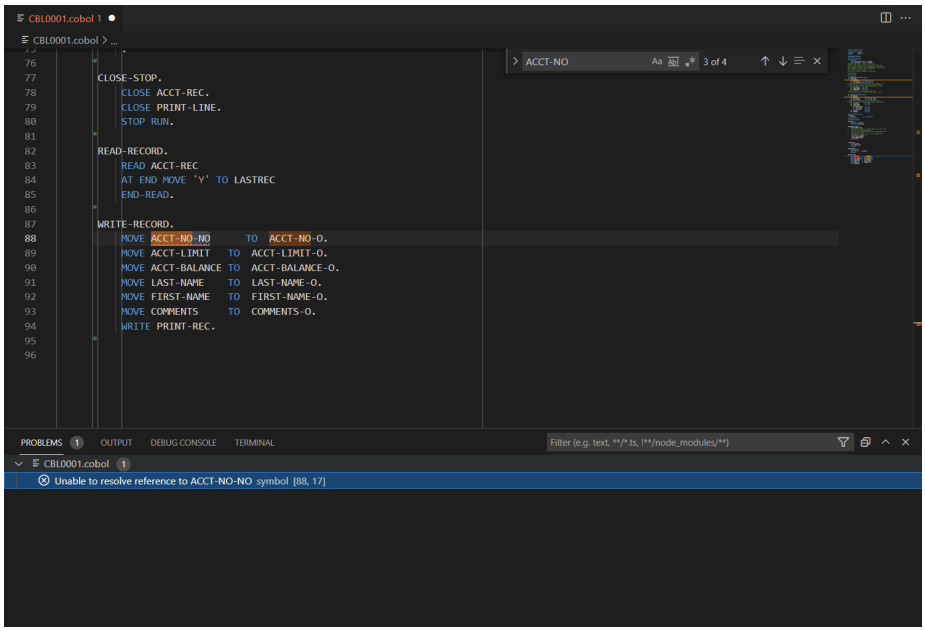


Figure 10. Erreur de surbrillance dans le code source

Maintenant que vous voyez où se trouve l'erreur, elle peut maintenant être corrigée. Dès que l'erreur a été corrigée, le problème disparaît de la vue des problèmes.

3.5 Sommaire

Dans ce chapitre, vous avez pu découvrir certaines des fonctionnalités d'édition de Z Open Editor for VSCode. Ces fonctionnalités rendent l'édition COBOL, PL/I et JCL beaucoup plus conviviale et plus facile qu'avec certains des autres éditeurs du marché.

4 VS Code avec le package d'extension Open-Source Code4z

Cette section présente le package d'extension Code4z, en particulier l'extension COBOL Language Support.

- **Qu'est-ce que Code4z ?**
- **Extensions de fichiers connues**
- **Surlignage et coloration syntaxique**
- **Controle syntaxique et sémantique**
- **Navigation du Code**
  - **Go To à la définition**
  - **Trouvez toutes les références**
- **Support des Copybook**
- **Auto-complétion**
- **Sommaire**

4.1 Qu'est-ce que Code4z ?

Code4z est un package d'extension mainframe open source tout-en-un pour Visual Studio Code. Le package Code4z contient des extensions qui fournissent une prise en charge linguistique pour le langage COBOL et le High Level Assembler, un débogueur pour les programmes COBOL, ainsi que des outils permettant aux développeurs d'accéder aux ensembles de données mainframe et aux référentiels de code CA Endevor à l'aide de l'interface Visual Studio Code. Ce guide se concentre sur l'extension COBOL Language Support. L'extension Zowe Explorer est également incluse dans le package Code4z.

L'extension COBOL Language Support exploite le Language Server Protocol pour fournir des fonctionnalités de saisie semi-automatique, de surbrillance et de diagnostic pour le code COBOL. Avec Zowe Explorer, vous pouvez charger du code COBOL à partir d'un ensemble de données mainframe et le modifier en tirant parti des fonctionnalités LSP de l'extension. Une fois les modifications terminées, vous pouvez enregistrer le fichier sur l'ordinateur central et en stocker une copie localement.

Le pack d'extension Code4z peut être installé dans VS Code en recherchant "Code4z" sur le marché des extensions dans VS Code et en sélectionnant installer. Le pack d'extensions contient un certain nombre d'extensions qui peuvent être exploitées lors de l'utilisation du mainframe, y compris l'extension COBOL Language Support qui fournit des fonctionnalités similaires à l'extension Z Open Editor évoquée précédemment. Par conséquent, assurez-vous qu'une seule de ces deux extensions est activée. Les autres extensions incluses dans le pack fonctionneront soit avec COBOL Language Support, soit avec Z Open Editor. Pour voir des instructions plus détaillées sur l'installation de cette extension, reportez-vous à "Installation de VSCode et des extensions".

4.2 Extensions de fichiers connues

Code4z reconnaît les fichiers avec les extensions .COB et .CBL en tant que fichiers COBOL. Cela s'applique à la fois aux fichiers locaux et aux fichiers contenus dans un PDS sur l'ordinateur central. Les fonctionnalités de prise en charge du langage COBOL sont automatiquement activées lorsque vous ouvrez un fichier avec une extension l'identifiant comme un fichier COBOL.

4.3 Mise en évidence et coloration de la syntaxe

L'extension COBOL Language Support permet de colorer les mots-clés, les paragraphes et les variables dans différentes couleurs pour faciliter la navigation dans le code.

4.4 Vérification de la syntaxe et de la sémantique

L'extension COBOL Language Support vérifie les erreurs et les erreurs dans le code COBOL. La fonction de vérification de la syntaxe examine l'ensemble du contenu du code, met en évidence les erreurs et suggère des correctifs.

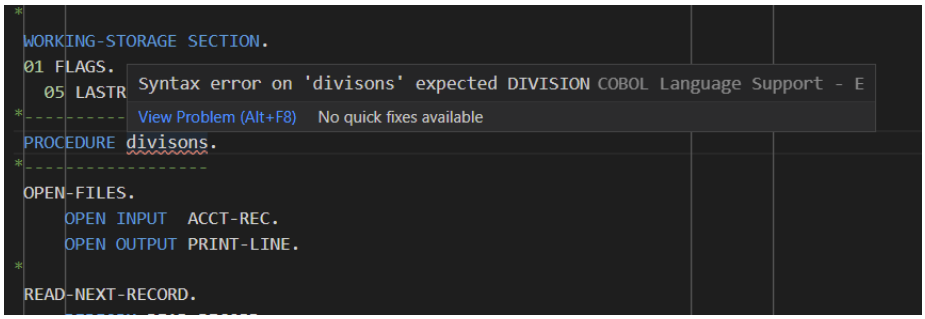


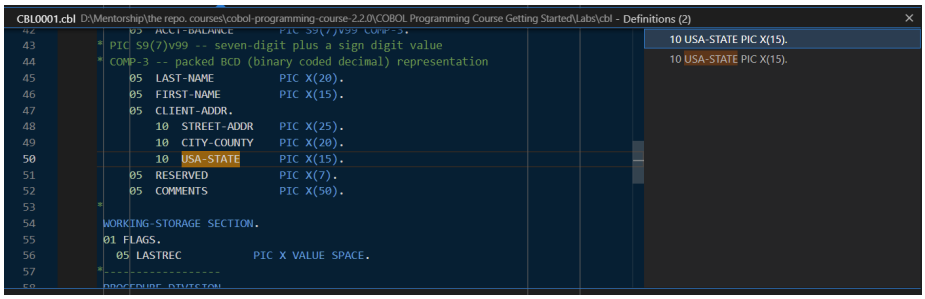
Figure 1. La fonction de vérification syntaxique et sémantique met en évidence une erreur.

4.5 Navigation du code

L'extension COBOL Language Support permet plusieurs fonctionnalités pour faciliter la navigation dans le code.

4.5.1 Aller à la définition

Pendant que votre curseur est placé sur une variable ou un nom de paragraphe, vous pouvez appuyer sur **F12** ou **CTRL+clic** pour utiliser la fonctionnalité **Aller à la définition** pour afficher le point dans le code où la variable ou paragraphe est défini.



La figure 2. Aller à la définition montre le point auquel la variable USA-STATE est définie pour la première fois.

4.5.2 Trouver toutes les références

La fonctionnalité **Rechercher toutes les références** (**MAJ+ALT+F12**) met en évidence toutes les références à une variable ou à un paragraphe et les affiche dans une liste dans la barre latérale, afin que vous puissiez facilement naviguer entre elles.

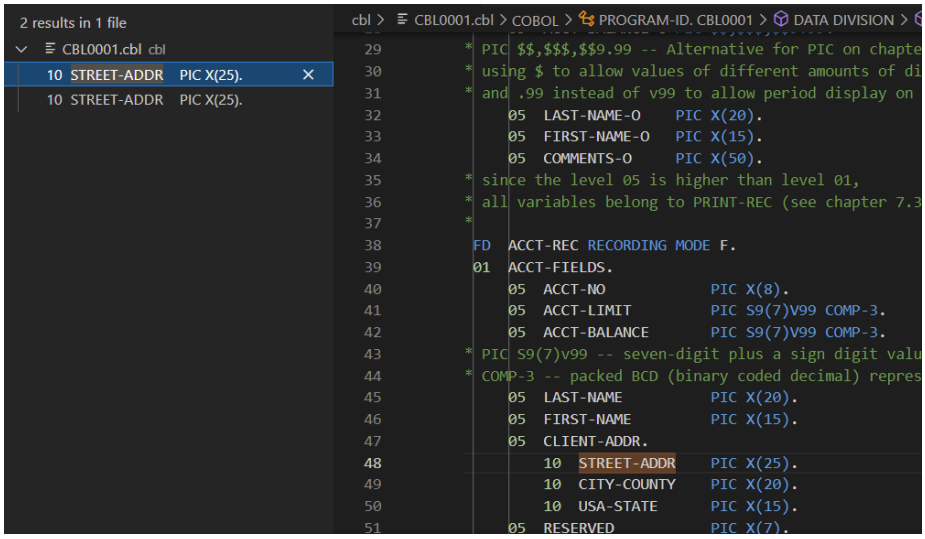


Figure 3. Find All References répertorie toutes les références à la variable STREET-ADDR dans le code.

4.6 Prise en charge du cahier

Les copybooks sont des morceaux de code source stockés dans des ensembles de données séparés qui sont référencés dans un programme. L'extension COBOL Language Support vous permet de télécharger tous les copybooks référencés dans votre programme depuis l'ordinateur central vers un dossier de votre espace de travail. Pour que cette fonctionnalité fonctionne, vous devez installer et

configurer un profil Zowe CLI zosmf. Vous pouvez également activer la prise en charge des cahiers stockés localement dans des dossiers de votre espace de travail. Ceci est utile lorsque vous travaillez avec un projet COBOL stocké dans un référentiel Github.

L'extension COBOL Language Support permet de garantir que les copybooks appelés dans le code restent compatibles grâce à l'analyse sémantique des mots-clés, des variables et des paragraphes dans les copybooks, et assure la cohérence du code en définissant des variables et des paragraphes à travers les copybooks. L'extension aide également à protéger contre les erreurs indésirables causées par des cahiers récursifs ou manquants.

Les fonctionnalités **Aller à la définition** et **Rechercher toutes les références** sont étendues pour fonctionner pour les occurrences de variables et de paragraphes provenant de cahiers appelés dans le programme ainsi que du programme lui-meme. Vous pouvez également utiliser la fonction **Aller à la définition** sur un nom de cahier afin de l'ouvrir.

4.7 Saisie automatique

L'extension COBOL Language Support fournit des suggestions en direct pendant que vous tapez des mots-clés COBOL, ainsi que des variables et des paragraphes qui sont déjà référencés dans le code ou dans les cahiers utilisés par le programme.

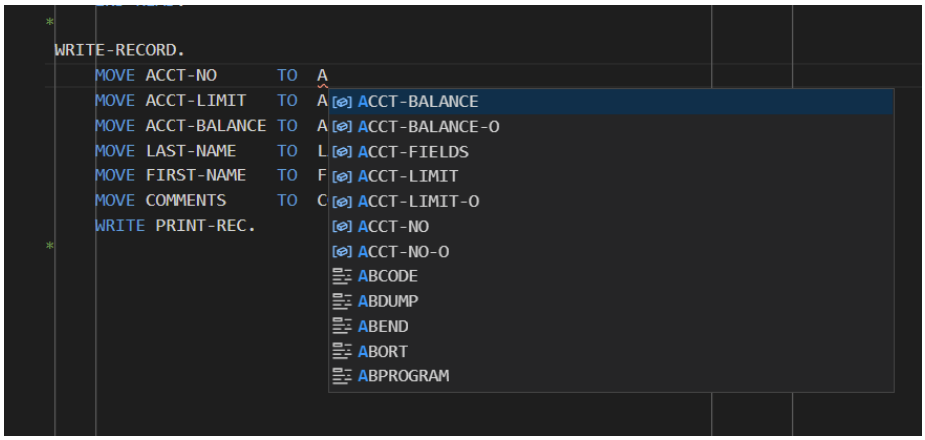


Figure 4. La saisie semi-automatique répertorie les variables et les mots-clés possibles commencant par la chaîne saisie dans une liste.

4.8 Sommaire

Dans ce chapitre, vous avez été présenté à toutes les fonctionnalités de prise en charge du langage COBOL du package Code4z d'extensions open source pour VS Code.

5 Zowe CLI et plug-ins Zowe CLI

Dans ce chapitre, nous expliquerons ce qu'est une CLI et pourquoi vous l'utiliseriez, comment utiliser la CLI Zowe de manière interactive, comment faire abstraction des commandes CLI dans des scripts utiles et comment la CLI Zowe permet d'utiliser des outils open source familiers tout en développant des applications COBOL sur l'ordinateur central.

- **Qu'est-ce qu'une CLI et pourquoi l'utiliseriez-vous ?**
- **Qu'est-ce que Zowe CLI ?**
- **Utilisation interactive de la CLI Zowe - Installation de Zowe CLI - Aide interactive - Profils Zowe - Interaction avec les ensembles de données z/OS - Interaction avec les travaux z/OS**
- **Automatisation des tâches à l'aide de Zowe CLI - Soumission automatisée des travaux - Utilisation d'autres langages de programmation et intégration continue - Exemples supplémentaires**
- **Le monde des outils open source modernes**
- **Sommaire** ## Qu'est-ce qu'une CLI et pourquoi l'utiliseriez-vous ?

CLI signifie Command Line Interface. C'est un programme qui permet l'interaction de l'utilisateur via une entrée basée sur du texte. Au début de l'informatique, les interfaces de ligne de commande étaient le seul moyen d'interagir avec les systèmes d'exploitation. L'invention de la souris et le développement d'interfaces utilisateur graphiques ont conduit à l'expérience que nous connaissons aujourd'hui. Des interfaces graphiques bien conçues offrent certainement une expérience interactive améliorée. Cependant, les CLI sont encore très utilisées aujourd'hui et sont très puissantes. Le shell Windows et le bash sont des exemples courants de terminaux où les outils de ligne de commande sont exécutés de manière interactive.

Si des interfaces graphiques bien conçues offrent une expérience interactive améliorée, pourquoi utiliseriez-vous une CLI ? En termes simples, l'automatisation. Les interfaces de ligne de commande peuvent être utilisées de manière interactive, ce qui permet une exploration rapide des commandes disponibles. Ils sont également généralement autoguidés et certains offrent meme des affichages d'aide modernes en lançant du contenu dans un navigateur. Mais, ce sont aussi des interfaces de programmation où les séquences de commandes et les tâches peuvent être facilement résumées dans des scripts.

5.1 Qu'est-ce que Zowe CLI ?

Zowe CLI est une CLI open source pour le mainframe. C'est un outil qui peut être exécuté sur Windows, Linux et Mac offrant un moyen d'interagir avec le mainframe à partir d'un environnement où des outils open source modernes sont disponibles. Les plateformes cloud comme Amazon Web Services, Azure et

Google Cloud Platform fournissent toutes des CLI très utilisées. La CLI Zowe permet d'interagir avec le mainframe comme d'interagir avec d'autres services cloud.

À la base, Zowe CLI fournit une interaction à distance avec les ensembles de données et les travaux z/OS, les fichiers Unix System Services, les commandes TSO et Console et les services de provisionnement. Zowe CLI est également une technologie extensible et de nombreux plug-ins existent qui étendent sa portée aux sous-systèmes z/OS et aux logiciels des fournisseurs.

Zowe CLI est un outil de passerelle entre les systèmes distribués et le mainframe. Choisissez votre langage ou outil open source préféré et utilisez-le pour le développement mainframe avec l'aide de la CLI Zowe. Vous souhaitez développer l'automatisation en Python ? Vous voulez écrire des tests dans Node ? Vous souhaitez exécuter des pipelines Jenkins pour une intégration continue ? Vous voulez utiliser des frameworks de test open source comme Mocha ou Facebook's Jest ? Vous voulez tirer parti d'outils de qualité de code comme SonarQube ? Fonce!

Les CLI sont utiles pour automatiser les tâches répétées. Pour les applications COBOL mainframe, Zowe CLI peut vous aider à automatiser vos processus de création, de déploiement et de test. Consultez ce blog pour plus d'informations et l'exemple de code qui a rendu cela possible ! Zowe CLI peut également vous aider à automatiser les tâches administratives.

La plupart des IDE ont également des terminaux intégrés afin que la CLI puisse être exploitée à partir de votre environnement de développement distribué préféré, y compris VS Code !

5.2 Utilisation interactive de la CLI Zowe

La CLI Zowe peut être utilisée pour une exploration rapide des services z/OS ainsi que pour émettre des commandes qui ne sont pas encore disponibles dans l'IDE de votre choix. Avant de développer l'automatisation, il est courant d'accomplir d'abord une tâche fréquemment répétée à partir de la CLI de manière interactive.

5.2.1 Installation de l'interface de ligne de commande Zowe

La CLI Zowe est un package de nœuds et fait partie des plus de 1,2 million de packages de nœuds disponibles sur le registre public npm. Une fois Node.js et npm installés sur la machine cliente, la CLI principale peut être installée en émettant simplement « npm install -g @zowe/cli@zowe-v1-lts ». Il existe une méthode d'installation alternative si votre poste de travail n'a pas accès au registre public. Plus de détails sur l'installation de Zowe CLI et des plug-ins Zowe CLI sont fournis dans une future section intitulée « Installation de Zowe CLI et des plug-ins ».

5.2.2 Aide interactive

Pour commencer, vous pouvez simplement ouvrir un terminal et émettre zowe. Cela donnera l'aide de haut niveau.

```
DESCRIPTION
-----------

  Welcome to Zowe CLI!

  Zowe CLI is a command line interface (CLI) that provides a simple and
  streamlined way to interact with IBM z/OS.

  For additional Zowe CLI documentation, visit https://docs.zowe.org

  For Zowe CLI support, visit https://www.zowe.org


USAGE
-----

  zowe <group>

  Where <group> is one of the following:

GROUPS
------

  auth                          Connect to Zowe API ML authentication service
  config                        Manage configuration and overrides
  plugins                       Install and manage plug-ins
  profiles                      Create and manage configuration profiles
  provisioning | pv             Perform z/OSMF provisioning tasks
  zos-console | console         Issue z/OS console commands and collect
                                responses
  zos-files | files             Manage z/OS data sets
  zos-jobs | jobs               Manage z/OS jobs
  zos-ssh | ssh | zos-uss | uss Issue z/OS USS commands and receive responses
  zos-tso | tso                 Interact with TSO
  zos-workflows | wf            Create and manage z/OSMF workflows
  zosmf                         Interact with z/OSMF

OPTIONS
-------

  --version  | -V (boolean)
```

Figure 1. Aide de Zowe CLI Dans l'exemple ci-dessus, plusieurs extensions sont installées. La structure des commandes est zowe <group> <action> <object> suivie de divers paramètres et options spécifiques à la commande. Par exemple, une commande valide est zowe files list data−set "HLQ.∗". Cette commande listera les ensembles de données correspondant à un modèle de "HLQ.*". Vous pouvez ajouter −h à n'importe quelle commande pour obtenir plus d'informations. Se référer fréquemment à l'aide peut être difficile et prendre du temps, donc si votre environnement a accès à un navigateur Web, ajoutez simplement −−help−web ou −−hw à n'importe quelle commande pour lancer l'aide Web interactive.

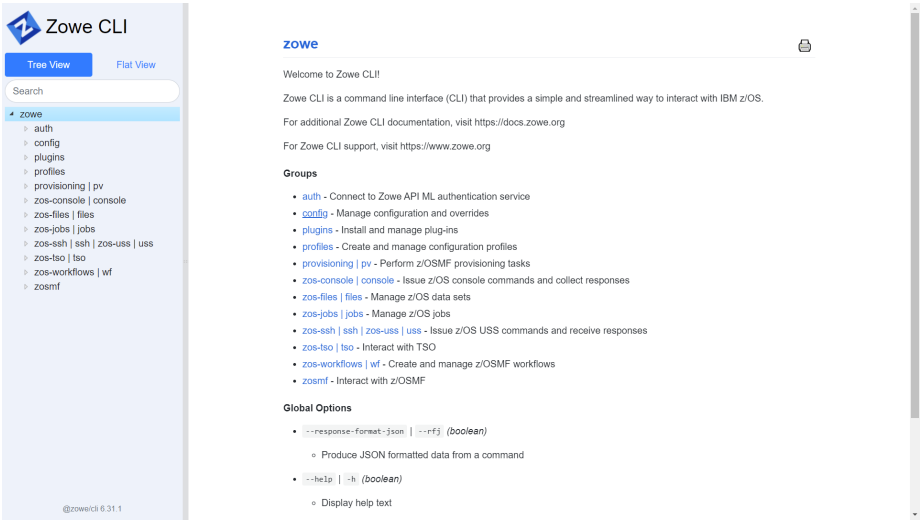


Figure 2. Aide Web Zowe CLI

Vous n'avez pas encore installé la CLI ? Vous pouvez également consulter une copie de l'aide Web pour les plug-ins Zowe CLI et Zowe principaux [ici] (https://docs.zowe.org/stable/web_help/index.html).

5.2.3 Profils Zowe

Les technologies clientes Zowe telles que Zowe CLI et l'extension de code Zowe Explorer VS stockent les informations de connexion dans des fichiers communément appelés profils. Cela est pratique car une fois les profils de services créés, les utilisateurs n'ont pas à fournir constamment ces informations. Pour le stockage sécurisé des informations d'identification, il existe le plug-in Secure Credential Store qui est abordé plus en détail dans une section ultérieure intitulée « Installation de Zowe CLI et des plug-ins ». Le magasin d'informations d'identification sécurisé fournit un moyen de stocker des informations d'identification dans le coffre-fort d'informations d'identification sécurisé du système d'exploitation.

Lors de la création de profils, vous pouvez également spécifier le mot-clé « prompt* » pour que votre nom d'utilisateur et votre mot de passe soient demandés afin qu'ils soient masqués sur la ligne de commande. La figure 3 montre un exemple de commande pour créer un profil zosmf. Cela éliminera le besoin de fournir ces détails sur les commandes futures.

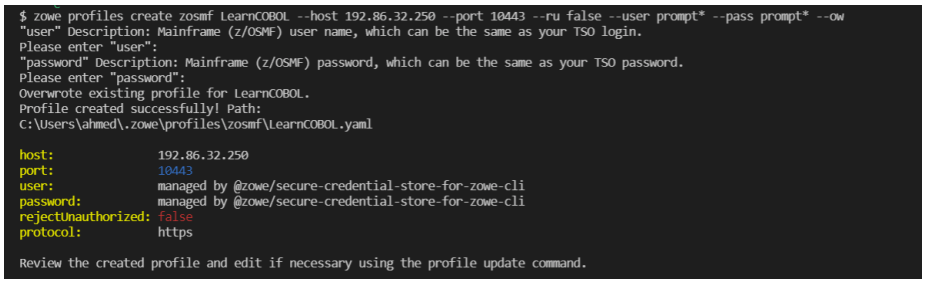


Figure 3. Commande de création de profil Zowe CLI z/OSMF

5.2.4 Interaction avec les ensembles de données z/OS

Zowe CLI fournit une suite importante de fonctionnalités d'interaction d'ensembles de données z/OS. Voir les figures suivantes pour plus de détails sur les actions disponibles et un exemple de commande de liste.

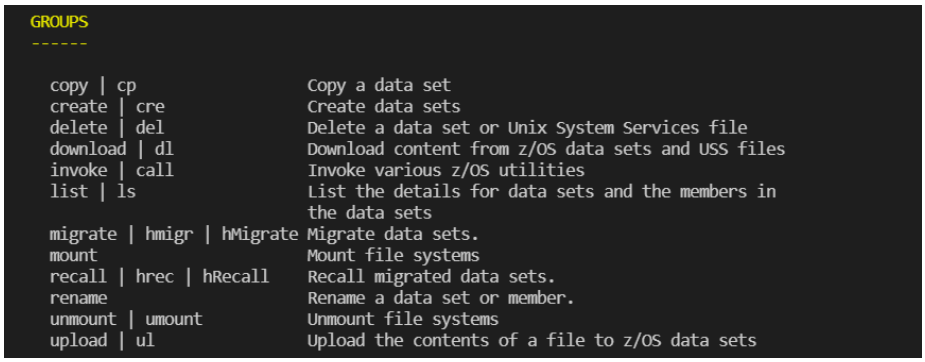


Figure 4. Actions Zowe CLI Zos-files

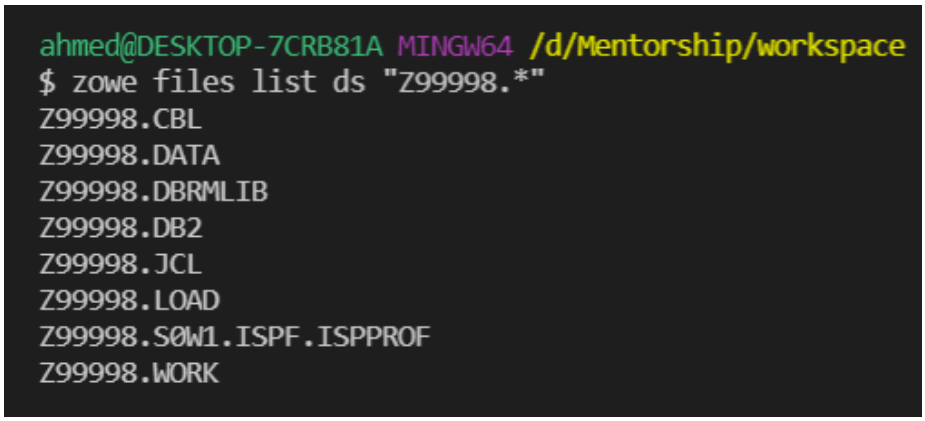


Figure 5. Exemple de commande Zowe CLI zos-files list ds

5.2.5 Interaction avec les travaux z/OS

Zowe CLI fournit une suite importante de fonctionnalités d'interaction de tâches z/OS. Consultez les figures suivantes pour plus de détails sur les actions disponibles et un exemple de commande de soumission de tâche.

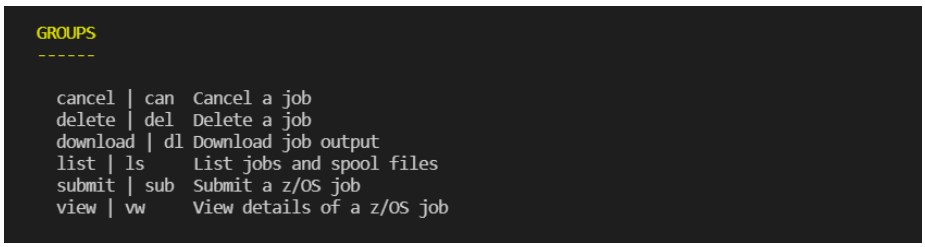


Figure 6. Actions Zowe CLI Zos-jobs

Figure 7. Exemple de commande Zowe CLI zos-jobs submit ds

5.3 Automatisation des tâches à l'aide de Zowe CLI

L'exécution de commandes de manière interactive est un excellent moyen d'apprendre les capacités de l'interface de ligne de commande Zowe. Cependant, la création d'une automatisation personnalisée pour vos tâches fréquemment répétées et l'utilisation d'outils de développement précieux constituent une valeur importante. Pour le développement COBOL, un temps considérable peut être consacré à l'examen de la sortie du compilateur et au test des programmes. Ces tâches répétitives sont d'excellents candidats pour l'automatisation.

5.3.1 Soumission automatisée des tâches

Examinons l'automatisation de la soumission d'un travail et la vérification que le code de retour est 0. Bien sur, nous pourrions également analyser la sortie du spool pour des messages d'intérêt spécifiques, mais nous allons rester simples pour le moment. Pour cet exemple, nous utiliserons Node.js pour développer une nouvelle suite d'automatisation. Pour commencer, je vais créer un fichier package.json pour faciliter la gestion et l'installation du projet par les autres. Il contiendra la liste des dépendances de mon projet ainsi que les tâches d'automatisation que je développerai. Un moyen rapide de créer un package.json consiste à lancer npm init et à répondre aux invites. Une fois

créé, j'ajouterai une tâche submitJob. Vous pouvez ajouter n'importe quelle automatisation que vous voulez ici. Mon package.json final est illustré dans la figure suivante. Vous pouvez en savoir plus sur les fichiers package.json [ici] (https://docs.npmjs.com/creating-a-package-json-file).

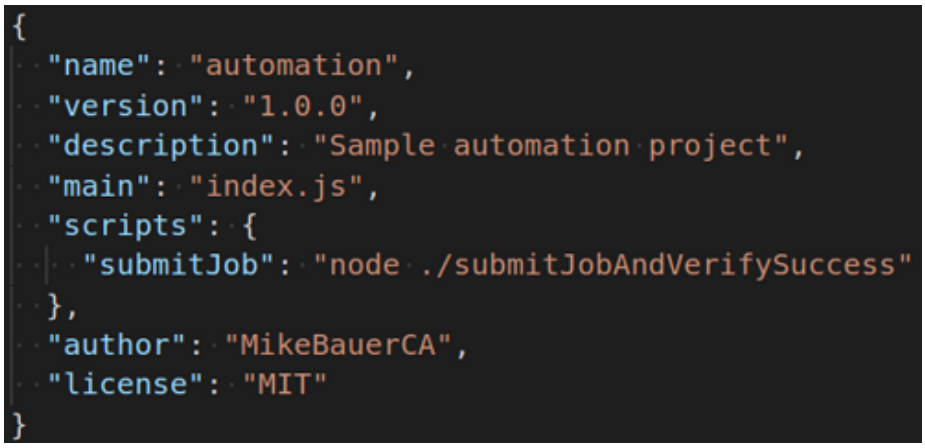


Figure 8. Exemple de package.json

Ensuite, je vais créer un fichier config.json pour stocker toutes les variables que je souhaite modifier pour mon projet. Dans ce cas, nous définirons le travail à soumettre et le code de retour maximum autorisé pour ce travail.

Figure 9. Exemple de config.json

Ensuite, nous allons écrire notre automatisation. La CLI Zowe a été conçue en pensant aux scripts et peut générer des réponses au format JSON qui peuvent être facilement analysées.

Figure 10. Exemple de sortie JSON au format de réponse Zowe CLI

Maintenant, au lieu d'émettre cette commande et de l'examiner pour voir si le retcode est inférieur ou égal à 4, nous voulons l'automatiser. Voir l'implémentation dans un script de nœud ci-dessous.

```
const { exec } = require('child_process'),
    config = require('./config.json');

exec('zowe jobs submit data-set "' + config.job + '" --wait-for-output --rfj', (err, stdout, stderr) => {
  if (err) {
    //some err occurred
    console.error(err)
  } else if (stderr){
    console.log(new Error("\nCommand:\n" + command + "\n" + "stderr:\n" + stderr));
  } else {
    data = JSON.parse(stdout).data;
    retcode = data.retcode;

    //retcode should be in the form CC nnnn where nnnn is the return code
    if (retcode.split(" ")[1] <= config.maxRC) {
      console.log("Job completed successfully");
    } else {
      console.log(new Error("Job did not complete successfully. Additional diagnostics:" + JSON.stringify(data,null,1)));
    }
  }
});
```

Figure 11. Exemple de code pour soumettre le travail et vérifier que la sortie est inférieure ou égale à un RC maximum autorisé J'ai dû investir pour écrire cette automatisation, mais pour les futures soumissions de travaux, je peux simplement émettre "npm run submitJob". Les IDE comme VS Code peuvent visualiser ces tâches, rendant mes tâches fréquemment répétées aussi simples que de cliquer sur un bouton :). Ce travail pourrait compiler, lier et/ou exécuter un programme COBOL.

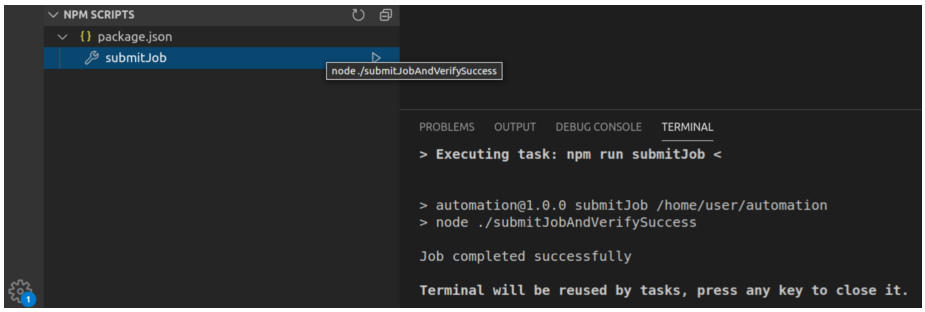


Figure 12. Visualisation du script npm et exécution de l'échantillon

Un code plus avancé automatisant la compilation, le déploiement dans l'environnement de test et le test d'une application COBOL CICS est décrit dans ce blog.

5.3.2 Utilisation d'autres langages de programmation et intégration continue

Un autre bon exemple d'automatisation des tâches à l'aide de Zowe CLI est lorsque vous souhaitez intégrer d'autres langages de programmation dans votre développement COBOL. Semblable à 3.4.1, vous pouvez utiliser d'autres langages tels que Typescript pour écrire un générateur de programme COBOL et utiliser Zowe CLI pour créer un processus "en un clic" pour créer votre programme. La figure ci-dessous est une représentation de ce processus automatisé "en un clic" où plusieurs tâches sont exécutées telles que la création de votre programme COBOL, son téléchargement dans le mainframe, sa compilation et l'exécution de votre JCL pour le tester.

```
> template@1.0.0 build C:\Users\jp669971\Documents\GitHub\Community\Cobol-Made-Easy
> npm run generate && npm run upload && npm run compile:cobol && npm run run:job


> template@1.0.0 generate C:\Users\jp669971\Documents\GitHub\Community\Cobol-Made-Easy
> node ./scripts/lib/append && node ./scripts/lib/generate

Created COBOL Template
Added Process
Added Closing
Generated custom JCL to ./build/compiler.jcl
Generated COBOL pgm to ./build/source.cbl
Generated run job to ./build/runjob.jcl

> template@1.0.0 upload C:\Users\jp669971\Documents\GitHub\Community\Cobol-Made-Easy
> zowe zos-files upload file-to-data-set "./build/source.cbl" "Z40607.SOURCE(CBLSRC)" --zosmf-p ztrial

success: true
from:    C:\Users\jp669971\Documents\GitHub\Community\Cobol-Made-Easy\build\source.cbl
to:      Z40607.SOURCE(CBLSRC)


file_to_upload: 1
success:        1
error:          0
skipped:        0


Data set uploaded successfully.

> template@1.0.0 compile:cobol C:\Users\jp669971\Documents\GitHub\Community\Cobol-Made-Easy
> zowe jobs submit lf "./build/compiler.jcl" --directory "./output" --zosmf-p ztrial

  ___________| 30%  0 | Waiting for JOB07745 to enter OUTPUT
```

Figure 13. Processus de construction COBOL « One Click »

Vous pouvez ensuite améliorer ce processus en tirant parti d'un pipeline CI/CD. Qu'est-ce qu'un pipeline CI/CD ? Il s'agit d'un moyen automatisé de créer, de tester et de déployer votre application et vous pouvez faire de même avec votre développement COBOL. La figure ci-dessous montre le pipeline pour les memes tâches automatisées que nous avons effectuées précédemment.

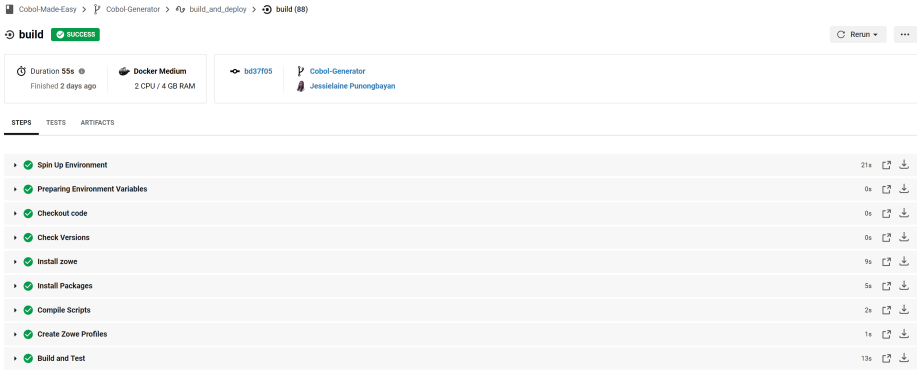


Figure 14. Pipeline CI/CD du processus de construction COBOL "en un clic"

Pour en savoir plus sur ce sujet, consultez this.

5.3.3 Exemples supplémentaires

Si vous cherchez un exemple sur la façon d'utiliser Zowe Explorer et Zowe CLI avec les procédures stockées Db2, consultez ce blog.

Si vous souhaitez utiliser des outils open source dans votre développement, vous pouvez consulter ce blog où il est question d'utiliser Zowe CLI pour tirer parti des outils d'analyse de code statique lors du développement d'applications COBOL.

Pour d'autres blogs et articles sur l'utilisation des technologies Zowe, consultez https://medium.com/zowe.

5.4 Le monde des outils open source modernes

Nous n'avons fait qu'effleurer la surface de l'utilisation d'outils et de langages modernes pour le développement mainframe et l'incorporation d'applications mainframe dans les pipelines DevOps d'entreprise. En tant qu'outil de passerelle, la CLI Zowe permet l'utilisation d'une pléthore d'outils développés par une énorme communauté pour le développement mainframe. Si vous êtes nouveau sur le mainframe, j'espère que cela vous offrira une certaine familiarité lors de votre transition vers cet espace. Si vous êtes un mainframer expérimenté, j'espère que vous trouverez le temps d'essayer certaines de ces technologies disponibles pour voir si elles peuvent vous aider.

5.5 Sommaire

As both a user and programmatic interface, command line interfaces offer significant value in simplifying complex repeatable processes into single tasks. CLIs are commonly used when developing on popular cloud platforms like Amazon Web Services. The Zowe CLI is the CLI for the mainframe that has been extended via numerous plug-ins. Zowe CLI acts as a bridge tool enabling the use of distributed development tooling while working with mainframe applications. Numerous resources and articles are available for using Zowe CLI to create custom automation, build CI pipelines, and incorporate static analysis into your COBOL development processes. Development tooling created by the distributed open source community can now be effectively leveraged for mainframe development.

6 Installation de VSCode et extensions

Ce chapitre couvre tous les aspects du téléchargement et de l'installation du code Visual Studio (VS) et toutes les conditions préalables nécessaires. Il comprend:

- **Installer les prérequis**
  - **Installer node.js**
  - **Installer le SDK Java**
- **Installer VSCode**
- **Installer les extensions VSCode**
  - **Explorateur Zowe**
  - **Éditeur ouvert IBM Z**
  - **Code4z**
- **Sommaire**

6.1 Installer les prérequis

Cette section couvrira les étapes et les informations nécessaires pour télécharger et installer les prérequis nécessaires pour les ateliers suivants de ce livre. Ce logiciel est nécessaire pour une ou plusieurs des applications que nous utiliserons dans nos laboratoires tout au long du livre. Les prérequis comprennent :

- Installer node.js
- Installer le SDK Java

6.1.1 Installer node.js

1. Recherchez l'installation de node.js et vérifiez que le numéro de version est v8 ou supérieur.

   Ouvrez la version de l'invite de commande de votre poste de travail (appelée Terminal sous Mac OS X). Une fois l'invite de commande ouverte, utilisez la commande de l'exemple 1. pour vérifier si votre poste de travail dispose actuellement d'une version de node.js installée.

```
C:\Users\User> node -v
V12.16.1
```

Exemple 1. Version Node.js

Si vous ne voyez pas de numéro de version après avoir soumis la commande, vous n'avez pas installé node.js, ou s'il affiche une version inférieure à v8, vous devez continuer à suivre ces instructions. Si vous voyez un numéro de version et qu'il est v8 ou supérieur, vous pouvez passer à la section Installer Java SDK.

2. Si la version de node.js est inférieure à v8, ou si node n'est pas du tout installé.

   La mise a jour de node.js vers le numéro de version approprié est un processus relativement simple car le programme d'installation s'occupe de la plupart des "gros travaux". Tout ce que vous aurez à faire est de visiter le site de téléchargement Node.js, fourni ci-dessous et de suivre les instructions de téléchargement et d'installation pour votre plate-forme de poste de travail spécifique. Faites ce meme processus si vous n'avez pas déjà installé node.js.

https://nodejs.org/en/download/

Ce processus installera les dernières versions de Node.js et du gestionnaire de packages de nœuds (npm) et écrasera tous les fichiers d'anciennes versions de votre système. Cela supprime l'étape de devoir désinstaller manuellement les versions précédentes au préalable.

3. Une fois terminé, vérifiez l'installation et le bon numéro de version, comme indiqué précédemment dans l'exemple 1.

Remarque : Les numéros de version dans nos exemples sont fournis uniquement à titre de référence et peuvent ne pas refléter les dernières versions du logiciel.

6.1.2 Installer le SDK Java

1. Vérifiez l'installation de Java et vérifiez que le numéro de version est v8 ou supérieur.

   Ouvrez la version de l'invite de commande de votre poste de travail, si elle n'est pas déjà ouverte. Une fois l'invite de commande ouverte, utilisez la commande de l'exemple 2. pour vérifier si votre poste de travail dispose actuellement d'une version de Java installée. Java SDK 8 est la version préférée pour ces laboratoires, cependant, toute version supérieure suffira.

```
C:\Utilisateurs\Utilisateur> java -version

version java "1.8.0_241"

Environnement Java(TM) SE (version 1.8.0_241-b07)

Machine virtuelle serveur Java HotSpot(TM) 64 bits (build 25.241-b07, mode mixte
```

Exemple 2. Version Java

Si vous ne voyez pas de numéro de version après avoir soumis la commande, si Java n'est pas installé ou s'il affiche une version inférieure à v8, vous devez continuer à suivre ces instructions. Le format d'affichage du numéro de version pour Java est légèrement différent de celui affiché pour node.js. Avec Java, la deuxième valeur du numéro de version affiché, c'est-à-dire le "8" dans l'exemple

2. , est le numéro de version. Ainsi, notre exemple montre Java SDK version 8. Si vous voyez un numéro de version et qu'il est v8 ou supérieur, vous pouvez passer à la section Installer VSCode.

2. Si votre version de Java affichée est inférieure à v8, vous devez désinstaller la version actuelle sur votre poste de travail et réinstaller la bonne version. Suivez le lien ci-dessous pour désinstaller les instructions qui représentent le système d'exploitation (OS) de votre poste de travail.

-Linux :

https://www.java.com/en/download/help/linux_uninstall.xml -Mac:

https://www.java.com/en/download/help/mac_uninstall_java.xml

-Windows:

https://www.java.com/en/download/help/uninstall_java.xml

3. Une fois Java désinstallé de votre poste de travail, vous pouvez cliquer sur le lien de téléchargement Java JDK 8 ci-dessous et suivre les instructions d'installation pour votre système d'exploitation spécifique.

https://www.oracle.com/java/technologies/javase−jdk8−downloads.html

4. Vérifiez l'installation et le numéro de version approprié comme indiqué dans l'exemple 2.

Remarque : Vous serez invité à enregistrer un nouveau compte Oracle afin de télécharger le fichier d'installation, veuillez le faire. Si vous avez un compte existant, vous pouvez l'utiliser pour vous connecter et continuer.

6.2 Installer VSCode

Si vous n'avez pas déjà installé VSCode sur votre poste de travail, veuillez le faire maintenant en suivant les instructions de téléchargement et d'installation sur le lien ci-dessous :

https://code.visualstudio.com/download

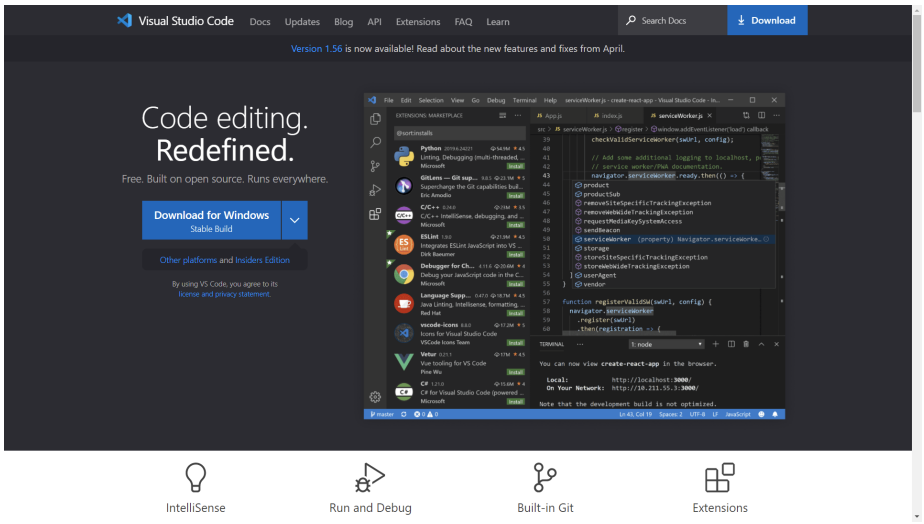


Figure 1. Site de téléchargement VSCode

Remarque : Assurez-vous de sélectionner le fichier d'installation approprié pour le système d'exploitation respectif de votre poste de travail, illustré à la figure 1.

6.3 Installer les extensions VSCode

Cette section présente deux extensions VSCode, Zowe Explorer et IBM Z Open Editor répertoriées dans la figure 2. , et des instructions sur la façon de les installer.

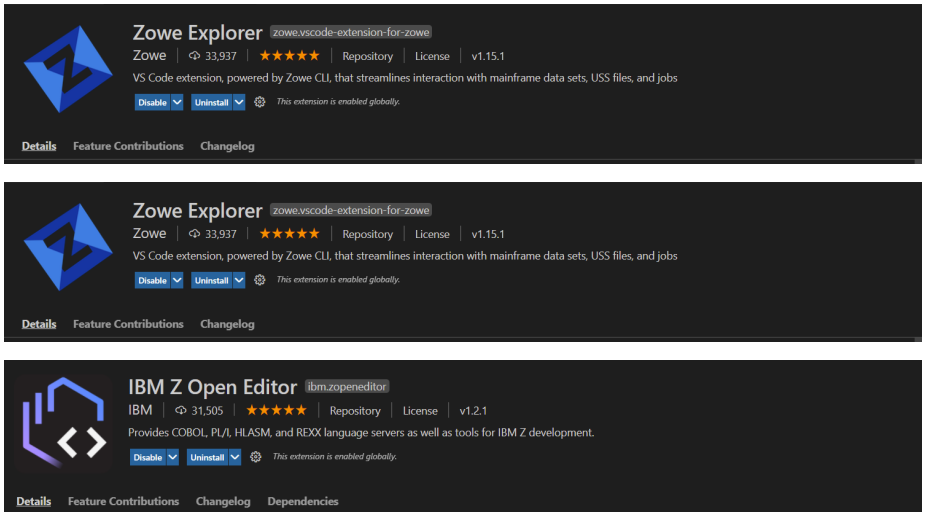


Figure 2. VSCode required extensions

6.3.1 Explorateur Zowe

Zowe est un nouveau et le premier framework open source pour z/OS et fournit des solutions aux équipes de développement et d'exploitation pour gérer, controler, scripter et développer en toute sécurité sur le mainframe comme toute autre plate-forme cloud. Prêt à l'emploi, Zowe Explorer fournit de nombreuses fonctionnalités permettant aux développeurs z/OS d'accéder aux travaux, aux ensembles de données et aux fichiers (USS) sur un serveur z/OS. Soutenus par la CLI Zowe et z/OSMF, les développeurs disposent désormais de fonctionnalités puissantes qui facilitent l'utilisation de z/OS dans l'environnement VSCode familier. Cette extension peut être utilisée pour éditer les fichiers COBOL et PL/I ouverts sur z/OS MVS™ et USS à l'aide des ensembles de données et des vues USS de l'extension Zowe. Il peut meme exécuter JCL et vous permet de parcourir les fichiers spoule des travaux. Pour plus d'informations sur Zowe Explorer et son interaction avec z/OS, veuillez visiter :

https://ibm.github.io/zopeneditor-about/Docs/interact_zos_zowe_explorer.html

6.3.1.1 Installer Zowe Explorer Ouvrez VSCode et dans le menu outil de gauche, sélectionnez **Extensions**. À partir de là, dans le champ de recherche « Rechercher des extensions sur le marché », tapez « Zowe Explorer ». Les résultats de la recherche commenceront à se remplir, sélectionnez **"Zowe Explorer"** et cliquez sur **installer**, illustré à la figure 3.

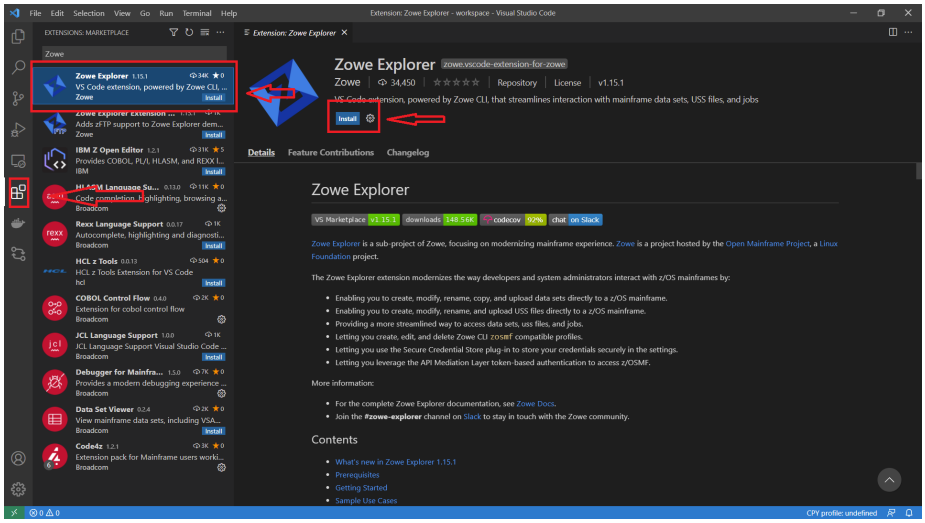


Figure 3. Installer Zowe Explorer dans VSCode

La communauté Zowe propose un certain nombre de vidéos en ligne qui expliquent les étapes requises pour installer, configurer et utiliser Zowe Explorer, voir Zowe Explorer VSC Extension (part 1).

6.3.2 Éditeur ouvert IBM Z

IBM Z Open Editor apporte la prise en charge des langages COBOL et PL/I à Microsoft VSCode. Il s'agit de l'une des nombreuses fonctionnalités de nouvelle génération pour une expérience de développement ouverte pour z/OS®. Il fonctionne également en association avec le plugin Zowe Explorer. Pour plus d'informations sur IBM Z Open Editor, veuillez visiter :

https://ibm.github.io/zopeneditor-about/Docs/introduction.html#key-capabilities

6.3.2.1 Installer IBM Z Open Editor Ouvrez VSCode et dans le menu outil de gauche, sélectionnez **Extensions**. À partir de là, dans le champ de recherche « Rechercher des extensions sur le marché », saisissez « IBM Z Open Editor ». Les résultats de la recherche commenceront à se remplir, sélectionnez **" IBM Z Open Editor "** et cliquez sur **installer**, illustré à la figure 4.

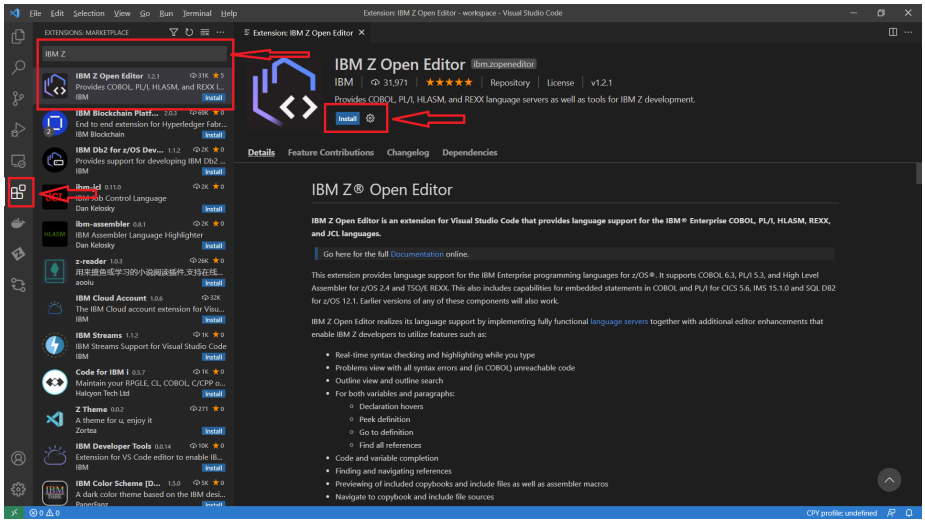


Figure 4. Installer IBM Z Open Editor dans VSCode

Remarque : Il peut y avoir certaines limitations avec IBM Z Open Editor si vous exécutez une version Java 32 bits sous Windows.

6.3.3 Code4z

Code4z est un package d'extension mainframe open source tout-en-un pour les développeurs travaillant avec des applications z/OS, adapté à tous les niveaux d'expérience mainframe, meme les débutants. Les développeurs d'applications mainframe peuvent utiliser le package Code4z pour une expérience moderne, familière et transparente, ce qui permet de surmonter les réserves ou les préoccupations de certains développeurs concernant l'expérience utilisateur

traditionnelle du mainframe. Pour en savoir plus sur Code4z, veuillez visiter https://github.com/BroadcomMFD/code4z.

6.3.3.1 Installer Code4z Ouvrez VSCode et dans le menu outil de gauche, sélectionnez **Extensions**. À partir de là, dans le champ de recherche « Rechercher des extensions sur le marché », tapez « Code4z ». Les résultats de la recherche commenceront à se remplir, sélectionnez **" Code4z "** et cliquez sur **installer**.

Le pack d'extensions contient un certain nombre d'extensions qui peuvent être exploitées lors de l'utilisation du mainframe, y compris l'extension COBOL Language Support qui fournit des fonctionnalités similaires à l'extension Z Open Editor. Par conséquent, assurez-vous qu'une seule de ces extensions est activée. Les extensions peuvent être désactivées dans VS Code en localisant l'extension dans le menu Extensions, en cliquant sur l'engrenage des paramètres et en sélectionnant « Désactiver ». Les autres extensions incluses dans le pack Code4z fonctionneront avec la prise en charge du langage COBOL ou avec l'éditeur ouvert Z.

6.4 Sommaire

Dans ce chapitre, vous avez été présenté à VSCode et à certains des outils d'extension disponibles. Nous avons parcouru le processus d'installation des logiciels prérequis, Node.js et Java SDK, ainsi que VSCode, Zowe Explorer, IBM Z Open Editor et Code4z. Vous avez également été brièvement présenté à l'utilité de ces extensions dans VSCode. Dans les chapitres suivants, nous approfondirons comment et quand les utiliser et nous nous entraînerons à travers des travaux de laboratoire.

7 Installation de Zowe CLI et des plug-ins

Ce chapitre couvre tous les aspects du téléchargement et de l'installation de Zowe CLI et des plug-ins Zowe CLI.

- **Installer les prérequis - Node.js**
- **Installer Zowe CLI**
  - **Registre npm public**
  - ** Forfait de Zowe.org **
- **Installer les plug-ins CLI Zowe**
  - **Registre npm public**
  - ** Forfait de Zowe.org **
- **Sommaire**

7.1 Installer les prérequis - Node.js

Avant d'installer Zowe CLI, assurez-vous qu'une version LTS de Node v8.0 ou supérieure est installée. Veuillez vous référer à la section intitulée "Installer Node.js" si vous ne l'avez pas déjà terminée. Veuillez également vérifier que vous disposez d'une version de Node Package Manager (npm) compatible avec votre version de Node.js. Pour une liste des versions compatibles, voir https://nodejs.org/en/download/releases/. npm est inclus avec l'installation de Node.js. Exécutez la commande npm −−version pour afficher la version de npm installée.

7.2 Installer Zowe CLI

Il existe deux méthodes recommandées pour installer la CLI Zowe. Si vous avez accès au registre public npm depuis votre poste de travail, nous vous recommandons d'utiliser cette méthode d'installation car l'extraction des mises à jour est transparente. Si vous n'avez pas accès à ce registre, nous vous recommandons de télécharger le package sur zowe.org et de l'installer à partir du package fourni.

7.2.1 Installer à partir du registre public npm

Exécutez la commande suivante dans votre terminal (par exemple, Invite de commandes ou si vous utilisez VS Code, Terminal -> Nouveau terminal) :

```
npm install -g @zowe/cli@zowe-v1-lts
```

Si la commande renvoie une erreur EACCESS, reportez-vous à [Résoudre les erreurs d'autorisations EACCESS lors de l'installation des packages globalement](https://docs.npmjs.com/resolving-eacces-permissions-errors-when-installing-packages-globally) dans le npm Documentation. Si d'autres problèmes sont

rencontrés dans votre environnement, veuillez consulter les problèmes connus de Zowe CLI pour des solutions .

Nous vous recommandons également fortement d'installer le plug-in Secure Credential Store avant d'utiliser la CLI. Le plug-in Secure Credential Store pour Zowe CLI vous permet de stocker vos informations d'identification en toute sécurité dans le gestionnaire d'informations d'identification par défaut du système d'exploitation de votre ordinateur. Sous Linux, libsecret devra être installé.

Si vous utilisez Linux, veuillez exécuter la commande suivante pour votre distribution Linux :

- Debian/Ubuntu : sudo apt−get install libsecret −1−dev
- Basé sur Red Hat : sudo yum install libsecret −devel
- Arch Linux : sudo pacman −S libsecret

Pour installer le plug-in Secure Credential Store pour Zowe CLI, exécutez la commande suivante :

```
Les plugins zowe installent @zowe/secure−credential−store−for−zowe−cli@zowe−v1−l
```

Les profils utilisateur, qui contiennent des informations de connexion pour interagir avec divers services z/OS, créés après l'installation du plug-in stockeront automatiquement vos informations d'identification en toute sécurité.

Pour stocker en toute sécurité les informations d'identification dans les profils utilisateur existants (profils que vous avez créés avant d'installer le plug-in SCS), exécutez la commande suivante :

```
mise a jour zowe scs
```

7.2.2 Installer à partir du package groupé

Accédez à [Téléchargements Zowe.org] (https://www.zowe.org/#download) et cliquez sur le bouton CLI Core pour télécharger le package de base qui inclut Zowe CLI et le plug-in Secure Credential Store. Après avoir accepté le CLUF pour Zowe, un package nommé zowe−cli−package−v.r.m.zip sera téléchargé sur votre machine. Décompressez le contenu de zowe−cli−package−v.r.m.zip dans un emplacement préféré sur votre machine.

Ouvrez votre terminal (par exemple, invite de commande ou si vous utilisez VS Code, Terminal -> Nouveau terminal), modifiez votre répertoire de travail à l'endroit où vous avez décompressé le contenu et exécutez la commande suivante :

```
npm install −g zowe−cli.tgz
```

Si la commande renvoie une erreur EACCESS, reportez-vous à [Résoudre les erreurs d'autorisations EACCESS lors de l'installation des packages globalement]

(https://docs.npmjs.com/resolving-eacces-permissions-errors-when-installing-packages-globally) dans le npm Documentation. Si d'autres problèmes sont rencontrés dans votre environnement, veuillez consulter les problèmes connus de Zowe CLI pour des solutions .

Le plug-in Secure Credential Store hautement recommandé pour Zowe CLI vous permet de stocker vos informations d'identification en toute sécurité dans le gestionnaire d'informations d'identification par défaut du système d'exploitation de votre ordinateur. Sous Linux, libsecret devra être installé.

Si vous utilisez Linux, veuillez exécuter la commande suivante pour votre distribution Linux :

- Debian/Ubuntu : sudo apt−get install libsecret −1−dev
- Basé sur Red Hat : sudo yum install libsecret −devel
- Arch Linux : sudo pacman −S libsecret

Pour installer le plug-in Secure Credential Store pour Zowe CLI, exécutez la commande suivante à partir de laquelle vous avez décompressé le contenu principal du package CLI :

```
Les plugins zowe installent secure−credential−store−for−zowe−cli.tgz
```

Les profils utilisateur, qui contiennent des informations de connexion pour interagir avec divers services z/OS, créés après l'installation du plug-in stockeront automatiquement vos informations d'identification en toute sécurité.

Pour stocker en toute sécurité les informations d'identification dans les profils utilisateur existants (profils que vous avez créés avant d'installer le plug-in SCS), exécutez la commande suivante :

```
mise a jour zowe scs
```

7.3 Installer les plug-ins Zowe CLI

Zowe CLI est une technologie extensible qui peut être améliorée en installant des plug-ins. Zowe propose un certain nombre de [plug-ins] (https://-docs.zowe.org/stable/user-guide/cli-extending.html). Au moment d'écrire ces lignes, ceux-ci incluent des plug-ins pour CICS, Db2, FTP, IMS et MQ. Il existe également de nombreux plug-ins de fournisseurs, dont beaucoup sont disponibles sur le [registre public] (https://www.npmjs.com/search?q=zowe-cli). Au moment d'écrire ces lignes, ceux-ci incluent des plug-ins pour CA Endevor, CA Endevor Bridge for Git, CA File Master Plus, CA OPS/MVS, [CA View] (https://www.npmjs.com/package/@broadcom/caview-for-zowe-cli), [Génération et déploiement d'IBM CICS Bundle] (https://www.npmjs.com/package/zowe-cli-cics-deploy-plugin) et IBM z/OS Connect EE.

7.3.1 Installer à partir du registre public npm

Pour installer un plug-in Zowe CLI à partir du registre, localisez simplement le plug-in que vous souhaitez installer, par ex. @zowe/cics−for−zowe−cli, recherchez la balise de distribution pour la distribution que vous souhaitez installer, par ex. zowe−v1−lts et exécutez la commande suivante :

```
Les plugins zowe installent <name>@<distTag>
```

Par exemple,

```
Les plugins zowe installent @zowe/cics−for−zowe−cli@zowe−v1−lts
```

Plusieurs plug-ins peuvent être installés en une seule commande. Par exemple, pour installer tous les plug-ins Zowe CLI disponibles auprès de l'organisation Zowe, vous pouvez émettre :

```
Les plugins zowe installent @zowe/cics−for−zowe−cli@zowe−v1−lts @zowe/ims−for−zo
```

Les plug-ins des fournisseurs sur le registre sont installés de la meme manière. Par exemple, pour installer le plug-in CA Endevor, vous devez émettre

```
Les plugins zowe installent @broadcom/endevor−for−zowe−cli@zowe−v1−lts
```

7.3.2 Installer à partir du package groupé

Accédez à [Téléchargements Zowe.org] (https://www.zowe.org/#download) et cliquez sur le bouton Plugins CLI pour télécharger le package qui inclut tous les plug-ins CLI Zowe pour l'organisation Zowe. Après avoir accepté le CLUF pour Zowe, un package nommé zowe−cli−plugins−v.r.m.zip sera téléchargé sur votre machine. Décompressez le contenu de zowe−cli−plugins−v.r.m.zip dans un emplacement préféré sur votre machine. Vous pouvez sélectionner les plug-ins que vous souhaitez installer. Le plug-in IBM Db2 nécessite une configuration supplémentaire lors de l'installation à partir d'un package local . Pour installer tous les plug-ins, vous pouvez exécuter :

```
Les plugins zowe installent cics−for−zowe−cli.tgz zos−ftp−for−zowe−cli.tgz ims−f
```

Pour l'installation hors ligne des plug-ins du fournisseur, veuillez contacter le fournisseur spécifique pour plus de détails.

7.4 Sommaire

Dans ce chapitre, nous avons parcouru le processus d'installation des logiciels prérequis, Node.js et npm, ainsi que Zowe CLI et divers plug-ins.

Partie 2 - Apprentissage du COBOL

8 COBOL de base

Ce chapitre présente les bases de la syntaxe COBOL. Il montre ensuite comment afficher et exécuter un programme COBOL de base dans VSCode.

- **Caractéristiques COBOL**
  - **Entreprise COBOL**
  - **Objectifs du chapitre**
- ** Que doit savoir un programmeur COBOL novice pour être un programmeur COBOL expérimenté ? **
  - **Quelles sont les règles de codage et le format de référence ?**
  - **Quelle est la structure de COBOL ?**
  - **Quels sont les mots réservés COBOL ?**
  - **Qu'est-ce qu'une instruction COBOL ?**
  - **Quelle est la signification d'un terminateur de portée ?**
  - **Qu'est-ce qu'une phrase COBOL ?**
  - **Qu'est-ce qu'un paragraphe COBOL ?**
  - **Qu'est-ce qu'une section COBOL ?**
  - **Comment exécuter un programme COBOL sur z/OS ?**
- **Divisions COBOL**
  - **Structure Divisions COBOL**
  - **Quelles sont les quatre Divisions du COBOL ?**
- **DIVISION DE LA PROCÉDURE expliquée**
- **Information additionnelle**
  - **Manuels professionnels**
  - **En savoir plus sur les récents progrès de COBOL**
- **Laboratoire**
- **Laboratoire - Zowe CLI et automatisation**
  - **Zowe CLI - Utilisation interactive**
  - **Zowe CLI - Utilisation par programmation**

8.1 Caractéristiques COBOL

COBOL est un langage informatique de type anglais permettant au code source COBOL d'être plus facile à lire, à comprendre et à maintenir. Apprendre à programmer en COBOL comprend la connaissance des règles du code source COBOL, les mots réservés COBOL, la structure COBOL et la capacité de

localiser et d'interpréter la documentation COBOL professionnelle. Ces caractéristiques COBOL doivent être comprises pour maîtriser la lecture, l'écriture et la maintenance des programmes COBOL.

8.1.1 Entreprise COBOL

COBOL est une norme et n'appartient à aucune entreprise ou organisation. "Enterprise COBOL" est le nom du langage de programmation COBOL compilé et exécuté dans le système d'exploitation IBM Z, z/OS. Les détails et les explications du COBOL dans les chapitres suivants s'appliquent au COBOL d'entreprise.

Enterprise COBOL a des décennies d'avancées, y compris de nouvelles fonctions, des extensions de fonctionnalités, des performances améliorées, des interfaces de programmation d'applications (API), etc. Il fonctionne avec les technologies d'infrastructure modernes avec une prise en charge native de JSON, XML et Java®.

8.1.2 Objectifs du chapitre

L'objet de ce chapitre est d'exposer le lecteur à la terminologie COBOL, aux règles de codage et à la syntaxe, tandis que les chapitres restants incluent plus de détails avec des laboratoires pour pratiquer ce qui est introduit dans ce chapitre.

8.2 Que doit savoir un programmeur COBOL novice pour être un programmeur COBOL expérimenté ?

Cette section fournira au lecteur les informations nécessaires pour mieux comprendre les questions et les réponses posées dans chaque rubrique suivante.

8.2.1 Quelles sont les règles de codage et le format de référence ?

Le code source COBOL dépend de la colonne, ce qui signifie que les règles de colonne sont strictement appliquées. Chaque ligne de code source COBOL comporte cinq zones, chacune de ces zones ayant une colonne de début et de fin.

Le texte source COBOL doit être écrit au format de référence COBOL. Le format de référence se compose des zones illustrées à la figure 1. sur une ligne de 72 caractères.

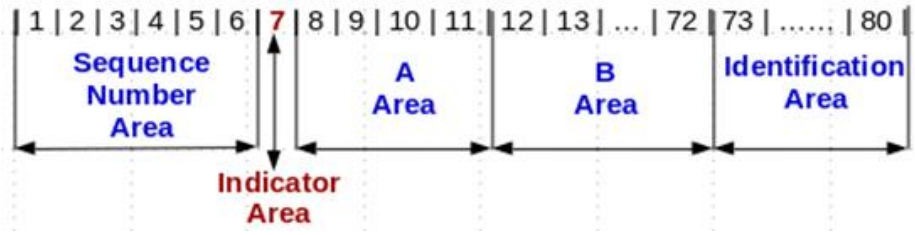


Figure 1. Format de référence COBOL

Le format de référence COBOL est formaté comme suit :

8.2.1.1 Zone de numéro de séquence (colonnes 1 - 6)

- Vide ou réservé aux numéros de séquence de ligne.

8.2.1.2 Zone d'indicateur (colonne 7)

- Un espace polyvalent :
  - Ligne de commentaire (généralement un astérisque)
  - Ligne de continuation (généralement un trait d'union)
  - Ligne de débogage (D ou d)
  - Formatage de la liste des sources (un symbole slash)

8.2.1.3 Zone A (colonnes 8 - 11)

- Certains éléments doivent commencer dans la zone A, ce sont :
  - Indicateurs de niveau
  - Déclaratif
  - En-têtes de division, de section, de paragraphe
  - Noms de paragraphe
- La colonne 8 est appelée la marge A

8.2.1.4 Zone B (colonnes 12 - 72)

- Certains éléments doivent commencer dans la zone B, ce sont :
  - Entrées, phrases, déclarations et clauses
  - Lignes de continuation
- La colonne 12 est appelée la marge B

8.2.1.5 Zone d'identification (colonnes 73 - 80)

- Ignoré par le compilateur.
- Peut être vide ou éventuellement utilisé par le programmeur à n'importe quelle fin.

8.2.2 Quelle est la structure de COBOL ?

COBOL est une structure hiérarchique composée et dans l'ordre descendant de :

- Divisions
- Sections
- Les paragraphes
- Phrases
- Déclarations

8.2.3 Que sont les mots réservés COBOL ?

Le langage de programmation COBOL comporte de nombreux mots ayant une signification spécifique pour le compilateur COBOL, appelés mots réservés. Ces mots réservés ne peuvent pas être utilisés comme noms de variables choisis par le programmeur ou comme noms de types de données choisis par le programmeur.

Quelques mots réservés COBOL pertinents pour ce livre sont : PERFORM, MOVE, COMPUTE, IF, THEN, ELSE, EVALUATE, PICTURE, etc. Vous pouvez trouver un tableau de tous les mots COBOL réservés à l'adresse :

https://www.ibm.com/docs/en/cobol-zos/6.3?topic=appendixes-reserved-words

8.2.4 Qu'est-ce qu'une instruction COBOL ?

Des mots réservés COBOL spécifiques sont utilisés pour modifier le flux d'exécution en fonction des conditions actuelles. Les « déclarations » n'existent qu'au sein de la division des procédures, la logique de traitement du programme. Des exemples de mots réservés COBOL utilisés pour modifier le flux d'exécution sont :

- SI
- Évaluer
- Effectuer

8.2.5 Quelle est la signification d'un terminateur de portée ?

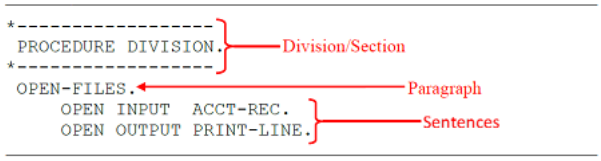
Un terminateur de portée peut être explicite ou implicite. Un terminateur de portée explicite marque la fin de certaines instructions PROCEDURE DIVISION avec le mot réservé COBOL « END- ». Tout verbe COBOL qui est toujours conditionnel (IF, EVALUATE) ou qui a une clause conditionnelle (COMPUTE, PERFORM, READ) aura un terminateur de portée correspondant. Un terminateur de portée implicite est un point (.) qui termine la portée de toutes les instructions précédentes qui n'ont pas encore été terminées.

8.2.6 Qu'est-ce qu'une phrase COBOL ?

Une « phrase » COBOL est une ou plusieurs « déclarations » suivies d'un point (.), où le point sert de fin de portée.

8.2.7 Qu'est-ce qu'un paragraphe COBOL ?

Un « paragraphe » COBOL est un nom défini par l'utilisateur ou prédéfini suivi d'un point. Un « paragraphe » se compose de zéro ou plusieurs phrases et est la subdivision d'une « section » ou d'une « division », voir l'exemple 1 ci-dessous.



Exemple 1. Division -> paragraphe -> phrases

8.2.8 Qu'est-ce qu'une section COBOL ?

Une « section » est soit un nom défini par l'utilisateur, soit un nom prédéfini suivi d'un point et se compose de zéro ou plusieurs phrases. Une « section » est une collection de paragraphes.

8.2.9 Comment exécuter un programme COBOL sous z/OS ?

Lorsque vous utilisez COBOL sous z/OS, vous rencontrerez JCL ou Job Control Language. JCL est un ensemble d'instructions qui indiquent au système d'exploitation z/OS les tâches que vous souhaitez qu'il exécute.

Pour que votre programme COBOL soit exécutable sous z/OS, vous devez indiquer au système d'exploitation de compiler et de modifier le code avant de l'exécuter. Tout cela se fera à l'aide de JCL.

La première chose que votre JCL doit faire est de compiler le programme COBOL que vous avez écrit. Dans cette étape, votre programme est transmis au compilateur COBOL pour être traité en code objet. Ensuite, la sortie du compilateur passera par l'étape d'édition de lien. Ici, un classeur reprendra le code objet et toutes les bibliothèques et options nécessaires spécifiées dans le JCL pour créer un programme exécutable. Dans cette étape, vous pouvez également indiquer au JCL d'inclure des ensembles de données supplémentaires que votre programme COBOL lira. Ensuite, vous pouvez exécuter le programme.

Pour simplifier les choses, Enterprise COBOL for z/OS fournit trois procédures JCL pour compiler votre code. Lors de l'utilisation d'une procédure JCL, nous pouvons fournir la partie variable pour répondre à un cas d'utilisation spécifique. Voici les procédures qui s'offrent à vous :

1. Procédure de compilation (IGYWC)

2. Procédure de compilation et d'édition de liens (IGYWCL)
3. Compiler, éditer des liens et exécuter la procédure (IGYWCLG)

Ce cours étant un cours COBOL, le JCL nécessaire à la réalisation des Labs vous est fourni. Par conséquent, vous rencontrerez les procédures répertoriées ci-dessus sur le JCL. Si vous souhaitez créer un nouveau programme COBOL, vous pouvez copier l'un des JCL fournis et le modifier en conséquence.

Pour en savoir plus sur JCL, visitez le centre de connaissances IBM :

https://www.ibm.com/docs/en/zos-basic-skills?topic=collection-basic-jcl-concepts

8.3 Divisions COBOL

Cette section présente les quatre divisions COBOL et décrit brièvement leur objectif et leurs caractéristiques.

8.3.1 Structure des divisions COBOL

Les divisions sont subdivisées en sections.

Les sections sont subdivisées en paragraphes.

Les paragraphes sont subdivisés en phrases.

Les phrases consistent en des déclarations.

Les déclarations commencent par des mots réservés COBOL et peuvent être subdivisées en « Phrases »

8.3.2 Quelles sont les quatre divisions du COBOL ?

8.3.2.1 IDENTIFICATION DIVISION L' IDENTIFICATION DIVISION identifie le programme avec un nom et, facultativement, donne d'autres informations d'identification, telles que le nom de l'auteur, la date de compilation du programme (dernière modification), etc.

8.3.2.2 ENVIRONNEMENT DIVISION L' ENVIRONMENT DIVISION décrit les aspects de votre programme qui dépendent de l'environnement informatique, tels que la configuration de l'ordinateur et les entrées et sorties de l'ordinateur.

8.3.2.3 DATA DIVISION La DATA DIVISION est l'endroit où les caractéristiques des données sont définies dans l'une des sections suivantes :

- FILE SECTION :

  Définit les données utilisées dans les opérations d'entrée-sortie.

- LINKAGE SECTION:

  Décrit les données d'un autre programme. Lors de la définition des données développées pour le traitement interne.

- WORKINS-STORAGE SECTION:

  Stockage alloué et restant pour la durée de vie du programme.

- LOCAL-STORAGE SECTION:

  Stockage alloué à chaque fois qu'un programme est appelé et désalloué à la fin du programme.

8.3.2.4 PROCEDURE DIVISION La PROCEDURE DIVISION contient des instructions relatives à la manipulation des données et des interfaces avec d'autres procédures sont précisées.

8.4 PROCEDURE DIVISION expliquée

La PROCEDURE DIVISION est l'endroit où le travail est effectué dans le programme. Les déclarations se trouvent dans la PROCÉDURE DIVISION où elles sont des actions à entreprendre par le programme. La DIVISION DE PROCÉDURE est requise pour que les données soient traitées par le programme. PROCÉDURE DIVISION d'un programme est divisé en sections et paragraphes, qui contiennent des phrases et des déclarations, comme décrit ici :

- **Section** - Une subdivision logique de votre logique de traitement. Une section a un en-tête et est éventuellement suivie d'un ou plusieurs paragraphes. Une section peut faire l'objet d'une instruction PERFORM. Un type de section est pour les déclaratifs. Les déclaratives sont un ensemble d'une ou plusieurs sections à usage spécial. Les sections à usage spécial sont exactement ce à quoi elles ressemblent, des sections écrites à des fins spéciales et peuvent contenir des éléments tels que la description des entrées et des sorties. Elles sont écrites au début de la PROCEDURE DIVISION, dont la première est précédée du mot-clé DECLARATIVES et dont la dernière est suivie du mot-clé FIN DECLARATIVES.
- **Paragraphe** - Une subdivision d'une section, d'une procédure ou d'un programme. Un paragraphe peut faire l'objet d'un énoncé.
- **Phrase** - Une série d'une ou plusieurs instructions COBOL se terminant par un point.
- **Instruction** - Une action à entreprendre par le programme, telle que l'ajout de deux nombres.
- **Phrase** - Une petite partie d'un énoncé (c'est-à-dire une subdivision), analogue à un adjectif ou une préposition anglais

8.5 Information additionnelle

Cette section fournit des ressources utiles sous forme de manuels et de vidéos pour vous aider à en apprendre davantage sur les bases de COBOL.

8.5.1 Manuels professionnels

Au fur et à mesure que l'expérience Enterprise COBOL progresse, le besoin de documentation professionnelle augmente. Une recherche sur Internet des manuels Enterprise COBOL inclut : « Enterprise COBOL for z/OS documentation library - IBM », lien fourni ci-dessous. Le contenu du site comporte des onglets pour chaque niveau de version COBOL. Depuis avril 2020, la version actuelle d'Enterprise COBOL est la V6.3. Mettez en surbrillance l'onglet V6.3, puis sélectionnez la documentation du produit.

https://www.ibm.com/support/pages/enterprise-cobol-zos-documentation-library

Trois manuels « Enterprise COBOL for z/OS » sont référencés dans les chapitres en tant que sources d'informations supplémentaires, pour référence et pour faire progresser le niveau de connaissances. Elles sont:

1. Référence du langage - Décrit le langage COBOL tel que la structure du programme, les mots réservés, etc.

   http://publibfp.boulder.ibm.com/epubs/pdf/igy6lr30.pdf

2. Guide de programmation - Décrit des sujets avancés tels que les options du compilateur COBOL, l'optimisation des performances du programme, la gestion des erreurs, etc.

   http://publibfp.boulder.ibm.com/epubs/pdf/igy6pg30.pdf

3. Messages et codes - Pour mieux comprendre certains messages du compilateur COBOL et codes de retour pour diagnostiquer les problèmes.

   http://publibfp.boulder.ibm.com/epubs/pdf/c2746481.pdf

8.5.2 En savoir plus sur les récents progrès de COBOL

- Quoi de neuf dans Enterprise COBOL pour z/OS V6.1 :

  https://www.ibm.com/support/pages/cobol-v61-was-announced-whats-new

- Nouveautés d'Enterprise COBOL pour z/OS V6.2 :

  https://www.ibm.com/support/pages/cobol-v62-was-announced-whats-new

- Quoi de neuf dans Enterprise COBOL pour z/OS V6.3 :

https://www.ibm.com/support/pages/cobol-v63-was-announced-whats-new

8.6 Labo

Dans cet exercice de laboratoire, vous allez vous connecter à un système IBM Z, afficher un simple programme COBOL hello world dans VSCode, soumettre JCL pour compiler le programme COBOL et afficher la sortie. Référez-vous à "Installation de VSCode et extensions" pour configurer VSCode si vous ne l'avez pas déjà fait. Vous pouvez soit utiliser Z Open Editor et Zowe Explorer, soit Code4z.

1. Le laboratoire suppose l'installation de VSCode avec les extensions Z Open Editor et Zowe Explorer, comme illustré à la Figure 2a, ou le pack d'extension Code4z, comme illustré à la Figure 2b.

   Cliquez sur l'icone **Extensions**. Si vous avez installé Z Open Editor et Zowe Explorer, la liste doit inclure :

   1. Éditeur IBM Z Open
   2. Explorateur Zowe

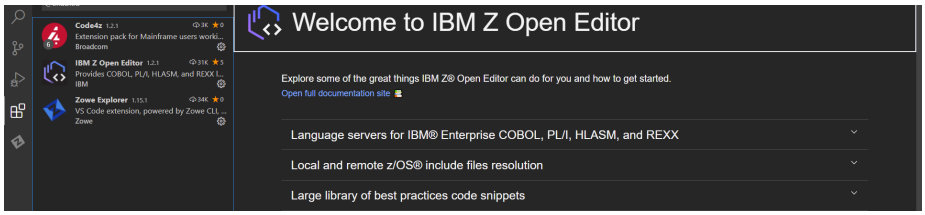


Figure 2a. Les extensions Z Open Editor et Zowe Explorer VSCode

Si vous avez installé Code4z, la liste doit inclure :

1. Prise en charge du langage COBOL
2. Explorateur Zowe
3. Explorateur pour Endevor
4. Prise en charge du langage HLASM
5. Débogueur pour mainframe
6. Flux de controle COBOL

   Dans ces exercices, vous n'utiliserez que les extensions COBOL Language Support et Zowe Explorer.

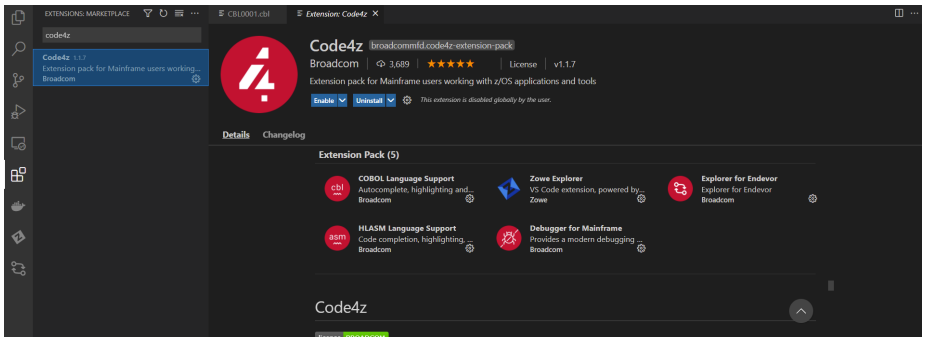


Figure 2b. Le package Code4z d'extensions VS Code.

Remarque : Si votre liste contient à la fois Z Open Editor et COBOL Language Support, désactivez l'un d'entre eux en cliquant sur l'icone **cog** à coté de l'extension dans la liste des extensions et en sélectionnant **désactiver** .

2. Cliquez sur l'icone Zowe Explorer comme illustré à la Figure 3.

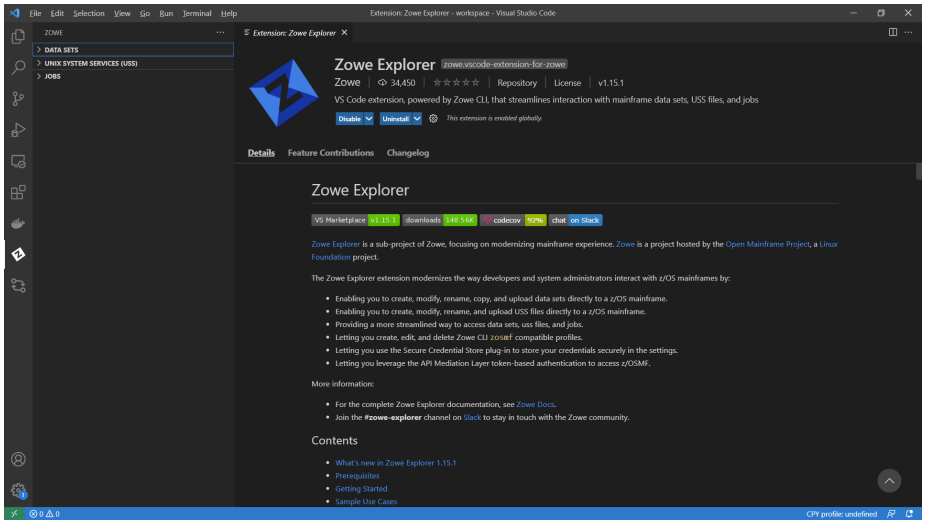


Figure 3. Zowe Explorer Icone Zowe Explorer

3. Zowe Explorer peut répertorier les ensembles de données, les fichiers Unix System Services (USS) et la sortie des tâches, comme illustré à la figure 4. + apparaîtra lorsque vous survolez tout à droite sur la ligne DATA SETS. Cliquez sur le + pour définir un profil VSCode.

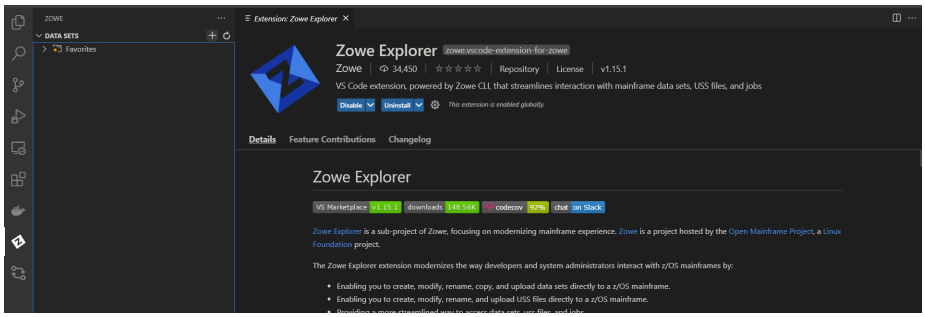


Figure 4. Explorateur Zowe

4. Une boîte apparaît pour définir un nouveau profil. Cliquez sur + à gauche de Créer une nouvelle connexion à z/OS, comme illustré à la Figure 5.

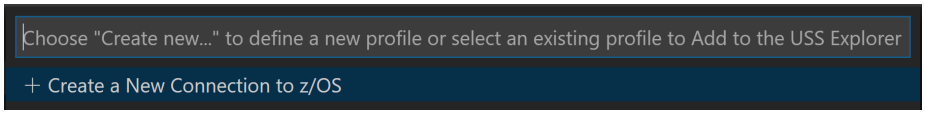


Figure 5. Créer une nouvelle connexion à z/OS

5. Sélectionnez un nom à saisir, puis saisissez. La figure 6. a utilisé « LearnCOBOL » comme nom de connexion sélectionné.

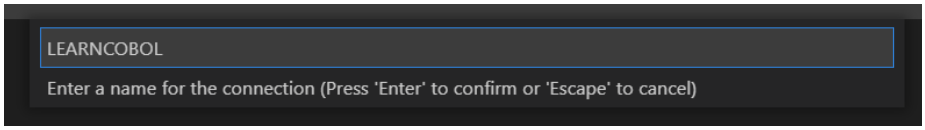


Figure 6. Définir le nom de la connexion

6. VSCode demande l'URL et le port z/OSMF, comme illustré à la Figure 7. L'URL et le port z/OSMF seront normalement fournis par l'administrateur système z/OS.

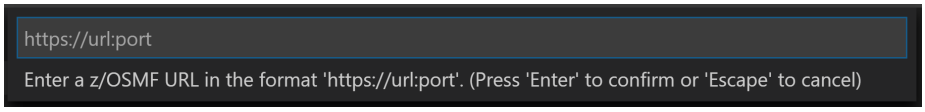


Figure 7. URL z/OSMF

7. Un exemple d'URL et de port z/OSMF est entré, comme illustré à la Figure 8.

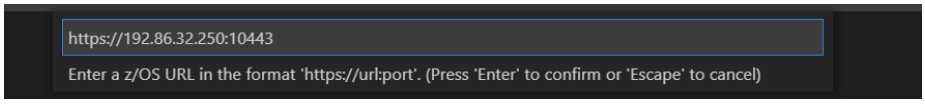


Figure 8. URL z/OSMF spécifiée

8. La connexion demande le nom d'utilisateur, comme illustré à la figure 9.

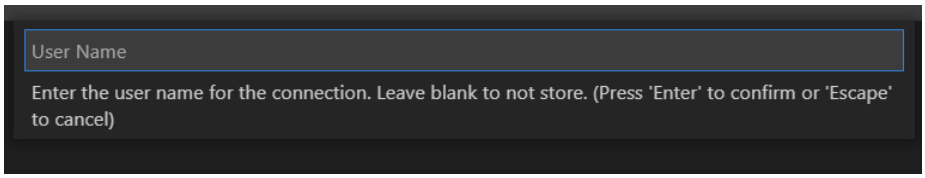


Figure 9. Invite de nom d'utilisateur

9. **Veuillez saisir le nom d'utilisateur qui vous a été attribué ! N'utilisez pas l'exemple de nom d'utilisateur de Z99998.** Un exemple de nom d'utilisateur est saisi comme illustré à la Figure 10. L'ID est attribué par l'administrateur système.

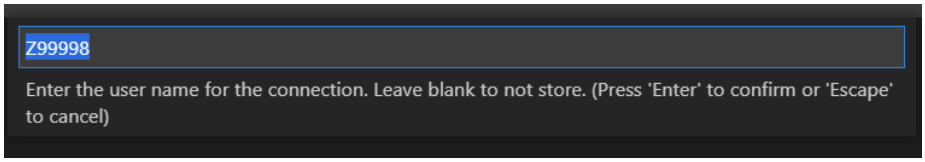


Figure 10. Nom d'utilisateur spécifié

10. La connexion demande le mot de passe, comme illustré à la Figure 11.

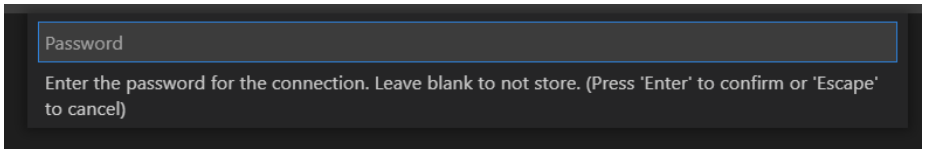


Figure 11. Invite de mot de passe

11. Saisissez le mot de passe comme illustré à la Figure 12.

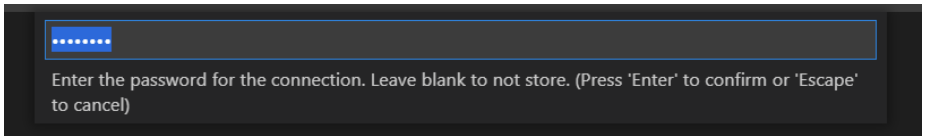


Figure 12. Mot de passe spécifié

12. Sélectionnez **False - Accepter les connexions avec des certificats auto-signés** pour autoriser la connexion au poste de travail, comme illustré à la Figure 13.

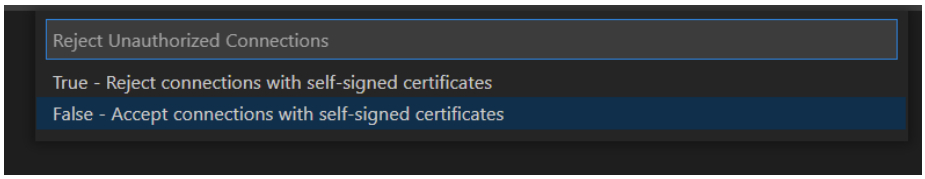
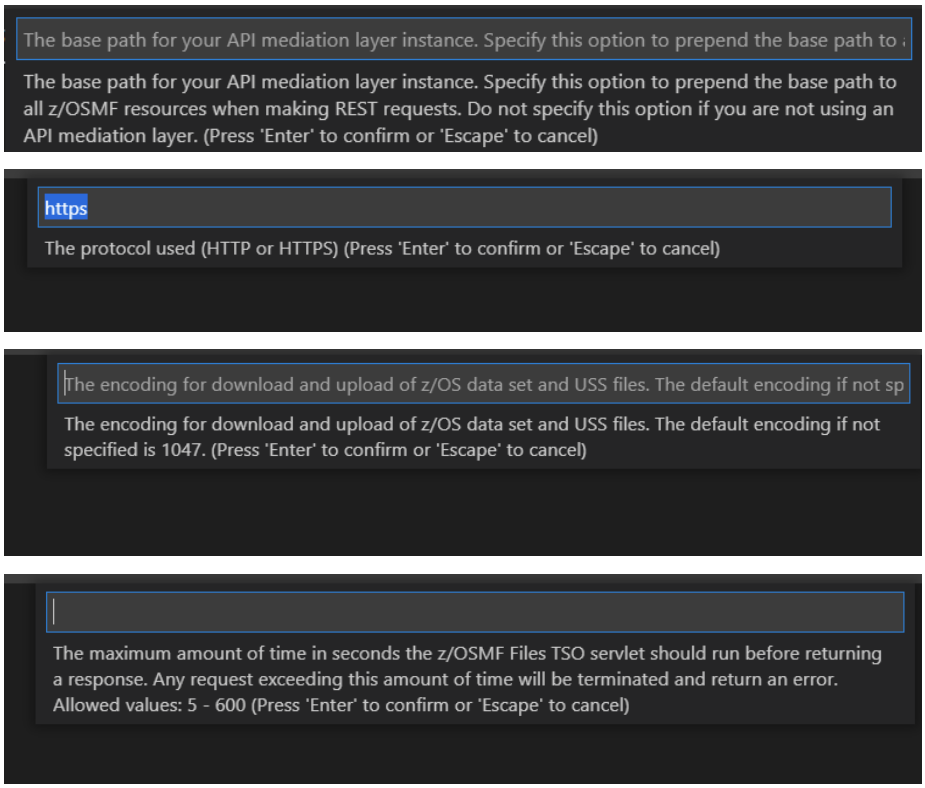


Figure 13. Acceptez les connexions avec des certifications auto-signées

13. Appuyez sur **Entrée** pour le reste des invites pour choisir les valeurs par défaut



14. Votre connexion sera ajoutée automatiquement à la liste **DATASET** sous **Favoris** comme illustré à la Figure 14.

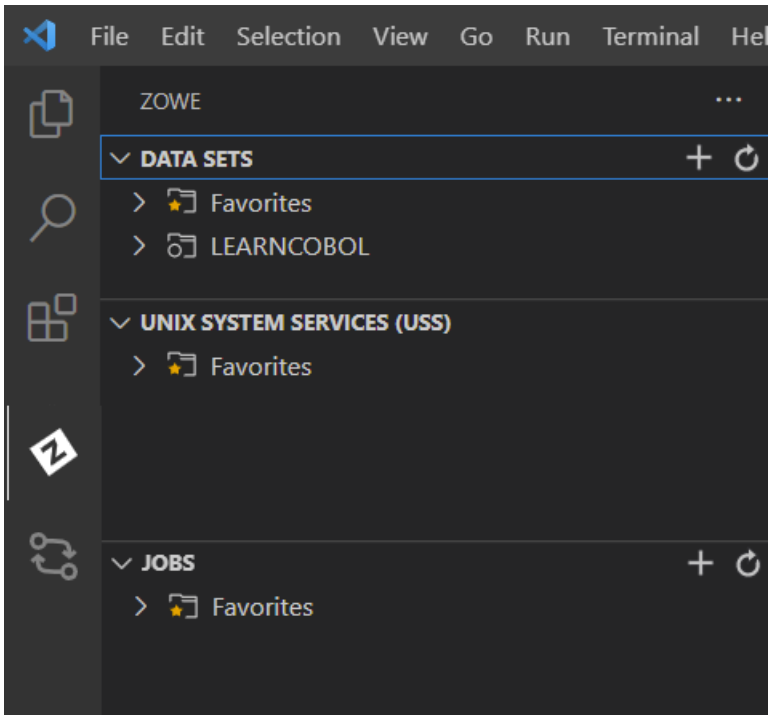


Figure 14. Connexion LEARNCOBOL

- Remarque : *si vous cliquez sur le + à l'extrême droite de la sélection des ensembles de données, une autre invite pour créer une nouvelle connexion à z/OS apparaîtra.*
  - toutes les autres connexions que vous avez seront répertoriées comme illustré à la Figure 15.

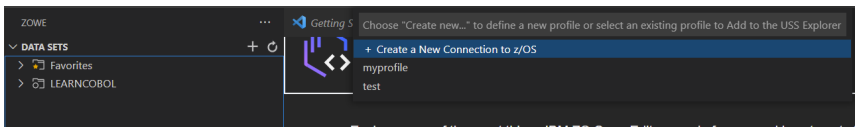


Figure 15. liste des connexions

15. L'extension de LEARNCOBOL indique « Utilisez le bouton de recherche pour afficher les ensembles de données ». Cliquez sur le bouton de recherche comme illustré à la Figure 16.

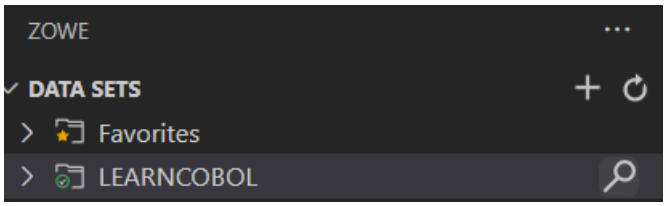


Figure 16. Bouton de recherche

16. Une invite pour “Sélectionner un filtre” apparaît pour votre nom d’utilisateur. Sélectionnez le + pour « Créer un nouveau filtre » comme le montre la figure 17.

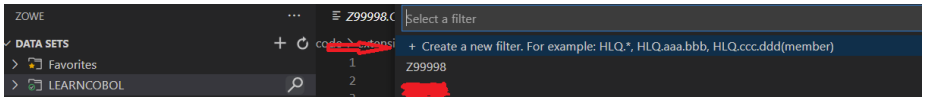


Figure 17. Sélectionnez un filtre

17. Une invite apparaît pour saisir le nom du filtre à rechercher, comme illustré à la Figure 18.

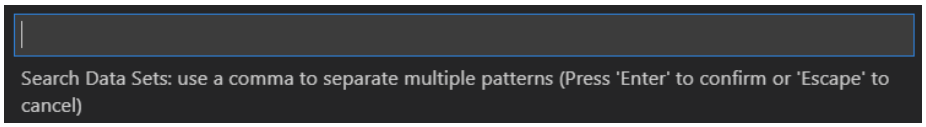


Figure 18. Nom du filtre à rechercher

18. Chaque utilisateur a un qualificatif de haut niveau qui est le meme que son nom d’utilisateur. Par conséquent, entrez le nom d’utilisateur qui vous a été attribué comme critère de recherche, comme illustré à la Figure 19. **Veuillez utiliser votre nom d’utilisateur, pas Z99998 !**

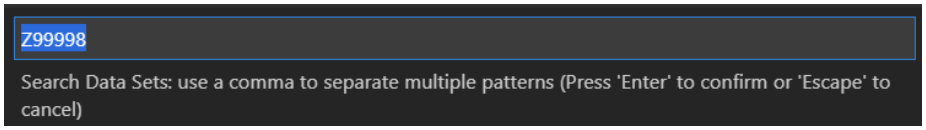


Figure 19. Nom du filtre saisi

19. Une liste de noms d’ensembles de données commençant par votre nom d’utilisateur de z/OS Connection LEARNCOBOL apparaît, comme illustré à la Figure 20.

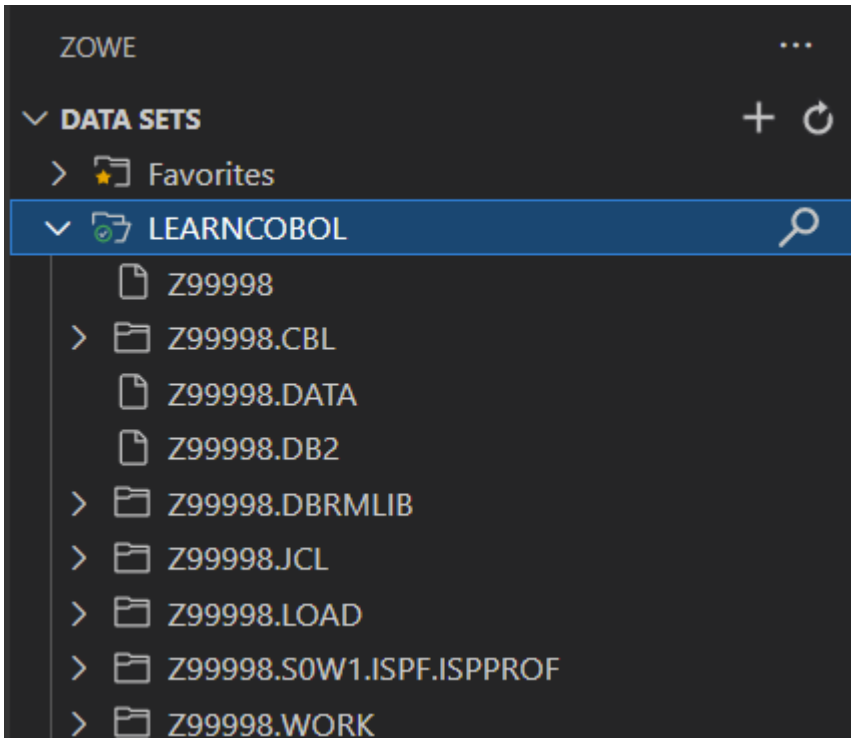


Figure 20. Noms des jeux de données filtrés

20. Développez ***<USERNAME>.CBL** pour afficher les membres source COBOL, puis sélectionnez le membre **HELLO** pour voir un programme COBOL « Hello World ! » simple, comme illustré à la Figure 21.

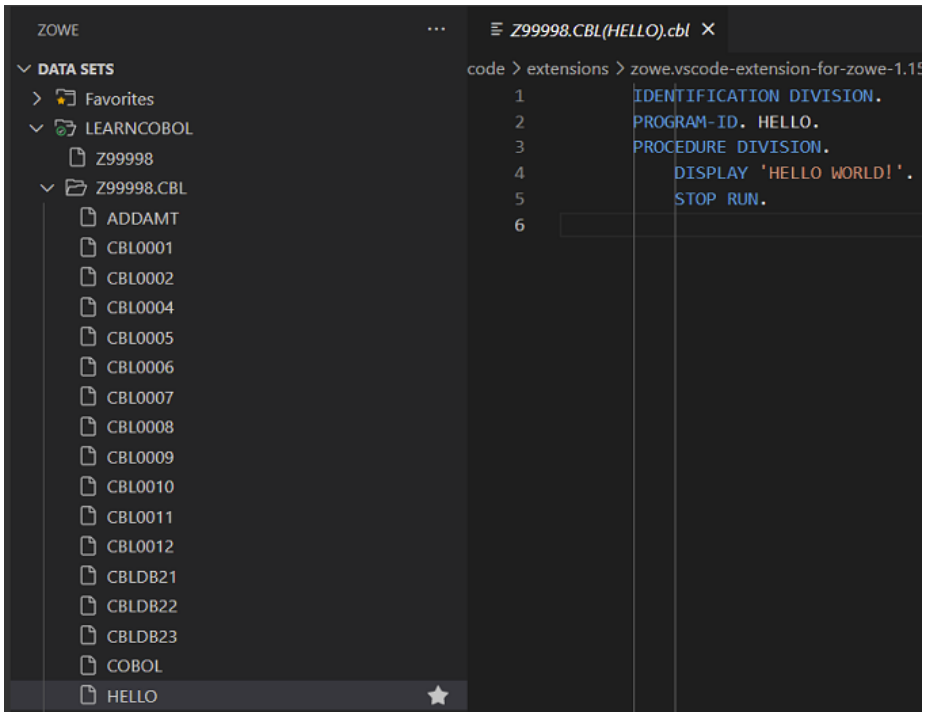


Figure 21. <USERNAME>.CBL

21. Développez **<USERNAME>.JCL** pour afficher le JCL et sélectionnez le membre HELLO qui est le JCL pour compiler et exécuter le code source COBOL simple « Hello World ! », comme illustré à la Figure 22.

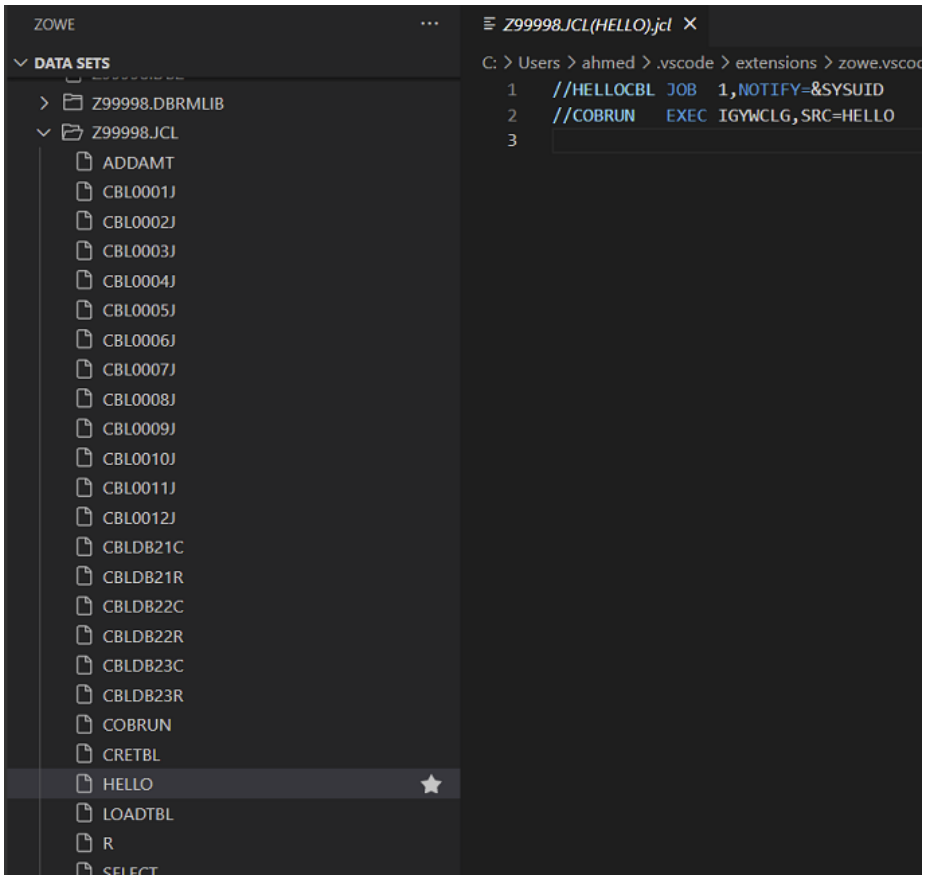


Figure 22. <USERNAME>.JCL

22. Faites un clic droit sur le membre JCL **HELLO** . Une zone de section apparaît. Sélectionnez **Submit Job** pour que le système traite le JCL HELLO, comme illustré à la Figure 23. Le travail JCL soumis compile le code source COBOL HELLO, puis exécute le programme COBOL HELLO.

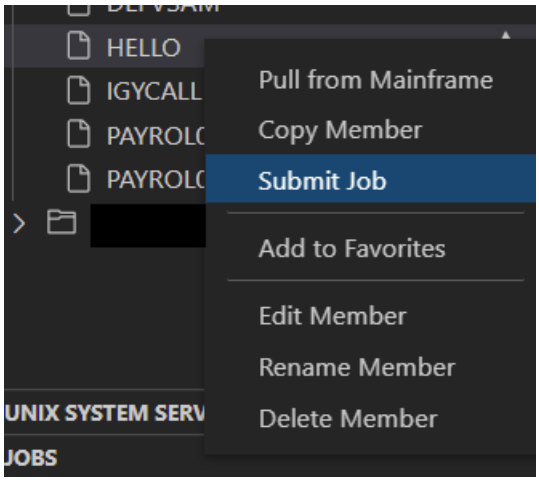


Figure 23. Soumettre le travail

23. Observez la section « Emplois » dans Zowe Explorer, comme illustré à la Figure 24.

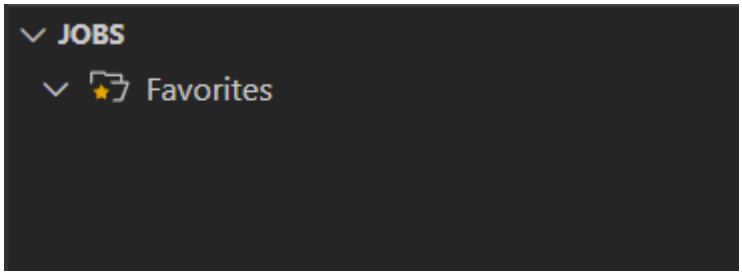


Figure 24. Section JOBS

24. Encore une fois, cliquez sur le + à l'extrême droite de la sélection Jobs. Le résultat est une autre invite pour « Créer un nouveau ». Sélectionnez **LearnCOBOL** dans la liste, comme illustré à la Figure 25.

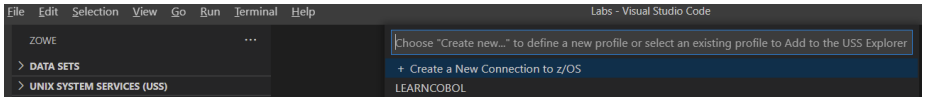


Figure 25. + Sélectionnez la connexion LEARNCOBOL

25. Par conséquent, les travaux JCL appartenant à votre nom d'utilisateur s'affichent. HELLOCBL est le nom du travail JCL précédemment soumis. Développez la sortie **HELLOCBL** pour afficher des sections de la sortie, comme illustré à la Figure 26.

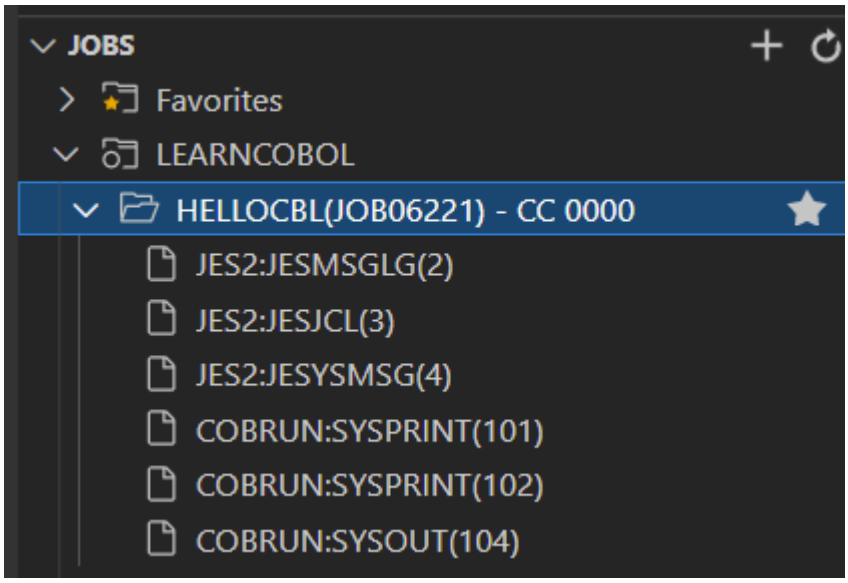


Figure 26. Sortie HELLOCBL

26. Sélectionnez **COBRUN:SYSPRINT(101)** pour afficher la sortie du compilateur COBOL. Faites défiler la compilation COBOL vers l'avant pour localiser le code source COBOL compilé dans un module exécutable, comme illustré à la Figure 27. Observez la zone d'indicateur dans la colonne 7, la zone A commençant à la colonne 8 et la zone B commençant à la colonne 12. Observez également les points de terminaison (.) de portée dans la source COBOL.

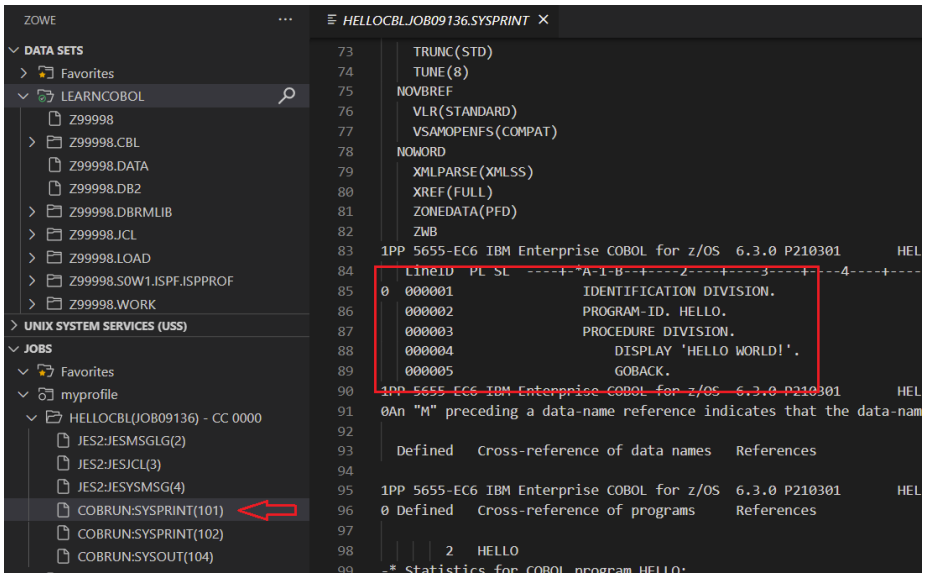


Figure 27. Sortie du compilateur COBOL

27. Affichez l'exécution du programme COBOL en sélectionnant **COBRUN:SYSOUT(104)** dans LEARNCOBOL dans la section Jobs de Zowe Explorer, comme illustré à la Figure 28.

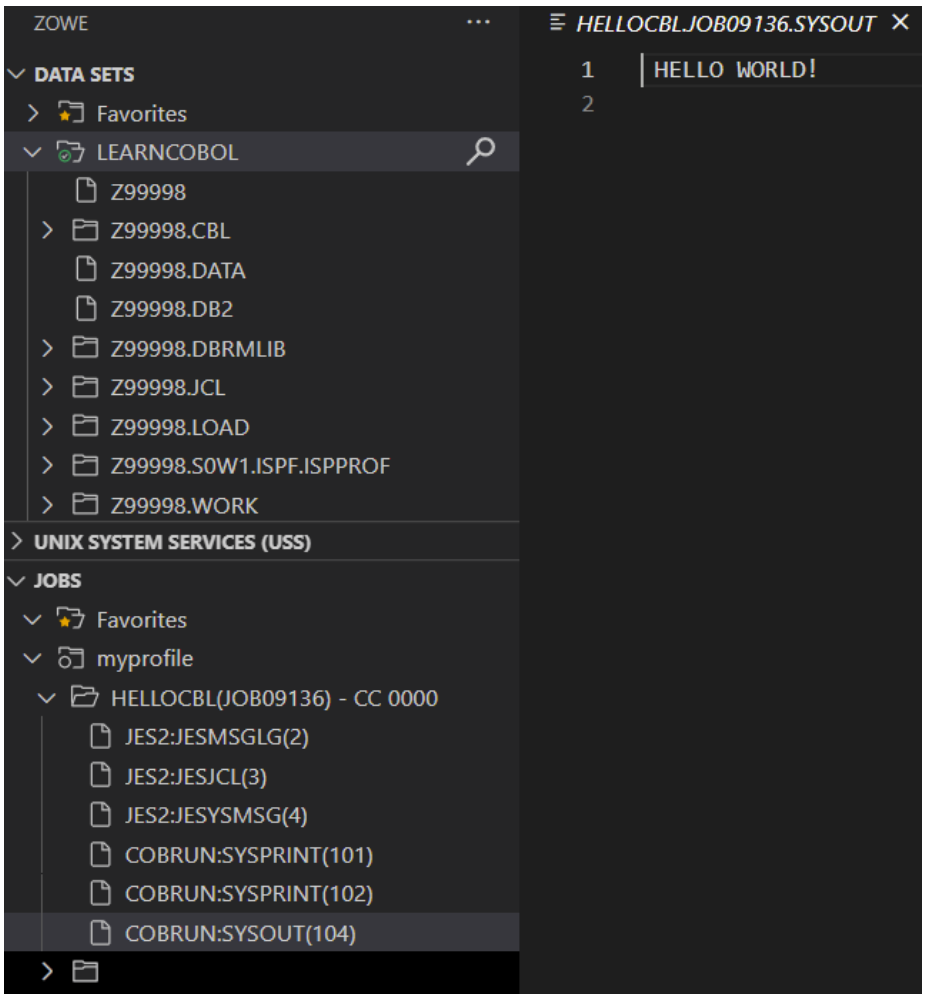


Figure 28. Exécution du programme COBOL

28. L'URL suivante est un autre excellent document décrivant les détails ci-dessus de VSCode et Zowe Explore avec des exemples : https://marketplace.visualstudio.com/items?itemName=Zowe.vscode-extension-for-zowe

8.7 Lab - Zowe CLI et automatisation

Dans cet exercice de laboratoire, vous utiliserez l'interface de ligne de commande Zowe pour automatiser la soumission du JCL pour compiler, lier et exécuter le programme COBOL et télécharger la sortie du spool. Reportez-vous à la section "Installation de Zowe CLI et des plug-ins" pour installer Zowe CLI si vous ne l'avez pas déjà fait. Avant de développer l'automatisation, nous allons d'abord exploiter la CLI Zowe de manière interactive.

8.7.1 Zowe CLI - Utilisation interactive

Dans cette section, nous utiliserons la CLI Zowe de manière interactive pour afficher les membres de l'ensemble de données, soumettre des tâches et examiner la sortie du spool.

1. Dans VS Code, ouvrez le terminal intégré (Terminal -> Nouveau terminal). Dans le terminal, exécutez « zowe –version » pour confirmer que la CLI Zowe est installée comme illustré dans la figure suivante. S'il n'est pas installé, reportez-vous à la section "Installation de Zowe CLI et des plug-ins". Notez également que le shell par défaut sélectionné (encadré en rouge) est « bash ». Je recommanderais de sélectionner le shell par défaut en tant que « bash » ou « cmd » pour ce laboratoire.

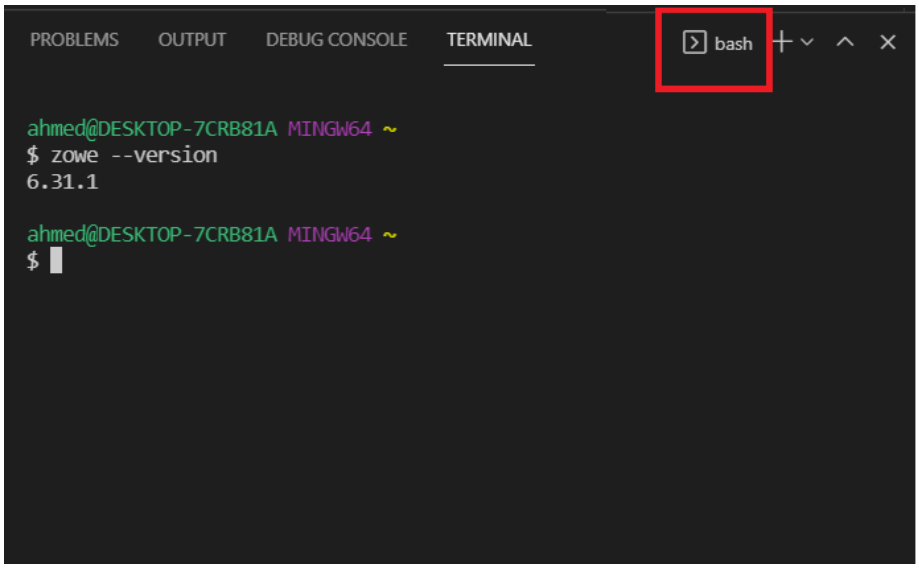


Figure 29. Commande zowe ––version dans le terminal intégré de code VS (sélection du shell encadrée en rouge)

2. Pour que Zowe CLI puisse interagir avec z/OSMF, la CLI doit connaître les détails de la connexion tels que l'hôte, le port, le nom d'utilisateur, le mot de passe, etc. Bien que vous puissiez saisir ces informations sur chaque commande, Zowe offre la possibilité de stocker ces informations. dans des

configurations communément appelées profils. Zowe CLI et l'extension de code Zowe VS partagent des profils. Donc, si vous avez créé un profil de connexion dans le premier laboratoire, vous pouvez naturellement l'exploiter ici.

Pour créer un profil LEARNCOBOL (et l'écraser s'il existe déjà), exécutez la commande suivante avec les détails de votre système (l'utilisation de prompt∗ vous demandera certains champs et n'affichera pas l'entrée) :

```
zowe profiles create zosmf LEARNCOBOL --host 192.86.32.250 --port 10443 --ru fal
```

De nombreux profils peuvent être créés pour interagir avec différentes instances z/OSMF. S'il ne s'agissait pas de votre premier profil, vous souhaiterez le définir comme profil par défaut pour les exercices de laboratoire suivants. Émettez la commande suivante :

```
zowe profiles set zosmf LearnCOBOL
```

La figure suivante illustre cette séquence de commandes.

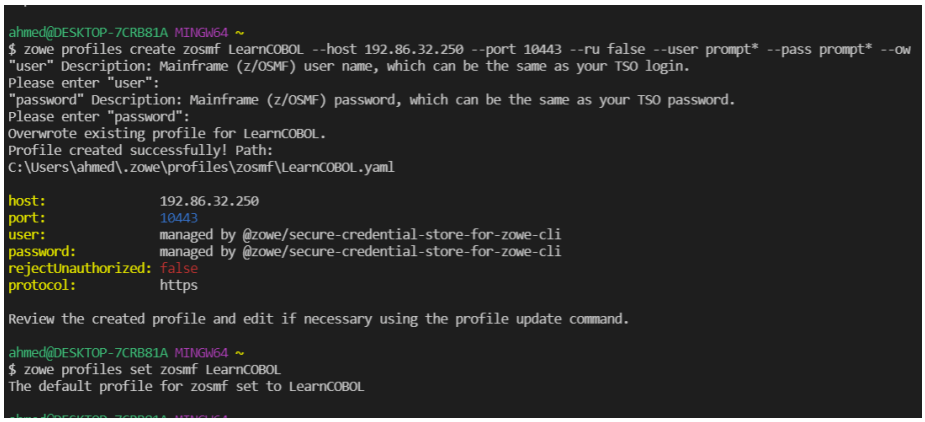


Figure 30. Créer et définir un profil z/OSMF (le plug-in de stockage d'informations d'identification sécurisé est utilisé)

3. Confirmez que vous pouvez vous connecter à z/OSMF en exécutant la commande suivante :

```
zowe zosmf check status
```

4. Répertoriez les ensembles de données sous votre ID en émettant une commande similaire à (voir l'exemple de sortie dans la figure suivante) :

```
liste de fichiers zowe ds "Z99998.*"
```

Vous pouvez également répertorier tous les membres d'un ensemble de données partitionné en exécutant une commande similaire à (voir l'exemple de sortie dans la figure suivante) :

zowe files list am "Z99998.CBL"

```
ahmed@DESKTOP-7CRB81A MINGW64 ~
$ zowe files list ds "Z99998.*"
Z99998.CBL
Z99998.DATA
Z99998.DBRMLIB
Z99998.DB2
Z99998.JCL
Z99998.LOAD
Z99998.S0W1.ISPF.ISPPROF
Z99998.WORK

ahmed@DESKTOP-7CRB81A MINGW64 ~
$ zowe files list am "Z99998.CBL"
ADDAMT
CBLDB21
CBLDB22
CBLDB23
CBL0001
CBL0002
CBL0004
CBL0005
CBL0006
CBL0007
CBL0008
CBL0009
CBL0010
CBL0011
CBL0012
COBOL
HELLO
SQL
```

*Figure 31. Commande de liste de fichiers ds et am

5. Ensuite, nous allons télécharger nos membres d'ensembles de données COBOL et JCL sur notre machine locale. Tout d'abord, créez et ouvrez

un nouveau dossier dans votre explorateur de fichiers. Notez que vous pouvez également créer un espace de travail pour gérer plusieurs projets. Consultez la figure suivante pour obtenir de l'aide :

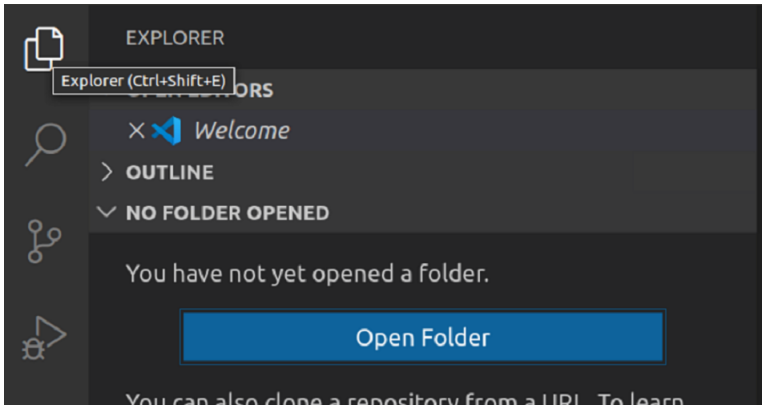


Figure 32. Vue de l'explorateur de fichiers pour montrer l'ouverture d'un nouveau dossier

Une fois que vous avez un dossier vide ouvert, retournez au terminal intégré, assurez-vous que vous êtes dans votre dossier et lancez des commandes similaires à :

```
zowe files download am "Z99998.CBL" -e ".cbl"
zowe files download am "Z99998.JCL" -e ".jcl"
```

Ensuite, ouvrez hello.cbl dans votre explorateur de fichiers. Un exemple terminé est illustré dans la figure suivante :

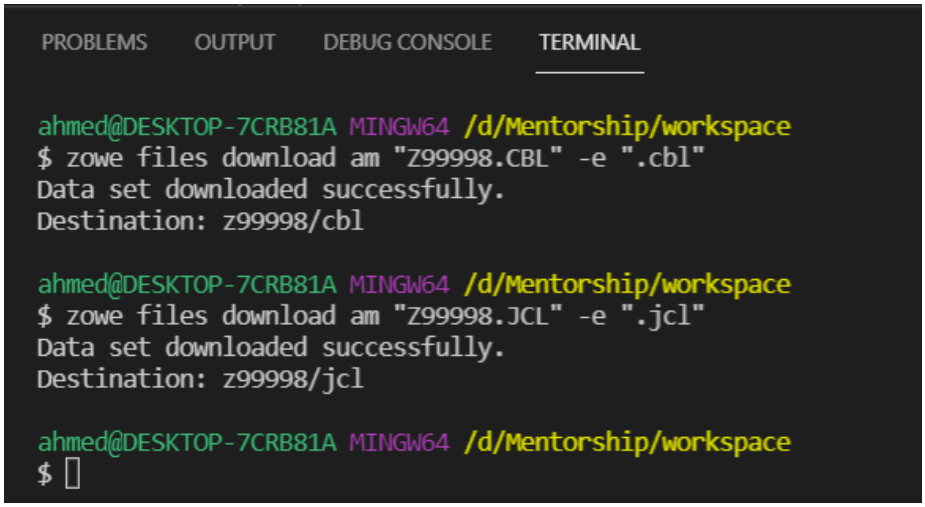


Figure 33. Telechargez et affichez les membres de l'ensemble de données à l'aide

de la CLI

6. Ensuite, nous soumettrons le travail dans le membre Z99998.JCL(HELLO). Pour soumettre le travail, attendez qu'il se termine et affichez tout le contenu du spool, lancez :

```
zowe jobs submit ds "Z99998.JCL(HELLO)" --vasc
```

Nous pourrions également effectuer cette étape au coup par coup pour obtenir la sortie d'un fichier spool spécifique. Voir la figure suivante pour un exemple des commandes à venir. Pour soumettre le travail et attendre qu'il passe à l'état SORTIE, exécutez :

```
zowe jobs submit ds "Z99998.JCL(HELLO)" --wfo
```

Pour répertorier les fichiers spool associés à cet ID de tâche, exécutez :

```
zowe jobs list sfbj JOB09413
```

où JOB09413 a été renvoyé par la commande précédente.

Pour afficher un fichier spool spécifique (COBRUN:SYSOUT), exécutez :

```
zowe jobs view sfbi JOB09413 104
```

où "JOB09413" et "104" sont obtenus à partir des commandes précédentes.

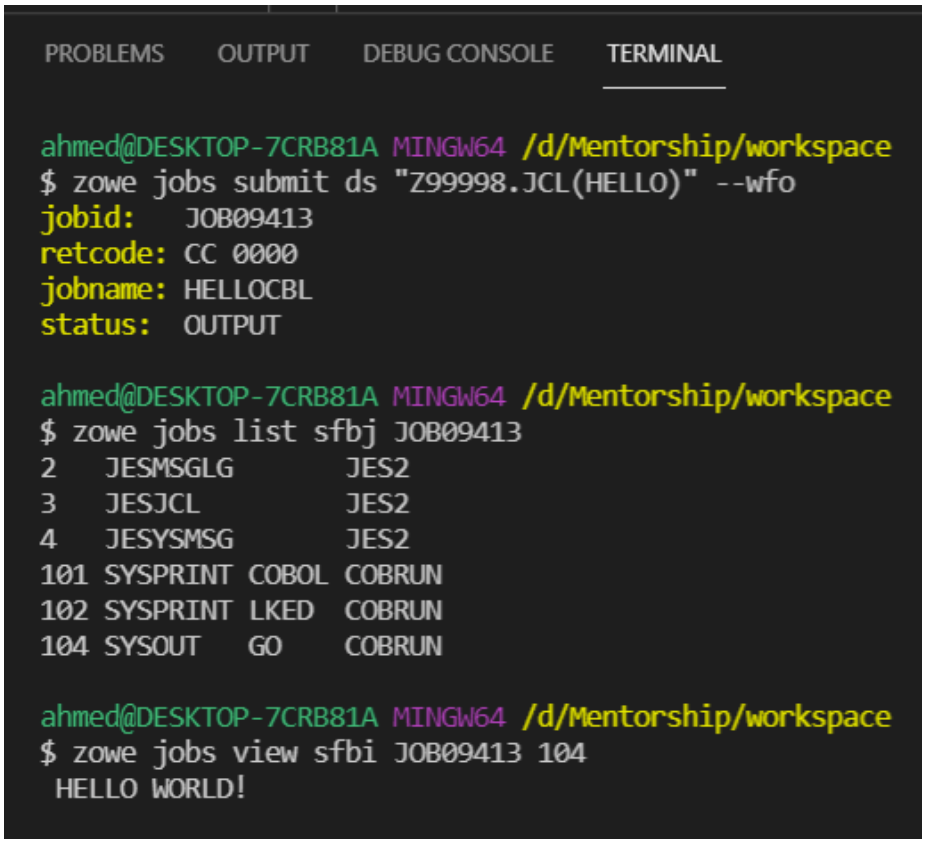


Figure 34. Soumettez une tâche, attendez qu'elle se termine, puis répertoriez les fichiers spoule pour la tâche et affichez un fichier spoule spécifique

Si vous le souhaitez, vous pouvez également facilement soumettre un travail, attendre qu'il se termine et télécharger le contenu du spool à l'aide de la commande suivante (voir la figure suivante pour l'état terminé) :

zowe jobs submit ds "Z99998.JCL(HELLO)" –d .

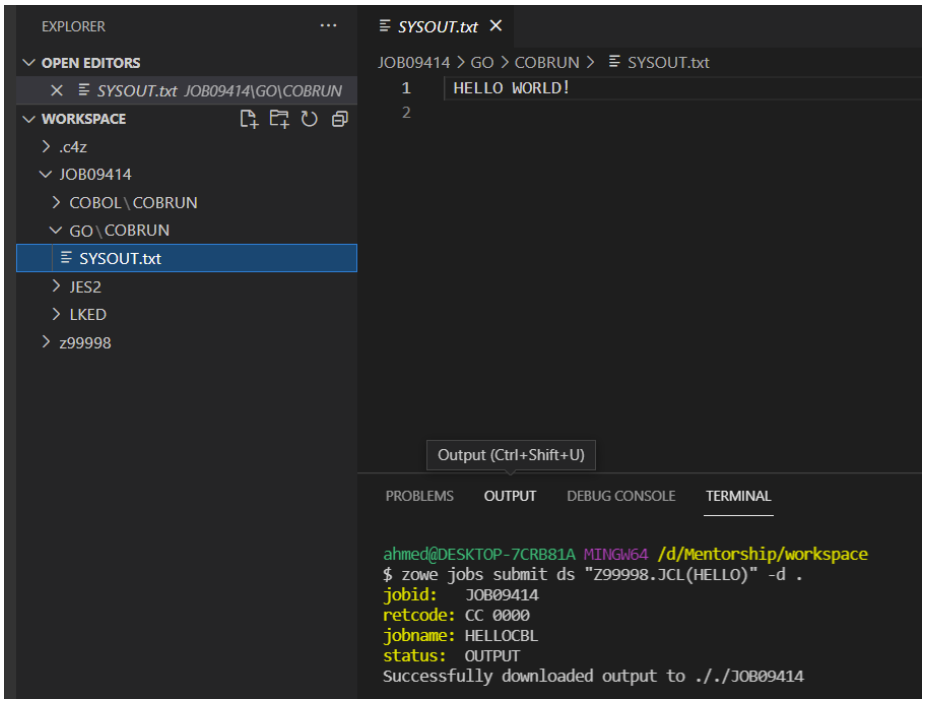


Figure 35. Soumettez un travail, attendez qu'il se termine, téléchargez et affichez les fichiers spoule

La CLI Zowe a été conçue en pensant aux scripts. Par exemple, vous pouvez utiliser l'indicateur ––rfj pour recevoir une sortie au format JSON pour une analyse facile. Voir la figure suivante pour un exemple.

Figure 36. L'indicateur ––rfj permet une utilisation programmatique facile

8.7.2 Zowe CLI - Utilisation par programmation

Dans cette section, nous utiliserons la CLI Zowe par programmation pour automatiser la soumission du JCL pour compiler, lier et exécuter le programme COBOL et télécharger la sortie du spool. Une fois que vous avez le contenu

localement, vous pouvez utiliser n'importe quel nombre d'outils de script et de test distribués pour éliminer le besoin d'examiner manuellement le contenu du spool lui-meme. Historiquement, dans Mainframe, nous utilisons REXX EXEC, etc. pour l'automatisation, mais aujourd'hui, nous allons utiliser la CLI et les outils distribués.

1. Puisque nous avons déjà installé Node et npm, créons simplement un projet de nœud pour notre automatisation. Pour initialiser un projet, lancez « npm init » dans le dossier de votre projet et suivez les invites. Vous pouvez accepter les valeurs par défaut en appuyant simplement sur Entrée. Seuls les champs description et auteur doivent être modifiés. Voir la figure suivante.

```
user@ubuntu-base:~/Mainframe$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (mainframe)
version: (1.0.0)
description: Automation for COBOL Program
entry point: (index.js)
test command:
git repository:
keywords:
author: Michael Bauer
license: (ISC)
About to write to /home/user/Mainframe/package.json:

{
  "name": "mainframe",
  "version": "1.0.0",
  "description": "Automation for COBOL Program",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Michael Bauer",
  "license": "ISC"
}
```

Figure 37. Utilisation de npm init pour créer package.json pour le projet

2. Maintenant que nous avons notre package.json, remplacez simplement le script test par un script clg qui exécute la commande zowe suivante (remplacez Z99998 par votre qualificateur de haut niveau):

```
zowe jobs submit ds 'Z99998.JCL(HELLO)' -d .
```

Vous pouvez nommer le script comme vous le souhaitez. J'ai seulement suggéré clg parce que le CLG dans le proc IGYWCLG (qui est ce que le JCL exploite) signifie compiler, lier, aller. Maintenant, lancez simplement "npm run clg" dans votre terminal pour tirer parti de l'automatisation pour compiler, lier et exécuter le programme COBOL et télécharger la sortie pour examen. Un exemple du package.json terminé et de l'exécution de la commande est illustré dans la figure suivante.

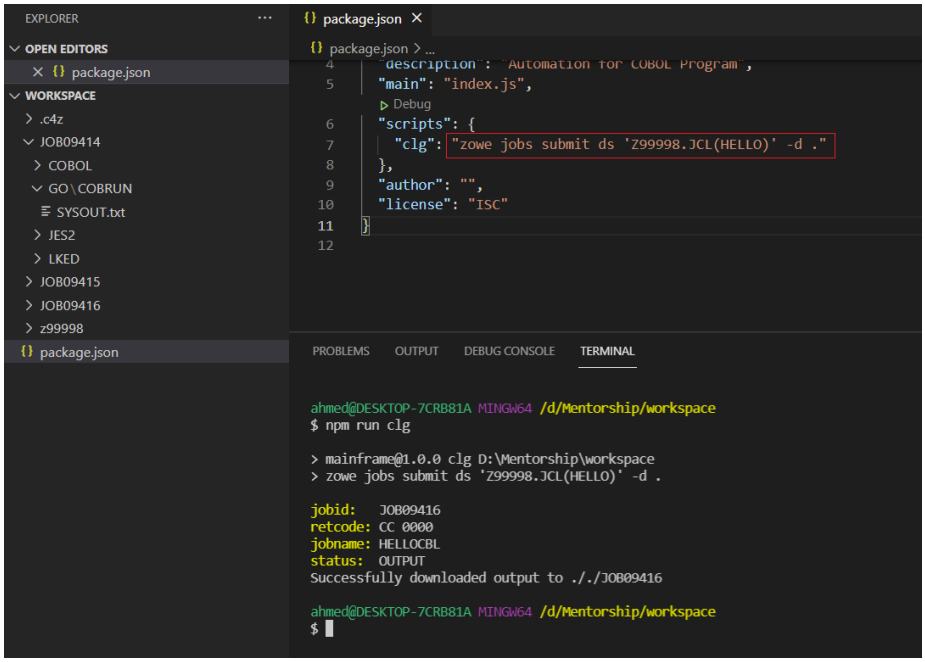


Figure 38. Exécution finale de package.json et npm run clg

3. Si vous préférez un déclencheur graphique, vous pouvez tirer parti du code VS, comme illustré dans la figure suivante. Essentiellement, la CLI vous permet de créer rapidement vos propres boutons pour vos tâches z/OS personnalisées. Vous pouvez également invoquer un script plutôt qu'une seule commande pour s'adapter à des scénarios plus complexes.

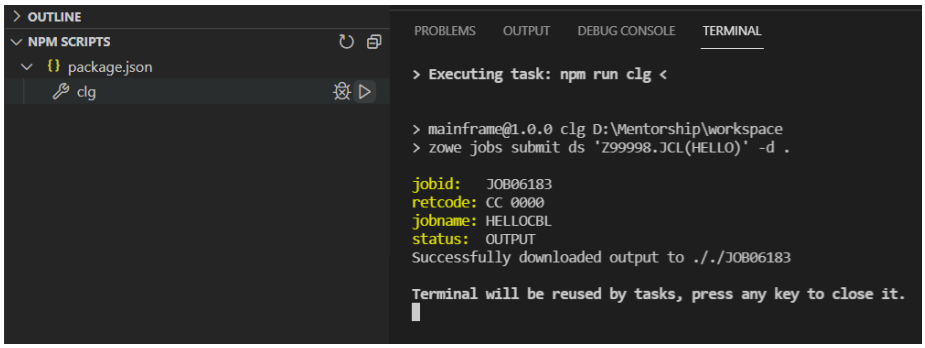


Figure 39. Tâche clg déclenchée via un bouton

9 Data division

La compréhension des variables COBOL et le traitement des variables par programme sont essentiels pour apprendre efficacement le langage COBOL. Un programmeur COBOL expérimenté doit maîtriser les caractéristiques des variables COBOL et le traitement du programme à l'aide des variables présentées dans ce chapitre. L'objectif est d'initier le lecteur aux bases des variables COBOL tout en exposant le lecteur aux nombreuses options avancées de variables COBOL.

À la suite de ce chapitre se trouve un laboratoire disponible pour compiler et exécuter le code source COBOL fourni plus loin dans le chapitre. Après la compilation et l'exécution réussies d'un programme fourni, un deuxième programme COBOL fourni avec une modification mineure est disponible pour la compilation. Le deuxième programme a une erreur intégrée et échouera à la compilation. L'échec de la compilation est l'occasion d'identifier l'erreur associée à la signification des types de données de la clause PICTURE associés à l'opération de l'instruction COMPUTE (discutée dans ce chapitre) et comment résoudre l'erreur.

- **Variables / Data-items**
  - **Variable / Data-item name restrictions and data types**
- **PICTURE clause**
  - **PIC clause symbols and data types**
  - **Coding COBOL variable / data-item names**
  - **PICTURE clause character-string representation**
- **Literals**
  - **Figurative constants**
  - **Data relationships**
  - **Levels of data**
- **MOVE and COMPUTE**
- **Labo**

9.1 Variables / Éléments de données

Une variable COBOL, également appelée élément de données, est un nom et est choisie par le programmeur COBOL. La variable nommée est codée pour contenir des données où la valeur des données peut varier, d'où le terme générique « variable ». Un nom de variable COBOL est également appelé « Nom de données ». Un nom de variable COBOL a des restrictions.

9.1.1 Restrictions de nom de variable / élément de données et types de données

Une liste de restrictions ou de règles de nom de variable COBOL est :

- Ne doit pas être un mot réservé COBOL.
- Ne doit pas contenir d'espace dans le nom.
- Le nom contient des lettres (A-Z), des chiffres (0-9), des traits de soulignement (_) et des tirets (-).
- Longueur maximale de 30 caractères.
- Un tiret ne peut pas apparaître comme premier ou dernier caractère.
- Un trait de soulignement ne peut pas apparaître comme premier caractère.

Remarque : Une liste complète des mots réservés COBOL est disponible dans la référence du langage COBOL d'entreprise, annexe E.

Lorsque le code source COBOL est compilé dans un programme exécutable, le compilateur COBOL s'attend à ce qu'une variable COBOL nommée possède des attributs tels qu'une longueur et un type de données. Pendant l'exécution du programme, la variable représente une zone définie de la mémoire de traitement où l'emplacement mémoire a une longueur maximale et un type de données désigné.

Voici une liste des types de données COBOL les plus courants :

- Numérique (0-9)
- Alphabétique (A-Z), (a-z) ou un espace
- Combinaison alphanumérique numérique et alphabétique

9.2 Clause PICTURE

Le mot réservé COBOL, PICTURE (PIC), détermine la longueur et le type de données d'un nom de variable sélectionné par le programmeur. Les types de données décrits par PIC sont communément appelés clause d'image ou clause pic. Voici quelques clauses pic simples :

- PIC 9 - valeur numérique unique où la longueur est un
- PIC 9(4) - quatre valeurs numériques où la longueur est quatre
- PIC X - valeur alphanumérique unique (caractère) où la longueur est un
- PIC X(4) - quatre valeurs alphanumériques où la longueur est quatre

9.2.1 Symboles de clause PIC et types de données

La longueur maximale d'une clause d'image dépend du type de données et des options du compilateur. Le mot réservé PIC a beaucoup plus de types de données que numérique (PIC 9) et alphanumérique (PIC X). Par exemple, un type de données uniquement alphabétique est défini comme PIC A. Les autres symboles de clause PIC sont :

B E G N P S U V Z 0 / + − , . ∗ CR DB cs

Où cs est un symbole de devise valide tel que le signe dollar ($).

Tous les symboles de clause PIC sont décrits dans le [Manuel de référence du langage Enterprise COBOL for z/OS] (http://publibfp.boulder.ibm.com/epubs/pdf/igy6lr30.pdf).

9.2.2 Codage des noms de variables COBOL / éléments de données

Une clause PIC décrit le type de données d'un nom de variable/élément de données. Le codage d'une variable/donnée se fait dans la DATA DIVISION. Le code COBOL décrivant un nom de variable/élément de données est réalisé à l'aide d'un numéro de niveau et d'une clause d'image.

- Numéro de niveau - Une hiérarchie de champs dans un enregistrement.
- Nom de variable / Nom de l'élément de données - Attribue un nom à chaque champ à référencer dans le programme et doit être unique dans le programme.
- Clause d'image - Pour la vérification du type de données.

La figure 1 ci-dessous est un exemple de numéros de niveau COBOL avec les noms de variables/d'éléments de données et la clause d'image respectifs.

9.2.3 Représentation sous forme de chaîne de caractères de la clause PICTURE

Certains symboles de clause PIC ne peuvent apparaître qu'une seule fois dans une chaîne de caractères de clause PIC, tandis que d'autres peuvent apparaître plusieurs fois. Par exemple:

- La clause PIC pour contenir la valeur 1123,45 est codée comme suit, où le V représente la position décimale.

  PIC 9(4)V99

- La clause PIC pour une valeur telle que 1 123,45 $ est codée comme suit :

  PIC $9,999V99

9.3 Littéraux

Un littéral COBOL est une valeur de données constante, ce qui signifie que la valeur ne changera pas comme le peut une variable. L'instruction COBOL, DISPLAY "HELLO WORLD!",' est un mot reserve COBOL,DISPLAY, suivi d'un litteral ,HELLO WORLD¡

9.3.1 Constantes figuratives

Les constantes figuratives sont des mots réservés qui nomment et font référence à des valeurs constantes spécifiques. Voici des exemples de constantes figurées :

- ZERO, ZEROS, ZEROES
- SPACE, SPACES
- HIGH-VALUE, HIGH-VALUES
- LOW-VALUE, LOW-VALUES
- QUOTE, QUOTE
- NULL, NULL

9.3.2 Relations de données

Les relations entre toutes les données à utiliser dans un programme sont définies dans la DATA DIVISION, à travers un système d'indicateurs de niveau et de numéros de niveau. Un indicateur de niveau, avec son entrée descriptive, identifie chaque fichier dans un programme. Les indicateurs de niveau représentent le niveau le plus élevé de toute hiérarchie de données auquel ils sont associés. Un numéro de niveau, avec son entrée descriptive, indique les propriétés de données spécifiques. Les numéros de niveau peuvent être utilisés pour décrire une hiérarchie de données ; ils peuvent indiquer que ces données ont une finalité particulière.

9.3.2.1 Numéros de niveau Une relation hiérarchique de numéro de niveau structurée est disponible pour toutes les sections DATA DIVISION. La figure 1. montre la relation hiérarchique des numéros de niveau avec les numéros de niveau choisis par le programmeur, les noms de variables et les clauses PIC dans la section Fichier où « 01 PRINT-REC » fait référence au groupe de variables de niveau « 05 » suivant et le « 01 ACCT-FIELDS » fait référence au groupe de variables de niveau “05” suivant. Observez que CLIENT-ADDR de niveau 05 est subdivisé en plusieurs noms de niveau 10. Le code COBOL référençant le nom CLIENT-ADDR inclut les noms de niveau 10.

```
*-------------
 DATA DIVISION.
*-------------
 FILE SECTION.
 FD  PRINT-LINE RECORDING MODE F.
 01  PRINT-REC.
     05  ACCT-NO-O      PIC X(8).
     05  ACCT-LIMIT-O   PIC $$,$$$,$$9.99.
     05  ACCT-BALANCE-O PIC $$,$$$,$$9.99.
* PIC $$,$$$,$$9.99 -- Alternative for PIC on chapter 7.2.3,
* using $ to allow values of different amounts of digits
* and .99 instead of v99 to allow period display on output
     05  LAST-NAME-O    PIC X(20).
     05  FIRST-NAME-O   PIC X(15).
     05  COMMENTS-O     PIC X(50).
* since the level 05 is higher than level 01,
* all variables belong to PRINT-REC (see chapter 7.3.3)
*
 FD  ACCT-REC RECORDING MODE F.
 01  ACCT-FIELDS.
     05  ACCT-NO            PIC X(8).
     05  ACCT-LIMIT         PIC S9(7)V99 COMP-3.
     05  ACCT-BALANCE       PIC S9(7)V99 COMP-3.
* PIC S9(7)v99 -- seven-digit plus a sign digit value
* COMP-3 -- packed BCD (binary coded decimal) representation
     05  LAST-NAME          PIC X(20).
     05  FIRST-NAME         PIC X(15).
     05  CLIENT-ADDR.
         10  STREET-ADDR    PIC X(25).
         10  CITY-COUNTY    PIC X(20).
         10  USA-STATE      PIC X(15).
     05  RESERVED           PIC X(7).
     05  COMMENTS           PIC X(50).
*
 WORKING-STORAGE SECTION.
 01 FLAGS.
```

Figure 1. Relation hiérarchique des numéros de niveau

9.3.3 Niveaux de données

Une fois qu'un enregistrement est défini, il peut être subdivisé pour fournir des références de données plus détaillées comme le montre la figure 1. Un numéro de niveau est un nombre entier à un ou deux chiffres compris entre 01 et 49, ou l'un des trois numéros de niveau spéciaux : 66, 77 ou 88 où les noms de variables se voient attribuer des attributs différents des numéros de niveau 01-49. La relation entre les numéros de niveau au sein d'un élément de groupe définit la hiérarchie des données au sein de ce groupe. Un élément de groupe comprend tous les éléments de groupe et élémentaires qui le suivent jusqu'à ce qu'un numéro de niveau inférieur ou égal au numéro de niveau de ce groupe soit rencontré. Un élément élémentaire est un élément qui ne peut pas être subdivisé davantage. Ces éléments ont une clause PIC car ils réservent un espace de stockage pour l'élément.

9.4 MOVE et COMPUTE

Les instructions de mots réservés MOVE et COMPUTE modifient la valeur des noms de variables. Chaque MOVE illustré à la figure 2 génère un littéral stocké dans un nom de variable à 77 niveaux. L'instruction COMPUTE, également illustrée à la figure 2. , stocke la valeur de « HOURS * TAUX » dans « PAY-GRUT ». instruction COMPUTE.

```
*         COBOL reference format (Figure 1., page 32)
*Columns:
*  1        2         3         4         5         6         7
*89012345678901234567890123456789012345678901234567890123456789012
*<A-><----------------------------B-------------------------------->
*Area                           Area
*<---Sequence Number Area                 Identification Area---->
*-----------------------
 IDENTIFICATION DIVISION.
*-----------------------
 PROGRAM-ID. PAYROL00.
*-------------
 DATA DIVISION.
*-------------
 WORKING-STORAGE SECTION.
****** Variables for the report
* level number
* |  variable name
* |  |           picture clause
* |  |           |
* V  V           V
 77  WHO         PIC X(15).
 77  WHERE       PIC X(20).
 77  WHY         PIC X(30).
 77  RATE        PIC 9(3).
 77  HOURS       PIC 9(3).
 77  GROSS-PAY   PIC 9(5).

* PIC X(15) -- fiftheen alphanumeric characters
* PIC 9(3)  -- three-digit value
*------------------
 PROCEDURE DIVISION.
*------------------
****** COBOL MOVE statements - Literal Text to Variables
     MOVE  "Captain COBOL" TO WHO.
     MOVE "San Jose, California" TO WHERE.
      MOVE "Learn to be a COBOL expert" TO WHY.
      MOVE 19 TO HOURS.
      MOVE 23 TO RATE.
* The string "Captain COBOL" only contains 13 characters,
* the remaining positions of variable WHO are filled with spaces
* The value 19 only needs 2 digits,
* the leftmost position of variable HOURS is filled with zero
****** Calculation using COMPUTE reserved word verb
      COMPUTE GROSS-PAY = HOURS * RATE.
* The result of the multiplication only needs 3 digits,
* the remaining leftmost positions are filled with zeroes
****** DISPLAY statements
      DISPLAY "Name: " WHO.
      DISPLAY "Location: " WHERE
      DISPLAY "Reason: " WHY
      DISPLAY "Hours Worked: " HOURS.
      DISPLAY "Hourly Rate: " RATE.
      DISPLAY "Gross Pay: " GROSS-PAY.
      DISPLAY WHY " from " WHO.
      GOBACK.
```

Figure 2. Exemple de déplacement et de calcul

9.5 Labo

Remarque : Le chargement de tous les segments de cet atelier peut prendre quelques secondes. Si les fichiers ne se chargent pas, appuyez sur le bouton d'actualisation de la liste qui apparaît lorsque vous survolez la barre de section.

1. Affichez le membre du code source COBOL PAYROL00 dans l'ensemble de données 'id'.CBL.
2. Soumettez le membre JCL, PAYROL00, à partir de id.JCL, où id est votre id, liste déroulante. C'est là que id.JCL(PAYROL00) compile et exécute avec succès le PAYROL00. programme.

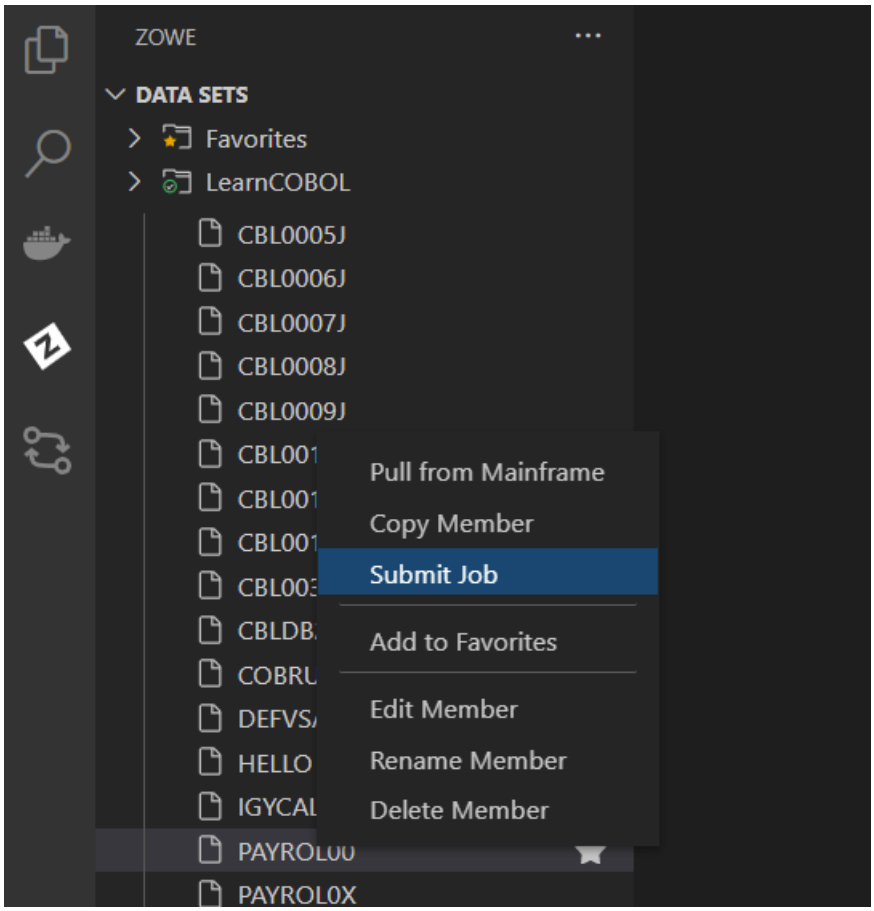
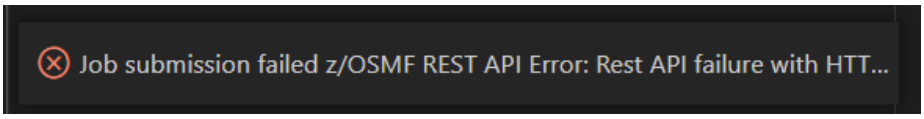


Figure 3. Soumettre le travail PAYROL00

Remarque : Si vous recevez ce message d'erreur après avoir soumis le travail :



C'est parce que vous avez soumis le travail à partir de l'ensemble de données .CBL et non de l'ensemble de données .JCL.

3. Affichez a la fois la compilation et l'exécution de la sortie du travail PAYROL00, referencee dans la Figure 4.

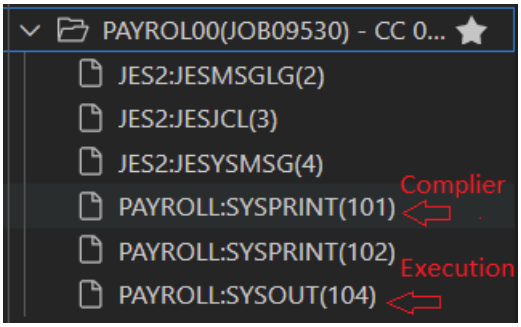


Figure 4. Output PAYROL00

4. Ensuite, affichez le membre du code source COBOL PAYROL0X dans l'ensemble de donnees id.CBL.
5. Affichez et soumettez le membre JCL, PAYROL0X, à partir de la liste déroulante id.JCL. C'est ici que id.JCL(PAYROL0X) compile et exécute le programme PAYROL0X.
6. Affichez la compilation de la sortie du travail PAYROLL0X, notez qu'il n'y a pas de sortie d'exécution.

   Remarquez-vous une différence entre cette compilation et la compilation de travail précédente illustrée à la figure 5. ?

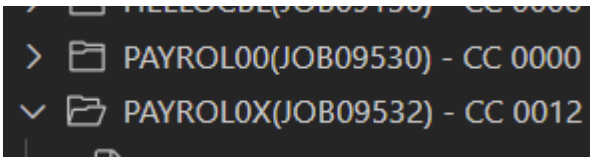


Figure 5. Comparer les compilations de jobs

La différence est le code de retour/code de complétion associe à chaque sortie de job, situé à la fois à côté du nom de la sortie de tâche dans la section JOBS comme indiqué ci-dessus, ou à la fin de la sortie de compilation sous la forme 0Return code ##. Un code de retour de 12 signifie qu'il y a eu une erreur, mais comment savoir quelle était cette erreur ? Continuez à le découvrir !

7. Observez le texte associé à IGYPA3146-S à la ligne 137 dans la sortie du travail (compilation), illustré à la figure 6.

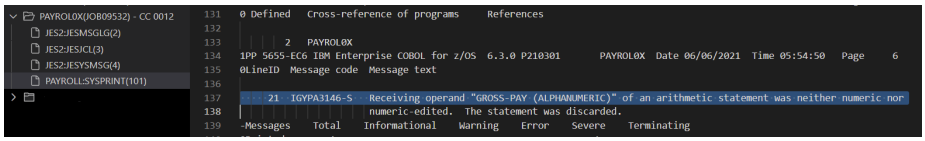


Figure 6. Message IGYPA3146-S

Notez que cette ligne vous indique de vous concentrer sur la clause d'image GROSS-PAY afin d'identifier le problème. Utilisez ces informations, modifiez le code source COBOL PAYROL0X pour corriger l'erreur. Assurez-vous que vous éditez le bon code.

8. Après la modification, soumettez à nouveau le JCL PAYROL0X pour vérifier que le problème a été identifié et corrigé, ce qui entraîne une compilation et une exécution réussies avec un code de retour de zéro, illustré à la figure 7.

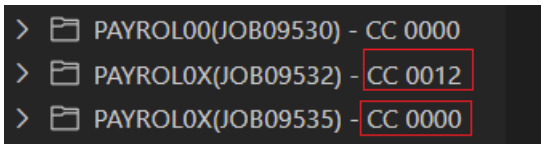


Figure 7. Comparez les codes retour

10 Traitement des tableaux

Cette section présente le concept de tableaux, qui sont une collection d'éléments de données ayant la même description. Les éléments subordonnés sont appelés éléments de table. Une table est l'équivalent COBOL des tableaux.

L'objectif de ce chapitre est de fournir des informations pour que le lecteur soit capable de manipuler les tableaux à l'intérieur des programmes COBOL.

10.1 Définir une table

Pour coder une table, nous devons donner à la table un nom de groupe et définir un élément subordonné que nous répétons n fois.

```
01   TABLE-NAME.
     05   SUBORDINATE-NAME OCCURS n TIMES.
          10   ELEMENT1   PIC X(2).
          10   ELEMENT2   PIC 9(2).
```

Pour coder une table, nous devons donner à la table un nom de groupe et définir un élément subordonné que nous répétons n fois.

```
01   TABLE-NAME.
     05   SUBORDINATE OCCURS n TIMES      PIC X(10).
```

Dans ce cas, TABLE-NAME contient n elements SUBORDINATE, chacun pouvant contenir jusqu'a 10 caracteres alphanumeriques.

Nous pouvons egalement imbriquer plusieurs elements OCCURS pour creer un tableau de dimensions supplementaires, dans la limite de sept dimensions. Notez l'exemple ci-dessous :

```
01   PROGRAM-DETAILS.
     05   PROGRAM-DEGREE                    PIC X(32).
     05   COURSE-DETAILS OCCURS 10 TIMES.
          10   COURSE-NAME                   PIC X(32).
          10   INSTRUCTOR-ID                 PIC 9(10).
          10   ASSIGNMENT-DETAILS OCCURS 8 TIMES.
               15   ASSIGNMENT-NAME          PIC X(32).
               15   ASSIGMMENT-WEIGHTAGE     PIC 9(03).
```

Ici, nous definissons un programme diplomant qui comporte 10 cours et chaque cours comportera 8 devoirs. Que se passe-t-il si nous ne savons pas combien de fois un element de table se produira ? Pour resoudre ce probleme, nous pouvons utiliser une table de longueur variable, en utilisant la clause OCCURS DEPENDING ON (ODO) que nous detaillerons dans une section ulterieure.

10.2 Référence à un item de la table

Alors qu'un element de table a un nom collectif, les elements individuels qu'il contient n'ont pas de nom unique. Pour faire reference a un element, nous pouvons utiliser un indice, un index ou une combinaison des deux.

10.2.1 Indice

L'indice utilise le nom de donnees de l'element de table, ainsi que son numero d'occurrence (appele indice). Le numero d'indice le plus bas possible est 1, qui definit la premiere occurrence d'un element de table. Nous pouvons egalement utiliser un nom litteral ou de donnees comme indice. Notez que si vous utilisez un nom de donnees, il doit s'agir d'un entier numerique elementaire.

```
01   TABLE-NAME.
     05   TABLE-ELEMENT OCCURS 3 TIMES      PIC X(03) VALUE "ABC".
...
     MOVE "DEF" TO TABLE-ELEMENT (2)
```

Dans l'exemple ci-dessus, le deuxieme element de table contiendra "DEF" au lieu de "ABC".

10.2.2 Indexation

Alternativement, nous pouvons creer un index en utilisant la phrase INDEXED BY de la clause OCCURS. Cet index est ajoute a l'adresse de la table pour localiser un element (comme un deplacement depuis le debut de la table). Par exemple,

```
05   TABLE-ELEMENT OCCURS 10 TIMES INDEXED BY INX-A
PIC X(03).
```

Ici, INX-A est un nom d'index. Le compilateur calculera la valeur dans l'index comme le nombre d'occurrence moins 1 multiplie par la longueur de l'element de table. Ainsi, par exemple, pour la deuxieme occurrence de TABLE-ELEMENT, la valeur binaire contenue dans INX-A est (2-1) * 3, ou 3.

S'il vous arrive d'avoir une autre table avec le meme nombre d'elements de table de la meme longueur, vous pouvez utiliser un nom d'index comme reference pour les deux tables.

Nous pouvons egalement definir une donnee d'index a l'aide de la clause USAGE IS INDEX. Ces elements de donnees d'index peuvent etre utilises avec n'importe quelle table. Par exemple,

```
77   INX-B   USAGE IS INDEX.
...
     SET INX-A TO 10.
     SET INX-B TO INX-A.
```

```
PERFORM VARYING INX–A FROM 1 BY 1 UNTIL INX–A > INX–B
    DISPLAY TABLE–ELEMENT (INX–A)
    ...
END–PERFORM.
```

Le nom d'index INX-A est utilise pour parcourir la table TABLE-ELEMENT, tandis que INX-B est utilise pour contenir l'index du dernier element de la table. En faisant cela, nous minimisons le calcul des decalages et aucune conversion ne sera necessaire pour la condition UNTIL.

On peut aussi incrementer ou decrementer un nom d'index par une donnee entiere elementaire. Par exemple,

```
SET INX–A DOWN BY 3
```

Le nom d'index INX-A est utilise pour parcourir la table TABLE-ELEMENT, tandis que INX-B est utilise pour contenir l'index du dernier element de la table. En faisant cela, nous minimisons le calcul des decalages et aucune conversion ne sera necessaire pour la condition UNTIL.

On peut aussi incrementer ou decrementer un nom d'index par une donnee entiere elementaire. Par exemple,

```
01  TABLE–2D.
    05  TABLE–ROW OCCURS 2 TIMES INDEXED BY INX–A.
        10  TABLE–COL OCCURS 5 TIMES INDEXED BY INX–B
PIC X(4).
```

Supposons que nous codions l'index suivant :

```
TABLE–COL (INX–A + 2, INXB – 1)
```

Ceci provoquera le calcul du deplacement à l'element TABLE-COL:

```
(contents of INX–A) + (20 * 2) + (contents of INX–B) – (4 * 1)
```

Le calcul est base sur la longueur des elements. Chaque occurrence de TABLE-ROW a une longueur de 20 octets (5 * 4) et chaque occurrence de TABLE-COL a une longueur de 4 octets.

10.3 Chargement d'une table avec des donnees

Il existe de nombreuses façons de charger une table. La première consiste à charger la table de maniere dynamique, à partir d'un ecran, d'un fichier ou d'une base de donnees. Nous pouvons egalement utiliser la clause REDEFINES sur les valeurs de champ codees en dur avec une clause OCCURS. La troisième methode consiste à utiliser l'instruction INITIALIZE, et enfin, nous pouvons egalement utiliser la clause VALUE lors de la definition de la table.

10.3.1 Charger une table dynamiquement

Pour charger une table dynamiquement, nous devons utiliser l'instruction PERFORM avec indice ou indexation. Ce faisant, nous devons nous assurer que les donnees ne depassent pas l'espace alloue à la table. Nous discuterons de la gestion des fichiers et de l'utilisation de la clause PERFORM dans un chapitre ulterieur. Par exemple,

```
PROCEDURE DIVISION
    ...
    PERFORM READ-FILE.
    PERFORM VARYING SUB FROM 1 BY 1 UNTIL END-OF-FILE
        MOVE DATA TO WS-DATA(SUB)
        PERFORM READ-FILE
    END-PERFORM.
```

Dans l'exemple ci-dessus, nous executons un paragraphe qui lit les fichiers, puis nous parcourrons chaque ligne du fichier jusqu'à la fin, en mettant chaque valeur dans le tableau.

10.3.2 REDEFINES une valeur codee en dur

Considerons l'exemple suivant,

```
WORKING-STORAGE SECTION.
01  NUMBER-VALUES.
    05  FILLER  PIC X(05) VALUE "One  "
    05  FILLER  PIC X(05) VALUE "Two  "
    05  FILLER  PIC X(05) VALUE "Three"
    05  FILLER  PIC X(05) VALUE "Four "
    05  FILLER  PIC X(05) VALUE "Five "

01  NUMBER-TABLES REDEFINES NUMBER-VALUES.
    05  WS-NUMBER  PIC X(05)  OCCURS 5 TIMES.
```

Ici, nous prenons les valeurs codees en dur des nombres 1 à 5 et les chargeons dans une table à l'aide d'une clause REDEFINES.

10.3.3 INITIALISER une table

Nous pouvons egalement utiliser l'instruction INITIALIZE pour charger des donnees dans une table. Le tableau sera traite comme un element de groupe et chaque element de donnees elementaire qu'il contient sera reconnu et traité. Par exemple, supposons que nous ayons le tableau suivant :

```
01  TABLE-ONE.
    05  TABLE-ELEMENT OCCURS 10 TIMES.
        10  NUMBER-CODE    PIC 9(02) VALUE 10.
        10  ITEM-ID        PIC X(02) VALUE "R3".
```

Ici, nous avons un tableau qui contient 10 elements, chacun avec son propre NUMBER-CODE (avec une valeur de 10) et ITEM-ID (avec une valeur de “R3”).

On peut deplacer la valeur 3 dans chacune des donnees numeriques elementaires et la valeur “X” dans chacune des donnees alphanumeriques elementaires du tableau :

```
INITIALIZE TABLE-ONE REPLACING NUMERIC DATA BY 3.
INITIALIZE TABLE-ONE REPLACING ALPHANUMERIC DATA BY "X".
```

Apres avoir execute les deux instructions INITIALIZE, NUMBER-CODE contiendra la valeur 3, tandis que ITEM-ID contiendra la valeur “X”.

10.3.4 Attribution de valeurs à l’aide de la clause VALUE

Si une table doit contenir des valeurs stables, nous pouvons les definir lors de la definition de la table. Prenons par exemple les TABLES SEMAINE-JOUR et TABLE-UN des sections precedentes. Les deux ont des valeurs assignees lorsqu’elles sont definies. Voici d’autres exemples :

```
01   TABLE-TWO                                  VALUE "1234".
     05   TABLE-TWO-DATA OCCURS 4 TIMES         PIC X.
```

Dans l’exemple ci-dessus le group data item TABLE-TWO utilise une clause VALUE qui initialise chaque élément de la TABLE-TWO-DARA. Donc après l’initialisation TABLE-TWO-DATA(1) contiendra la valuer alphinumérique ‘1’, TABLE-TWO-DATA(2) la valeur ‘2’ etc.

10.4 Tableaux de longueur variable

Si nous ne savons pas avant l’exécution combien de fois un élément de table se produira, nous pouvons définir une table de longueur variable à l’aide de la clause OCCURS DEPENDING ON (ODO).

```
X OCCURS 1 TO 10 TIMES OF Y
```

Dans l’exemple ci-dessus, X est le sujet ODO et Y est l’objet ODO.

Plusieurs facteurs affectent la manipulation réussie des enregistrements de longueur variable :

- Calcul correct des longueurs d’enregistrement

Ici, la longueur de la partie variable est le produit de l’objet de la phrase DEPENDING ON et de la longueur du sujet de la clause OCCURS.

- Conformité des données dans l’objet de la clause OCCURS DEPENDING ON à sa clause PICTURE

Nous devons nous assurer que l'objet ODO spécifie correctement le nombre d'occurrences des éléments de la table, sinon le programme pourrait se terminer anormalement.

L'exemple suivant montre comment utiliser une clause OCCURS DEPENDING ON :

```
WORK-STORAGE SECTION
01 MAIN ZONE.
     03 REC-1.
          05 FIELD-1 PIC 9.
          05 FIELD-2 OCCURS 1 TO 5 TIMES
               ACCORDING TO CHAMP-1 PIC X (05).
01 REC-2.
     03 REC-2-DATA PIC X (50).
```

Si nous déplaçons REC-1 vers REC-2, la longueur de REC-1 sera déterminée immédiatement à l'avance en utilisant la valeur actuelle de FIELD-1. Si FIELD-1 n'est pas conforme à sa clause PICTURE, le résultat est imprévisible. Ainsi, nous devons nous assurer que l'objet ODO (FIELD-1) a la bonne valeur avant de déplacer REC-1 vers REC-2.

En revanche, si nous passons à REC-1, la longueur est déterminée en utilisant le nombre maximum d'occurrences. Cependant, si REC-1 est suivi d'un groupe à localisation variable, l'objet ODO sera utilisé dans le calcul de la longueur réelle de REC-1. Un exemple d'un tel cas est fourni ci-dessous :

```
01   MAIN-AREA.
     03   REC-1.
          05   FIELD-1                          PIC 9.
          05   FIELD-3                          PIC 9.
          05   FIELD-2 OCCURS 1 TO 5 TIMES
               DEPENDING ON FIELD-1              PIC X(05).
     03   REC-2.
          05   FIELD-4 OCCURS 1 TO 5 TIMES
               DEPENDING ON FIELD-3              PIC X(05).
```

Ainsi, dans le cas ci-dessus, la valeur de l'objet ODO doit être définie avant d'utiliser l'élément de groupe comme champ de réception.

10.5 Recherche dans une table

Il existe deux techniques pour rechercher une table : série et binaire.

Une recherche binaire peut être plus efficace qu'une recherche en série, mais elle nécessite que les éléments du tableau soient déjà triés.

10.5.1 Recherche en série

Nous pouvons effectuer une recherche en série en utilisant l'instruction SEARCH. La recherche commencera au réglage d'index actuel et se poursuivra jusqu'à ce que la condition de la phrase WHEN soit remplie. Pour modifier le paramètre d'index, nous pouvons utiliser l'instruction SET. S'il y a plusieurs conditions dans la phrase WHEN, la recherche se terminera lorsque l'une des conditions sera satisfaite et l'index continuera à pointer vers l'élément qui a satisfait la condition.

Par exemple, supposons que nous ayons une liste de noms :

```
77   PEOPLE-SEARCH-DATA                    PIC X(20).
01   PEOPLE-SERIAL.
     05   PEOPLE-NAME   OCCURS 50 TIMES
          INDEXED BY PL-IDX                PIC X(20).
...
PROCEDURE-DIVISION.
     ...
     SET PL-IDX TO 1.
     SEARCH PEOPLE-NAME VARYING PL-IDX
          AT END DISPLAY "Not found"
          WHEN PEOPLE-SEARCH-DATA = PEOPLE-NAME(PL-IDX)
               DISPLAY "Found".
```

Le code ci-dessus recherchera la liste des noms à partir d'un index de 1. S'il a trouvé le contenu de PEOPLE-SEARCH-DATA, il affichera "Found", sinon il affichera "Not found".

Pour un cas d'utilisation plus complexe, nous pouvons également utiliser des instructions SEARCH imbriquées. Nous devrons délimiter chaque instruction SEARCH imbriquée avec END-SEARCH.

10.5.2 Recherche binaire

Pour effectuer une recherche binaire, nous pouvons utiliser une instruction SEARCH ALL. Nous n'avons pas besoin de définir l'index, mais il utilisera celui associé dans la clause OCCURS. Pour utiliser l'instruction SEARCH ALL, la table doit spécifier les expressions ASCENDING ou DESCENDING KEY de la clause OCCURS, ou les deux, et elle doit être ordonnée sur la clé spécifiée.

En utilisant la phrase WHEN, vous pouvez tester n'importe quelle touche nommée dans les phrases ASCENDING ou DESCENDING KEY. Le test doit être une condition égale à, et la phrase WHEN doit spécifier soit une clé, soit un nom de condition associé à la clé.

Par exemple, supposons que nous ayons une liste de noms triés par ordre croissant :

```
77   PEOPLE-SEARCH-DATA                    PIC X(20).
01   PEOPLE-TABLE-BINARY.
     05   PEOPLE-NAME   OCCURS 50 TIMES
          ASCENDING KEY IS PEOPLE-NAME
          INDEXED BY PL-IDX                PIC X(20).
...
PROCEDURE-DIVISION.
     ...
     SEARCH ALL PEOPLE-NAME
          AT END DISPLAY "Not found"
          WHEN PEOPLE-SEARCH-DATA = PEOPLE-NAME(PL-IDX)
               DISPLAY "Found".
```

Le code ci-dessus recherchera la liste des noms triés par ordre alphabétique. S'il a trouvé le contenu de PEOPLE-SEARCH-DATA, il affichera "Found", sinon, il affichera "Not found".

Laboratoire

Remarque : Le chargement de tous les segments de cet atelier peut prendre quelques secondes. Si les fichiers ne se chargent pas, appuyez sur le bouton d'actualisation de la liste qui apparaît lorsque vous survolez la barre de section.

1. Affichez le membre du code source SRCHSER COBOL dans l'ensemble de données 'id'.CBL.
2. Soumettez le membre JCL, SRCHSERJ, à partir de id.JCL, où id est votre id, la liste déroulante. C'est là que id.JCL(SRCHSERJ) compile et exécute avec succès le programme SRCHSER.
3. Affichez à la fois la compilation et l'exécution de la sortie du travail SRCHSERJ.
4. Ensuite, affichez le membre du code source SRCHBIN COBOL dans l'ensemble de données id.CBL.
5. Affichez et soumettez le membre JCL, SRCHBINJ, à partir de la liste déroulante id.JCL. C'est là que id.JCL(SRCHBINJ) compile et exécute le programme SRCHBIN.
6. Affichez la compilation et l'exécution de la sortie du travail SRCHBINJ.
7. Comparez SRCHSER avec SRCHBIN. Remarquez-vous les différences?

   une. Observez comment les tables sont définies.

   b. Observez comment les tables sont chargées à partir de l'ensemble de données id.DATA.

   c. Observez les instructions SEARCH et SEARCH ALL.

11 La gestion des fichiers

Le chapitre et l'atelier precedents se sont concentres sur les variables et le deplacement de litteraux dans des variables, puis sur l'ecriture du contenu des variables a l'aide de l'instruction COBOL DISPLAY. Cette section presente la lecture des enregistrements de fichiers dans des variables, le deplacement des variables vers les variables de sortie et l'ecriture des variables de sortie dans un fichier different. Un programme COBOL simple pour lire chaque enregistrement d'un fichier et ecrire chaque enregistrement dans un fichier different est utilise pour illustrer le code COBOL necessaire pour lire les enregistrements a partir d'une source de donnees externe d'entree et ecrire les enregistrements dans une source de donnees externe de sortie.

Un programmeur COBOL experimente peut repondre a la question : " Comment un programme COBOL d'entreprise lit-il des donnees a partir d'une source de donnees externe d'entree et ecrit-il des donnees dans une source de donnees externe de sortie ? " L'objectif de ce chapitre est de fournir suffisamment d'informations completes pour que le lecteur puisse repondre a cette question.

- **Code COBOL utilise pour le traitement sequentiel des fichiers**
  - **Entrees et sorties COBOL**
  - **paragraphe FILE-CONTROL**
  - **Source de donnees externe COBOL**
  - **Ensembles de donnees, enregistrements et champs**
  - **Blocs**
  - **clause AFFECTATION**
- **PROCEDURE DIVISION traitement sequentiel des fichiers**
  - **Entree et sortie ouvertes pour lecture et ecriture**
  - **Fermer entree et sortie**
- **Techniques de programmation COBOL pour lire et ecrire des enregistrements de maniere sequentielle**
  - **Execution du paragraphe READ-NEXT-RECORD**
  - **paragraphe LIRE-ENREGISTRER**
  - **paragraphe ECRIRE-ENREGISTRER**
  - **Traitement iteratif du paragraphe READ-NEXT-RECORD**
- **Laboratoire**

11.1 Code COBOL utilise pour le traitement sequentiel des fichiers

Le code COBOL utilise pour le traitement sequentiel des fichiers implique :

- ENVIRONMENT DIVISION.
  - clauses SELECT
  - AFFECTATION des clauses
- DATA DIVISION.
  - Declarations FD
- PROCEDURE DIVISION.
  - Declarations OUVERTES
  - FERMER les declarations
  - Declaration LIRE DANS
  - instruction WRITE FROM

11.1.1 Entrees et sorties COBOL

L' ENVIRONMEN DIVISION et la DATA DIVISION decrivent les entrees et les sorties utilisees dans la logique du programme DIVISION PROCeDURE. Les chapitres precedents ont introduit des descriptions de variables dans la DATA DIVISION et les litteraux ont ete deplaces dans les variables definies. Le role de la DIVISION ENVIRONNEMENT et plus particulierement, le paragraphe INPUT-OUTPUT SECTION, FILE-CONTROL introduit l'acces a des sources de donnees externes ou les donnees provenant de sources externes sont deplacees dans des variables definies.

11.1.2 Paragraphe FILE-CONTROL

Le paragraphe FILE-CONTROL associe chaque nom de fichier interne COBOL a un nom de jeu de donnees externe. Dans le paragraphe FILE-CONTROL, la clause SELECT cree un nom de fichier interne et la clause ASSIGN cree un nom d'ensemble de donnees externe. La figure 1. montre le nom de fichier interne PRINT-LINE associe au nom de jeu de donnees externe PRTLINE et le nom de fichier interne ACCT-REC associe au nom de jeu de donnees externe ACCTREC. La section intitulee Assign Clause explique plus en detail la relation SELECT ASSIGN TO.

```
*---------------------
 ENVIRONMENT DIVISION.
*---------------------
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT PRINT-LINE ASSIGN TO PRTLINE.
     SELECT ACCT-REC   ASSIGN TO ACCTREC.
*SELECT clause creates an internal file name
*ASSIGN clause creates a name for an external data source,
*which is associated with the JCL DDNAME used by the z/OS
*e.g. ACCTREC is linked in JCL file CBL0001J to &SYSUID..DATA
*where &SYSUID. stands for Your z/OS user id
*e.g. if Your user id is Z54321,
*the data set used for ACCTREC is Z54321.DATA
```

Figure 1. FILE-CONTROL

Alors que SELECT donne un nom a un fichier interne et ASSIGN donne un nom au nom de l'ensemble de donnees externe, un programme COBOL a besoin de plus d'informations sur les deux. Le compilateur COBOL recoit plus d'informations sur les deux dans la DATA DIVISION, FILE SECTION.

Le mot reserve COBOL 'FD' est utilise pour donner au compilateur COBOL plus d'informations sur les noms de fichiers internes dans FILE-SECTION. Le code sous l'instruction FD est le cliche d'enregistrement. Le cliche d'enregistrement se compose de numeros de niveau, de noms de variables, de types de donnees et de longueurs, comme le montre la figure 2.

```
*-------------
 DATA DIVISION.
*-------------
 FILE SECTION.
 FD  PRINT-LINE RECORDING MODE F.
 01  PRINT-REC.
     05  ACCT-NO-O      PIC X(8).
     05  ACCT-LIMIT-O   PIC $$,$$$,$$9.99.
     05  ACCT-BALANCE-O PIC $$,$$$,$$9.99.
* PIC $$,$$$,$$9.99 -- Alternative for PIC on chapter 7.2.3,
* using $ to allow values of different amounts of digits
* and .99 instead of v99 to allow period display on output
     05  LAST-NAME-O    PIC X(20).
     05  FIRST-NAME-O   PIC X(15).
     05  COMMENTS-O     PIC X(50).
* since the level 05 is higher than level 01,
* all variables belong to PRINT-REC (see chapter 7.3.3)
```

Figure 2. FILE-SECTION

11.1.3 Source de donnees externe COBOL

Le code source Enterprise COBOL se compile et s'execute sur le materiel IBM Z Mainframe ou z/OS est le logiciel du systeme d'exploitation. z/OS stocke les donnees dans des ensembles de donnees et des fichiers Unix. z/OS inclut de nombreuses methodes de stockage de donnees. Ce chapitre se concentrera sur la methode de stockage de donnees sequentiel z/OS. Un ensemble de donnees

sequentiel est une collection d'enregistrements.

11.1.4 Ensembles de donnees, enregistrements et champs

Un ensemble de donnees a de nombreux enregistrements. Un enregistrement est une seule ligne dans l'ensemble de donnees et a une longueur definie. Chaque enregistrement peut etre subdivise en champs ou chaque champ a une longueur definie. Par consequent, la somme de toutes les longueurs de champ serait egale a la longueur de l'enregistrement. Observez la figure 3.

11.1.5 Blocs

Chaque enregistrement lu par le programme peut entrainer un acces au stockage sur disque. Un programme lit generalement 1 enregistrement a la fois dans un ordre sequentiel jusqu'a ce que tous les enregistrements soient lus. Lorsqu'un enregistrement est lu, l'enregistrement extrait du disque est stocke en memoire pour l'acces au programme. Lorsque chaque lecture d'enregistrement suivante necessite de recuperer l'enregistrement a partir du disque, les performances du systeme sont affectees negativement. Les enregistrements peuvent etre bloques lorsqu'un bloc est un groupe d'enregistrements. Le resultat est que lorsque le premier enregistrement est lu, un bloc entier d'enregistrements est lu en memoire en supposant que le programme lira les deuxieme, troisieme, etc. enregistrements, evitant ainsi les recuperations de disque inutiles et les performances systeme negatives. La memoire contenant un enregistrement ou un bloc d'enregistrements a lire par le programme est appelee tampon. La clause COBOL BLOCK CONTAINS est disponible pour specifier la taille du bloc dans le tampon. Observez la figure 3.

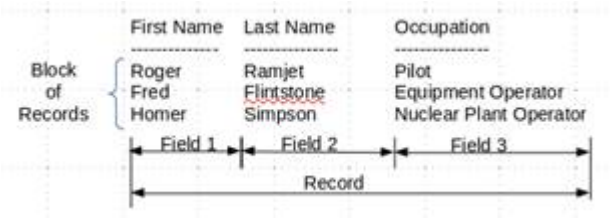


Figure 3. Enregistrements, champs et blocs

11.1.6 Clause ASSIGN

Alors que le nom de la clause SELECT est un nom de fichier interne, le nom de la clause ASSIGN decrit une source de donnees externe au programme. z/OS utilise des operations JCL, Job Control Language, pour indiquer au systeme quel programme charger et executer, suivis des noms d'entree et de sortie requis par le programme. Les noms d'entree et de sortie JCL sont des DDNAME d'appel. L'instruction JCL DDNAME inclut une operation JCL DD ou DD est l'abreviation de Data Definition. Sur la meme instruction DDNAME se trouve le nom de l'ensemble de donnees controle par le systeme.

Le code COBOL “SELECT ACCT-REC ASSIGN TO ACCTREC” necessite un JCL DDNAME ACCTREC avec un DD redirigeant ACCTREC vers un nom de jeu de donnees controle par z/OS, MY.DATA. Le programme COBOL est illustre dans l'exemple 1.

Le but de la redirection d'ACCT-REC, via ASSIGN TO, vers JCL DDNAME, ACCTREC est la flexibilite. ACCT-REC est utilise dans le programme lui-meme, ACCTREC est un pont vers JCL, illustre dans l'exemple 1. , et une instruction DD JCL relie ACCTREC a un ensemble de donnees reel, illustre dans l'exemple 2. Cette flexibilite permet au meme programme COBOL d'acceder a un source de donnees differente avec une simple modification JCL evitant de devoir changer le code source pour referencer la source de donnees alternative.

```
SeLECTIONNER ACCT-REC ASSIGN TO **ACCTREC**
```

Exemple 1. Programme COBOL

L'instruction JCL requise par le programme COBOL compile pendant l'execution pour rediriger ACCTREC vers l'ensemble de donnees controle par MY.DATA z/OS est illustree dans l'exemple 2.

```
//**ACCTREC** DD DSN=MY.DATA,DISP=SHR
```

Exemple 2. Instruction JCL

En resume, ACCT-REC est le nom de fichier interne. ACCTREC est le nom externe ou un JCL DDNAME doit correspondre au nom COBOL ASSIGN TO ACCTREC. Lors de l'execution du programme, l'instruction JCL ACCTREC DDNAME est redirigee vers le nom de l'ensemble de donnees identifie immediatement apres l'operation JCL DD.

ACCT−REC >>> ACCTREC >>> //ACCTREC >>> DD >>> MY.DATA

Par consequent, le nom de fichier ACCT-REC interne COBOL lit les enregistrements de donnees a partir de l'ensemble de donnees sequentiel nomme MY.DATA.

JCL est une competence technique z/OS distincte. L'introduction a COBOL explique juste assez sur JCL pour comprendre comment le nom de fichier interne COBOL localise le nom de l'ensemble de donnees sequentiel externe. Pour en savoir plus sur JCL, visitez le centre de connaissances IBM :

https://www.ibm.com/docs/en/zos-basic-skills?topic=collection-basic-jcl-concepts

11.2 PROCEDURE DIVISION traitement sequentiel des fichiers

Lors de l'execution du programme COBOL, SELECT ASSIGN TO a JCL DDNAME est obligatoire. Si le nom ASSIGN TO ne s'associe pas a un JCL

DDNAME de la meme orthographe, lors de l'execution, une erreur d'execution du programme se produit lorsque l'operation OPEN est tentee. Un message apparait dans la sortie d'execution indiquant que le DDNAME n'a pas ete trouve. READ et WRITE dependent de la reussite de l'operation OPEN. Le compilateur ne peut pas detecter l'erreur d'execution car le compilateur ne connait pas le nom de l'ensemble de donnees JCL DDNAME d'execution soumis a OPEN, READ ou WRITE. Le mappage du FD, File Descriptor, des champs d'enregistrement de donnees necessite qu'un OPEN reussi soit rempli par les operations de LECTURE ou d'eCRITURE suivantes.

11.2.1 Ouvrir l'entree et la sortie pour la lecture et l'ecriture

Les entrees et sorties COBOL doivent etre ouvertes pour connecter le nom interne selectionne au nom externe attribue. La figure 4. ouvre le nom de fichier ACCT-REC comme entree de programme et le nom de fichier PRINT-LINE comme sortie de programme.

```
*-----------------
 PROCEDURE DIVISION.
*-----------------
 OPEN-FILES.
     OPEN INPUT  ACCT-REC.
     OPEN OUTPUT PRINT-LINE.
 OPEN-FILES-END.
*OPEN-FILES-END -- consists of an empty paragraph suffixed by
*-END that ends the past one and serves as a visual delimiter
*
```

Figure 4. OPEN-FILES

11.2.2 Fermer l'entree et la sortie

Les entrees et sorties COBOL doivent etre fermees a la fin du programme ou mieux encore lorsque le programme a termine de lire ou d'ecrire dans le nom de fichier interne. La figure 5. ferme le nom de fichier interne ACCT-REC et le nom de fichier interne PRINT-LINE, puis arrete le traitement, STOP RUN.

```
*
 CLOSE-STOP.
      CLOSE ACCT-REC.
      CLOSE PRINT-LINE.
      STOP RUN.
*
```

Figure 5. CLOSE-STOP

11.3 Techniques de programmation COBOL pour lire et ecrire des enregistrements de maniere sequentielle

Lors de la lecture d'enregistrements, le programme doit d'abord verifier s'il n'y a plus d'enregistrements a lire ou s'il n'y a plus d'enregistrements a lire. Si un enregistrement existe, les champs de l'enregistrement lu remplissent les noms de variables definis par la clause FD. COBOL utilise une instruction PERFORM pour l'iteration. En programmation informatique, iteratif est utilise pour decrire une situation dans laquelle une sequence d'instructions ou d'instructions peut etre executee plusieurs fois. Un passage dans la sequence s'appelle une iteration. L'execution iterative est egalement appelee boucle. Dans d'autres langages de programmation, les instructions 'DO' ou 'FOR' sont utilisees pour l'execution iterative. COBOL utilise une instruction PERFORM pour une execution iterative. La figure 6. montre quatre noms de paragraphe choisis par le programmeur dans la PROCEDURE DIVISION.

- READ-NEXT-RECORD
- CLOSE-STOP
- READ-RECORD
- WRITE-RECORD

READ-NEXT-RECORD execute a plusieurs reprises READ-RECORD et WRITE-RECORD jusqu'a ce qu'un dernier enregistrement soit rencontre. Lorsque le dernier enregistrement est rencontre, CLOSE-STOP est execute pour arreter le programme.

```
 READ-NEXT-RECORD.
     PERFORM READ-RECORD
*     The previous statement is needed before entering the loop.
*     Both the loop condition LASTREC = 'Y'
*     and the call to WRITE-RECORD depend on READ-RECORD having
*     been executed before.
*     The loop starts at the next line with PERFORM UNTIL
      PERFORM UNTIL LASTREC = 'Y'
      PERFORM WRITE-RECORD
      PERFORM READ-RECORD
      END-PERFORM
     .
*
 CLOSE-STOP.
     CLOSE ACCT-REC.
     CLOSE PRINT-LINE.
     STOP RUN.
*
 READ-RECORD.
     READ ACCT-REC
     AT END MOVE 'Y' TO LASTREC
     END-READ.
*
 WRITE-RECORD.
     MOVE ACCT-NO       TO  ACCT-NO-O.
     MOVE ACCT-LIMIT    TO  ACCT-LIMIT-O.
     MOVE ACCT-BALANCE  TO  ACCT-BALANCE-O.
     MOVE LAST-NAME     TO  LAST-NAME-O.
     MOVE FIRST-NAME    TO  FIRST-NAME-O.
     MOVE COMMENTS      TO  COMMENTS-O.
     WRITE PRINT-REC.
```

Figure 6. Lecture et ecriture d'enregistrements

Remarque : COBOL ressemble à l'anglais et les mots reserves COBOL sont de type anglais. Le programmeur est libre d'utiliser des noms de variables de type anglais pour se souvenir de l'objectif des noms de variables. La structure de PROCEDURE DIVISION est de type anglais. Un paragraphe contient une ou plusieurs phrases. Une phrase contient une ou plusieurs declarations. Le terminateur de portee implicite, un point (.), termine une phrase ou termine plusieurs declarations consecutives qui seraient analogues a une phrase composee ou " et " joint des phrases potentiellement independantes. ###

11.3.1 Execution du paragraphe READ-NEXT-RECORD

Le paragraphe READ-NEXT-RECORD est une technique de programmation COBOL utilisee pour lire tous les enregistrements d'un fichier sequentiel JUSQU'a CE QUE le dernier enregistrement soit lu. Le paragraphe contient une phrase composee terminee par un point de terminaison de portee implicite (.) sur une ligne distincte apres l'instruction END-PERFORM. PERFORM UNTIL a END-PERFORM, terminateur de portee explicite, est execute a plusieurs reprises jusqu'a ce que la variable LASTREC contienne Y. Le premier PERFORM READ-RECORD aboutit a un branchement vers le paragraphe READ-RECORD. Observez #1 dans la figure 7.

11.3.2 Paragraphe READ-RECORD

Le paragraphe READ-RECORD execute l'instruction COBOL READ resultant dans le fichier sequentiel externe remplissant les variables associees au nom de fichier interne ACCT-REC. Si 'AT END' des enregistrements lus, alors Y est deplace dans la variable LASTREC. L'instruction READ se termine par un terminateur de portee explicite, END-READ. Le paragraphe se termine par une terminaison de portee implicite, un point (.). Le controle est renvoye au paragraphe READ-NEXT-RECORD pour executer l'instruction suivante, PERFORM WRITE-RECORD.

11.3.3 Paragraphe WRITE-RECORD

Le paragraphe WRITE-RECORD contient plusieurs phrases terminees par un terminateur de portee implicite, point (.). Les instructions MOVE entrainent le deplacement de chaque nom de variable de fichier d'entree vers un nom de variable de fichier de sortie. La derniere phrase du paragraphe ecrit la collection de noms de variables de fichier de sortie, PRINT-REC.

PRINT-REC est affecte a PRTREC. Le JCL est utilise pour executer le programme COBOL. Un JCL PRTREC DDNAME associe redirige la sortie ecrite vers un nom d'ensemble de donnees controle par z/OS, etc. a l'aide de l'operation JCL DD sur l'instruction JCL DDNAME. Observez #2 dans la figure 7.

11.3.4 Traitement iteratif du paragraphe READ-NEXT-RECORD

Une fois que toutes les instructions du paragraphe WRITE-RECORD sont executees, le controle revient au paragraphe READ-NEXT-RECORD ou la phrase suivante a executer est la deuxieme instruction PERFORM READ-RECORD.

Encore une fois, le paragraphe READ-RECORD execute l'instruction COBOL READ, ce qui fait que le fichier sequentiel externe remplit les variables associees au nom de fichier interne ACCT-REC. Si 'AT END' des enregistrements lus, Y est deplace dans la variable LASTREC, puis renvoie le controle au paragraphe READ-NEXT-RECORD. Le paragraphe READ-NEXT-RECORD continuerait le processus iteratif JUSQU'a CE QUE Y soit trouve dans la variable LASTREC. Observez #3 dans la figure 7.

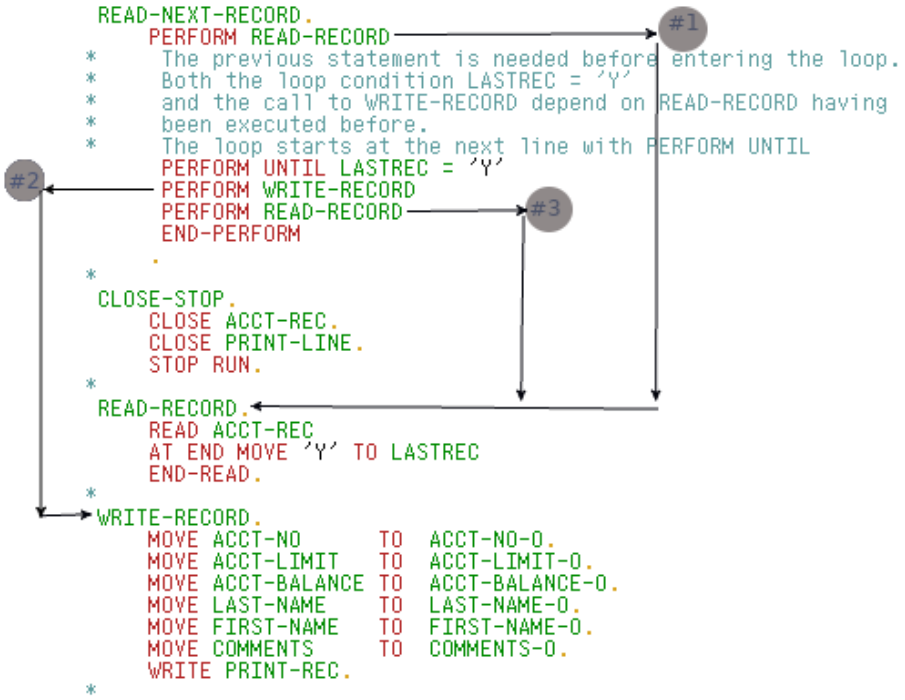


Figure 7. Traitement iteratif

11.4 Labo

Le laboratoire associe a ce chapitre demontre la technique de codage COBOL de " fin de fichier " pour lire tous les enregistrements de donnees a partir d'un fichier sequentiel. Si une etape est accompagnee d'un asterisque (*), un indice sera associe a la fin du contenu de l'atelier.

1. Si ce n'est deja fait, ouvrez VSCode et selectionnez Zowe Explorer dans la barre laterale gauche.

Remarque : Si vous ouvrez une nouvelle instance de VSCode (c'est-a-dire que vous l'avez fermee apres l'utilisation precedente), vous devrez peut-etre a nouveau " Selectionner un filtre ". Vous pouvez le faire en selectionnant l'icone de recherche a cote de votre connexion nommee dans la section ENSEMBLES DE DONNeES, puis en reselectionnant le filtre precedemment utilise. Il devrait figurer dans les filtres repertories apres avoir selectionne le symbole de recherche.

2. Affichez ces membres du code source COBOL repertories dans l'ensemble de donnees id.CBL :

- CBL0001
- CBL0002

3. Affichez ces trois membres JCL dans l'ensemble de donnees id.JCL :

- CBL0001J
- CBL0002J
- CBL0003J

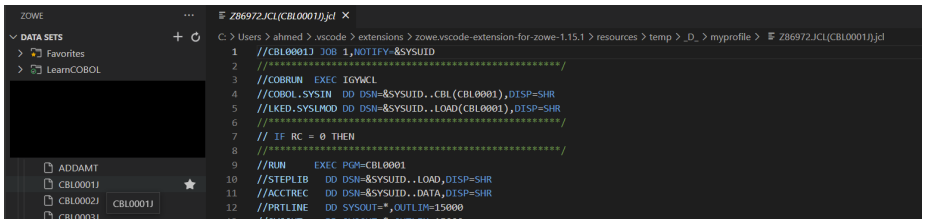


Figure 8. Id.JCL(CBL0001J).jcl

4. Soumettez le travail, JCL(CBL0001J), dans la section DATA SET.
5. Affichez la sortie de ce travail a l'aide de la section JOBS.

- COBRUN:SYSPRINT(101) - Sortie du compilateur de programme COBOL
- RUN:PRTLINE(103) - Sortie d'execution du programme COBOL, illustree a la Figure 9.

Figure 9. RUN:PRTLINE(103) pour JCL(CBL0001J)

6. Soumettez le travail, JCL(CBL0002J), dans la section DATA SET.
7. Affichez la sortie de ce travail a l'aide de la section JOBS.

    - COBRUN:SYSPRINT(101) - Sortie du compilateur de programme COBOL

    Localisez le message severe du compilateur COBOL IGYPS2121-S dans le fichier de sortie auquel il est fait reference a l'etape 7, illustre a la figure 10.

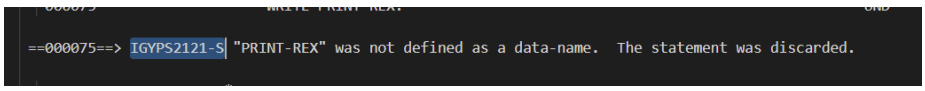


Figure 10. Message IGYPS2121-S

8. Modifier le CBL (CBL0002) :
    - Determinez l'orthographe appropriee de PRINT-REX, corrigez-la dans le code source et enregistrez le code source mis a jour.
9. Soumettez a nouveau le travail, JCL(CBL0002J), a l'aide de la section DATA SET et affichez le resultat dans la section JOBS.
    - COBRUN:SYSPRINT(101) sortie du compilateur de programme COBOL
    - RUN:PRTLINE(103) est la sortie d'execution du programme COBOL (si la correction est reussie)
10. Soumettez le travail, JCL(CBL0003J), a l'aide de la section DATA SET.
11. Affichez la sortie CBL0003J ABENDU4038 a l'aide de la section JOBS :
    - Affichez le message de fin anormale IGZ00355 dans RUN:SYSOUT(104) a partir de la sortie d'execution du programme COBOL.

- IGZ00355 lit, le programme est incapable d'ouvrir ou de fermer le nom de fichier ACCTREC, illustre a la figure 11. vous guidant vers la racine de l'erreur.

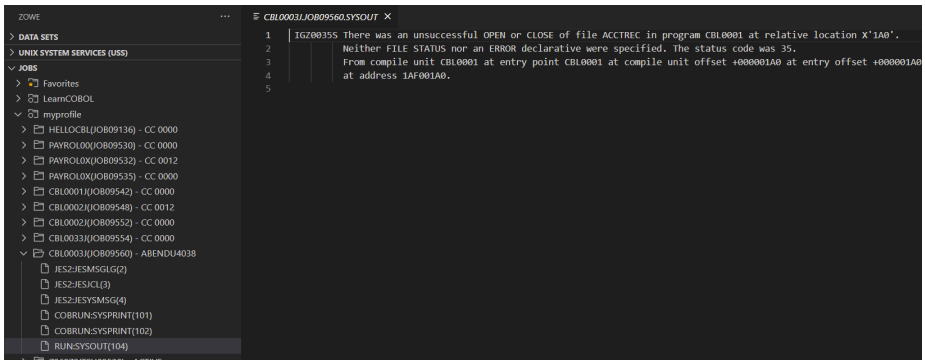


Figure 11. Message RUN:SYSOUT(104)

12. Corrigez cette erreur en modifiant le JCL(CBL0003J) :
    - Determinez le DDNAME necessaire, mais manquant ou mal orthographie.
    - Corrigez-le dans le code et enregistrez-le
13. Soumettez a nouveau le travail, JCL(CBL0003J), a l'aide de la section DATA SET.
14. Affichez la sortie CBL0003J a l'aide de la section JOBS, votre sortie devrait ressembler a la Figure 12.

- RUN:PRTLINE - Sortie d'execution du programme COBOL (si la correction est reussie)

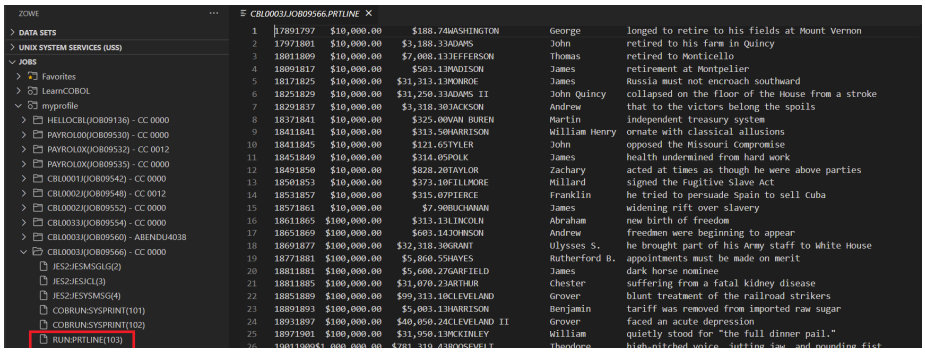


Figure 12. RUN:PRTLINE(103) pour JCL(CBL0003J)

Conseils de laboratoire

13. L'erreur se situe a la ligne 11, ajustez 'ACCTREX' en consequence.

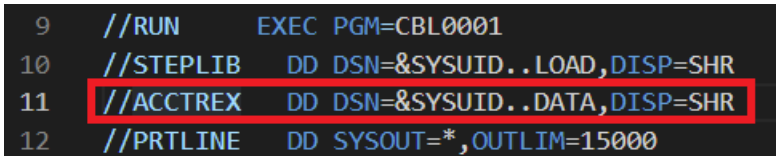


Figure 13. Erreur dans id.JCL(CBL0003J).jcl

12 Structure du programme

Dans ce chapitre, nous discutons du concept de programmation structuree et de ses liens avec COBOL. Nous mettons en evidence les techniques cles du langage COBOL qui vous permettent d'ecrire de bons programmes bien structures.

- **Styles de programmation**
  - **Qu'est-ce que la programmation structuree**
  - **Qu'est-ce que la programmation orientee objet**
  - **Style de programmation COBOL**
- **Structure de la Division Procedure**
  - **Controle du programme et flux a travers un programme de base**
  - **Instructions d'execution en ligne et hors ligne**
  - **Utiliser performs pour coder une boucle**
  - **Apprentissage des mauvais comportements a l'aide du mot-cle GO TO**
- **Paragraphes sous forme de blocs de code**
  - **Concevoir le contenu d'un paragraphe**
  - **Ordre et denomination des paragraphes**
- **Controle du programme avec paragraphes**
  - **TEMPS D'EXeCUTION**
  - **EFFECTUER PAR**
  - **EFFECTUER JUSQU'a**
  - **EFFECTUER UNE VARIATION**
- **Utilisation de sous-programmes**
  - **En precisant le programme cible**
  - **Specification des variables du programme**
  - **Specifier la valeur de retour**
- **a l'aide de cahiers**
- **Sommaire**
- **Laboratoire**

12.1 Styles de programmation

Avant de discuter plus en detail de la maniere de structurer un programme ecrit en COBOL, il est important de comprendre le type de langage COBOL et en quoi il est different des autres langages et comment il affecte la facon dont vous pouvez structurer vos programmes.

12.1.1 Qu'est-ce que la programmation structuree

La programmation structuree est le nom donne a un ensemble de styles de programmation qui peuvent inclure, entre autres, des fonctions fonctionnelles, procedurales. La technique de programmation structuree rend la logique du programme plus facile a comprendre et a maintenir. Des exemples de langages de programmation structures sont C, PL/I, Python et bien sur COBOL. Ces langages, etant donne des structures de flux de controle specifiques telles que des boucles, des fonctions et des methodes, permettent a un programmeur d'organiser son code de maniere significative.

Les constructions de programmation non structurees, egalement connues sous le nom de code spaghetti, sont des concepts tels que GOTO ou JUMP qui permettent au flux d'execution de se ramifier de maniere sauvage autour du code source. Un tel code comme celui-ci est difficile a analyser et a lire. Bien que COBOL contienne ces structures, il est important de les utiliser avec parcimonie et non comme l'epine dorsale d'un code bien structure.

Un code bien structure est a la fois facile a comprendre et a maintenir. Il est fort probable qu'a un moment donne de votre carriere, vous serez amene a lire et a travailler a partir du code de quelqu'un d'autre, souvent une decennie apres sa redaction initiale. Cela vous serait extremement utile si l'auteur original structurait bien son code et de meme s'il s'agit de votre code que quelqu'un d'autre lit.

12.1.2 Qu'est-ce que la programmation orientee objet

La programmation orientee objet, ou programmation OO, differe de la programmation structuree, bien qu'elle emprunte beaucoup des memes concepts. Dans la programmation OO, le code est divise en plusieurs classes, chacune representant un acteur au sein du systeme. Chaque classe est constituee de variables et d'une sequence de methodes. Les instanciations d'une classe ou d'objets peuvent executer des methodes d'un autre objet. Chaque classe d'un programme OO peut etre consideree comme un programme structure, car elle contiendra toujours des methodes et des constructions d'iteration. Cependant, c'est la composition du programme a partir d'un ensemble de classes individuelles qui rend la programmation OO differente. Il est possible d'ecrire du COBOL oriente objet ; cependant, il n'est pas pris en charge par certains des produits middleware qui fournissent des API COBOL. Il n'est generalement pas utilise sur le marche et n'est donc pas couvert dans ce cours.

12.1.3 Style de programmation COBOL

COBOL n'a pas directement certains des composants d'un langage de programmation structure comme vous les connaissez peut-etre si vous avez etudie un langage comme C ou Java. COBOL ne contient pas de boucles for ou while, ni de fonctions ou methodes definies. Parce que COBOL est cense etre un langage facile a lire, ces concepts sont incarnes par l'utilisation du mot-cle PERFORM et le concept de paragraphes. Cela permet au programmeur de toujours creer ces structures, mais d'une maniere facile a lire et a suivre.

12.2 Structure de la Division des procedures

Comme vous le savez deja, un programme COBOL est divise en plusieurs divisions, dont l'identification, l'environnement et les donnees. Cependant, ce chapitre s'interesse a la facon dont vous structurez le contenu de la division des procedures pour qu'il soit facile a lire, comprehensible et maintenable a l'avenir.

12.2.1 Controle du programme et flux a travers un programme de base

En regle generale, l'execution dans un programme COBOL commence a la premiere instruction dans la division de procedure et progresse sequentiellement a travers chaque ligne jusqu'a ce qu'elle atteigne la fin du code source. Par exemple, jetez un oeuil a l'exemple 1. Extrait de TOTEN1. Il s'agit d'un programme simple qui affiche un message simple comptant jusqu'a dix.

```
OUVRIR LA LIGNE D'IMPRESSION DE SORTIE.

DePLACEZ « LE NUMeRO EST : » a MSG-EN-TÊTE DE IMPRIMER-REC.

AJOUTER 1 AU COMPTOIR DONNER LE COMPTEUR.
DePLACEZ LE COMPTEUR a MSG-TO-WRITE.
eCRIRE IMPRIMER-REC.

AJOUTER 1 AU COMPTOIR DONNER LE COMPTEUR.
DePLACEZ LE COMPTEUR a MSG-TO-WRITE.
eCRIRE IMPRIMER-REC.

...

FERMER LA LIGNE D'IMPRESSION.
ARRÊTEZ DE COURIR.
```

Exemple 1. Extrait de TOTEN1

Bien que ce code soit tres simple a lire, il n'est pas tres elegant, il y a beaucoup de repetition de code au fur et a mesure que le nombre augmente. evidemment, nous

voulons donner une certaine structure au programme. Il existe trois mots-cles que nous pouvons utiliser pour transferer le controle a une section differente du code source et fournir la structure dont nous avons besoin. Ces mots-cles sont PERFORM, GO TO et CALL.

12.2.2 Instructions d'execution en ligne et hors ligne

Le mot cle PERFORM est un element tres flexible du langage COBOL, car il permet de saisir des fonctions et des boucles. Au niveau le plus basique, un PERFORM permet de transferer le controle a une autre section du code. Une fois cette section executee, le controle revient a la ligne de code suivante. Prenons l'exemple suivant :

```
    OUVRIR LA LIGNE D'IMPRESSION DE SORTIE.

    DePLACEZ " LE NUMeRO EST : " a MSG-EN-TETE DE IMPRIMER-REC.

    EFFECTUER eCRIRE-NOUVEAU-ENREGISTREMENT.
    EFFECTUER eCRIRE-NOUVEAU-ENREGISTREMENT.
    EFFECTUER eCRIRE-NOUVEAU-ENREGISTREMENT.
    EFFECTUER eCRIRE-NOUVEAU-ENREGISTREMENT.
    EFFECTUER eCRIRE-NOUVEAU-ENREGISTREMENT.
    EFFECTUER eCRIRE-NOUVEAU-ENREGISTREMENT.
    EFFECTUER eCRIRE-NOUVEAU-ENREGISTREMENT.
    EFFECTUER eCRIRE-NOUVEAU-ENREGISTREMENT.
    EFFECTUER eCRIRE-NOUVEAU-ENREGISTREMENT.
    EFFECTUER eCRIRE-NOUVEAU-ENREGISTREMENT.


    FERMER LA LIGNE D'IMPRESSION.
    ARRETEZ DE COURIR.


eCRIRE-NOUVEAU-ENREGISTREMENT.
    AJOUTER 1 AU COMPTOIR DONNER LE COMPTEUR
    DePLACER LE COMPTEUR VERS MSG-TO-WRITE
    eCRIRE IMPRIMER-REC.
```

Exemple 2. Extrait de TOTEN2

Dans cet exemple, les trois lignes de code qui ont construit une nouvelle ligne de sortie et l'ont imprimee ont ete extraites dans un nouveau paragraphe appele WRITE-NEW-RECORD. Ce paragraphe est ensuite execute dix fois a l'aide du mot cle PERFORM. Chaque fois que le mot cle PERFORM est utilise, l'execution saute au paragraphe WRITE-NEW-RECORD, execute les trois lignes contenues dans ce paragraphe avant de revenir a la ligne suivant l'instruction PERFORM. Le concept de paragraphe sera traite plus en profondeur plus loin dans ce

chapitre.

12.2.3 Utilisation de performs pour coder une boucle

Le code que nous avons construit jusqu'a present n'est toujours pas optimal, la repetition de l'instruction perform dix fois est inelegante et peut etre optimisee. Observez l'extrait de code suivant :

```
DePLACEZ « LE NUMeRO EST : » a MSG-EN-TÊTE DE IMPRIMER-REC.

EFFECTUER UN COMPTEUR VARIABLE DE 01 PAR 1 JUSQU'A COMPTEUR eGAL 11
DePLACER LE COMPTEUR VERS MSG-TO-WRITE
eCRIRE IMPRIMER-REC
FIN-PERFORMER.

FERMER LA LIGNE D'IMPRESSION.
ARRÊTEZ DE COURIR.
```

Exemple 3. Extrait de TOTEN2

Dans cet exemple, nous utilisons le mot cle PERFORM d'une maniere similaire a une boucle for dans d'autres langages. La boucle s'execute du mot cle PERFORM au mot cle END-PERFORM. A chaque iteration de l'execution sur la boucle, la valeur de COUNTER est incrementee et testee de un. A titre de comparaison, la meme boucle serait ecrite en Java comme ceci :

```
for(int compteur =0; compteur<11; compteur++){
    // deplacer le compteur vers msg-to-write
    //ecrire print-rec
}
```

Exemple 4. Exemple Java

Bien que la version COBOL soit peut-etre plus detaillee qu'une boucle for dans d'autres langues, elle est plus facile a lire et n'oubliez pas que vous avez toujours la saisie semi-automatique (si vous utilisez un bon editeur) pour vous aider avec la frappe.

12.2.4 Apprendre un mauvais comportement a l'aide du mot-cle GO TO

Les programmeurs ont tendance a avoir de fortes convictions quant au choix de l'editeur, des onglets ou des espaces et de nombreuses discussions animees ont eu lieu sur de tels sujets. Cependant, s'il y a une chose sur laquelle nous pouvons nous mettre d'accord, c'est que l'utilisation de GO TO est generalement une mauvaise idee. Pour demontrer pourquoi GO TO peut etre une mauvaise idee, nous examinerons a nouveau TOTEN2 et remplacerons la deuxieme instance du mot cle PERFORM par GO TO, illustre dans l'exemple 5.

```
PERFORM WRITE-NEW-RECORD.
2802 GO TO WRITE-NEW-RECORD.
2803 PERFORM WRITE-NEW-RECORD.
```

Exemple 5. ALLER a l'exemple

Si nous devions compiler et executer le programme, vous verriez que bien que le travail ABENDS (se termine anormalement) avec un code 4038-abend, il a execute une partie du code et ecrit les deux premieres lignes de la sortie. Si vous deviez regarder la sortie plus en detail, vous verriez un message comme le suivant :

```
IGZ0037S Le flux de controle dans le programme TOTEN1 s'est poursuivi au-dela de
```

Exemple 6. Abend de l'exemple GO TO

Alors, qu'est-ce qui s'est si mal passe lorsque nous avons utilise la commande GO TO ? Pour repondre a cette question, nous devons comprendre la principale difference entre GO TO et PERFORM. Sur la premiere ligne, nous avons utilise le mot-cle PERFORM, qui a transfere le controle au paragraphe WRITE-NEW-RECORD. Une fois que l'execution a atteint la fin de ce paragraphe, l'execution est revenue a la ligne suivant l'instruction PERFORM. La ligne suivante a utilise le mot-cle GOTO pour transferer a nouveau le controle au paragraphe WRITE-NEW-RECORD, qui imprime la deuxieme ligne de sortie. Cependant, une fois ce paragraphe termine, l'execution s'est poursuivie jusqu'a la ligne suivante suivant le paragraphe WRITE-NEW-RECORD. Puisqu'il n'y a pas de lignes de code apres ce paragraphe, le processeur a essaye d'executer du code au-dela du programme, z/OS a detecte cela comme un probleme et a interrompu le programme.

Comme on peut le voir, l'utilisation de GO TO provoque une branche d'execution qui ne revient pas a la ligne de code qui l'a emise. Montrons a quel point ce code peut etre desordonne :

```
0  01   FLAG                PIC 9(1) VALUE 1.

1   OPEN OUTPUT PRINT-LINE.
2      GO TO SAY-HELLO-WORLD DEPENDING ON FLAG.
3
4  PRINT-NEW-MESSAGE.
5      MOVE 2 TO FLAG
6      GO TO SAY-HELLO-COBOL DEPENDING ON FLAG
7      GO TO END-RUN.
8
9  SAY-HELLO-WORLD.
10       MOVE "Hello World" TO MSG-TO-WRITE
11       WRITE PRINT-REC
12       GO TO PRINT-NEW-MESSAGE.
```

```
13
14 SAY-HELLO-COBOL.
15     MOVE "Hello COBOL" TO MSG-TO-WRITE
16     WRITE PRINT-REC
17     GO TO END-RUN.
18
19 END-RUN.
20     CLOSE PRINT-LINE
21     STOP RUN.
```

Exemple 7. Code desordonne en utilisant GO TO

Cet exemple utilise un melange d'instructions GO TO conditionnelles et non conditionnelles, et des numeros de ligne sont inclus pour faciliter le suivi du code. La ligne 2 s'execute et se branche sur SAY-HELLO-WORLD a la ligne 9, si la variable indicateur est definie sur 1. Dans ce cas, c'est le cas, nous progressons donc dans les lignes 9 a 12 et nous passons aux lignes 4 a 6 ou la valeur du drapeau est mis a jour et teste a nouveau pour voir si nous devons passer a SAY-HELLO-COBOL. Puisque la valeur de flag n'est plus 1, l'execution continue juste a la ligne 7 avant de sauter a la ligne 19 et de terminer la course. Prenez ce programme et commentez la ligne 5 et executez a nouveau le programme. Suivre l'execution du programme. Desordonne, non ?

Remarque : Les parties TO et ON de l'instruction conditionnelle GO TO peuvent etre omises, donnant une instruction qui ressemble a GO SAY-HELLO-WORLD DEPENDING FLAG. Ce qui, bien que moins verbeux, n'en est pas moins facile a comprendre.

Alors pourquoi vous apprendre quelque chose dont nous avons dit qu'il est desordonne et deconseille ? Eh bien, en vous donnant une certaine comprehension de son comportement, vous serez mieux equipe pour parcourir le code existant et le maintenir.

12.3 Les paragraphes sous forme de blocs de code

Jusqu'a present dans cette section, nous avons utilise quelques exemples de paragraphes sans vraiment expliquer ce qu'ils sont, comment ils fonctionnent et a quoi ils peuvent servir. Cette section traite de cela.

La facon la plus analogue de penser a un paragraphe en COBOL est de penser a une fonction ou une methode dans un autre langage qui n'accepte aucun parametre, ne renvoie aucune reponse et modifie les variables globales. Il s'agit essentiellement d'un bloc de code qui execute une sequence d'actions pouvant etre utilisees plusieurs fois dans le meme programme.

Un paragraphe est defini dans la division de procedure et commence a la colonne huit et peut avoir n'importe quel nom que l'utilisateur aime, a l'exception d'un mot-cle COBOL, et la declaration du paragraphe est completee par un point (.).

Un paragraphe peut contenir une a plusieurs phrases COBOL et se termine soit par le debut d'un autre paragraphe, soit par la fin physique du programme.

Remarque : Un paragraphe peut egalement se terminer par END-PROGRAM, END-METHOD, END FACTORY OU END-OBJECT. La plupart d'entre eux sont utilises dans le COBOL oriente objet qui n'est pas aborde ici.

Considerant qu'un programme peut etre compose de plusieurs paragraphes et que le mot-cle PERFORM peut etre utilise pour appeler le paragraphe, soit de maniere conditionnelle, soit dans le cadre d'une boucle, il est facile de voir qu'une bonne conception de paragraphe aide vraiment a rendre votre COBOL plus structure et lisible.

12.3.1 Concevoir le contenu d'un paragraphe

Il n'y a aucune restriction quant au contenu pouvant etre insere dans un paragraphe, cependant, il existe deux raisons principales pour lesquelles vous pouvez vouloir refactoriser le code pour qu'il soit a l'interieur d'un paragraphe :

1. Pour regrouper une sequence de phrases COBOL qui realisent une fonction ou une tache particuliere, comme ouvrir tous les fichiers qu'une application utilise, calculer une fonction particuliere ou effectuer une validation de donnees. Regrouper de telles phrases dans un paragraphe vous permet de leur donner un nom qui explique le but des lignes de code.
2. La sequence de phrases sera utilisee dans une boucle. Extraire ces lignes dans un paragraphe puis utiliser le mot-cle PERFORM pour creer une boucle peut rendre le code tres comprehensible.

N'oubliez pas que vous pouvez egalement executer d'autres paragraphes dans des paragraphes existants. Cet appel imbrique de paragraphes peut encore une fois aider a structurer votre code.

12.3.2 Ordre et nommage des paragraphes

Il n'y a aucune exigence concernant l'ordre dans lequel les paragraphes doivent apparaitre dans un programme COBOL. Un paragraphe peut etre appele à partir d'un point avant ou après l'endroit où il est déclaré. Bien qu'il n'y ait aucune restriction imposée par la langue, il existe certaines techniques que vous pouvez suivre qui rendront les programmes plus volumineux plus faciles à suivre et à comprendre. Certaines de ces techniques et meilleures pratiques sont :

- Nommez chaque paragraphe en fonction de sa fonction ou de son comportement. Un paragraphe nommé OPEN-INPUT-FILES. est beaucoup plus compréhensible que celui nommé DO-FILE-STUFF.
- Classez les paragraphes dans l'ordre général dans lequel ils seront exécutés à l'exécution. Faire cela présente deux avantages principaux. L'utilisation de la vue hiérarchique dans un IDE moderne vous permettra de « lire » le

nom de chaque paragraphe de haut en bas, ce qui vous permettra d'établir la structure générale du programme et son comportement.

- Certains programmeurs COBOL préfixent le nom des paragraphes avec un nombre qui augmente tout au long du code source selon l'exemple 8.
- Parce que les paragraphes sont numérotés et apparaissent dans le code source dans cet ordre, lorsqu'une phrase fait référence à un paragraphe, il est plus facile de savoir où dans le programme ce paragraphe peut apparaître. Lors de la structuration initiale d'un programme de cette manière, les nombres utilisés n'incrémenteraient que le chiffre significatif le plus élevé, permettant d'insérer de nouveaux paragraphes entre les deux si nécessaire. Bien que l'essor des éditeurs modernes, qui permettent d'esquisser et de sauter instantanément à une référence ou à une déclaration, rend cette technique moins nécessaire, il est toujours utile de la comprendre.

```
PERFORM 1000-OPEN-FILES.
        PERFORM 2000-READ-NEXT-RECORD.
        GO TO 3000-CLOSE-STOP.
 1000-OPEN-FILES.
        OPEN INPUT  ACCT-REC.
        OPEN OUTPUT PRINT-LINE.
*
 2000-READ-NEXT-RECORD.
        PERFORM 4000-READ-RECORD
        PERFORM UNTIL LASTREC = 'Y'
        PERFORM 5000-WRITE-RECORD
        PERFORM 4000-READ-RECORD
        END-PERFORM.
*
 3000-CLOSE-STOP.
        CLOSE ACCT-REC.
        CLOSE PRINT-LINE.
        STOP RUN.
*
 4000-READ-RECORD.
        READ ACCT-REC
        AT END MOVE 'Y' TO LASTREC
        END-READ.
*
 5000-WRITE-RECORD.
        MOVE ACCT-N -      TO  ACCT-NO-O.
        MOVE ACCT-LIMIT    TO  ACCT-LIMIT-O.
        MOVE ACCT-BALANCE TO  ACCT-BALANCE-O.
        MOVE LAST-NAME     TO  LAST-NAME-O.
        MOVE FIRST-NAME    TO  FIRST-NAME-O.
        MOVE COMMENTS      TO  COMMENTS-O.
```

```
                WRITE PRINT-REC.
```

Exemple 8. Paragraphes numérotés

- Enfin, il est courant de terminer explicitement un paragraphe en codant un paragraphe vide à la suite de chaque paragraphe, voir exemple 9. Ce paragraphe vide ne contient aucun code, porte le même nom que le paragraphe qu'il ferme, suffixé par -END et est à son tour clôturé par le début d'un paragraphe suivant. Mais il peut être utilisé comme délimiteur visuel et est utile lors de l'utilisation du mot-clé PERFORM THRU, qui est abordé plus loin dans ce chapitre. Certains programmeurs Java qui ont appris COBOL ont indiqué que cela équivaut à l'accolade fermante ("}") à la fin d'un bloc de code.

```
 1000-OPEN-FILES.
     OPEN INPUT  ACCT-REC.
     OPEN OUTPUT PRINT-LINE.
1000-OPEN-FILES-END.
*
2000-READ-NEXT-RECORD.
     PERFORM 4000-READ-RECORD
     PERFORM UNTIL LASTREC = 'Y'
     PERFORM 5000-WRITE-RECORD
     PERFORM 4000-READ-RECORD
     END-PERFORM.
2000-READ-NEXT-RECORD-END.
```

Exemple 9. Paragraphes explicitement fermés

12.4 Contrôle de programme avec paragraphes

Jusqu'à présent dans ce chapitre, nous avons discuté de l'importance d'utiliser des paragraphes pour structurer votre code. Ce faisant, nous avons utilisé le mot-clé PERFORM plusieurs fois pour exécuter les paragraphes que nous avions créés. Plus précisément, nous avons utilisé le mot-clé seul et l'avons utilisé avec le mot-clé VARYING pour construire une boucle. Dans cette section, nous verrons plus en détail comment le mot-clé PERFORM peut être utilisé.

12.4.1 EFFECTUER LES TEMPS

Le moyen le plus simple de répéter une instruction perform est peut-être d'utiliser le mot-clé TIMES pour exécuter un paragraphe ou des sections de code un nombre statique de fois, comme illustré dans l'exemple 10.

```
PERFORM 10 TIMES
  MOVE FIELD-A TO FIELD-B
  WRITE RECORD
END-PERFORM.
```

Exemple 10. TIMES

Le nombre requis de fois que le code doit être exécuté peut être soit un littéral, comme ci-dessus, soit la valeur d'une variable numérique comme indiqué dans l'exemple 11. où le mot clé PERFORM est utilisé pour exécuter un paragraphe.

```
PERFORM MY-NEW-PARAGRAPH COUNTER TIMES.
```

Exemple 11. TIMES 2

12.4.2 PERFORM THROUGH

Vous pouvez exiger qu'une liste séquentielle de paragraphes soit exécutée à tour de rôle, au lieu de les exécuter individuellement. Le mot clé THROUGH ou THRU peut être utilisé pour lister les paragraphes de début et de fin de la liste. L'exécution progressera dans chacun des paragraphes tels qu'ils apparaissent dans le code source, du début à la fin, avant de revenir à la ligne suivant l'instruction perform initiale, observez l'exemple 12.

```
1000-PARAGRAPH-A.
      PERFORM 2000-PARAGRAPH-B THRU
                  3000-PARAGRAPH-C.
*
2000-PARAGRAPH-B.
      ...
*
3000-PARAGRAPH-C.
      ...
*
4000-PARAGRAPH-D.
```

Exemple 12. PERFORM THRU

Remarque : L'utilisation du mot-clé THRU peut également être utilisée avec les mots-clés TIMES, UNTIL et VARYING, pour permettre l'exécution de la liste des paragraphes plutôt qu'un seul paragraphe ou des blocs de code.

12.4.3 PERFORM UNTIL

L'ajout du mot clé UNTIL à une phrase perform vous permet d'itérer sur un groupe de phrases jusqu'à ce que la condition booléenne soit remplie. Vous permettant effectivement de programmer des boucles while en COBOL, prenons cet exemple basique :

```
MOVE 0 TO COUNTER.
 PERFORM UNTIL COUNTER = 10
   ADD 1 TO COUNTER GIVING COUNTER
   MOVE COUNTER TO MSG-TO-WRITE
   WRITE PRINT-REC
```

```
END-PERFORM.
```

Exemple 13. EFFECTUER JUSQU'À

Cela équivaudrait au code Java :

```
while(compteur != 10){
    //compteur++
    // déplacer le compteur vers msg-to-write
    //écrire print-rec
}
```

Exemple 14. Boucle while Java

Dans ce cas, la condition booléenne est évaluée avant l'exécution de la boucle. Cependant, si vous souhaitez que la boucle soit exécutée au moins une fois avant que la condition ne soit évaluée, vous pouvez modifier la phrase pour lire :

```
PERFORM WITH TEST AFTER UNTIL COUNTER = 10
  ADD 1 TO COUNTER GIVING COUNTER
  MOVE COUNTER TO MSG-TO-WRITE
  WRITE PRINT-REC
END-PERFORM.
```

Exemple 15. EFFECTUER AVEC TEST APRÈS JUSQU'À

Cela serait similaire à une boucle "do while" en Java :

```
do{
    //counter++
    //move counter to msg-to-write
    //write print-rec
}
While(counter != 10);
```

Exemple 16. Java while loop

12.4.4 PERFORM VARYING

Nous avons déjà utilisé le mot clé VARYING plus tôt dans la section intitulée Using performs to code a loop, rappelez-vous :

```
PERFORM VARYING COUNTER FROM 01 BY 1 UNTIL COUNTER EQUAL 11
...
END-PERFORM.
```

Exemple 17. Boucle élémentaire

Dans cet exemple, le compteur de variable est testé pour voir s'il est égal à 11, tant que ce n'est pas le cas, il est incrémenté et les phrases imbriquées

dans l'instruction perform sont exécutées. Cette construction peut être étendue, illustrée dans l'exemple 18.

```
PERFORM VARYING COUNTER FROM 01 BY 1 UNTIL COUNTER EQUAL 11
      AFTER COUNTER–2 FROM 01 BY 1 UNTIL COUNTER–2 EQUAL 5
...
FIN–PERFORMER.
```

Exemple 18. Boucle étendue

Cela peut sembler complexe, mais comparez-le à ce pseudo-code Java :

```
for(int counter = 0; counter < 11; counter++){
    for(int counter2 = 0; counter2 < 5; counter2++{
        //move counter to msg–to–write
        //write print–rec
    }
}
```

Exemple 19. Boucle étendue Java

Il s'agit en réalité de deux boucles for imbriquées l'une dans l'autre. Cette construction est très utile lors de l'itération sur des tables ou des structures d'enregistrement imbriquées. Comme pour chaque boucle de la boucle variable externe, la boucle interne sera exécutée cinq fois. Comme mentionné précédemment, le test de la condition sera supposé par COBOL être au début de la boucle, cependant, il peut être spécifié pour être évalué à la fin de la boucle en ajoutant la phrase WITH TEST AFTER à la phrase d'exécution initiale .

12.5 Utilisation de sous-programmes

Jusqu'à présent, nous n'avons examiné que la structure interne d'un seul programme COBOL. Au fur et à mesure que les programmes augmentent en fonction et en nombre, il est courant qu'un programmeur souhaite que certains aspects d'une fonction de programme soient mis à disposition d'autres programmes au sein du système. Faire abstraction des fonctions génériques dans leur propre programme et leur permettre d'être appelées à partir d'autres programmes peut réduire la quantité de duplication de code au sein d'un système et donc diminuer les coûts de maintenance, car les correctifs des modules partagés ne doivent être effectués qu'une seule fois.

Remarque : Bien que nous décrivions ici la manière native COBOL d'appeler un autre programme, notez que certains produits middleware fourniront des API qui pourraient le faire de manière améliorée.

Lors de l'appel d'un autre programme, nous devons prendre en compte trois préoccupations principales : comment nous allons référencer le programme que nous souhaitons appeler, les paramètres que nous voulons envoyer au programme cible et le paramètre que nous souhaitons que le programme cible renvoie.

12.5.1 Spécification du programme cible

Pour appeler un programme cible, nous utiliserons le mot-clé CALL suivi d'une référence au programme cible que nous souhaitons appeler. Les deux principales façons de le faire sont par une valeur littérale ou en référençant une variable, illustrée dans l'exemple 20.

```
CALL 'PROGA1' ...
...
MOVE 'PROGA2' TO PROGRAM-NAME.
CALL PROGRAM-NAME ...
```

Exemple 20. APPEL de base

Il est également possible de référencer la plate-forme cible en passant une référence de pointeur au programme cible. Si vous pensiez que passer une référence de pointeur à une fonction n'était que quelque chose que les langages ultra-modernes avaient, non, COBOL est arrivé en premier !

12.5.2 Spécification des variables du programme

Maintenant que nous avons identifié le nom du programme que nous souhaitons appeler ; nous devons identifier les variables que le programme appelant pourrait vouloir envoyer. Ceux-ci sont spécifiés individuellement par le mot-clé USING. COBOL prend en charge à la fois le passage par référence et le passage par copie, ainsi qu'un concept de passage par valeur. Chacune des techniques de transmission prises en charge peut être appliquée à tous les éléments de données transmis ou utilisée sélectivement contre différents éléments.

Par défaut, COBOL transmettra les éléments de données par référence. Cela signifie que le programme appelant et le programme cible pourront lire et écrire dans la même zone de mémoire que celle représentée par la variable. Cela signifie que si le programme cible met à jour le contenu de la variable, ce changement sera visible pour le programme appelant une fois l'exécution terminée.

La phrase BY CONTENT permet à une copie de la variable transmise d'être transmise au programme cible. Bien que le programme cible puisse mettre à jour la variable, ces mises à jour ne seront pas visibles pour le programme appelant.

Remarque : Lors de la transmission de variables PAR RÉFÉRENCE ou PAR CONTENU, notez que vous pouvez envoyer des éléments de données de n'importe quel niveau. Ce qui signifie que vous pouvez transmettre des structures de données entières, ce qui est pratique pour traiter des enregistrements courants.

Vous pouvez également voir l'expression BY VALUE être utilisée dans une phrase CALL. BY VALUE est similaire à BY CONTENT, car une copie du contenu de la variable est transmise. La différence est que seul un sous-ensemble de types de données COBOL est pris en charge et vous ne pouvez spécifier que des éléments de données élémentaires. En effet, BY VALUE est principalement utilisé lorsque COBOL appelle un programme d'un autre langage (tel que C).

12.5.3 Spécification de la valeur de retour

Enfin, la phrase RETURNING est utilisée pour spécifier la variable qui doit être utilisée pour stocker la valeur de retour. Il peut s'agir de n'importe quel élément de données élémentaire déclaré dans la division de données. Notez que ceci est facultatif. Certains programmes peuvent ne rien renvoyer, ou vous pouvez avoir transmis des valeurs PAR RÉFÉRENCE au programme cible, auquel cas les mises à jour de ces variables seront visibles une fois le programme cible renvoyé.

12.6 Utilisation de cahiers

Si votre programme contient des séquences de code fréquemment utilisées, nous pouvons écrire la séquence de code une fois et la placer dans une bibliothèque de copie COBOL. Ces séquences de codes sont appelées cahiers. Ensuite, nous pouvons utiliser l'instruction COPY pour récupérer ces séquences de code et les inclure au moment de la compilation. L'utilisation de cahiers de cette manière éliminera le codage répétitif.

Nous aurions besoin de spécifier une bibliothèque de copie dans le JCL que nous avons utilisé. Si vous utilisez la procédure fournie (IGYWC, IGYWCL ou IGYWCLG), vous pouvez fournir une instruction DD au paramètre SYSLIB de l'étape COBOL dans la procédure. Par exemple:

```
//COBOL.SYSLIB  DD  DISP=SHR,DSN=Z99998.COPYLIB
```

Exemple 21. Instruction JCL SYSLIB

Cela indiquera au compilateur de rechercher les copybooks sur l'ensemble de données fourni.

Voyons un exemple d'utilisation de l'instruction COPY dans un programme.

Supposons qu'un COPYBOOK portant le nom de DOWORK soit stocké et contienne l'instruction suivante :

```
COMPUTE SPACE-AVAILABLE = TOTAL-FREE-SPACE
MOVE SPACE-AVAILABLE TO PRINT-SPACE
```

Exemple 22. Contenu du cahier DOWORK

Nous pouvons récupérer le cahier en utilisant l'instruction COPY :

```
PROCEDURE DIVISION.
...
    DISPLAY "RETRIEVE COPYBOOK".
    COPY DOWORK.
```

Exemple 23. COPIE de base

Les instructions à l'intérieur de la procédure DOWORK suivront alors l'instruction DISPLAY.

Contrairement aux sous-programmes, l'utilisation de copybooks ne transfère pas le contrôle sur un autre programme. Mais le code écrit à l'intérieur des copybooks ne sera transféré qu'une seule fois lors de la compilation. Ainsi, d'autres modifications apportées aux cahiers nécessiteront une recompilation du programme.

D'autre part, le code à l'intérieur d'un sous-programme ne sera invoqué que pendant l'exécution du programme. Par conséquent, en supposant que le sous-programme est lié dynamiquement, nous pouvons le modifier sans avoir besoin de recompiler le programme appelant.

12.7 Sommaire

En résumé, ce chapitre devrait fournir les bases nécessaires pour comprendre la programmation structurée et son lien avec COBOL et son importance pour la compréhension et la maintenance du code. De nombreux exemples de comment, quand et pourquoi mettre en œuvre des techniques clés ont été fournis et expliqués pour une meilleure compréhension. Vous devriez être capable d'identifier les différences fondamentales entre la programmation structurée (COBOL) et la programmation OO (Java). Vous devez également comprendre le concept général des meilleures pratiques dans la structure de la Procedure Division en référence à la conception et au contenu des paragraphes, aux options de contrôle du programme et aux moyens d'appeler d'autres programmes au sein du même système.

12.8 Laboratoire

Ce laboratoire utilise le programme COBOL CBL0033, situé dans votre ensemble de données id.CBL, ainsi que le travail JCL CBL0033J, situé dans votre ensemble de données id.JCL. Les travaux JCL sont utilisés pour compiler et exécuter les programmes COBOL, comme expliqué dans les chapitres précédents.

12.8.0.1 Utilisation de VSCode et Zowe Explorer

1. Prenez un moment et regardez le code source du programme COBOL fourni : CBL0033.
2. Comparez CBL0033 avec CBL0001 et CBL0002 du laboratoire précédent. Remarquez-vous les différences?

   une. Observez la nouvelle ligne COUNTER dans WORKING-STORAGE > DATA DIVISION.

   b. Observez que les paragraphes sont numérotés et qu'ils sont tous explicitement terminés par une phrase -END.

   c. Observez les nouveaux paragraphes READ-FIRST-RECORD, READ-TEN-RECORDS, READ-ANOTHER-RECORD, READ-NEXT-

RECORDS et CALLING-SUBPROGRAM dans la PRECEDURE DIVISION.

   d. Ces paragraphes effectuent la même boucle que dans CBL0001, mais en utilisant l'instruction PERFORM de différentes manières. Le SOUS-PROGRAMME D'APPEL appelle le programme HELLO, déjà présenté dans le deuxième Lab de ce cours.

3. Soumettre le travail : CBL0033J. Ce JCL compile d'abord le programme HELLO, compile ensuite CBL0033 et lie le résultat des deux compilations ensemble.

4. Affichez la sortie CBL0033J à l'aide de la section JOBS et ouvrez RUN:PRT-LINE, observez que le rapport est identique à CBL0001.

5. Affichez la sortie du programme cible HELLO à l'aide de la section JOBS et ouvrez RUN:SYSOUT.

13 Fichier de sortie

La conception d'une mise en page structurée qui est facile à lire et à comprendre est nécessaire pour formater la sortie. La conception d'une mise en page structurée implique des en-têtes de colonnes et un alignement variable utilisant des espaces, un format numérique, un format monétaire, etc. Ce chapitre vise à expliquer ce concept en utilisant un exemple de code COBOL pour concevoir des en-têtes de colonnes et aligner les noms de données sous ces en-têtes. À la fin du chapitre, vous êtes invité à effectuer un laboratoire qui pratique la mise en œuvre des composants couverts.

Une capacité de formatage de sortie de données COBOL qui mérite d'être notée mais qui n'est pas abordée dans ce chapitre est que COBOL est un langage informatique compatible avec le Web. COBOL inclut une transformation facile et rapide du code COBOL existant pour écrire JSON (JavaScript Object Notation) où la sortie est ensuite formatée pour un navigateur, un smartphone, etc. Fréquemment, les données critiques accessibles par un smartphone, comme un solde bancaire, est stocké et contrôlé par z/OS où un programme COBOL est chargé de récupérer et de renvoyer le solde bancaire sur le smartphone.

- **Examen du processus de sortie d'écriture COBOL**
  - **ENVIRONMENT DIVISION**
- **DESCRIPTEUR DE FICHIER**

  FILLER
- **En-têtes de rapport et de colonne**
  - **HEADER-2**
- **PROCEDURE DIVISION**
  - **MOVE sentence**
  - **PRINT-REC FROM sentences**
- **Laboratoire**

13.1 Examen du processus de sortie d'écriture COBOL

Cette section passe brièvement en revue certains aspects de la ENVIRONMENT DIVISION afin de comprendre comment elle s'articule avec le contenu de ce chapitre.

13.1.1 DIVISION ENVIRONNEMENT

La section "Traitement des fichiers" couvrait les noms choisis par le programmeur pour SELECT et ASSIGN respectif, tandis que ce chapitre se concentre sur la

sortie. La figure 1. montre un exemple de codage utilisant PRINT-LINE comme programmeur choisi le nom de fichier interne COBOL pour la sortie.

```
*--------------------
 ENVIRONMENT DIVISION.
*--------------------
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT PRINT-LINE ASSIGN TO PRTLINE.
     SELECT ACCT-REC   ASSIGN TO ACCTREC.
*SELECT clause creates an internal file name
*ASSIGN clause creates a name for an external data source,
*which is associated with the JCL DDNAME used by the z/OS
*e.g. ACCTREC is linked in JCL file CBL0001J to &SYSUID..DATA
*where &SYSUID. stands for Your z/OS user id
*e.g. if Your user id is Z54321,
*the data set used for ACCTREC is Z54321.DATA
```

Figure 1. SELECT and ASSIGN

13.2 FILE DESCRIPTOR

L'entrée File Description (FD), précédemment décrite dans la section de paragraphe FILE-CONTROL, représente le plus haut niveau d'organisation dans la FILE SECTION. L'entrée FD décrit la disposition du fichier définie par une instruction FILE-CONTROL SELECT précédente. Par conséquent, l'entrée FD relie le nom de fichier SELECT avec une disposition définie du nom de fichier. Un exemple de descripteur de fichier, FD, pour PRINT-LINE est illustré à la Figure 2. Ce qui suit le descripteur de fichier est une mise en page définie de PRINT-LINE.

13.2.1 FILLER

Observez le nom de données FILLER. Alors que la plupart des champs de données ont des noms uniques, FILLER est un nom de données de mot réservé COBOL, qui est utile pour le formatage de sortie. C'est en partie parce que FILLER alloue de l'espace mémoire sans avoir besoin d'un nom. De plus, la mémoire allouée par FILLER a une longueur définie dans la ligne de sortie et peut contenir des espaces ou n'importe quel littéral. La figure 2. montre plusieurs ESPACES DE VALEUR pour FILLER. Les ESPACES créent un espace blanc entre les éléments de données dans la sortie, ce qui est précieux pour garder le code lisible. Plus précisément dans la figure 2. FILLER PIC X(02) VALUE SPACES, représente la ligne de sortie contenant deux espaces.

```
*-------------
 DATA DIVISION.
*-------------
 FILE SECTION.
 FD  PRINT-LINE RECORDING MODE F.
*FD -- describes the layout of PRINT-LINE file,
*including level numbers, variable names, data types and lengths
*
 01  PRINT-REC.
     05  ACCT-NO-O      PIC X(8).
     05  FILLER         PIC X(02) VALUE SPACES.
*    FILLER -- COBOL reserved word used as data name to remove
*    the need of variable names only for inserting spaces
*
     05  LAST-NAME-O    PIC X(20).
     05  FILLER         PIC X(02) VALUE SPACES.
*    SPACES -- used for structured spacing data outputs rather
*    than using a higher PIC Clause length as in CBL0001.cobol,
*    which makes a good design practice and a legible output
*
 01 WS-CURRENT-DATE-DATA.
     05  ACCT-LIMIT-O   PIC $$,$$$,$$9.99.
*    The repeated $ characters revert to spaces and then one $
*    in front of the printed amount.
*
     05  FILLER         PIC X(02) VALUE SPACES.
     05  ACCT-BALANCE-O PIC $$,$$$,$$9.99.
     05  FILLER         PIC X(02) VALUE SPACES.
```

Figure 2. FILLER

13.3 En-têtes de rapport et de colonne

La rédaction d'en-têtes de rapport ou de colonne nécessite une mise en page de sortie structurée conçue par le programmeur. La figure 3. illustre une telle structure. La mise en page de la structure de sortie conçue est mise en œuvre au sein de la DATA DIVISION et comprend les en-têtes répertoriés et définis ci-dessous.

- **HEADER-1:**
  - Écrit un littéral
  - Exemple : « Rapport financier pour »

HEADER-2:

- Écrit des littéraux
- Exemples :
  - « Année » suivi d'un nom de variable
  - « Mois » suivi d'un nom de variable
  - « Jour » suivi d'un nom de variable

HEADER-3:

- Écrit des littéraux
- Exemples :
  - 'Accounts' suivi de l'espacement FILLER
  - « Last Name » suivi de l'espacement FILLER
  - 'Limit' suivi de l'espacement FILLER
  - 'Balance; suivi de l'espacement FILLER

- **EN-TÊTE-4 :**
  - Écrit des tirets suivis de l'espacement FILLER

```
WORKING-STORAGE SECTION.
01  HEADER-1.
    05  FILLER          PIC X(20) VALUE 'Financial Report for'.
    05  FILLER          PIC X(60) VALUE SPACES.
01  HEADER-2.
    05  FILLER          PIC X(05) VALUE 'Year '.
    05  HDR-YR          PIC 9(04).
    05  FILLER          PIC X(02) VALUE SPACES.
    05  FILLER          PIC X(06) VALUE 'Month '.
    05  HDR-MO          PIC X(02).
    05  FILLER          PIC X(02) VALUE SPACES.
    05  FILLER          PIC X(04) VALUE 'Day '.
    05  HDR-DAY         PIC X(02).
    05  FILLER          PIC X(56) VALUE SPACES.
01  HEADER-3.
    05  FILLER          PIC X(08) VALUE 'Account '.
    05  FILLER          PIC X(02) VALUE SPACES.
    05  FILLER          PIC X(10) VALUE 'Last Name '.
    05  FILLER          PIC X(15) VALUE SPACES.
    05  FILLER          PIC X(06) VALUE 'Limit '.
    05  FILLER          PIC X(06) VALUE SPACES.
    05  FILLER          PIC X(08) VALUE 'Balance '.
    05  FILLER          PIC X(40) VALUE SPACES.
01  HEADER-4.
    05  FILLER          PIC X(08) VALUE '--------'.
    05  FILLER          PIC X(02) VALUE SPACES.
    05  FILLER          PIC X(10) VALUE '----------'.
    05  FILLER          PIC X(15) VALUE SPACES.
    05  FILLER          PIC X(10) VALUE '----------'.
    05  FILLER          PIC X(02) VALUE SPACES.
    05  FILLER          PIC X(13) VALUE '-------------'.
    05  FILLER          PIC X(40) VALUE SPACES.
*
*HEADER -- structures for report or column headers,
*that need to be setup in WORKING-STORAGE so they can be used
*in the PROCEDURE DIVISION
*
```

Figure 3. Mise en page de la structure de sortie

13.3.1 HEADER-2

HEADER-2 inclut l'année, le mois, le jour du rapport ainsi que la zone FILLER, créant des espaces vides entre l'année, le mois et le jour, comme vous pouvez le voir sur la figure 3. La figure 4. est un exemple de la disposition du nom des données utilisée pour stocker les valeurs de CURRENT-DATE. Les informations fournies par COBOL dans CURRENT-DATE sont utilisées pour remplir le fichier de sortie dans HEADER-2.

```
 01 WS-CURRENT-DATE-DATA.
     05  WS-CURRENT-DATE.
         10  WS-CURRENT-YEAR         PIC 9(04).
         10  WS-CURRENT-MONTH        PIC 9(02).
         10  WS-CURRENT-DAY          PIC 9(02).
     05  WS-CURRENT-TIME.
         10  WS-CURRENT-HOURS        PIC 9(02).
         10  WS-CURRENT-MINUTE       PIC 9(02).
         10  WS-CURRENT-SECOND       PIC 9(02).
         10  WS-CURRENT-MILLISECONDS PIC 9(02).
*    This data layout is organized according to the ouput
*    format of the FUNCTION CURRENT-DATE.
*
```

Figure 4. Fonction intrinsèque CURRENT-DATE

13.4 PROCEDURE DIVISION

Les figures 1 à 4 sont une présentation de données conçue qui comprend une ligne de données et des en-têtes de rapport. En utilisant le stockage mappé par la ligne de données et les en-têtes de rapport, la logique de traitement COBOL peut écrire les en-têtes suivis par chaque ligne de données. La figure 5. est un exemple d'une logique d'exécution résultante utilisée pour écrire la structure de mise en page d'en-tête dans un programme COBOL.

```
 WRITE-HEADERS.
     MOVE FUNCTION CURRENT-DATE TO WS-CURRENT-DATE-DATA.
*         The CURRENT-DATE function returns an alphanumeric value
*         that represents the calendar date and time of day
*         provided by the system on which the function is
*         evaluated.
     MOVE WS-CURRENT-YEAR  TO HDR-YR.
     MOVE WS-CURRENT-MONTH TO HDR-MO.
     MOVE WS-CURRENT-DAY   TO HDR-DAY.
     WRITE PRINT-REC FROM HEADER-1.
     WRITE PRINT-REC FROM HEADER-2.
     MOVE SPACES TO PRINT-REC.
     WRITE PRINT-REC AFTER ADVANCING 1 LINES.
     WRITE PRINT-REC FROM HEADER-3.
     WRITE PRINT-REC FROM HEADER-4.
     MOVE SPACES TO PRINT-REC.
```

Figure 5. Logique d'exécution pour écrire la structure de mise en page de l'en-tête

13.4.1 MOVE sentences

La phrase COBOL MOVE, à la ligne 1, dans le paragraphe WRITE-HEADERS collecte les informations de date actuelle du système et stocke ces informations dans une disposition de nom de données définie, WS-CURRENT-DATE-DATA. L'utilisation du mot réservé FONCTION signifie que ce qui suit est une fonction intrinsèque COBOL. Les phrases des lignes 2, 3 et 4 stockent les informations de date, l'année, le mois et le jour, dans les zones de nom de données définies par HEADER-2, HDR-YR, HDR-MO et HDR-DAY. La phrase de la ligne 11, la dernière phrase du paragraphe, écrit des espaces dans la zone PRINT-REC pour effacer le stockage de ligne en vue de l'écriture des lignes de données.

13.4.2 PRINT-REC FROM phrases

PRINT-REC est ouvert pour la sortie résultant en PRINT-REC FROM suivi d'une écriture PRINT-REC FROM un en-tête différent ou une disposition de nom de données définie. Les phrases des lignes 5 et 6 écrivent les noms de données d'en-tête définis PRINT-REC FROM, HEADER-1 et HEADER-2, de la figure 3. Les noms de données de descripteur de fichier PRINT-REC de la figure 2. sont effectivement remplacés par le contenu du noms de données d'en-tête dans la figure 3. écrits dans la sortie. Les phrases des lignes 7 et 8 donnent lieu à une ligne vierge écrite entre les en-têtes. Les phrases des lignes 9 et 10 écrivent les noms de données PRINT-REC FROM définis HEADER-3 et HEADER-4 de la figure 3. Les noms de données de descripteur de fichier PRINT-REC de la figure 2. sont effectivement remplacés par le contenu des noms de données d'en-tête dans la figure 3.

13.5 Labo

Ce laboratoire utilise deux programmes COBOL, CBL0004 et CBL0005, situés dans votre ensemble de données id.CBL, ainsi que deux travaux JCL, CBL0004J et CBL0005J, situés dans votre ensemble de données id.JCL. Les travaux JCL sont utilisés pour compiler et exécuter les programmes COBOL, comme expliqué dans les chapitres précédents.

13.5.0.1 Utilisation de VSCode et Zowe Explorer

1. Soumettre le job: CBL0004J
2. Observez le rapport écrit avec des en-têtes comme la figure 6 ci-dessous.

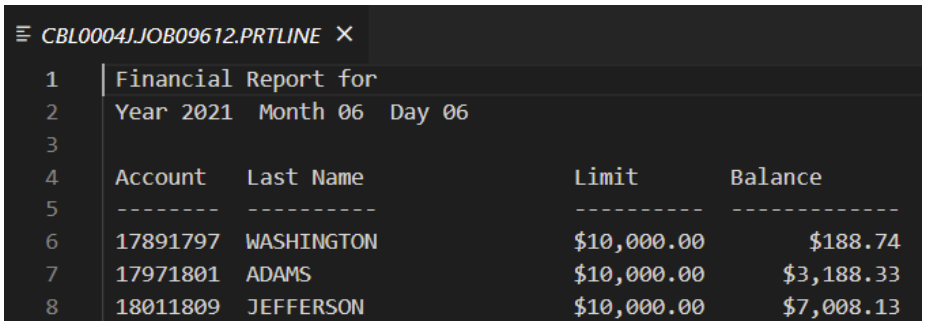


Figure 6. Rapport avec en-têtes

3. Soumettre le travail : CBL0005J
4. Observez que les lignes de données du rapport sont écrites sans le symbole monétaire du dollar, illustré à la figure 7.

Figure 7. Aucun symbole de devise dans la sortie

5. Modifiez id.CBL(CBL0005) pour inclure le symbole de devise dollar dans le rapport.

   Indice : comparer avec CBL0004 ligne 25

6. Re-soumettre le job : CBL0005J
7. Observez que les lignes de données du rapport doivent maintenant inclure le symbole monétaire du dollar.

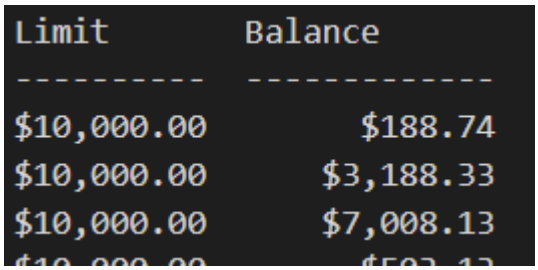


Figure 8. Symbole monétaire ajouté à la sortie

14 Expressions conditionnelles

Ce chapitre explique comment les programmes prennent des décisions en fonction de la logique écrite du programmeur. Plus précisément, les programmes prennent ces décisions au sein de la PROCEDURE DIVISION du code source. Nous développerons plusieurs sujets concernant les expressions conditionnelles écrites en COBOL à travers des explications utiles, des exemples et éventuellement une mise en œuvre en laboratoire.

- **Logique booléenne, opérateurs, opérandes et identifiants**
  - **Expressions conditionnelles et opérateurs COBOL**
  - **Exemples d'expressions conditionnelles utilisant des opérateurs booléens**
- **Mots et terminologie réservés à l'expression conditionnelle**
  - **IF, EVALUATE, PERFORM et SEARCH**
  - **États conditionnels**
  - **Noms conditionnels**
- **Opérateurs conditionnels**
- **Expressions conditionnelles**
  - **instructions IF ELSE (THEN)**
  - **instructions EVALUATE**
  - **instructions PERFORM**
  - **instructions SEARCH**
- **Conditions**
  - **Conditions relationnelles**
  - **Conditions de classes**
  - **Conditions de signature**
- **Labo**

14.1 Logique booléenne, opérateurs, opérandes et identifiants

Les programmes prennent des décisions basées sur la logique écrite du programmeur. Les décisions de programme sont prises en utilisant la logique booléenne où une expression conditionnelle est soit vraie ou fausse, oui ou non. Un exemple simple serait une variable nommée « LANGUAGE ». De nombreux langages de programmation existent ; par conséquent, la valeur de la variable LANGUE pourrait être Java, COBOL, etc… Supposons que la valeur

de LANGUE soit COBOL. La logique booléenne est, SI LANGUE = COBOL, ALORS AFFICHAGE COBOL, SINON AFFICHAGE NON COBOL. IF déclenche la logique booléenne pour déterminer la condition vrai/faux, oui/non, appliquée à LANGUE = COBOL qui est l'expression conditionnelle. Le résultat de la condition IF exécute ce qui suit THEN lorsque la condition est vraie et exécute ce qui suit ELSE lorsque la condition est fausse.

Le verbe booléen SI opère sur deux opérandes ou identifiants. Dans l'exemple ci-dessus, LANGUE est un opérande et COBOL est un opérande. Un opérateur relationnel booléen compare les valeurs de chaque opérande.

14.1.1 Expressions conditionnelles et opérateurs COBOL

Trois des types les plus courants d'expressions conditionnelles COBOL sont :

1. Condition de relation générale
2. Condition de classe
3. État du signe

Une liste d'opérateurs relationnels booléens COBOL pour chacun des types courants d'expressions conditionnelles COBOL est représentée dans les figures 1, 2 et 3 ci-dessous.

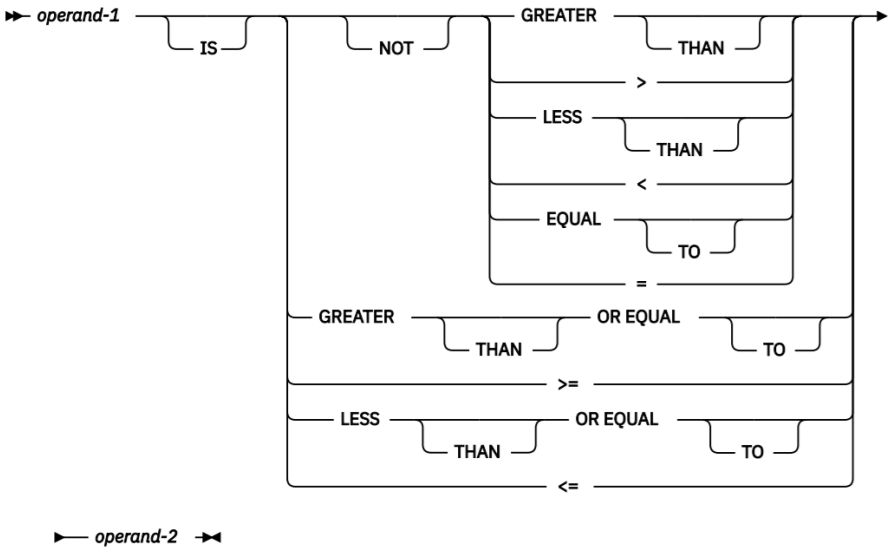


Figure 1. Opérateurs de condition de relation générale

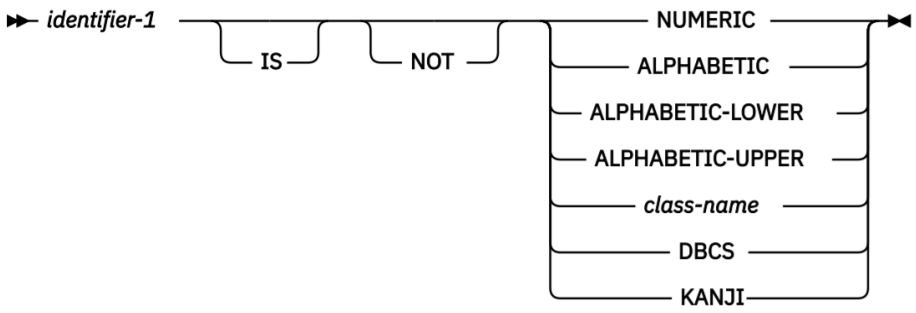


Figure 2. Opérateurs de condition de classe

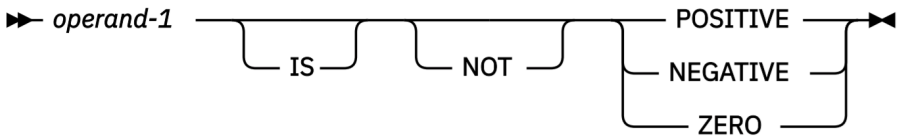


Figure 3. Opérateurs de condition de signe

14.1.2 Exemples d'expressions conditionnelles utilisant des opérateurs booléens

Une expression conditionnelle simple peut s'écrire sous la forme :

SI 5 > 1 THEN DISPLAY '5 is greater than 1' ELSE DISPLAY '1 is greater than 5'.

Les expressions conditionnelles composées sont mises entre parenthèses et leurs opérateurs booléens sont :

'AND'

"OR"

L'extrait de code ci-dessous montre une expression conditionnelle composée à l'aide de l'opérateur booléen AND.

IF (5 > 1 AND 1 > 2) THEN .... ELSE ....

Cette expression conditionnelle est évaluée à faux car si 5 > 1 est vrai, 1 > 2 est faux. L'opération AND requiert que les deux expressions soient vraies pour retourner vrai pour l'expression de condition composée. Montrons une autre implémentation, cette fois en utilisant l'opérateur booléen OR.

IF (5 > 1 OU 1 > 2) THEN .... ELSE ....

Cette expression conditionnelle est évaluée à vrai car si 1 > 2 est faux, 5 > 1 est vrai. L'opération OR requiert qu'une seule des expressions soit vraie pour renvoyer la valeur vraie pour l'ensemble de l'expression de condition composée. D'autres opérateurs conditionnels utilisés pour les conditions de relation, de classe et de signe sont abordés plus loin dans le chapitre.

14.2 Expression conditionnelle mots réservés et terminologie

Jusqu'ici dans ce livre, nous avons évoqué la nécessité et l'utilisation des mots réservés COBOL. Cette section vise à développer le sujet des mots réservés, en particulier ceux qui sont utilisés lors du traitement des expressions conditionnelles.

14.2.1 IF, EVALUATE, PERFORM and SEARCH

Ce sont des mots réservés COBOL disponibles pour le traitement des expressions conditionnelles, où une condition est un état qui peut être défini ou modifié.

14.2.2 États conditionnels

TRUE et FALSE sont parmi les états conditionnels les plus courants.

14.2.3 Noms conditionnels

Un nom conditionnel est un nom de variable défini par le programmeur avec l'état de condition TRUE. Les noms conditionnels sont déclarés dans la WORKING STORAGE SECTION avec un numéro de niveau 88. L'objectif du niveau 88 est d'améliorer la lisibilité en simplifiant les instructions IF et PERFORM UNTIL.

Le nom de données conditionnel de niveau 88 reçoit une valeur au moment de la compilation. Le programme ne peut pas modifier le nom de données à 88 niveaux pendant l'exécution du programme. Cependant, le programme peut modifier la valeur du nom de données dans le numéro de niveau au-dessus du nom de données conditionnel à 88 niveaux. USA-STATE de niveau 01 dans l'exemple 1. peut être modifié. Une expression de programme faisant référence au nom de données de niveau 88 n'est vraie que lorsque la valeur actuelle du nom de données de niveau précédent, USA-STATE, est égale à la valeur attribuée au nom de données conditionnel de niveau WORKING-STORAGE 88.

Observez l'exemple 1. « L'État n'est pas le Texas » est écrit à la suite du premier IF STATE car la valeur de USA-STATE est AZ, ce qui n'est pas égal au nom de données conditionnel de niveau 88, TX. Le deuxième IF STATE écrit : « L'État est le Texas » car la valeur de USA-STATE est égale à la valeur de niveau 88 attribuée de TX.

```
WORKING-STORAGE.
01 USA-STATE       PIC X(2) VALUE SPACES.
   88 STATE        VALUE 'TX'.
 ....
 ....
PROCEDURE DIVISION.
 ....
 ....
MOVE 'AZ' TO USA-STATE.
```

```
....
....
IF STATE DISPLAY 'The State is Texas'
   ELSE DISPLAY  'The State is not Texas'
END-IF.
....
....
MOVE 'TX' TO USA-STATE.
....
....
IF STATE DISPLAY 'The State is Texas'
   ELSE DISPLAY  'The State is not Texas'
END-IF.
```

Exemple 1. Utilisation d'un nom conditionnel de niveau 88

De nombreux noms de données conditionnels de niveau 88 peuvent suivre un nom de données de niveau 01. Par conséquent, une référence IF à une expression de nom de données de niveau 01 peut avoir de nombreuses valeurs qui retourneraient true.

D'autres noms de données de numéro de niveau nécessitent que l'expression de condition inclue un opérateur booléen, comme illustré dans l'exemple 2. , où une valeur peut être stockée dans le nom de données STATE de niveau 05 pour être comparée à une autre valeur stockée. Par conséquent, un peu de codage supplémentaire est nécessaire.

```
WORKING-STORAGE.
01 USA-STATE.
   05 STATE       PIC X(2) VALUE SPACES.
....
....
PROCEDURE DIVISION.
....
....
MOVE 'AZ' TO STATE.
....
....
IF STATE = 'TX' DISPLAY 'The State is Texas'
   ELSE DISPLAY  'The State is not Texas'
END-IF.
....
....
MOVE 'TX' TO STATE.
....
....
IF STATE = 'TX' DISPLAY 'The State is Texas'
```

```
    ELSE DISPLAY  'The State is not Texas'
END–IF .
```

Exemple 2. Sans nom conditionnel de niveau 88

14.3 Opérateurs conditionnels

Les opérateurs relationnels comparent des données numériques, des chaînes de caractères ou des données logiques. Le résultat de la comparaison, vrai (1) ou faux (0), peut être utilisé pour prendre une décision concernant le déroulement du programme. Le tableau 1 affiche une liste d'opérateurs relationnels, leur écriture et leur signification.

Relational operator	Can be written	Meaning
IS GREATER THAN	IS >	Greater than
IS NOT GREATER THAN	IS NOT >	Not greater than
IS LESS THAN	IS <	Less than
IS NOT LESS THAN	IS NOT <	Not less than
IS EQUAL TO	IS =	Equal to
IS NOT EQUAL TO	IS NOT =	Not equal to
IS GREATER THAN OR EQUAL TO	IS >=	Is greater than or equal to
IS LESS THAN OR EQUAL TO	IS <=	Is less than or equal to

Tableau 1. Opérateur relationnel

14.4 Expressions conditionnelles

Une expression conditionnelle amène le programme objet à sélectionner des chemins de contrôle alternatifs, en fonction de la valeur de vérité d'un test. Les expressions conditionnelles sont spécifiées dans les instructions EVALUATE, IF, PERFORM et SEARCH.

14.4.1 Instructions IF ELSE (THEN)

Les instructions IF sont utilisées pour implémenter ou évaluer des opérations relationnelles. IF ELSE permet de coder un choix entre deux traitements et l'inclusion du mot THEN est facultative. Lorsqu'une instruction IF est présente, les instructions qui suivent l'instruction IF sont traitées en fonction de la véracité de l'expression conditionnelle. Les instructions sont traitées jusqu'à ce qu'une instruction END-IF ou ELSE soit rencontrée. L'instruction ELSE peut apparaître sur n'importe quelle ligne avant END-IF. Les instructions IF, quel que soit le nombre de lignes, se terminent explicitement à l'aide de END-IF.

Considérez ceci, pendant le traitement du programme, quelque chose se produit pour changer la valeur du nom de données, FACIAL-EXP. Les instructions

suivantes, l'expression conditionnelle, doivent vérifier la valeur du nom de données pour décider de la manière de procéder dans le programme. Exemplifié dans l'exemple 3. par les instructions THEN DISPLAY et ELSE DISPLAY.

```
IF FACIAL–EXP = 'HAPPY' THEN
    DISPLAY 'I am glad you are happy'
ELSE DISPLAY 'What can I do to make you happy'
END–IF.
```

Exemple 3. IF, THEN, ELSE, END-IF instruction

14.4.2 Instructions EVALUATE

Les instructions EVALUATE sont utilisées pour coder un choix parmi trois actions possibles ou plus. Le terminateur explicite d'une instruction EVALUATE est END-EVALUATE. L'instruction EVALUATE est une forme développée de l'instruction IF qui vous permet d'éviter l'imbrication d'instructions IF, une source courante d'erreurs logiques et de problèmes de débogage. EVALUATE fonctionne à la fois sur les valeurs de chaîne de texte et sur les variables numériques. En utilisant le nom-conditionnel FACIAL-EXP, observez le code COBOL implémentant une instruction EVALUATE, illustré dans l'exemple 4.

```
EVALUATE FACIAL–EXP
 WHEN 'HAPPY'
   DISPLAY 'I am glad you are happy'
 WHEN 'SAD'
   DISPLAY 'What can I do to make you happy'
 WHEN 'PERPLEXED'
   DISPLAY 'Can you tell me what you are confused about'
 WHEN 'EMOTIONLESS'
   DISPLAY 'Do you approve or disapprove'
END–EVALUATE
```

```*Exemple 4. ÉVALUATION de l'instruction*

### Instructions PERFORM

Une expression PERFORM avec UNTIL est une expression conditionnelle. Dans le for

WORKING-STORAGE. 01 FACIAL-EXP PIC X(11) VALUE SPACES. 88 HAPPY VALUE 'HAPPY'. .... .... PROCEDURE DIVISION. .... .... PERFORM SAY-SOMETHING-DIFFERENT UNTIL HAPPY END-PERFORM.

*Exemple 5. Instruction PERFORM avec un nom conditionnel à 88 niveaux*

Il est également possible d'utiliser l'instruction PERFORM sans utiliser de nom
```

WORKING-STORAGE. 01 FACIAL-EXP PIC X(11) VALUE SPACES. …. …. PROCEDURE DIVISION. …. …. PERFORM SAY-SOMETHING-DIFFERENT UNTIL FACIAL-EXP = “HAPPY” END-PERFORM.

∗Exemple 6. Instruction PERFORM sans nom conditionnel à 88 niveaux∗

instructions SEARCH

L'instruction SEARCH recherche dans une table un élément qui satisfait la condit

WORKING-STORAGE. 01 FACIAL-EXP-TABLE REDEFINES FACIAL-EXP-LIST. 05 FACIAL-EXP PIC X(11) OCCURS n TIMES INDEXED BY INX-A. 88 HAPPY VALUE “HAPPY”. …. …. PROCEDURE DIVISION. …. …. SEARCH FACIAL-EXP WHEN HAPPY(INX-A) DISPLAY ‘I am glad you are happy’ END-SEARCH “‘*Exemple 7. Instruction RECHERCHE QUAND*

14.5 Conditions

Une expression conditionnelle peut être spécifiée dans des conditions simples ou des conditions complexes. Les conditions simples et complexes peuvent être entourées de n'importe quel nombre de parenthèses appariées ; les parenthèses, cependant, ne changent pas si la condition est simple ou complexe. Cette section couvrira trois des cinq conditions simples :

- Relation
- Classer
- Signe

14.5.1 Conditions relationnelles

Une condition de relation spécifie la comparaison de deux opérandes. L'opérateur relationnel qui joint les deux opérandes spécifie le type de comparaison. La condition de relation est vraie si la relation spécifiée existe entre les deux opérandes ; la condition de relation est fausse si la relation spécifiée n'existe pas. Une liste de quelques comparaisons définies est fournie :

- Comparaisons numériques - Deux opérandes de classe numérique
- Comparaisons alphanumériques - Deux opérandes de classe alphanumérique
- Comparaisons DBCS (Double Byte Character Set) - Deux opérandes de la classe DBCS
- Comparaisons nationales - Deux opérandes de classe nationale

14.5.2 Conditions de classe

La condition de classe détermine si le contenu d'un élément de données est alphabétique, alphabétique-inférieur, alphabétique-supérieur, numérique, DBCS, KANJI, ou contient uniquement les caractères de l'ensemble de caractères spécifié par la clause CLASS, comme défini dans la clause SPECIAL- paragraphe NOMS de la division environnement. Vous trouverez ci-dessous une liste de quelques formulaires valides sur la condition de classe pour différents types d'éléments de données.

- Numérique
  - IS NUMERIC or IS NOT NUMERIC
- Alphabétique
  - IS ALPHABETIC or IS NOT ALPHABETIC
  - IS ALPHABETIC-LOWER / ALPHABETIC-UPPER
  - IS NOT ALPHABETIC-LOWER / ALPHABETIC-UPPER
- DBCS
  - IS DBCS or IS NOT DBCS
  - IS KANJI or IS NOT KANJI

14.5.3 Conditions de signe

La condition de signe détermine si la valeur algébrique d'un opérande numérique est supérieure, inférieure ou égale à zéro. Un opérande non signé est POSITIF ou ZÉRO. Lorsqu'une variable conditionnelle numérique est définie avec un signe, les éléments suivants sont disponibles :

- IS POSITIVE
- IS NEGATIVE
- IS ZERO

Remarque : Pour en savoir plus sur ces conditions, veuillez visiter le lien :

https://www.ibm.com/docs/en/cobol-zos/6.3?topic=structure-conditional-expressions

14.6 Laboratoire

Ce laboratoire nécessite deux programmes COBOL, CBL0006 et CBL0007 et deux jobs JCL respectifs, CBL0006J et CBL0007J, pour compiler et exécuter les programmes COBOL. Tout cela vous est fourni dans votre VSCode - Zowe Explorer.

14.6.0.1 Utilisation de VSCode et Zowe Explorer :

1. Prenez un moment et regardez le code source des deux programmes COBOL fournis : CBL0006 et CBL0007.
2. Comparez CBL0006 avec CBL0005 du laboratoire précédent. Remarquez-vous les différences?

   une. Observez la nouvelle ligne CLIENTS-PER-STATE au sein de la DIVISION WORKING-STORAGE > PROCEDURE.

   b. Observez le nouveau paragraphe IS-STATE-VIRGINIA au sein de cette même division.

   c. Ce paragraphe vérifie si le client est originaire de Virginie. Si cette condition est remplie (vrai), alors le programme doit ajouter 1 au total de clients de Virginie.

   ré. Le programme écrit « Virginia Clients = », dans la dernière ligne du rapport.
3. Soumettre CBL0006J
4. Affichez le résultat du travail dans la section JOBS et vérifiez que les étapes mentionnées ci-dessus ont été exécutées.

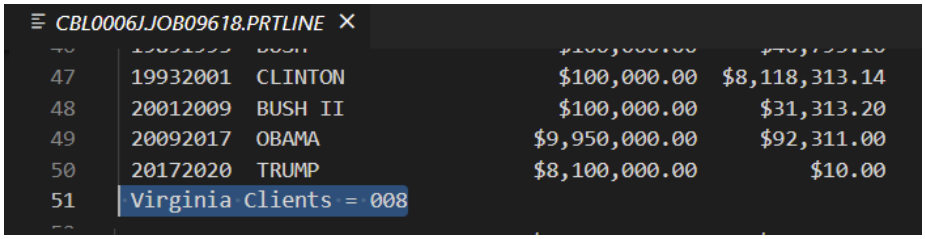


Figure 4. Sortie Id.JCL(CBL0006J)

5. Soumettre CBL0007J
6. Recherchez l'erreur de compilation, IGYPS2113-E, dans la sortie du travail.
7. Allez-y et modifiez id.CBL(CBL0007) pour corriger l'erreur de syntaxe décrite par le message IGYPS2113-E.*
8. Re-soumettre CBL0007J
9. Vérifiez que l'erreur de syntaxe a été corrigée en obtenant un fichier de sortie sans erreur.

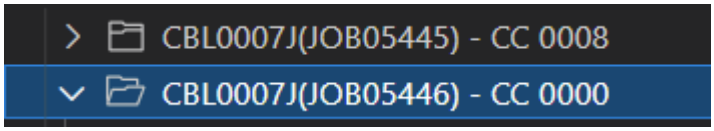


Figure 5. Compilation réussie

Conseils de laboratoire

```
*
IS-STATE-VIRGINIA.
    IF USA-STATE = 'Virginia' THEN
        ADD 1 TO VIRGINIA-CLIENTS
    END-IF.
*
```

15 Expressions arithmétiques

Ce chapitre vise à introduire le concept d'implémentation d'expressions arithmétiques dans les programmes COBOL. Nous passerons en revue le concept de base des expressions arithmétiques, des opérateurs, des instructions, des limitations, des opérandes d'instructions, ainsi que la priorité des opérations dans les expressions. Vous pourrez suivre un exemple complet illustrant l'utilisation d'expressions arithmétiques dans un programme COBOL que vous avez vu dans les chapitres et les laboratoires précédents. Après le chapitre est un laboratoire pour pratiquer la mise en œuvre de ce que vous avez appris.

- **Qu'est-ce qu'une expression arithmétique ?**
  - **Opérateurs arithmétiques**
  - **Déclarations arithmétiques**
- **Règles de priorité des expressions arithmétiques**
  - **Parenthèses**
- **Limites des expressions arithmétiques**
- **Opérandes d'instructions arithmétiques**
  - **Taille des opérandes**
- **Exemples d'énoncés arithmétiques COBOL**
- **Labo**

15.1 Qu'est-ce qu'une expression arithmétique ?

Les expressions arithmétiques sont utilisées comme opérandes de certaines instructions conditionnelles et arithmétiques. Une expression arithmétique peut être constituée de l'un des éléments suivants :

1. Un identifiant décrit comme une rubrique élémentaire numérique (y compris les fonctions numériques).
2. Un littéral numérique.
3. La constante figurative ZÉRO.
4. Identifiants et littéraux, tels que définis aux points 1, 2 et 3, séparés par des opérateurs arithmétiques.
5. Deux expressions arithmétiques, telles que définies aux éléments 1, 2, 3 ou 4, séparées par un opérateur arithmétique.
6. Une expression arithmétique, telle que définie aux éléments 1, 2, 3, 4 ou 5, entre parenthèses.
7. Toute expression arithmétique peut être précédée d'un opérateur unaire.

Les identificateurs et les littéraux qui apparaissent dans les expressions arithmétiques doivent représenter soit des éléments élémentaires numériques, soit des littéraux numériques sur lesquels l'arithmétique peut être effectuée. Si la valeur d'une expression à élever à une puissance est zéro, l'exposant doit avoir une valeur supérieure à zéro. Sinon, la condition d'erreur de taille existe. Dans tous les cas où aucun nombre réel n'existe à la suite d'une évaluation, la condition d'erreur de taille existe.

15.1.1 Opérateurs arithmétiques

Cinq opérateurs arithmétiques binaires et deux opérateurs arithmétiques unaires peuvent être utilisés dans les expressions arithmétiques. Ces opérateurs sont représentés par des caractères spécifiques qui doivent être précédés et suivis d'un espace. Cependant, aucun espace n'est requis entre une parenthèse gauche et un opérateur unaire. Ces opérateurs arithmétiques binaires et unaires sont répertoriés dans le tableau 1.

Binary operator	Meaning	Unary operator	Meaning
+	Addition	+	Multiplication by +1
-	Subtraction	-	Multiplication by -1
*	Multiplication		
/	Division		
**	Exponentiation		

Tableau 1. Opérateurs arithmétiques

15.1.2 Instructions arithmétiques

Les déclarations arithmétiques sont utilisées pour les calculs. Les opérations individuelles sont spécifiées par les instructions ADD, SUBTRACT, MULTIPLY et DIVIDE. Ces opérations individuelles peuvent être combinées symboliquement dans une formule qui utilise l'instruction COMPUTE pour faciliter la programmation et les performances. L'instruction COMPUTE affecte la valeur d'une expression arithmétique à un ou plusieurs éléments de données. Avec l'instruction COMPUTE, les opérations arithmétiques peuvent être combinées sans les restrictions de réception d'éléments de données imposées par les règles des instructions ADD, SUBTRACT, MULTIPLY et DIVIDE. Lorsque des opérations arithmétiques sont combinées, l'instruction COMPUTE peut être plus efficace que les instructions arithmétiques distinctes écrites dans une série. Pour ces raisons, il est recommandé d'utiliser l'instruction COMPUTE pour la plupart des évaluations arithmétiques plutôt que les instructions ADD , SUBTRACT , MULTIPLY et DIVIDE. Souvent, vous ne pouvez coder qu'une seule instruction COMPUTE au lieu de plusieurs instructions arithmétiques individuelles. L'instruction « COMPUTE » affecte le résultat d'une expression arithmétique à un ou plusieurs éléments de données, par exemple :

COMPUTE z = a + b / c ** d − e

COMPUTE x y z = a + b / c ** d − e

Certains calculs arithmétiques peuvent être plus intuitifs en utilisant des instructions arithmétiques autres que COMPUTE . Vous pouvez également préférer utiliser l'instruction DIVIDE (avec sa phrase REMAINDER) pour la division dans laquelle vous souhaitez traiter un reste. La fonction intrinsèque "REM" permet également de traiter un reste.

15.2 Règles de priorité des expressions arithmétiques

Les règles de l'ordre des opérations ont été martelées dans votre tête au fil des années d'apprentissage des mathématiques, vous vous souvenez du PEMDAS classique (parenthèses, exposants, multiplier, diviser, additionner, soustraire) ? Les expressions arithmétiques en COBOL ne sont pas exemptes de ces règles et utilisent souvent des parenthèses pour spécifier l'ordre dans lequel les éléments doivent être évalués.

15.2.1 Parenthèses

Les parenthèses sont utilisées pour indiquer les modifications apportées à l'ordre normal des opérations (règles de priorité). Une expression arithmétique entre parenthèses est évaluée en premier et le résultat est utilisé dans le reste de l'expression. Lorsque des expressions sont contenues dans des parenthèses imbriquées, l'évaluation se fait de l'ensemble le moins inclusif à l'ensemble le plus inclusif. Cela signifie que vous travaillez de l'expression la plus intérieure entre parenthèses à la plus extérieure. La priorité pour la résolution d'une expression arithmétique dans Enterprise COBOL avec des parenthèses est :

1. Parenthèses (simplifiez l'expression à l'intérieur)
2. Opérateur unaire
3. Exposants
4. Multiplication et division (de gauche à droite)
5. Addition et soustraction (de gauche à droite)

Les parenthèses éliminent les ambiguïtés dans la logique où des opérations consécutives apparaissent au même niveau hiérarchique ou modifient la séquence hiérarchique normale d'exécution si nécessaire. Lorsque l'ordre des opérations consécutives au même niveau hiérarchique n'est pas complètement spécifié par des parenthèses, l'ordre est de gauche à droite.

Une expression arithmétique ne peut commencer que par une parenthèse gauche, un opérateur unaire ou un opérande (c'est-à-dire un identifiant ou un littéral). Il ne peut se terminer que par une parenthèse droite ou un opérande. Une expression arithmétique doit contenir au moins une référence à un identifiant ou un littéral.

Il doit y avoir une correspondance un à un entre les parenthèses gauche et droite dans une expression arithmétique, chaque parenthèse gauche étant placée à gauche de sa parenthèse droite correspondante. Si le premier opérateur d'une expression arithmétique est un opérateur unaire, il doit être immédiatement précédé d'une parenthèse gauche si cette expression arithmétique suit immédiatement un identificateur ou une autre expression arithmétique.

15.3 Limitations des expressions arithmétiques

Les exposants des expressions exponentielles à virgule fixe ne peuvent pas contenir plus de neuf chiffres. Le compilateur tronquera tout exposant avec plus de neuf chiffres. En cas de troncature, le compilateur émettra un message de diagnostic si l'exposant est un littéral ou une constante ; si l'exposant est une variable ou un nom de données, un message de diagnostic est émis au moment de l'exécution.

L'explication détaillée des expressions exponentielles à virgule fixe est un sujet avancé et dépasse le cadre du chapitre. Cependant, il est fait référence à des expressions exponentielles à virgule fixe pour votre conscience à mesure que vous progressez dans votre niveau d'expérience avec la programmation COBOL et l'arithmétique appliquée aux représentations de données internes.

15.4 Opérandes d'instructions arithmétiques

Les descriptions de données des opérandes dans une instruction arithmétique n'ont pas besoin d'être les mêmes. Tout au long du calcul, le compilateur effectue toute conversion de données et alignement de la virgule décimale nécessaires.

15.4.1 Taille des opérandes

Si l'option du compilateur ARITH(COMPAT) est active, la taille maximale de chaque opérande est de 18 chiffres décimaux. Si l'option du compilateur ARITH(EXTEND) est active, la taille maximale de chaque opérande est de 31 chiffres décimaux.

Le composé d'opérandes est une donnée hypothétique résultant de l'alignement des opérandes à la virgule décimale puis de leur superposition les uns sur les autres. Le tableau 2 montre comment déterminer le composé d'opérandes pour les instructions arithmétiques.

Si l'option du compilateur ARITH(COMPAT) est active, le composite d'opérandes peut comporter au maximum 30 chiffres. Si l'option du compilateur ARITH(EXTEND) est active, le composite d'opérandes peut comporter au maximum 31 chiffres.

Statement	Determination of the composite of operands
SUBTRACT, ADD	Superimposing all operands in a given statement, except those following the word GIVING.
MULTIPLY	Superimposing all receiving data-items
DIVIDE	Superimposing all receiving data items except the REMAINDER data-item
COMPUTE	Restriction does not apply

Tableau 2. Comment le composite d'opérandes est déterminé

Dans toutes les instructions arithmétiques, il est important de définir des données avec suffisamment de chiffres et de décimales pour garantir la précision requise dans le résultat. Les détails de la précision arithmétique sont disponibles dans IBM Enterprise COBOL Programming Guide Annexe A.

De plus, le chapitre 20. « Instructions PROCEDURE DIVISION » de IBM Enterprise COBOL Language Reference comprend une explication détaillée des capacités des instructions DIVIDE et COMPUTE appliquées à la gestion de ROUNDING et ON SIZE ERROR.

15.5 Exemples d'instructions arithmétiques COBOL

Dans cette section, le code source COBOL utilisé dans les ateliers précédents sera modifié pour démontrer le traitement arithmétique. La figure 1. montre les éléments de données de numéro de niveau dans la section WORKING-STORAGE. Les éléments de données seront utilisés pour totaliser la limite du compte client et le solde du compte client. Observez que la valeur initiale est ZÉRO.

Figure 1. Éléments de données au niveau numérique (1)

La figure 2 montre un autre exemple d'éléments de données de niveau numérique dans la section WORKING-STORAGE. Ces éléments de données sont des lignes de fin de rapport qui sont utilisées pour écrire une limite de compte totale formatée et un solde de compte total pour tous les clients dans le rapport. Observez les éléments de données TLIMIT et TBALANCE avec de grandes clauses d'image de nombre de devises.

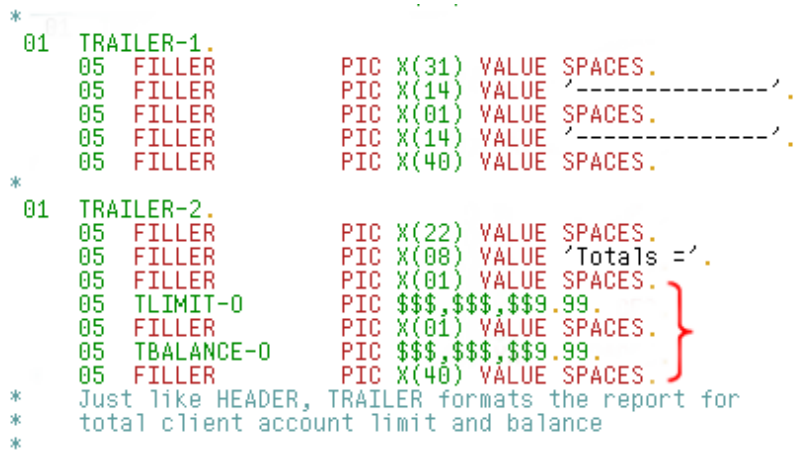


Figure 2. Éléments de données au niveau numérique (2)

Dans la figure 3. le paragraphe READ-NEXT-RECORD, situé dans la PROCEDURE DIVISION, comprend une instruction PERFORM LIMIT-BALANCE-TOTAL. Le résultat de cette instruction est de transférer le contrôle au paragraphe LIMIT-BALANCE-TOTAL, situé dans la PROCEDURE DIVISION, pour exécuter les instructions COMPUTE.

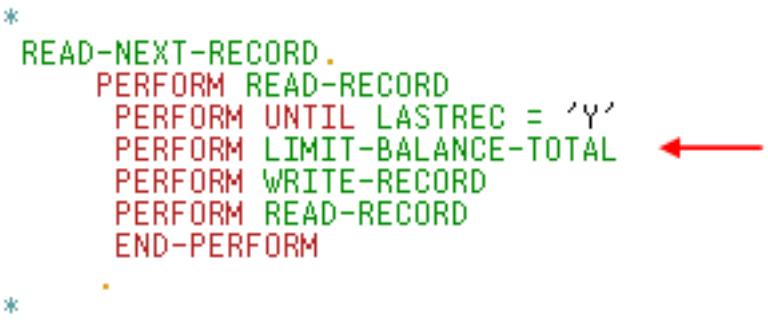


Figure 3. LIRE-ENREGISTREMENT SUIVANT.

La figure 4 est un exemple de deux instructions COMPUTE dans le paragraphe LIMIT-BALANCE-TOTAL. Notez que les résultats des instructions COMPUTE consistent à ajouter le client ACCT-LIMIT au TLIMIT actuel et à ajouter le client ACCT-BALANCE aux totaux TBALANCE chaque fois que le paragraphe est exécuté, ce qui correspond à une fois pour chaque enregistrement client lu dans notre exemple.

```
*     The LIMIT-BALANCE-TOTAL paragraph performs an arithmetic
*     statement for each client through the loop,
*     in order to calculate the final limit and balance report.
*
 LIMIT-BALANCE-TOTAL.
     COMPUTE TLIMIT   = TLIMIT   + ACCT-LIMIT   END-COMPUTE
     COMPUTE TBALANCE = TBALANCE + ACCT-BALANCE END-COMPUTE
     .
*    The COMPUTE verb assigns the value of the arithmetic
*    expression to the TLIMIT and TBALANCE data items.
*    Since the expression only includes an addition operation,
*    the statements can also be written as:
*    ADD ACCT-LIMIT TO TLIMIT.
*    ADD ACCT-BALANCE TO TBALANCE.
*    Or, alternatively specifying the target variable:
*    ADD ACCT-LIMIT TO TLIMIT GIVING TLIMIT.
*    ADD ACCT-BALANCE TO TBALANCE GIVING TLIMIT.
*    A END-COMPUTE or END-ADD stetement is optional.
*
```

Figure 4. Instructions COMPUTE

Le paragraphe WRITE-TLIMIT-TBALANCE illustré à la figure 5. est positionné dans la PROCEDURE DIVISION pour être exécuté immédiatement après la lecture de tous les enregistrements et avant le dernier paragraphe qui ferme les fichiers et termine l'exécution du programme.

```
*
 WRITE-TLIMIT-TBALANCE.
      MOVE TLIMIT   TO TLIMIT-O.
      MOVE TBALANCE TO TBALANCE-O.
      WRITE PRINT-REC FROM TRAILER-1.
      WRITE PRINT-REC FROM TRAILER-2.
*
```

Figure 5. WRITE-TLIMIT-TBALANCE

15.6 Labo

Ce laboratoire nécessite deux programmes COBOL, CBL0008 et CBL0009 et deux travaux JCL respectifs, CBL0008J et CBL0009J, pour compiler et exécuter les programmes COBOL. Tout cela vous est fourni dans votre VSCode - Zowe Explorer.

15.6.0.1 Utilisation de VSCode et Zowe Explorer

1. Prenez un moment et regardez le code source des deux programmes COBOL fournis : CBL0008 et CBL0009.
2. Soumettre CBL0008J
3. Observez le rapport écrit avec des remorques comprenant des totaux de limite et de solde au bas de la sortie.

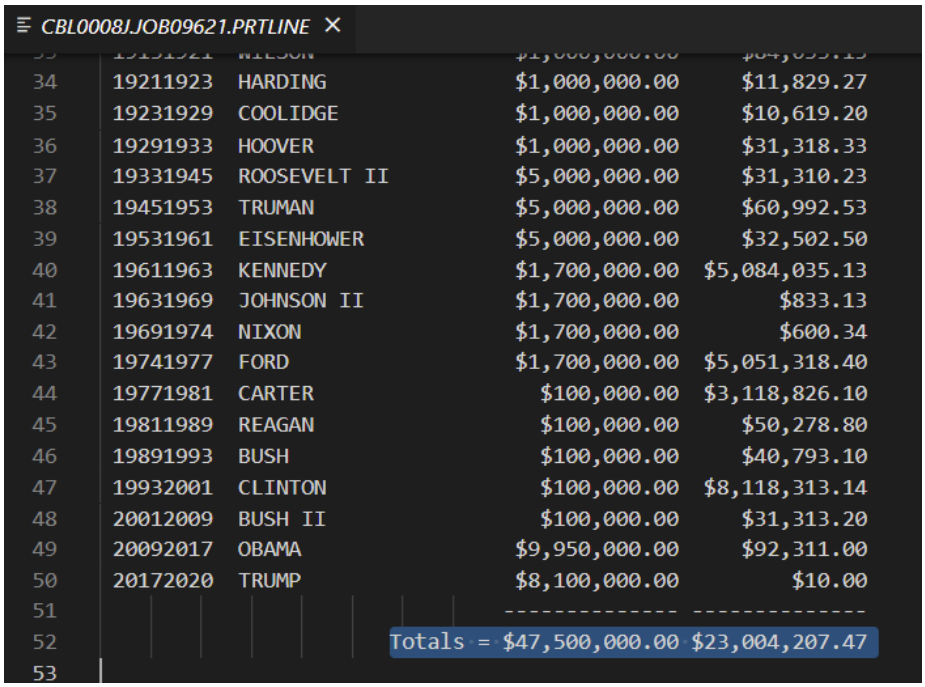


Figure 6. Totaux limites et soldes

4. Soumettre CBL0009J
5. Le travail a-t-il été réussi ? Sinon, recherchez le message d'erreur de compilation pour comprendre pourquoi.
6. Modifiez l'id.CBL(CBL0009) en corrigeant l'erreur de compilation.*

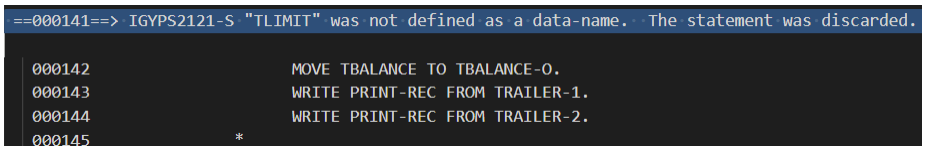


Figure 7. Message d'erreur IGYPS2121-S

7. Re-soumettre CBL0009J
8. Validez que l'erreur de syntaxe a été corrigée en obtenant un fichier de sortie sans erreur comme dans la figure 8. La correction doit rapporter écrite avec des remorques composées de totaux de limite et de solde, comme la figure 6.

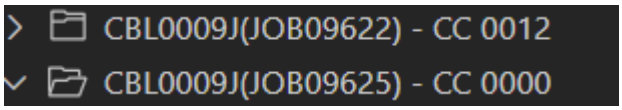


Figure 8. Compilation réussie

Conseils de laboratoire

```
01  TLIMIT-TBALANCE.
    05 TLIMIT              PIC S9(9)V99 COMP-3 VALUE ZERO.
    05 TBALANCE            PIC S9(9)V99 COMP-3 VALUE ZERO.
```

16 Types de données

Un programmeur COBOL doit être conscient que la représentation et le formatage des données internes stockées sur l'ordinateur peuvent différer, la différence devant être définie dans le code source COBOL. Pour comprendre la représentation des données internes de l'ordinateur, il faut se familiariser avec le binaire, l'hexadécimal, l'ASCII et l'EBCDIC. Packed-Decimal est nécessaire pour expliquer le format de données de calcul et d'affichage COBOL. Ce chapitre vise à familiariser le lecteur avec ces différents « types » de représentation des données.

- **Représentation des données**
  - **Représentation de la valeur numérique**
  - **Représentation textuelle**
- **AFFICHAGE COBOL vs CALCUL**
- **Laboratoire**

16.1 Représentation des données

Les données telles que les valeurs numériques et le texte sont représentées en interne par des zéros et des uns dans la plupart des ordinateurs, y compris les ordinateurs centraux utilisés par les entreprises. Bien que la représentation des données soit un sujet quelque peu complexe en informatique, un programmeur n'a pas toujours besoin de bien comprendre le fonctionnement de diverses représentations alternatives. Il est toutefois important de comprendre les différences et de savoir comment spécifier une représentation spécifique si nécessaire.

16.1.1 Représentation de la valeur numérique

COBOL a cinq représentations de valeurs de calcul (numériques). La connaissance de ces représentations est importante pour deux raisons principales. La première raison étant que lorsqu'un programme COBOL doit lire ou écrire des données, il doit comprendre comment les données sont représentées dans l'ensemble de données. La deuxième raison est lorsqu'il existe des exigences spécifiques concernant la précision et la plage des valeurs traitées. Pour plus de détails sur les systèmes de numérotation binaire et hexadécimale ainsi que sur ces représentations numériques, pensez à lire le chapitre « Représentation numérique des données » dans le cours sur les sujets avancés.

16.1.1.1 COMP-1 Ceci est également connu comme une représentation de nombre à virgule flottante simple précision. En raison de la nature à virgule flottante, une valeur COMP-1 peut être très petite et proche de zéro, ou elle peut être très grande (environ 10 à la puissance 38). Cependant, une valeur COMP-1 a une précision limitée. Cela signifie que même si une valeur COMP-1 peut aller jusqu'à 10 à la puissance 38, elle ne peut conserver qu'environ sept chiffres décimaux significatifs. Toute valeur comportant plus de sept chiffres significatifs

est arrondie. Cela signifie qu'une valeur COMP-1 ne peut pas représenter exactement un solde bancaire comme 1 234 567,89 $, car cette valeur comporte neuf chiffres significatifs. Au lieu de cela, le montant est arrondi. L'application principale de COMP-1 est le stockage de valeurs numériques scientifiques ainsi que le calcul.

16.1.1.2 COMP-2 Ceci est également connu sous le nom de représentation de nombres à virgule flottante double précision. COMP-2 étend la plage de valeurs pouvant être représentées par rapport à COMP-1. COMP-2 peut représenter des valeurs jusqu'à environ 10 à la puissance 307. Comme COMP-1, les valeurs COMP-2 ont également une précision limitée. En raison du format étendu, COMP-2 a des chiffres plus significatifs, environ 15 chiffres décimaux. Cela signifie qu'une fois qu'une valeur atteint certains quadrillions (sans décimales), elle ne peut plus être exactement représentée dans COMP-2.

COMP-2 remplace COMP-1 pour un stockage et un calcul plus précis des données scientifiques. Notez que COMP-1 et COMP-2 ont des applications limitées dans la représentation ou le calcul de données financières.

16.1.1.3 COMP-3 Ceci est également connu sous le nom de représentation BCD (décimale codée binaire) condensée. C'est, de loin, la représentation de valeur numérique la plus utilisée dans les programmes COBOL. Le BCD compact est également quelque peu unique et natif des ordinateurs centraux tels que l'architecture IBM z.

Contrairement au COMP-1 ou au COMP-2, le BCD compact n'a pas de limitation de précision inhérente indépendante de la plage de valeurs. En effet, COMP-3 est un format de largeur variable qui dépend du format de valeur réel. COMP-3 représente exactement les valeurs avec décimales. Une valeur COMP-3 peut avoir jusqu'à 31 chiffres décimaux.

16.1.1.4 COMP-4 COMP-4 n'est capable de représenter que des entiers. Comparé à COMP-1 et COMP-2, COMP-4 peut stocker et calculer exactement avec des valeurs entières (à moins qu'une division ne soit impliquée). Bien que COMP-3 puisse également être utilisé pour représenter des valeurs entières, COMP-4 est plus compact.

16.1.1.5 COMP-5 COMP-5 est basé sur COMP-4, mais avec la possibilité de spécifier la position d'un point décimal. COMP-5 a l'efficacité de l'espace de COMP-4, et l'exactitude de COMP-3. Contrairement à COMP-3, cependant, une valeur de COMP-5 ne peut pas dépasser 18 chiffres décimaux.

16.1.2 Représentation de textes

Les programmes COBOL doivent souvent représenter des données textuelles telles que des noms et des adresses.

16.1.2.1 EBCDIC Le code d'échange décimal codé binaire étendu (EBCDIC) est une norme de codage de caractères à huit chiffres binaires, où les huit positions numériques sont divisées en deux parties. EBCDIC a été conçu au début des années 1960 pour les ordinateurs IBM. EBCDIC est utilisé pour coder les données de texte afin que le texte puisse être imprimé ou affiché correctement sur des appareils qui comprennent également EBCDIC.

16.1.2.2 ASCII American Standard Code for Information Interchange, ASCII, est une autre norme de codage de caractères binaires.

16.1.2.3 EBCDIC vs ASCII Pourquoi ces deux normes alors qu'elles remplissent apparemment la même fonction ?

EBCDIC est une norme qui remonte aux cartes perforées conçues en 1931. ASCII, d'autre part, est une norme qui a été créée, sans rapport avec les cartes perforées IBM, en 1967. Un programme COBOL comprend nativement EBCDIC, et il peut traiter confortablement données à l'origine capturées dans des cartes perforées dès 1931.

L'ASCII est principalement utilisé par les ordinateurs non IBM.

COBOL peut encoder et traiter des données de texte en EBCDIC ou ASCII. Cela signifie qu'un programme COBOL peut traiter simultanément les données capturées lors d'un recensement il y a plusieurs décennies tout en exportant les données vers un service cloud utilisant ASCII ou Unicode. Il est important de souligner, cependant, que le programmeur doit avoir la conscience et choisir l'encodage approprié.

16.2 COBOL DISPLAY versus COMPUTATIONAL

Par défaut, Enterprise COBOL for z/OS utilise le codage EBCDIC. Cependant, il est possible de lire et d'écrire en ASCII dans z/OS. La représentation au format EBCDIC des caractères alphabétiques est au format DISPLAY. Les décimales zonées pour les nombres, sans le signe, sont au format DISPLAY. Les nombres décimaux, binaires et flottants compactés ne sont PAS au format DISPLAY. COBOL peut décrire des champs décimaux, binaires et à virgule flottante compactés à l'aide de mots réservés COMPUTATIONAL, COMP-1, COMP-2, COMP-3, COMP-4 et COMP-5.

16.3 Labo

La plupart des programmes de laboratoire COBOL avec lesquels vous avez travaillé jusqu'à présent lisent des enregistrements contenant deux champs décimaux emballés, la limite du compte client et le solde du compte client. Dans le laboratoire d'expressions arithmétiques, le total de toutes les limites et soldes des comptes clients utilisait une instruction COMPUTE, où les champs COMP-3 contenaient les données internes décimales compressées.

Que se passe-t-il lorsqu'un champ décimal condensé interne n'est pas décrit à l'aide de COMP-3 ? Sans utiliser COMP-3 pour décrire le champ, le programme COBOL traite les données comme des données DISPLAY (format EBCDIC). Ce laboratoire montre ce qui se passe pendant l'exécution d'un programme sans utiliser COMP-3.

16.3.0.1 Utilisation de VSCode et Zowe Explorer

1. Soumettez le travail, id.JCL(CBL0010J)
2. Observez que la compilation de la source COBOL a réussi, cependant, observez également que l'exécution du travail a échoué. Comment pouvez-vous dire?

   Il n'y a pas de code CC à côté de CBL0010J(JOB#), à la place il y a un message ABENDU4038. U4038 est une erreur de code utilisateur courante impliquant généralement une incompatibilité entre les données externes et la représentation COBOL des données.
3. Lisez attentivement le message d'exécution SYSOUT. Le message SYSOUT pense à tort que les enregistrements ont une longueur de 174 caractères alors que le programme pense que les enregistrements ont une longueur de 170 caractères.

   Explication : Le nombre décimal compacté (COMP-3) se développe en deux nombres là où un seul nombre existe généralement. Si le programme lit un champ décimal condensé sans décrire le champ comme COMP-3, l'exécution du programme devient confuse quant à la taille de l'enregistrement car la clause PIC, S9(7)V99, s'attend à stocker sept nombres plus un chiffre de signe lorsque seules trois positions de mot sont lues. Par conséquent, l'exécution signale un écart de position de longueur de quatre enregistrements.
4. Modifiez l'id.CBL(CBL0010) pour identifier et corriger le problème de code source.*
5. Soumettez l'id.JCL(CBL0010J) et vérifiez que la correction est réussie avec un code CC 0000.

Conseils pour le labo:

La clause ACCT-LIMIT PIC dans le paragraphe ACCT-FIELDS doit être la même que la clause PIC pour ACCT-BALANCE.

17 Fonctions intrinsèques

Le COBOL d'aujourd'hui n'est pas le COBOL de vos parents. Le COBOL d'aujourd'hui comprend des décennies d'avancées riches en fonctionnalités/fonctions et d'améliorations des performances. Des décennies de spécifications industrielles sont appliquées au COBOL pour répondre aux besoins croissants des entreprises. Ce que Enterprise COBOL for z/OS a promis et livré, ce sont des décennies de compatibilité ascendante avec les nouvelles versions de matériel et de logiciels de système d'exploitation. L'ADN original de COBOL a évolué pour devenir un langage informatique puissant, maintenable, fiable et éprouvé, sans fin en vue.

Parmi les nouvelles fonctionnalités COBOL, citons JSON GENERATE et JSON PARSE, qui fournissent un mécanisme de codage facile à utiliser pour transformer les éléments de données définis par DATA DIVISION en JSON pour un navigateur, un smartphone ou tout autre appareil IoT (Internet des objets) à formater en plus. à transformer le JSON reçu d'un navigateur, d'un téléphone intelligent ou de tout appareil IoT en éléments de données définis par DATA DIVISION pour le traitement. Fréquemment, les données critiques auxquelles accède un téléphone intelligent, comme un solde bancaire, sont stockées et contrôlées par z/OS où un programme COBOL est chargé de récupérer et de renvoyer le solde bancaire au téléphone intelligent. COBOL est devenu un langage informatique compatible avec le Web.

Les précédentes spécifications de l'industrie COBOL incluaient des fonctions intrinsèques, qui restent largement pertinentes aujourd'hui. Un programmeur COBOL expérimenté doit se familiariser avec les fonctions intrinsèques et rester au courant de toute nouvelle fonction intrinsèque introduite. Ce chapitre vise à couvrir le fondement des fonctions intrinsèques et leur utilisation en COBOL.

- **Qu'est-ce qu'une fonction intrinsèque ?**
  - **Syntaxe de fonction intrinsèque**
  - **Catégories de fonctions intrinsèques**
- **Fonctions intrinsèques dans Enterprise COBOL for z/OS V6.3**
  - **Exemple mathématique**
  - **Exemple statistique**
  - **Exemple date/heure**
  - **Exemple financier**
  - **Exemple de gestion de caractères**
- **Utilisation de fonctions intrinsèques avec modification de référence**
- **Labo**

17.1 Qu'est-ce qu'une fonction intrinsèque ?

Les fonctions intrinsèques sont effectivement du code réutilisable avec une implémentation de syntaxe simple et constituent une autre capacité COBOL puissante. Les fonctions intrinsèques permettent le traitement logique souhaité avec une seule ligne de code. Ils offrent également des capacités pour manipuler des chaînes et des nombres. Étant donné que la valeur d'une fonction intrinsèque est dérivée automatiquement au moment de la référence, vous n'avez pas besoin de définir ces fonctions dans la DATA DIVISION.

17.1.1 Syntaxe de fonction intrinsèque

Écrit comme :

FUNCTION nom−fonction (argument)

Où nom-fonction doit être l'un des noms de fonction intrinsèques. Vous pouvez référencer une fonction en spécifiant son nom, ainsi que tous les arguments requis, dans une instruction PROCEDURE DIVISION. Les fonctions sont des éléments de données élémentaires et renvoient des caractères alphanumériques, des caractères nationaux, des valeurs numériques ou entières.

```
01   Item−1    Pic x(30)  Value "Hello World!".
01   Item−2    Pic x(30).
. . .
     Display Item−1
     Display Function Upper−case(Item−1)
     Display Function Lower−case(Item−1)
     Move Function Upper−case(Item−1) to Item−2
     Display Item−2
```

Exemple 1. Utilisation du mot réservé COBOL FUNCTION

Le code illustré dans l'exemple 1 ci-dessus affiche les messages suivants sur le périphérique de sortie logique du système :

Hello World! HELLO WORLD! hello world! HELLO WORLD!

17.1.2 Catégories de fonctions intrinsèques

Les fonctions intrinsèques peuvent être regroupées en six catégories, selon le type de service effectué. Ils sont les suivants :

1. Mathématique
2. Statistique
3. Date/heure
4. Financier
5. Gestion des caractères

6. Générales

Les fonctions intrinsèques opèrent sur des éléments de données alphanumériques, nationaux, numériques et entiers.

- Les fonctions **alphanumériques** sont de classe et de catégorie alphanumérique. La valeur renvoyée a une utilisation implicite de DISPLAY. Le nombre de positions de caractère dans la valeur renvoyée est déterminé par la définition de la fonction.
- Les fonctions **National** sont de classe et catégorie nationale. La valeur renvoyée a une utilisation implicite de NATIONAL et est représentée en caractères nationaux (UTF-16). Le nombre de positions de caractère dans la valeur renvoyée est déterminé par la définition de la fonction.
- Les fonctions **Numériques** sont de classe et de catégorie numérique. La valeur renvoyée est toujours considérée comme ayant un signe opérationnel et est un résultat intermédiaire numérique.
- Les fonctions **Entier** sont de classe et de catégorie numérique. La valeur renvoyée est toujours considérée comme ayant un signe opérationnel et est un résultat intermédiaire entier. Le nombre de positions de chiffres dans la valeur renvoyée est déterminé par la définition de la fonction.

17.2 Fonctions intrinsèques dans Enterprise COBOL for z/OS V6.3

La version actuelle d'Enterprise COBOL pour z/OS V6.3 comprend 70 fonctions intrinsèques. Chacune de ces fonctions tombe dans l'une des six catégories susmentionnées. Alors qu'un livre entier pourrait être écrit sur les fonctions intrinsèques, un seul exemple pour chacune des six catégories est fourni dans cette section.

17.2.1 Exemple mathématique

L'exemple 2. stocke dans X le total de A + B + valeur résultant de C divisé par D. FONCTION SOMME permet l'opération arithmétique.

```
Compute x = Function Sum(a b (c / d))
```

Exemple 2. Fonction intrinsèque mathématique

17.2.2 Exemple statistique

L'exemple 3. montre trois fonctions COBOL, MEAN, MEDIAN et RANGE où les valeurs arithmétiques sont stockées dans Avg-Tax, Median-Tax et Tax-Range en utilisant les noms de données avec les valeurs de clause pic assignées.

```
01   Tax–S          Pic 99v999 value .045.
01   Tax–T          Pic 99v999 value .02.
```

```
01  Tax-W              Pic 99v999 value .035.
01  Tax-B              Pic 99v999 value .03.
01  Ave-Tax            Pic 99v999.
01  Median-Tax         Pic 99v999.
01  Tax-Range          Pic 99v999.
. . .
    Compute Ave-Tax      = Function Mean   (Tax-S Tax-T Tax-W Tax-B)
    Compute Median-Tax   = Function Median (Tax-S Tax-T Tax-W Tax-B)
    Compute Tax-Range    = Function Range  (Tax-S Tax-T Tax-W Tax-B)
```

Exemple 3. Fonction intrinsèque statistique

17.2.3 Exemple de date/heure

L'exemple 4 montre l'utilisation de trois fonctions COBOL, Current-Date, Integer-of-Date et Date-of-Integer appliquées aux instructions MOVE, ADD et COMPUTE.

```
01  YYYYMMDD           Pic 9(8).
01  Integer-Form       Pic S9(9).
. . .
    Move Function Current-Date(1:8) to YYYYMMDD
    Compute Integer-Form = Function Integer-of-Date(YYYYMMDD)
    Add 90 to Integer-Form
    Compute YYYYMMDD = Function Date-of-Integer(Integer-Form)
    Display 'Due Date: ' YYYYMMDD
```

Exemple 4. Fonction intrinsèque date/heure

17.2.4 Exemple financier

L'exemple 5 montre l'application de l'algorithme financier de la fonction COBOL ANNUITY où les valeurs du montant du prêt, des paiements, des intérêts et du nombre de périodes sont entrées dans la fonction ANNUITY.

```
01  Loan                 Pic 9(9)V99.
01  Payment              Pic 9(9)V99.
01  Interest             Pic 9(9)V99.
01  Number-Periods       Pic 99.
. . .
    Compute Loan = 15000
    Compute Interest = .12
    Compute Number-Periods = 36
    Compute Payment = Loan * Function Annuity((Interest / 12) Number-Periods)
```

Exemple 5. Fonction intrinsèque financière

17.2.5 Exemple de gestion de caractères

L'exemple 6. montre l'utilisation de la fonction COBOL MAJUSCULES où une chaîne ou une variable alphabétique traitée par MAJUSCULES traduira tous les caractères minuscules en majuscules.

```
MOVE FUNCTION UPPER-CASE("This is shouting!") TO SOME-FIELD
DISPLAY SOME-FIELD
Output – THIS IS SHOUTING!
```

Exemple 6. Fonction intrinsèque de gestion des caractères

17.3 Utilisation de fonctions intrinsèques avec des modificateurs de référence

Une modification de référence définit un élément de données en spécifiant la position de caractère la plus à gauche et une longueur facultative pour l'élément de données, où deux points (:) sont utilisés pour distinguer la position de caractère la plus à gauche de la longueur facultative, comme illustré dans l'exemple 7.

```
05 LNAME      PIC X(20).

LNAME(1:1)
LNAME(4:2)
```

Exemple 7. Modification de référence

La modification de référence, LNAME (1:1), renverrait uniquement le premier caractère de l'élément de données LNAME, tandis que la modification de référence, LNAME (4:2), renverrait les quatrième et cinquième caractères de LNAME comme résultat du début du quatrième caractère position avec une longueur de deux. Si LNAME de la valeur SMITH était l'élément de données référencé dans la fonction intrinsèque, la première référence produirait, S. Compte tenu de ces mêmes spécifications, la deuxième référence produirait, TH.

17.4 Labo

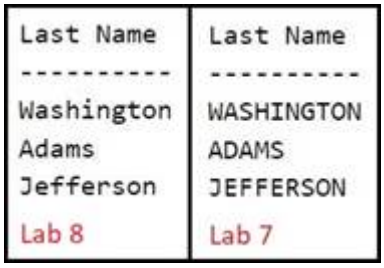
Cet atelier contient des données qui incluent un nom de famille, où le nom de famille est entièrement en majuscules. Il montre l'utilisation de fonctions intrinsèques avec la modification de référence pour mettre en minuscule les caractères du nom de famille, à l'exception du premier caractère du nom de famille.

Ce laboratoire nécessite deux programmes COBOL, CBL0011 et CBL0012 et deux jobs JCL respectifs, CBL0011J et CBL0012J, pour compiler et exécuter les programmes COBOL. Tout cela vous est fourni dans votre VSCode - Zowe Explorer.

17.4.0.1 Utilisation de VSCode et Zowe Explorer

1. Soumettez le travail, CBL0011J.
2. Observez la sortie du rapport, last name, avec le premier caractère en majuscule et les caractères restants en minuscule.

   La figure 1. , ci-dessous, illustre la différence de sortie de l'atelier Types de données par rapport à cet atelier. Notez que dans l'atelier précédent, les noms de famille étaient répertoriés avec tous les caractères en majuscule, alors que, comme indiqué précédemment, cette sortie de laboratoire n'a que le premier caractère du nom de famille en majuscule.



Last Name	Last Name
----------	----------
Washington	WASHINGTON
Adams	ADAMS
Jefferson	JEFFERSON
Lab 8	Lab 7

Figure 1. Labo courant versus types de données de sortie du laboratoire

3. Observer la fonction intrinsèque PROCEDURE DIVISION, en minuscule, dans le paragraphe WRITE-RECORD. Cette fonction intrinsèque est associée à une modification de référence résultant en la sortie du nom de famille avec le premier caractère en majuscule et le reste en minuscule.
4. Soumettre CBL0012J
5. Observez l'erreur de compilation.

Les programmes de laboratoire précédents utilisaient une fonction intrinsèque date/heure. La fonction intrinsèque date/heure de ce laboratoire comporte une erreur de syntaxe qui doit être identifiée et corrigée.

6. Modifiez l'id.CBL(CBL0012) en corrigeant l'erreur de compilation.*
7. Re-soumettre CBL0012J
8. Le code source CBL0012 corrigé doit compiler et exécuter le programme avec succès. Une compilation réussie entraînera la même sortie que CBL0011J.

Conseils de laboratoire

Reportez-vous à CBL0011 ligne 120 pour le formatage correct du nom de fonction provoquant l'erreur de compilation.

18 Traitement ABEND

Lorsque vous effectuez les travaux pratiques sur les chapitres précédents, vous pouvez avoir rencontré une fin anormale ou ABEND pour faire court. Il existe différentes catégories d'erreurs COBOL courantes qui provoquent des ABEND, et en production, les erreurs logicielles peuvent être coûteuses - à la fois en termes financiers et de réputation.

Ce chapitre présente ABEND et donne un aperçu des types ABEND fréquents qu'un programmeur d'application COBOL peut rencontrer. Nous passerons en revue les raisons possibles et les causes fréquentes des types ABEND que le programmeur doit déboguer. Nous passerons également en revue certaines meilleures pratiques courantes pour éviter ABEND et examinerons les raisons pour lesquelles un programmeur peut délibérément appeler une routine ABEND dans son application.

- **Pourquoi ABEND se produit-il ?**
- **Types ABEND fréquents**

last name,

- **Meilleures pratiques pour éviter ABEND**
- **Routines ABEND**

18.1 Pourquoi un ABEND se produit-il ?

Contrairement à votre poste de travail normal, le mainframe utilise une architecture de jeu d'instructions appelée z/Architecture. Ce jeu d'instructions décrit quelles instructions peuvent être exécutées au niveau inférieur du code machine.

Dans le cas où le système rencontre une instruction qui n'est pas autorisée dans le jeu d'instructions, un ABEND se produira. Cela peut se produire pendant la compilation, l'édition de liens ou l'exécution de votre programme COBOL.

18.2 Types d'ABEND fréquents

Vous trouverez ci-dessous neuf des ABEND courants pour vous aider à démarrer. Notez qu'il existe d'autres types et situations ABEND que vous pouvez rencontrer en tant que programmeur COBOL, et z/OS peut parfois produire un code ABEND différent selon que l'ABEND se produit ou non dans une couche du logiciel système.

Ces codes ABEND seraient parfois accompagnés d'un code de raison qui peut être utilisé pour affiner davantage la cause possible des erreurs.

- **S001** - Différence de longueur d'enregistrement/taille de bloc
- **S013** - Paramètres DCB en conflit
- **S0C1** - Instruction invalide

- **S0C4** - Exception de protection de stockage
- **S0C7** - Exception de données
- **S0CB** - Division par zéro
- **S222/S322** - Délai d'attente/Tâche annulée
- **S806** - Module non trouvé
- **B37/D37/E37** - Espace de l'ensemble de données ou de l'index PDS dépassé

Dans les sections suivantes, nous passerons en revue les ABENDs avec toutes les raisons possibles et les causes fréquentes des ABENDs. Notez que les raisons et les causes ne sont pas exhaustives.

18.2.1 S001 - Différence de longueur d'enregistrement/taille de bloc

z/OS gère les données à l'aide d'ensembles de données, c'est-à-dire un fichier contenant un ou plusieurs enregistrements. Ces ensembles de données ont une longueur d'enregistrement prédéterminée et une longueur maximale d'un bloc de stockage (taille de bloc) qui leur est associée lors de leur création. La plupart du temps, l'écart est dû à des erreurs de programmation.

Causes: - S001-0 : Conflit entre la spécification de longueur d'enregistrement (programme vs JCL vs étiquette de jeu de données) - S001-2 : Support de stockage endommagé ou erreur matérielle - S001-3 : Erreur QSAM fatale - S001-4 : Conflit entre spécifications de blocs (programme vs JCL) - S001-5 : Tentative de lecture après la fin du fichier

Causes fréquentes : - S001-0 : fautes de frappe dans l'instruction FD ou JCL - S001-2 : Disque corrompu ou jeu de données sur bande - S001-3 : Problème z/OS interne - S001-4 : Oubli de code BLOCK CONTAINS 0 RECORDS dans l'instruction FD - S001-5 : Erreur logique

18.2.2 S013 - Paramètres DCB en conflit

S013 ABEND se produit lorsque le programme s'attend à ce que l'instruction de définition de données (DD) ait un bloc de contrôle de données (DCB) spécifique, mais que le DD ait un DCB différent. Encore une fois, cela peut être quelque chose comme la taille du bloc, la longueur d'enregistrement ou le format d'enregistrement.

Pour en savoir plus sur les ensembles de données, visitez le centre de connaissances IBM :

https://www.ibm.com/docs/en/zos-basic-skills?topic=more-what-is-data-set

Causes: - S013-10 : l'ensemble de données dummy a besoin d'espace; spécifier BLKSIZE dans JCL - S013-14 : l'instruction DD doit spécifier un PDS - S013-18 : membre PDS introuvable - S013-1C : erreur I/O dans la recherche du répertoire PDS - S013-20 : la taille du bloc n'est pas un multiple de la longueur de l'enregistrement - S013-34 : la longueur de l'enregistrement est incorrecte -

S013-50 : J'ai essayé d'ouvrir une imprimante pour une entrée - S013-60 : la taille du bloc n'est pas égale à la longueur de l'enregistrement pour la taille non bloquée - S013-64 : Tentative de suppression d'un fichier indexé ou relatif - S013-68 : la taille du bloc est supérieure à 32752 - S013-A4 : SYSIN ou SYSOUT n'est pas un fichier QSAM - S013-A8 : Format d'enregistrement invalide pour SYSIN ou SYSOUT - S013-D0 : tentative de définition de PDS avec le format d'enregistrement FBS ou FS - S013-E4 : Tentative de concaténer plus de 16 PDS

Causes fréquentes : La plupart des raisons de ce code ABEND sont dues à des incohérences entre le JCL et le programme COBOL.

18.2.3 S0C1 - Instruction invalide

Dans S0C1, la CPU tente d'exécuter une instruction non valide ou non prise en charge.

Les raisons: - Instruction SYSOUT DD manquante - La valeur dans une clause AFTER ADVANCING est inférieure à 0 ou supérieure à 99 - Un index ou un indice est hors limites - Un verbe I/O a été émis contre un ensemble de données non ouvert - La liaison du sous-programme CALL ne correspond pas à la définition de l'enregistrement du programme appelant

Causes fréquentes : - Logique incorrecte dans le réglage de la clause AFTER ADVANCING - Logique incorrecte dans le code de gestion de la table ou débordement des entrées de la table

18.2.4 S0C4 - Exception de protection de stockage

Lorsque vous exécutez votre programme COBOL sous z/OS, le système d'exploitation alloue un bloc de mémoire virtuelle appelé espace d'adressage. L'espace d'adressage contiendra les adresses mémoire nécessaires à l'exécution du programme.

Raison: Avec S0C4, le programme tente d'accéder à une adresse mémoire qui n'est pas dans l'espace d'adressage alloué.

Causes fréquentes : - Instruction JCL DD manquante ou incorrecte - Logique incorrecte dans le code de gestion des tables - Débordement d'entrées de table - INITIALISER un fichier FD qui n'a pas été ouvert

18.2.5 S0C7 - Exception de données

Comme vous l'avez vu précédemment, le programme COBOL gère les données à l'aide de clauses PICTURE, qui déterminent le type de données de cette variable particulière. Mais parfois, vous pouvez rencontrer des données qui sont égarées.

Raison: Avec S0C7, le programme attend des données numériques, cependant, il a trouvé d'autres types de données invalides. Cela peut se produire lorsque vous essayez de DÉPLACER quelque chose de non numérique d'un champ PIC 9 vers un champ PIC X.

Causes fréquentes : - Variables mal initialisées ou non initialisées - Modifications de données manquantes ou incorrectes - PASSER d'un niveau 01 à un niveau 01 si le champ d'envoi est plus court que le champ de réception - DÉPLACEMENT des zéros vers les champs numériques au niveau du groupe - CORRESPONDANT DE DEPLACEMENT incorrect - Déclarations d'affectation incorrectes lors du déplacement d'un champ à un autre

18.2.6 S0CB - Division par zéro

Tout comme les mathématiques, tenter de diviser un nombre par 0 dans Enterprise COBOL est une opération indéfinie.

Raison: Le processeur a tenté de diviser un nombre par 0.

Causes fréquentes : - Variables mal initialisées ou non initialisées - Modifications de données manquantes ou incorrectes

18.2.7 S222/S322 - Délai d'attente / Travail annulé

Lorsque vous soumettez un JCL, il est possible de déterminer le temps que vous souhaitez allouer à un travail. Si le travail dépasse ce temps alloué, il expirera. Selon la configuration de votre système, un travail qui a pris un certain temps peut être annulé manuellement par l'opérateur ou automatiquement.

Raison: Délai d'attente, probablement dû au fait que la logique du programme est coincée dans une boucle sans sortie possible (boucle infinie). Pour être précis, S322 ABEND fait référence au délai d'attente, tandis que S222 fait référence au travail en cours d'annulation.

Causes fréquentes : - Logique invalide - Logique de fin de fichier invalide - Commutateur de fin de fichier écrasé - Indice pas assez grand - EFFECTUER PAR une mauvaise sortie - EFFECTUER JUSQU'À la fin du fichier sans changer le commutateur EOF

18.2.8 S806 - Module non trouvé

Nous avons vu précédemment qu'il est possible d'appeler un sous-programme en COBOL. Pour permettre au compilateur de savoir quel sous-programme nous voulons appeler, nous devons le spécifier sur le JCL. Si vous ne les indiquez pas, le compilateur tentera de vérifier d'abord les bibliothèques système avant d'échouer.

Raison: CALL fait à un sous-programme qui n'a pas pu être localisé.

Causes fréquentes : - Module supprimé de la bibliothèque - Nom du module mal orthographié - La bibliothèque de chargement avec le module n'est pas précisée sur le JCL - Erreur d'E/S lorsque z/OS a recherché dans le répertoire de la bibliothèque

18.2.9 B37/D37/E37 - Espace de l'ensemble de données ou de l'index PDS dépassé

Nous avons vu que les ensembles de données dans z/OS ont une taille qui leur est allouée. Lorsque nous créons de nombreuses données, à un moment donné, l'ensemble de données n'aura pas assez d'espace pour stocker quoi que ce soit de nouveau.

Codes de motif : - B37 - Volume de disque insuffisant - D37 - Espace primaire dépassé, aucune étendue secondaire définie - E37 - Extensions primaire et secondaire pleines - E37-04 - La table des matières du volume du disque est pleine

Causes fréquentes : - Pas assez d'espace pour allouer le(s) fichier(s) de sortie - Erreur de logique entraînant une boucle d'écriture infinie

18.3 Meilleures pratiques pour éviter les ABEND

Pour éviter les ABEND, nous pouvons faire quelque chose appelé programmation défensive. C'est une forme de programmation dans laquelle nous concevons notre code de manière défensive pour nous assurer qu'il fonctionne toujours dans des circonstances imprévues.

En faisant de la programmation défensive, nous pouvons réduire le nombre de bogues et rendre le programme plus prévisible quelles que soient les entrées.

Vous trouverez ci-dessous certaines choses que nous pouvons faire en COBOL :

- **INITIALISER les champs au début d'une routine.** Cela garantira que le champ a les données appropriées au début du programme. Cependant, des précautions particulières doivent être prises pour s'assurer que tous les indicateurs ou accumulateurs ont les données INITIALIZE appropriées.
- **Vérification des instructions d'E/S.** Cela peut se faire en utilisant la variable FILE STATUS et en les vérifiant avant d'effectuer toute autre opération d'E/S. De plus, nous devons vérifier les fichiers vides et d'autres exceptions possibles.
- **Vérification des champs numériques.** Une règle générale serait de ne pas faire confiance à un champ numérique sur lequel nous faisons des calculs. Supposons que l'entrée peut être invalide. Il serait recommandé d'utiliser les phrases ON OVERFLOW et ON SIZE ERROR pour détecter les données invalides ou anormales. Des précautions particulières doivent être prises lorsque nous devons effectuer des arrondis, car une troncature peut se produire dans certains cas.
- **Formatage du code.** Cela garantira que votre code est maintenable et facile à comprendre par quiconque le lit ou le maintient.
- **Utilisation cohérente des terminateurs de portée.** Il serait préférable de terminer explicitement une portée à l'aide de terminateurs de portée

tels que END-IF, END-COMPUTE ou END-PERFORM.

- **Test, vérification et évaluation par les pairs.** Des tests appropriés et une évaluation par les pairs peuvent être effectués pour détecter d'éventuelles erreurs qui ont pu se glisser dans votre programme. De plus, nous pouvons également nous assurer que la logique métier est correcte.

18.4 Routines ABEND

Même lorsqu'un système ABEND ne se produit pas, il existe des situations possibles dans lesquelles vous devrez appeler une routine ABEND. Cela peut être lorsque vous rencontrez des données d'entrée non valides pour votre programme ou une erreur renvoyée par un sous-programme.

Habituellement, ces routines seraient fournies par votre lieu de travail. Mais cela peut être aussi simple que l'exemple suivant :

```
IF abend-condition
    PERFORM ABEND-ROUTINE.
...
ABEND-ROUTINE.
    DISPLAY "Invalid data".
    STOP RUN.
```

Une telle routine peut afficher plus d'informations qui vous permettront de déterminer où et pourquoi exactement le programme a échoué.