IBM COBOL for Linux on x86 1.1

プログラミング・ガイド

SC43-5390-00

注記

本書および本書で紹介する製品をご使用になる前に、641 ページの『特記事項』に記載されている情報をお読みください。

本書は、IBM® COBOL for Linux® on x86 バージョン 1.1 (プログラム番号 5737-L11)、および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。製品のレベルに合った正しい版をご使用ください。

ソフトコピー資料は COBOL for Linux on x86 ライブラリーにおいて無償で表示またはダウンロードできます。

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原典：
SC28-3118-00
IBM COBOL for Linux on x86 1.1
Programming Guide
First edition

発行：
日本アイ・ビー・エム株式会社

担当：
トランスレーション・サービス・センター

© Copyright International Business Machines Corporation 2021.

目次

表

前書き

本書について

IBM COBOL for Linux on x86、IBM の Linux on x86 用 COBOL コンパイラーおよびランタイムをご利用いただき、ありがとうございます。

本書では、Linux on x86 用の IBM COBOL コンパイラーとランタイム環境 (本書では COBOL for Linux と呼びます) の使用法について説明します。

ホスト用の COBOL とワークステーション用の COBOL には多少の違いがあります。 COBOL for Linux と Enterprise COBOL for z/OS® の 言語およびシステムの違いについて詳しくは、521 ページの『付録 A IBM Enterprise COBOL for z/OS との違いのまとめ』を参照してください。.

本書の使い方

本書は、IBM COBOL for Linux on x86 プログラムの作成、コンパイル、リンク・エディット、および実行に役立ちます。

本書は、アプリケーション・プログラムの開発の経験と、COBOL に関する多少の知識を前提としています。 本書では、COBOL 言語の定義ではなく、プログラミングの目的に合った COBOL の使用に重点を置いています。 COBOL 構文の詳細については、「*COBOL for Linux on x86* 言語解説書」を参照してください。

また、本書は Linux の知識がある方を対象としています。Linux について詳しくは、お使いのオペレーティング・システムのマニュアルを参照してください。

略語

本書では、短縮形で使用される用語があります。 最も頻繁に使用される製品名の省略形を以下にリストします。

使用される用語	長形式
CICS® TXSeries® または TXSeries	IBM TXSeries for Multiplatforms
CICS TX	IBM CICS TX on Cloud
CICS	IBM TXSeries for Multiplatforms または IBM CICS TX on Cloud
COBOL for Linux	IBM COBOL for Linux on x86
Db2®	IBM Db2 for Linux、UNIX および Windows

これらの略語のほかに、本書では「85 COBOL 標準」という用語が使用されています。この用語は、以下の規格の組み合わせを示します。

- ISO 1989:1985、プログラム言語 - COBOL
- ISO/IEC 1989/AMD1:1992、プログラム言語 - COBOL - 組み込み関数モジュール
- ISO/IEC 1989/AMD2:1994、プログラム言語 - COBOL 用修正および説明改訂
- ANSI INCITS 23-1985、プログラム言語 - COBOL
- ANSI INCITS 23a-1989、プログラム言語 - COBOL 用組み込み関数モジュール
- ANSI INCITS 23b-1993、プログラム言語 - COBOL 用修正と改訂

その他の用語 (一般に理解されていない場合) は、初出時にイタリック体 で示され、用語集にリストされています。

© Copyright IBM Corp. 2021

構文図の読み方

本書中の構文図を読むには、以下の説明を参照してください。

- 構文図は、左から右、上から下へと線をたどって読んでください。

  >>--- 記号は、構文図の開始を示します。

  ---> 記号は、構文図が次の行に続くことを示します。

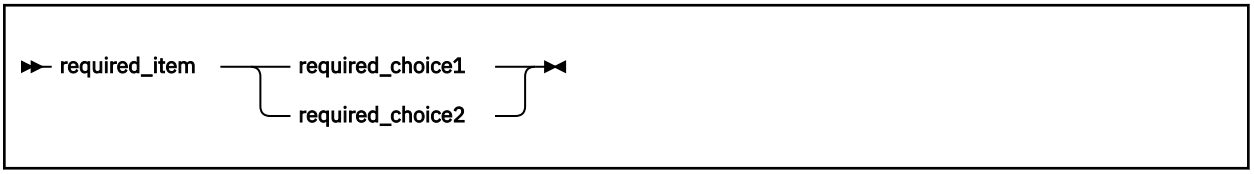
  >--- 記号は、構文図が前の行からの続きであることを示します。

  --->< 記号は、構文図の終わりを示します。

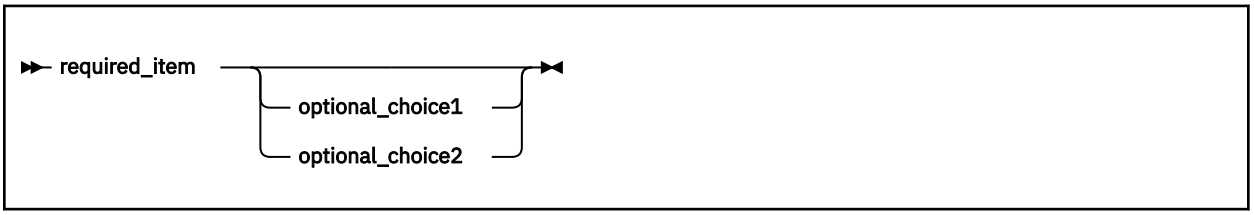
  完全なステートメントではない構文単位の図は、**>---** 記号で始まり、**--->** 記号で終わります。
- 必須項目は、横線 (幹線) 上に示されています。

- 任意指定項目は、幹線の下に示されています。

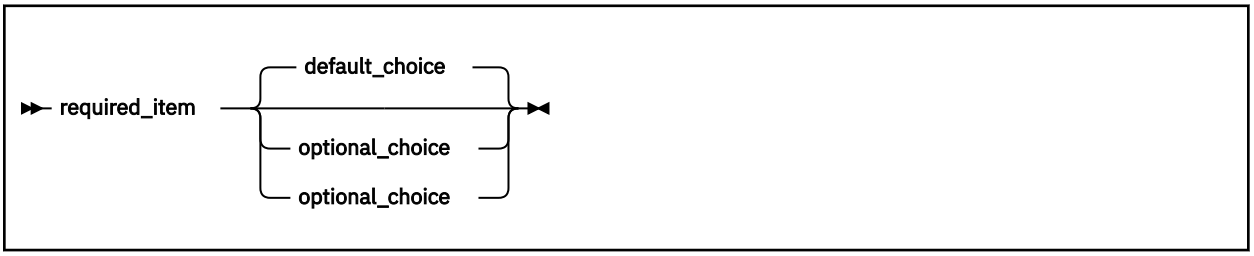
- 複数の項目から選択できる場合には、それらの項目は縦方向に重ねて示されます。 それらの項目のうち 1 つを選択しなければならない場合には、それらの項目のうちの 1 つが幹線上に示されます。



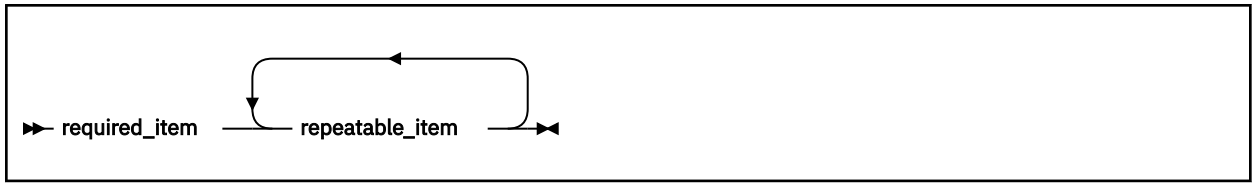
  いずれか 1 つの項目の選択が任意である場合は、重ねられた項目全体が主経路の下に示されます。



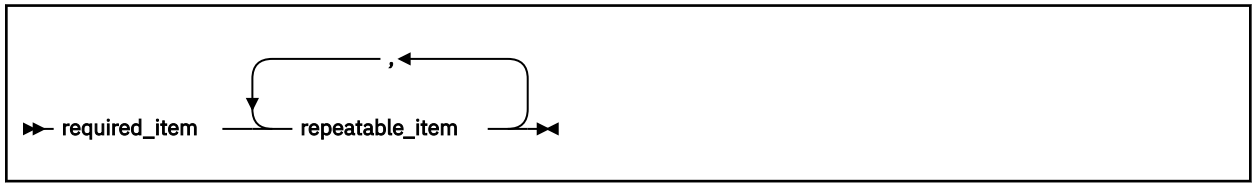
  項目のうちの 1 つがデフォルトであれば、それが幹線より上に示され、残りの選択項目は幹線より下に示されます。



- 幹線の上を左に戻る矢印は、その項目を繰り返して指定できることを示しています。



反復矢印にコンマが含まれている場合は、繰り返す項目をコンマで区切って指定する必要があります。



- キーワードは大文字で示されています (例えば、FROM)。 これは表記どおり正確につづらなければなりません。 変数は日本語または小文字で示されます (例えば *column-name*)。 それらの変数は、ユーザーが指定する名前や値を表します。
- 句読記号、括弧、算術演算子などの記号が示されている場合には、これらも構文の一部として入力しなければなりません。

例の示し方

本書では、コーディング技法を説明するために、サンプル COBOL ステートメント、プログラムの一部分、および短いプログラムの例が多く示されています。 プログラム・コードの例は、小文字、大文字、または大/小文字混合で書かれており、これらのいずれでもプログラムを作成できることを示しています。

例を説明テキストと明確に区別する場合は、例はモノスペース・フォントで示されます。

テキスト内の COBOL キーワードとコンパイラー・オプションは、通常は、SMALL UPPERCASE (小さい英大文字フォント) で示されます。 プログラム変数名などのようなその他の用語は、明確にするために、イタリック・フォント で示される場合があります。

関連情報

このプログラミング・ガイドに記載されている情報は、 オンラインの IBM COBOL for Linux 資料 (http://www.ibm.com/support/knowledgecenter/SS7FZ2_1.1.0) でもご覧いただけます。IBM Documentation Web サイトにも *COBOL for Linux on x86* 言語解説書 があります。

アクセシビリティー

アクセシビリティー機能は、運動障害または視覚障害などの障害を持つユーザーが情報技術製品を快適に使用できるように サポートします。

アクセシビリティー機能

IBM COBOL for Linux on x86 では、US Section 508 および Web Content Accessibility Guidelines (WCAG) 2.0 に確実に準拠するために、最新の W3C 標準、WAI-ARIA 1.0 が使用されています。アクセシビリティー機能を利用するには、最新リリースのスクリーン・リーダーを、この製品でサポートされる最新の Web ブラウザーと併用してください。

キーボード・ナビゲーション

この製品では、標準的なナビゲーション・キーを使用します。

インターフェース情報

Text-to-speech (TTS) ツールのような音声認識ソフトウェアを使用して、製品によって生成された出力を表示できます。

オンライン製品資料は、IBM の資料で入手でき、標準の Web ブラウザーで表示できます。

PDF ファイルでのアクセシビリティー・サポートは限定的です。 PDF 資料では、オプションのフォント拡大機能およびハイコントラスト表示設定を使用でき、キーボードのみでナビゲートできます。

スクリーン・リーダーを使用して、構文図、ソース・コード例、およびピリオドやコンマなどの PICTURE の記号を含むテキストを正確に読むには、句読点をすべて読むようにスクリーン・リーダーを設定する必要があります。

関連アクセシビリティー情報

標準の IBM ヘルプ・デスクとサポート Web サイトに加え、IBM は、聴覚が不自由なお客様が営業やサポート・サービスにアクセスするために使用できる TTY 電話サービスを立ち上げました。

TTY service 800-IBM-3383 (800-426-3383) (北アメリカ内)

IBM およびアクセシビリティー

IBM のアクセシビリティーに対する取り組みについて詳しくは、IBM アクセシビリティーを参照してください。

第 1 部 プログラムのコーディング

© Copyright IBM Corp. 2021

第 1 章 プログラムの構造

COBOL プログラムは、4 つの DIVISION (IDENTIFICATION DIVISION、ENVIRONMENT DIVISION、DATA DIVISION、および PROCEDURE DIVISION) から成ります。 それぞれの DIVISION には、固有の論理関数があります。

プログラムを定義するには、IDENTIFICATION DIVISION のみが必要です。

関連タスク

3 ページの『プログラムの識別』
5 ページの『コンピューター環境の記述』
9 ページの『データの記述』
13 ページの『データの処理』

プログラムの識別

IDENTIFICATION DIVISION は、プログラムの名前を指定し、また必要が あればその他の識別情報を与えるために使用されます。

オプションの AUTHOR、INSTALLATION、DATE-WRITTEN、および DATE-COMPILED 段落を使用して、プログラムに関する記述情報を指定することができます。 DATE-COMPILED 段落に入力したデータは、最新のコンパイル日付で置き換えられます。

```
IDENTIFICATION DIVISION.
Program-ID.    Helloprog.
Author.        A. Programmer.
Installation.  Computing Laboratories.
Date-Written.  06/30/2020.
Date-Compiled. 07/05/2020.
```

PROGRAM-ID 段落を使用して、プログラムの名前を指定します。 割り当てるプログラム名は、以下のように使用されます。

- プログラムを呼び出すために他のプログラムがその名前を使用します。
- プログラムのコンパイル時に生成されるプログラム・リストの各ページ (最初のページを除く) のヘッダーにその名前が入れられます。

ヒント: プログラム名に大/小文字の区別がある場合は、コンパイラーの探索対象である名前とのミスマッチが起こらないようにしてください。 PGMNAME コンパイラー・オプションでの該当する設定が有効であるか検査してください。

関連タスク

4 ページの『ソース・リストのヘッダーの変更』
4 ページの『プログラムを再帰的として識別する』
4 ページの『収容プログラムによってプログラムに呼び出し可能のマークを付ける』
4 ページの『プログラムを初期状態に設定する』

関連参照

コンパイラー限界値 (*COBOL for Linux on x86* 言語解説書)
プログラム名の規則 (*COBOL for Linux on x86* 言語解説書)

© Copyright IBM Corp. 2021

プログラムを再帰的として識別する

前の呼び出しがまだアクティブである間にプログラムに再帰的に再入できるようにするには、PROGRAM-ID 節に RECURSIVE 属性をコーディングしてください。

RECURSIVE は、コンパイル単位の最外部のプログラムにのみコーディングすることができます。 ネストされたサブプログラムも、 ネストされたサブプログラムを含むプログラムも、再帰的にすることはできません。

関連タスク
13 ページの『再帰的プログラムでのデータの共用』
444 ページの『再帰呼び出しの実行』

収容プログラムによってプログラムに呼び出し可能のマークを付ける

収容プログラムまたは収容プログラム内の任意のプログラムによってプログラムを呼び出せることを指定するには、PROGRAM-ID 段落で COMMON 属性を使用してください。 ただし、COMMON プログラムは、それ自体に含まれているプログラムによって呼び出すことはできません。

含まれているプログラムだけが COMMON 属性を持つことができます。

関連概念
433 ページの『ネストされたプログラム』

プログラムを初期状態に設定する

プログラムを呼び出すたびにプログラムおよびそのプログラムに含まれるネストされたプログラムを初期状態にすることを指定するには、PROGRAM-ID 段落の INITIAL 節を使用します。

プログラムが初期状態に設定されると、以下のようになります。

- VALUE 節を持つデータ項目が、指定された値に設定された。
- 変更された GO TO ステートメントおよび PERFORM ステートメントが、それぞれ初期状態になった。
- 非 EXTERNAL ファイルがクローズされた。

関連タスク
431 ページの『メインプログラムまたはサブプログラムの終了と再入』

関連参照
289 ページの『WSCLEAR』

ソース・リストのヘッダーの変更

ソース・リストの最初のページのヘッダーには、コンパイラーおよび現行リリース・レベルの識別、コンパイルの日時、およびページ番号が入っています。

以下の例はこれら 5 つのエレメントを示しています。

```
PP  5737-L11 IBM COBOL for Linux 1.1.0     Date 05/29/2020   Time 17:38:17   Page     1
```

ヘッダーは、コンパイル・プラットフォームを示します。 コンパイラー指示 TITLE ステートメントを使用すれば、リストの後続のページのヘッダーをカスタマイズすることができます。

関連参照
TITLE ステートメント (*COBOL for Linux on x86* 言語解説書)

コンピューター環境の記述

プログラムの ENVIRONMENT DIVISION では、コンピューター環境に依存するプログラムの局面について記述します。

CONFIGURATION SECTION を使用して、次の項目を指定します。

- プログラムをコンパイルするコンピューター (SOURCE-COMPUTER 段落)
- プログラムを実行するコンピューター (OBJECT-COMPUTER 段落)
- 通貨記号やシンボリック文字などの特殊な項目 (SPECIAL-NAMES 段落)
- ユーザー定義のクラス (REPOSITORY 段落)

INPUT-OUTPUT SECTION の FILE-CONTROL および I-O-CONTROL 段落は、以下の目的に使用します。

- プログラム内のファイルの特性を識別および記述する。
- ファイルを対応するシステム・ファイル名と直接的または間接的に関連付けます。
- オプションとして、ファウルに関連したファイル・システムを指定します (例: SFS または STL)。 これは、実行時にも行えます。
- ファイルのアクセス方法に関する情報を提供します。

5 ページの『例: FILE-CONTROL 段落』

関連タスク

6 ページの『照合シーケンスの指定』
7 ページの『シンボリック文字を定義する』
8 ページの『ユーザー定義のクラスを定義する』
8 ページの『オペレーティング・システムに対するファイルの識別 (ASSIGN)』

関連参照

セクションおよび段落 (*COBOL for Linux on x86* 言語解説書)

例: FILE-CONTROL 段落

次の例は、FILE-CONTROL 段落を使用して、COBOL プログラム内の各ファイルと、ファイル・システムに既知の物理ファイルを関連付ける方法を示しています。 この例は、索引付きファイル用の FILE-CONTROL 段落を示しています。

```
SELECT COMMUTER-FILE (1)
  ASSIGN TO COMMUTER (2)
  ORGANIZATION IS INDEXED (3)
  ACCESS IS RANDOM (4)
  RECORD KEY IS COMMUTER-KEY (5)
  FILE STATUS IS (5)
    COMMUTER-FILE-STATUS
    COMMUTER-STL-STATUS.
```

(1)
SELECT 文節は、COBOL プログラム内のファイルを対応するシステム・ファイルと関連付けます。

(2)
ASSIGN 文節は、プログラム内のファイル名を、システムに既知のファイル名と関連付けます。 COMMUTER は、システム・ファイル名または環境変数名を表す場合があります。 この実行時値は、オプションのディレクトリーおよびパス名とともに、システム・ファイル名として使用されます。

(3)
ORGANIZATION 節は、ファイルの編成を記述します。 この文節を省略した場合は、ORGANIZATION IS SEQUENTIAL が定義されたものと見なされます。

(4)
ACCESS MODE 文節は、ファイル内のレコードを処理できるようにする方式 (順次、ランダム、 または動的) を定義します。 この文節を省略した場合は、ACCESS IS SEQUENTIAL が定義されたものと見なされます。

(5)
使用するファイルおよびファイル・システムのタイプによって、FILE-CONTROL 段落に追加のステートメントを指定することができます。

関連タスク
5 ページの『コンピューター環境の記述』

照合シーケンスの指定

PROGRAM COLLATING SEQUENCE 節、および SPECIAL-NAMES 段落の ALPHABET 節を使用すれば、英数字項目に対するいくつかの操作で使用される照合シーケンスを設定できます。

これらの節では、英数字項目に対する以下の操作の照合シーケンスを指定します。

- 比較条件および条件名条件で明示的に指定された比較
- HIGH-VALUE および LOW-VALUE の設定
- SEARCH ALL
- SORT および MERGE (SORT または MERGE ステートメントの COLLATING SEQUENCE 句でオーバーライドされていない場合)

7 ページの『例: 照合シーケンスの指定』

以下のいずれかのアルファベットを基に、使用するシーケンスを選択できます。

- EBCDIC: EBCDIC 文字セットに関連付けられた照合シーケンスを参照します。
- NATIVE: ロケールの設定で指定された照合シーケンスを参照します。 ロケール設定は、コンパイル時に有効化された各国語ロケール名を参照します。 通常、インストール時に設定されます。
- STANDARD-1: *ANSI INCITS X3.4, Coded Character Sets - 7-bit American National Standard Code for Information Interchange (7-bit ASCII)* により定義された ASCII 文字セットに関連付けられた照合シーケンスを参照します。
- STANDARD-2: *ISO/IEC 646 -- Information technology -- ISO 7-bit coded character set for information interchange, International Reference Version* により定義されたコード化文字セットに関連付けられた照合シーケンスを参照します。
- SPECIAL-NAMES 段落で定義された ASCII シーケンスの代替。

自分で定義した照合シーケンスを指定することもできます。

制約事項 : コード・ページが DBCS、拡張 UNIX コード (EUC)、または UTF-8 の場合、ALPHABET 節を使用することはできません。

PROGRAM COLLATING SEQUENCE 節は国別または DBCS オペランドを含む比較に影響を及ぼしません。

関連タスク
159 ページの『代替照合シーケンスの選択』
194 ページの『国別 (UTF-16) データの比較』
201 ページの『第 11 章 ロケールの設定』
207 ページの『ロケール付きの照合シーケンスの制御』

例: 照合シーケンスの指定

次の例は、比較およびソート/マージの場合に大文字と小文字が同様に処理される照合シーケンスを指定する際に使用できる ENVIRONMENT DIVISION コーディングを示しています。

SPECIAL-NAMES 段落の ASCII シーケンスを変更すると、SPECIAL-NAMES 段落に含まれている文字の照合シーケンスだけでなく、全体の照合シーケンスが影響を受けます。

```
IDENTIFICATION DIVISION.
. . .
ENVIRONMENT DIVISION.
  CONFIGURATION SECTION.
   Object-Computer.
      Program Collating Sequence Special-Sequence.
   Special-Names.
      Alphabet Special-Sequence Is
          "A" Also "a"
          "B" Also "b"
          "C" Also "c"
          "D" Also "d"
          "E" Also "e"
          "F" Also "f"
          "G" Also "g"
          "H" Also "h"
          "I" Also "i"
          "J" Also "j"
          "K" Also "k"
          "L" Also "l"
          "M" Also "m"
          "N" Also "n"
          "O" Also "o"
          "P" Also "p"
          "Q" Also "q"
          "R" Also "r"
          "S" Also "s"
          "T" Also "t"
          "U" Also "u"
          "V" Also "v"
          "W" Also "w"
          "X" Also "x"
          "Y" Also "y"
          "Z" Also "z".
```

関連タスク

6 ページの『照合シーケンスの指定』

シンボリック文字を定義する

SYMBOLIC CHARACTERS 節を使用すると、指定したアルファベットの任意の文字に記号名を与えることができます。 序数位置を用いて文字を識別してください。位置 1 は文字 X'00' に対応します。

例えば、プラス文字 (ASCII アルファベットで、X'2B') に名前を与えるには、次のようにコーディングします。

```
SYMBOLIC CHARACTERS PLUS IS 44
```

ロケールによって指定されたコード・ページがマルチバイト文字のコード・ページである場合には、SYMBOLIC CHARACTERS 節を使用することはできません。

関連タスク

201 ページの『第 11 章 ロケールの設定』

ユーザー定義のクラスを定義する

CLASS 節は、節内にリストした文字のセットに名前を与えるために使用します。

例えば、次の節をコーディングして、数字のセットに名前を与えます。

```
CLASS DIGIT IS "0" THROUGH "9"
```

クラス名は、クラス条件でのみ参照できます。 (このユーザー定義クラスは、オブジェクト指向クラスと同じ概念ではありません。)

ロケールによって指定されたコード・ページがマルチバイト文字のコード・ページである場合には、CLASS 節を使用することはできません。

オペレーティング・システムに対するファイルの識別 (ASSIGN)

ASSIGN 文節は、プログラム内で既知のファイルの名前を、オペレーティング・システムで使用される関連ファイルに関連付けます。

ASSIGN 文節には、環境変数、システム・ファイル名、リテラル、データ名を使用することができます。 環境変数を割り当て名として指定した場合は、実行時に環境変数が評価され、その値が (オプションのディレクトリー およびパス名を含む) システム・ファイル名として使用されます。

デフォルト以外のファイル・システムを使用する場合は、システム・ファイル名の前にファイル・システム ID を 指定するなどの方法で、ファイル・システムを明示的に示す必要があります。 例えば、MYFILE が STL ファイルで、プログラム内でファイル名として F1 を使用する場合、ASSIGN 文節は次のようにコーディングできます。

```
SELECT F1 ASSIGN TO STL-MYFILE
```

MYFILE が環境変数ではない場合、または空ストリングが設定されている環境変数である場合、前述のコードは MYFILE をシステム・ファイル名として扱います。 MYFILE が、実行時に空ストリング以外の値を持つ環境変数である場合は、環境変数の値が使用されます。

例えば、MYFILE がコマンド `export MYFILE=RSD-YOURFILE` で設定されている場合、システム・ファイル名は YOURFILE となり、ファイルは RSD ファイルとして扱われ、ASSIGN 節にコーディングされたファイル・システム ID (STL) はオーバーライドされます。

割り当て名を引用符または一重引用符で囲んだ場合 (例: "STL-MYFILE")、環境変数に値が指定されていても無視されます。 リテラル割り当て名が使用されます。

関連タスク
8 ページの『実行時の入出力ファイルの変更』
113 ページの『ファイルの識別』

関連参照
116 ページの『ファイル・システム決定の優先順位』
300 ページの『FILESYS』
ASSIGN 節 (*COBOL for Linux on x86* 言語解説書)

実行時の入出力ファイルの変更

SELECT 節でコーディングした *file-name* は、COBOL プログラム全体にわたって定数として使用されますが、そのファイルの名前を実行時に異なるシステム・ファイルを関連付けることができます。

COBOL プログラム内の file-name を変更するには、入力ステートメントと出力ステートメントを変更し、プログラムを再コンパイルしなければなりません。 または、DD ステートメントの `export` の *assignment-name* コマンドを変更して、 実行時に異なるファイルを使用できます。

OPEN ステートメントの実行時に有効な 環境変数値が、COBOL ファイル名とシステム・ファイル名の関連付け (任意のパス指定を含む) に使用されます。

例: さまざまな入力ファイルの使用

この例では、プログラムの実行前に環境変数を設定することにより、同じ COBOL プログラムを使用してさまざまなファイルに アクセスできる場合を示します。

次の SELECT 節を含む COBOL プログラムを考えてみます。

```
SELECT MAINFILE ASSIGN TO MAINA
```

プログラムは、そのプログラム内の MAINFILE というファイルを 使用して、checking または savings ファイルにアクセスするものとします。 これを行うには、プログラムの実行前に次のうち該当する方のステートメントを使用して、MAIN 環境変数を設定します。なお、次のステートメントでは、checking ファイルと savings ファイルが /accounts ディレクトリーにあることを前提としています。

```
export MAINA=/accounts/checking
export MAINA=/accounts/savings
```

したがって、ソースを変更または再コンパイルしなくても、同じプログラムを使用して、プログラム内の MAINFILE というファイルとして checking または savings ファイルのどちらにでもアクセスすることができます。

データの記述

データの特性を定義し、データ定義をグループ化して、DATA DIVISION の 1 つ以上のセクションに入れなければなりません。

これらのセクションは、以下のタイプのデータを定義するときに使用できます。

- 入出力操作で使用するデータ: FILE SECTION
- 内部処理用に作成するデータ。
  - ストレージを静的に割り振り、実行単位 の存続期間中存在させる場合: WORKING-STORAGE SECTION
  - プログラムに入るたびにストレージを割り振り、プログラムからの戻り時に割り振りを解除させる場合: LOCAL-STORAGE SECTION
- 別のプログラムからのデータ: LINKAGE SECTION

COBOL for Linux コンパイラーでは、DATA DIVISION エレメントの最大サイズの制限があります。 詳細については、下記のコンパイラー限界値に関する関連参照を参照してください。

関連概念
11 ページの『WORKING-STORAGE と LOCAL-STORAGE の比較』

関連タスク
9 ページの『入出力操作でのデータの使用』
12 ページの『別のプログラムからのデータの使用』

関連参照
コンパイラー限界値 (*COBOL for Linux on x86* 言語解説書)

入出力操作でのデータの使用

入出力操作で使用するデータは、FILE SECTION で定義します。

データに関する以下の情報を提供してください。

- プログラムが使用する入出力ファイルの名前を指定します。 FD 記入項目を使用して、PROCEDURE DIVISION の入出力ステートメントで参照できるファイルに名前を与えます。

  FILE SECTION 内で定義されたデータ項目は、ファイルが正常にオープンされるまでは PROCEDURE DIVISION のステートメントにとって使用可能ではありません。

- FD 記入項目の後のレコード記述で、ファイル内のレコードおよびフィールドを記述します。 レコード名は、WRITE および REWRITE ステートメントの対象となります。

同じ実行単位内のプログラムは、同じ COBOL ファイル名を参照できます。

別々にコンパイルされたプログラムには EXTERNAL 節を使用することができます。 EXTERNAL として定義されたファイルは、実行単位の中でそのファイルを記述する任意のプログラムによって参照することができます。

ネストされた構造、すなわち含まれている構造の中のプログラムには、GLOBAL 節を使用できます。 あるプログラムが別のプログラムを (直接または間接的に) 含む場合には、どちらのプログラムも GLOBAL ファイル名を参照することによって共通のファイルにアクセスすることができます。

COBOL ソース・プログラムで外部またはグローバル・ファイル定義を使用せずに、物理ファイルを共有できます。 例えば、アプリケーションに 2 つの COBOL ファイル名があるが、それらの COBOL ファイルが 1 つのシステム・ファイルに関連付けられるように指定できます。

```
SELECT F1 ASSIGN TO MYFILE.
SELECT F2 ASSIGN TO MYFILE.
```

関連概念
433 ページの『ネストされたプログラム』

関連タスク
456 ページの『プログラム間でのファイルの共用 (外部ファイル)』

関連参照

10 ページの『FILE SECTION 記入項目』

FILE SECTION 記入項目

FILE SECTION で使用できる項目の要約を以下の表に示します。

表 *1.* ***FILE SECTION*** **記入項目**

節	**定義対象**
FD	PROCEDURE DIVISION の入出力ステートメント (OPEN、CLOSE、READ、START、および DELETE) 内で参照される *file-name*。 SELECT 節内の *file-name* と一致しなければなりません。 *file-name* は、*assignment-name* を通じてシステム・ファイルに関連付けられます。
RECORD CONTAINS *n*	論理レコード (固定長) のサイズ。 整数サイズは、レコード内のデータ項目の USAGE とは無関係に、レコードのバイト数を示します。
RECORD IS VARYING	論理レコード (可変長) のサイズ。 整数サイズが指定された場合は、レコード内のデータ項目の USAGE とは無関係に、レコードのバイト数を示します。
RECORD CONTAINS *n*TO *m*	論理レコード (可変長) のサイズ。 整数サイズは、レコード内のデータ項目の USAGE とは無関係に、レコードのバイト数を示します。
VALUE OF	ファイルに関連付けられているラベル・レコードの項目。 コメントのみ。
DATA RECORDS	ファイルに関連付けられているレコードの名前。 コメントのみ。
RECORDING MODE	順次ファイル用 レコード・タイプ

関連参照

FILE SECTION (*COBOL for Linux on x86* 言語解説書)

WORKING-STORAGE と LOCAL-STORAGE の比較

データ項目の割り振りと初期化がどのように行われるかは、それらの項目が WORKING-STORAGE SECTION の項目であるか LOCAL-STORAGE SECTION の項目であるかによって異なります。

プログラムの呼び出し時には、そのプログラムに関連付けられた WORKING-STORAGE が割り振られます。

VALUE 節を持つすべてのデータ項目は、そのときに適切な値に初期化されます。 その実行単位の存続期間中、WORKING-STORAGE 項目はそれぞれ最後に使われた状態を維持します。 例外は次のとおりです。

- PROGRAM-ID 段落に INITIAL が指定されたプログラム。

  この場合、WORKING-STORAGE データ項目は、プログラムに入るたびに再初期化されます。
- 動的に呼び出されてから取り消されるサブプログラム。

  この場合、WORKING-STORAGE データ項目は、CANCEL 後のプログラムへの最初の再入時に再初期化されます。

WORKING-STORAGE は、実行単位の終了時に割り振り解除されます。

COBOL クラス定義の WORKING-STORAGE については、関連タスクを参照してください。

LOCAL-STORAGE データの別のコピーがプログラムに対して割り振られ、プログラムから戻る時点で 解放されます。 LOCAL-STORAGE 項目に対して VALUE 節を指定すると、起動のたびに、項目はその値に初期化されます。 VALUE 節を 指定しないと、項目の初期値は未定義になります。

11 ページの『例: ストレージ・セクション』

関連タスク
431 ページの『メインプログラムまたはサブプログラムの終了と再入』

関連参照
WORKING-STORAGE SECTION (*COBOL for Linux on x86* 言語解説書)
LOCAL-STORAGE SECTION (*COBOL for Linux on x86* 言語解説書)

例: ストレージ・セクション

次の例は、WORKING-STORAGE と LOCAL-STORAGE の両方を使用する再帰的プログラムです。

```
CBL apost,pgmn(lu)
*********************************
* Recursive Program - Factorials
*********************************
 IDENTIFICATION DIVISION.
 Program-Id. factorial recursive.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  numb  pic 9(4)  value 5.
 01  fact  pic 9(8)  value 0.
 LOCAL-STORAGE SECTION.
 01  num  pic 9(4).
 PROCEDURE DIVISION.
     move numb to num.

     if numb = 0
        move 1 to fact
     else
        subtract 1 from numb
        call 'factorial'
        multiply num by fact
     end-if.
```

```
     display num '! = ' fact.
     goback.
 End Program factorial.
```

このプログラムは、次のような出力を生成します。

```
0000! = 00000001
0001! = 00000001
0002! = 00000002
0003! = 00000006
0004! = 00000024
0005! = 00000120
```

次の表は、プログラムの連続する再帰呼び出し (CALL) および結果として起こる戻り (GOBACK) における、LOCAL-STORAGE および WORKING-STORAGE 内のデータ項目の値の変化を示しています。 戻り時、fact は 5! の値を次々に累算します (5 階乗)。

再帰呼び出し (CALL)	LOCAL-STORAGE の num の値	WORKING-STORAGE の numb の値	WORKING-STORAGE の fact の値
メイン	5	5	0
1	4	4	0
2	3	3	0
3	2	2	0
4	1	1	0
5	0	0	0

戻り (GOBACK)	LOCAL-STORAGE の num の値	WORKING-STORAGE の numb の値	WORKING-STORAGE の fact の値
5	0	0	1
4	1	0	1
3	2	0	2
2	3	0	6
1	4	0	24
メイン	5	0	120

関連概念
11 ページの『WORKING-STORAGE と LOCAL-STORAGE の比較』

別のプログラムからのデータの使用

データを共用する方法は、プログラムのタイプによって異なります。 別個にコンパイルされたプログラムでは、ネストされたプログラムや、再帰的またはマルチスレッド化されたプログラムの場合とは異なる方法でデータを共用します。

関連タスク
13 ページの『別々にコンパイルされたプログラムでのデータの共用』
13 ページの『ネストされたプログラムでのデータの共用』
13 ページの『再帰的プログラムでのデータの共用』
447 ページの『データの受け渡し』

別々にコンパイルされたプログラムでのデータの共用

多くのアプリケーションは、相互に呼び出し、データを受け渡しする、別個にコンパイルされたプログラムから構成されます。 呼び出されるプログラム内の LINKAGE SECTION を用いて、別のプログラムから渡されるデータを 記述します。

呼び出し側プログラムでは、CALL . . . USING ステートメントをコーディングしてデータを渡してください。

関連タスク
447 ページの『データの受け渡し』
450 ページの『LINKAGE SECTION のコーディング』

ネストされたプログラムでのデータの共用

アプリケーションの中には、ネストされたプログラム (他のプログラムに含まれているプログラム) から構成されるものもあります。 レベル 01 のデータ項目には、GLOBAL 属性を指定できます。この属性を使用すると、宣言を含むネストされたプログラムがこれらのデータ項目にアクセスできるようになります。

ネストされたプログラムは、COMMON 属性で宣言された兄弟プログラム (同一の含まれているプログラム内で同じネスト・レベルにあるプログラム) のデータ項目にもアクセスできます。

関連概念
433 ページの『ネストされたプログラム』

再帰的プログラムでのデータの共用

プログラムが RECURSIVE 属性を持っている場合、プログラムの以降の呼び出しでは、LINKAGE SECTION で定義されたデータにアクセスできません。

LINKAGE SECTION のレコードをアドレッシングするには、以下のいずれかの技法を使用します。

- プログラムに引数を渡し、プログラムの USING 句に適切な位置のレコードを指定する。
- フォーマット-5 の SET ステートメントを使用する。

プログラムが RECURSIVE 属性を持っている場合、レコードのアドレスは、プログラム起動の特定のインスタンスについて有効です。 同じプログラムの別の実行インスタンスにあるレコードのアドレスは、その実行インスタンスに対して再設定する必要があります。 アドレスが設定されていないデータ項目を参照すると、予測できない結果が生じます。

関連タスク
444 ページの『再帰呼び出しの実行』

関連参照

SET ステートメント (*COBOL for Linux on x86* 言語解説書)

データの処理

プログラムの PROCEDURE DIVISION では、他の部で定義したデータを処理する実行可能ステートメントをコーディングします。 PROCEDURE DIVISION には、1 つまたは 2 つのヘッダーと、プログラムのロジックが入れられます。

PROCEDURE DIVISION は、部のヘッダーとプロシージャー名のヘッダーで始まります。 プログラムの部のヘッダーは、単に次のようにすることができます。

```
PROCEDURE DIVISION.
```

あるいは、USING 句を使用してパラメーターを受け取ったり、RETURNING 句を使用して値を戻すような部のヘッダーをコーディングすることができます。

参照によって (デフォルト) あるいは内容によって渡された引数を受け取るには、プログラムの部のヘッダーを次のようにコーディングしてください。

```
PROCEDURE DIVISION USING dataname
PROCEDURE DIVISION USING BY REFERENCE dataname
```

dataname は、DATA DIVISION の LINKAGE SECTION で定義しなければなりません。

値によって渡されたパラメーターを受け取るには、プログラムの部のヘッダーを次のようにコーディングしてください。

```
PROCEDURE DIVISION USING BY VALUE dataname
```

結果として値を戻すためには、部のヘッダーを次のようにコーディングしてください。

```
PROCEDURE DIVISION RETURNING dataname2
```

また、PROCEDURE DIVISION のヘッダーの中で USING と RETURNING を組み合わせることもできます。

```
PROCEDURE DIVISION USING dataname RETURNING dataname2
```

dataname および *dataname2* は、LINKAGE SECTION で必ず定義しなければなりません。

関連概念
14 ページの『PROCEDURE DIVISION 内でロジックが分割される方法』

関連タスク
450 ページの『LINKAGE SECTION のコーディング』
451 ページの『引数を受け渡すための PROCEDURE DIVISION のコーディング』
455 ページの『PROCEDURE DIVISION RETURNING . . .の使用』
509 ページの『反復コーディングの除去』

関連参照

手続き部ヘッダー (*COBOL for Linux on x86* 言語解説書)
USING 句 (*COBOL for Linux on x86* 言語解説書)
CALL ステートメント (*COBOL for Linux on x86* 言語解説書)

PROCEDURE DIVISION 内でロジックが分割される方法

プログラムの PROCEDURE DIVISION は、セクションと段落に分割されます。セクションおよび段落には、文、ステートメント、および句が含まれています。

セクション
処理ロジックの論理的な副区分。

セクションにはセクション・ヘッダーがあり、オプションとして後ろに 1 つ以上の段落が続きます。

セクションは、PERFORM ステートメントのサブジェクトにすることができます。 セクションのタイプの 1 つは宣言用です。

段落
セクション、プロシージャー、またはプログラムの再分割。

段落は、後ろにピリオドが付いた名前を持ち、その後に文が続く場合と続かない場合があります。

段落は、ステートメントのサブジェクトにすることができます。

文
1 つ以上の COBOL ステートメントの並びであって、ピリオドで終わるもの。

ステートメント
2 つの数値の加算など、COBOL 処理の定義されたステップを実行します。

ステートメントはワードの有効な組み合わせで、COBOL ステートメントで始まります。ステートメントには、命令ステートメント (無条件アクションを示す)、条件ステートメント、およびコンパイラー指示ステートメントがあります。 ステートメントの論理的な終わりを示すためには、ピリオドではなく明示範囲終了符号を使用することをお勧めします。

句
ステートメントの再分割。

関連概念
16 ページの『コンパイラー指示ステートメント』
16 ページの『範囲終了符号』
15 ページの『命令ステートメント』
15 ページの『条件ステートメント』
18 ページの『宣言』

関連参照
PROCEDURE DIVISION の構成 (*COBOL for Linux on x86* 言語解説書)

命令ステートメント

命令ステートメント (ADD、MOVE、CALL、または CLOSE など) は、取るべき無条件アクションを指示します。

命令ステートメントは、暗黙のまたは明示的な範囲終了符号を使用して終了することができます。

明示範囲終了符号で終わる条件ステートメントは、範囲区切りステートメント と呼ばれる命令ステートメントになります。 命令ステートメント (または範囲区切りステートメント) のみをネストすることができます。

関連概念
15 ページの『条件ステートメント』
16 ページの『範囲終了符号』

条件ステートメント

条件ステートメントには、単純な条件ステートメント (IF、EVALUATE、SEARCH) と、条件句またはオプションを含む命令ステートメントから構成された条件ステートメントの 2 種類があります。

条件ステートメントは、暗黙のまたは明示的な範囲終了符号を使用して終了することができます。 条件ステートメントを明示的に終わらせると、それは範囲区切りステートメント (命令ステートメント) になります。

範囲区切りステートメントは、次のような方法で使用できます。

- COBOL 条件ステートメントの操作の範囲を区切り、ネストのレベルを明示的に指示する場合。

  例えば、ネストされた IF の中の IF ステートメントの範囲を終了させるために、ピリオドではなく END-IF 句を使用します。
- COBOL 構文が命令ステートメントを必要とする条件ステートメントをコーディングする場合。

  例えば、インライン PERFORM のオブジェクトとして条件ステートメントをコーディングします。

```
PERFORM UNTIL TRANSACTION-EOF
    PERFORM 200-EDIT-UPDATE-TRANSACTION
    IF NO-ERRORS
        PERFORM 300-UPDATE-COMMUTER-RECORD
    ELSE
        PERFORM 400-PRINT-TRANSACTION-ERRORS
    END-IF
    READ UPDATE-TRANSACTION-FILE INTO WS-TRANSACTION-RECORD
```

```
        AT END
            SET TRANSACTION-EOF TO TRUE
    END-READ
END-PERFORM
```

インライン PERFORM ステートメントには、明示範囲終了符号が必須ですが、これはライン外の PERFORM ステートメントについては無効です。

追加のプログラム制御として、条件ステートメントと一緒に NOT 句を使用することもできます。 例えば、NOT ON SIZE ERROR のように、特定の例外が発生しない場合に実行される命令を指定することができます。 NOT 句は、CALL ステートメントの ON OVERFLOW 句とは一緒に使用できませんが、ON EXCEPTION 句とは一緒に使用できます。

条件ステートメントをネストしてはなりません。 ネストされるステートメントは、命令ステートメント (または範囲区切りステートメント) でなければならず、命令ステートメントの規則に従っていなければなりません。

以下に、範囲終了符号なしでコーディングされる場合の条件ステートメントの例を示します。

- ON SIZE ERROR が指定された算術ステートメント
- ON OVERFLOW が指定されたデータ操作ステートメント
- ON OVERFLOW が指定された CALL ステートメント
- INVALID KEY、AT END、または AT END-OF-PAGE が指定された I/O ステートメント
- AT END が指定された RETURN

関連概念
15 ページの『命令ステートメント』
16 ページの『範囲終了符号』

関連タスク
81 ページの『プログラム・アクションの選択』

関連参照
条件ステートメント (*COBOL for Linux on x86* 言語解説書)

コンパイラー指示ステートメント

コンパイラー指示ステートメントは、コンパイラーに、プログラム構造、COPY 処理、リスト制御について特定のアクションを行わせます。 制御フロー、または CALL インターフェース規約。

コンパイラー指示ステートメントは、プログラム・ロジックの一部ではありません。

関連参照
293 ページの『第 14 章 コンパイラー指示ステートメント』
コンパイラー指示ステートメント (*COBOL for Linux on x86* 言語解説書)

範囲終了符号

範囲終了符号はステートメントの終了を示します。 明示的または暗黙的な範囲終了符号にすることができます。

明示範囲終了符号は、文を終了させることなく、ステートメントを終了させます。 これは、END の後にハイフンと、終了させるステートメントの名前を続けたものから構成されます (例えば、END-IF)。 暗黙範囲終了符号はピリオド (.) で、まだ終了していないすべての先行ステートメントの範囲を終了します。

次のプログラム・フラグメントにおける 2 つのピリオドは、それぞれ IF ステートメントを終了させます。 そのため、このコードは、明示範囲終了符号を持つ以下の例と同等です。

```
IF ITEM = "A"
    DISPLAY "THE VALUE OF ITEM IS " ITEM
    ADD 1 TO TOTAL
    MOVE "C" TO ITEM
```

```
    DISPLAY "THE VALUE OF ITEM IS NOW " ITEM.
IF ITEM = "B"
    ADD 2 TO TOTAL.
```

```
IF ITEM = "A"
    DISPLAY "THE VALUE OF ITEM IS " ITEM
    ADD 1 TO TOTAL
    MOVE "C" TO ITEM
    DISPLAY "THE VALUE OF ITEM IS NOW " ITEM
END-IF
IF ITEM = "B"
    ADD 2 TO TOTAL
END-IF
```

暗黙の範囲終了符号を使用すると、ステートメントの終わる場所が不明確になることがあります。 結果として、ステートメントを意図に反して終了させ、プログラムのロジックが変わる可能性があります。 明示範囲終了符号を使用すると、プログラムが理解しやすくなり、ステートメントを意図に反して終了させることがなくなります。 例えば、次のプログラム・フラグメントで、最初の暗黙の範囲例の最初のピリオドの位置を変更すると、コードの意味が変更されます。

```
IF ITEM = "A"
    DISPLAY "VALUE OF ITEM IS " ITEM
    ADD 1 TO TOTAL.
    MOVE "C" TO ITEM
    DISPLAY " VALUE OF ITEM IS NOW " ITEM
IF ITEM = "B"
    ADD 2 TO TOTAL.
```

最初のピリオドが IF ステートメントを終わらせるため、その後の MOVE ステートメントおよび DISPLAY ステートメントは、字下げの意味を無視し、ITEM の値に関係なく実行されます。

プログラムをより読みやすくし、ステートメントの意図しない終了を防ぐために、特に段落内では、明示範囲終了符号を使用するようにすべきです。 暗黙の範囲終了符号は、段落の終わりまたはプログラムの終わりでのみ使用してください。

条件ステートメント内にネストされている命令ステートメントについての明示的範囲終了符号をコーディングする際には、注意が必要です。 範囲終了符号が、それが意図されたステートメントと対にされるようにしてください。 次の例では、範囲終了符号は最初の READ ステートメントと対になるように意図されましたが、実際には 2 つ目と対にされます。

```
READ FILE1
  AT END
    MOVE A TO B
    READ FILE2
END-READ
```

明示範囲終了符号が意図されたステートメントと対にされるようにするために、上記の例を次のようにコーディングし直すことができます。

```
READ FILE1
  AT END
    MOVE A TO B
    READ FILE2
    END-READ
END-READ
```

関連概念
15 ページの『条件ステートメント』
15 ページの『命令ステートメント』

宣言

宣言は、例外条件が起こったときに実行される 1 つ以上の特殊目的セクションを提供します。

各宣言セクションは、そのセクションの機能を識別する USE ステートメントで始まります。 プロシージャーの中に、条件が起こった場合に取るべきアクションを指定します。

関連タスク
304 ページの『入出力エラーの検出および処理』

関連参照
宣言 (*COBOL for Linux on x86* 言語解説書)

第 2 章 データの使用

ここに示す情報は、非 COBOL プログラマーが、他のプログラム言語で使用されているデータに関する用語を COBOL 用語と関連付けるのに役立ちます。 ここでは、COBOL の基礎である変数、構造、リテラル、定数、値の割り当てと表示、組み込み関数、およびテーブル (配列) とポインターについて紹介します。

関連タスク
19 ページの『変数、構造、リテラル、および定数の使用』
22 ページの『データ項目への値の割り当て』
31 ページの『画面上またはファイル内での値の表示 (DISPLAY)』
32 ページの『組み込み関数の使用 (組み込み関数)』
33 ページの『テーブル (配列) とポインターの使用』
179 ページの『第 10 章 国際環境でのデータの処理』

変数、構造、リテラル、および定数の使用

大半の高水準プログラム言語では、変数、構造 (グループ項目)、リテラル、または定数としてデータを表すという概念は同じです。

COBOL プログラムのデータは、英字、英数字、2 バイト文字セット (DBCS)、国別、または数値が可能です。 また指標名を定義したり、USAGE POINTER、USAGE FUNCTION-POINTER、または USAGE PROCEDURE-POINTER として記述されたデータ項目を定義したりできます。データ定義はすべて、プログラムの DATA DIVISION に入れます。

関連タスク
19 ページの『変数の使用』
20 ページの『データ項目とグループ項目の使用』
21 ページの『リテラルの使用』
22 ページの『定数の使用』
22 ページの『形象定数の使用』

関連参照
データのクラスおよびカテゴリー (*COBOL for Linux on x86* 言語解説書)

変数の使用

変数 は、プログラム実行時に値を変更できるデータ項目です。 ただし、値は、データ項目に名前と長さを与えるときに定義されたデータ型に制限されます。

例えば、お客様名がプログラム内の英数字データ項目であれば、次に示すように、そのお客様名を定義し使用することができます。

```
Data Division.
01  Customer-Name           Pic X(20).
01  Original-Customer-Name  Pic X(20).
. . .
Procedure Division.
    Move Customer-Name to Original-Customer-Name
    . . .
```

代わりに、PICTURE 節を Pic N(20) と指定し、その項目に USAGE NATIONAL 節を指定することにより、上記のお客様名を国別データ項目として定義できます。国別データ項目は Unicode UTF-16 で表され、その場合、ほとんどの文字が 2 バイトのストレージで表されます。

関連概念
180 ページの『Unicode および言語文字のエンコード』

© Copyright IBM Corp. 2021

関連タスク
180 ページの『COBOL での国別データ (Unicode) の使用』

関連参照

274 ページの『NSYMBOL』
193 ページの『文字データの保管』
PICTURE 節 (*COBOL for Linux on x86* 言語解説書)

データ項目とグループ項目の使用

連データ項目は、階層データ構造の一部となることができます。 従属データ項目を持たないデータ項目は、基本項目 と呼ばれます。 1 つ以上の従属データ項目で構成されるデータ項目をグループ項目と呼びます。

レコードは、基本項目またはグループ項目のどちらでも構いません。 グループ項目は、英数字グループ項目 または国別グループ項目 のいずれでも構いません。

例えば、以下の Customer-Record は 英数字グループ項目であり、それぞれが基本データ項目を含んでいる 2 つの従属英数字グループ項目 (Customer-Name と Part-Order) で構成されています。 これらのグループ項目は暗黙的に USAGE DISPLAY を持ちます。 以下に示すように、PROCEDURE DIVISION の MOVE ステートメントで、グループ項目全体またはグループ項目の一部を参照できます。

```
Data Division.
File Section.
FD  Customer-File
    Record Contains 45 Characters.
01  Customer-Record.
    05  Customer-Name.
        10 Last-Name        Pic x(17).
        10 Filler           Pic x.
        10 Initials         Pic xx.
    05  Part-Order.
        10 Part-Name        Pic x(15).
        10 Part-Color       Pic x(10).
Working-Storage Section.
01  Orig-Customer-Name.
    05  Surname            Pic x(17).
    05  Initials           Pic x(3).
01  Inventory-Part-Name     Pic x(15).
. . .
Procedure Division.
    Move Customer-Name to Orig-Customer-Name
    Move Part-Name to Inventory-Part-Name
    . . .
```

代わりに、以下に示すように DATA DIVISION の宣言を変更することにより、Customer-Record を、2 つの従属国別グループ項目で構成される国別グループ項目として定義できます。 国別グループ項目の振る舞いは、ほとんどの操作でカテゴリー国別の基本データ項目と同じです。 GROUP-USAGE NATIONAL 節は、グループ項目およびその従属グループ項目が国別グループであることを示します。 国別グループ内の従属基本項目は、明示的または暗黙的に USAGE NATIONAL として記述されている必要があります。

```
Data Division.
File Section.
FD  Customer-File
    Record Contains 90 Characters.
01  Customer-Record         Group-Usage National.
    05  Customer-Name.
        10 Last-Name        Pic n(17).
        10 Filler           Pic n.
        10 Initials         Pic nn.
    05  Part-Order.
        10 Part-Name        Pic n(15).
        10 Part-Color       Pic n(10).
Working-Storage Section.
01  Orig-Customer-Name      Group-Usage National.
    05  Surname            Pic n(17).
    05  Initials           Pic n(3).
01  Inventory-Part-Name     Pic n(15)  Usage National.
. . .
```

```
Procedure Division.
    Move Customer-Name to Orig-Customer-Name
    Move Part-Name to Inventory-Part-Name
    . . .
```

上記の例のグループ項目は、グループ・レベルで USAGE NATIONAL 節を指定することもできます。 グループ・レベルの USAGE 節は、グループ内のそれぞれの基本データ項目に適用されます (ですから、これは便利な省略表現と言えます)。 ただし、USAGE NATIONAL 節を指定しているグループは、グループ内の基本項目の表記にもかかわらず、国別グループではありません。 この USAGE 節を指定しているグループは英数字グループであり、多数の操作 (移動や比較など) において USAGE DISPLAY の基本データ項目と同様の振る舞いをします (ただし、データの編集や変換は行われません)。

関連概念
180 ページの『Unicode および言語文字のエンコード』
187 ページの『国別グループ』

関連タスク
180 ページの『COBOL での国別データ (Unicode) の使用』
191 ページの『国別グループの使用』

関連参照

10 ページの『FILE SECTION 記入項目』
193 ページの『文字データの保管』
グループ項目のクラスおよびカテゴリー (*COBOL for Linux on x86* 言語解説書)
PICTURE 節 (*COBOL for Linux on x86* 言語解説書)
MOVE ステートメント (*COBOL for Linux on x86* 言語解説書)
USAGE 節 (*COBOL for Linux on x86* 言語解説書)

リテラルの使用

リテラル とは、値が文字それ自体によって与えられる文字ストリングのことです。 データ項目に使用したい値が分かっている場合には、PROCEDURE DIVISION でそのデータ値のリテラル表記を使用できます。

値のデータ項目を定義したり、データ名を使用してデータ項目を参照したりする必要はありません。例えば、以下に示すように英数字リテラルを移動することにより、出力ファイル用のエラー・メッセージを準備できます。

```
Move "Name is not valid" To Customer-Name
```

以下に示すように数値リテラルを使用すれば、データ項目を特定の整数値と比較できます。 次の例では、"Name is not valid" は英数字リテラルであり、03519 は数字リテラルです。

```
01  Part-number     Pic 9(5).
. . .
    If Part-number = 03519 then display "Part number was found"
```

英数字リテラル内では、16 進表記形式 (X') を使用して、制御文字 X'00' から X'1F' を表すことができます。 これらの制御文字を英数字リテラルの基本形式で指定すると、結果は予測不能です。

NSYMBOL(NATIONAL) コンパイラー・オプションが有効であるときには、開始区切り文字 N" または N' を使用して国別リテラルを指定でき、 NSYMBOL(DBCS) コンパイラー・オプションが有効であるときには、同様にして DBCS リテラルを指定できます。

開始区切り文字 NX" または NX' を使用すれば、(NSYMBOL コンパイラー・オプションの設定とは無関係に) 16 進表記の国別リテラルを指定できます。 4 桁の 16 進数字のそれぞれのグループが、単一国別文字を指定します。

関連概念
180 ページの『Unicode および言語文字のエンコード』

関連タスク
182 ページの『国別リテラルの使用』
199 ページの『DBCS リテラルの使用』

関連参照
274 ページの『NSYMBOL』
リテラル (*COBOL for Linux on x86* 言語解説書)

定数の使用

定数 は、1 つの値しか持たないデータ項目です。 COBOL では、定数を表す構造を定義していません。 ただし、データ記述に VALUE 節をコーディングすることにより (INITIALIZE ステートメントをコーディングする代わりに)、初期値を使用してデータ項目を定義することができます。

```
Data Division.
01  Report-Header   pic x(50)  value "Company Sales Report".
. . .
01  Interest        pic 9v9999 value 1.0265.
```

上記の例は、英数字および数字データ項目を初期化します。 同様に、VALUE 節を、国別または DBCS 定数の定義に使用できます。

関連タスク
180 ページの『COBOL での国別データ (Unicode) の使用』
198 ページの『DBCS サポート用のコーディング』

形象定数の使用

通常用いられる特定の定数およびリテラルは、形象定数 と呼ばれる予約語として次のものが用意されています。ZERO、SPACE、HIGH-VALUE、LOW-VALUE、QUOTE、NULL、および ALL *literal*。 これらは固定値を表すため、形象定数はデータ定義を必要としません。

次に例を示します。

```
Move Spaces To Report-Header
```

関連タスク
186 ページの『国別文字形象定数の使用』
198 ページの『DBCS サポート用のコーディング』

関連参照

形象定数 (*COBOL for Linux on x86* 言語解説書)

データ項目への値の割り当て

データ項目を定義した後、いつでもそれに値を割り当てることができます。 COBOL における割り当ては、その背後にある目的によって多くの形式を取ります。

表 2. **プログラム内でのデータ項目の割り当て**	
目的	**方法**
データ項目または大きいデータ域に値を割り当てる。	以下のいずれかの方法を使用する。 • INITIALIZE ステートメント • MOVE ステートメント • STRING または UNSTRING ステートメント • VALUE 節 (データ項目を、プログラムが初期状態にあるときにそれに与えたい値に設定する場合)
算術の結果を割り当てる。	COMPUTE、 ADD、SUBTRACT、MULTIPLY、または DIVIDE ステートメントを使用します。
データ項目内の文字または文字グループを検査または置換します。	INSPECT ステートメントを使用します。
ファイルから値を受け取る。	READ (または READ INTO) ステートメントを使用する。
システム入力装置またはファイルから値を受け取る。	ACCEPT ステートメントを使用する。
定数を設定します。	データ項目の定義内で VALUE 節を使用し、そのデータ項目を受け取り側として使用しない。 このような項目は、コンパイラーが読み取り専用の定数としての扱いを強制しなくても、実質的に定数となります。
次のいずれかの処置。 • テーブル・エレメントに関連付けられた値を指標に設定する • 外部スイッチの状況を ON または OFF に設定する • 条件名にデータを移動して、条件を真にする • POINTER、PROCEDURE-POINTER、または FUNCTION-POINTER データ項目にアドレスを設定する	SET ステートメントを使用する。

23 ページの『例: データ項目の初期化』

関連タスク
27 ページの『構造の初期化 (INITIALIZE)』
28 ページの『基本データ項目への値の割り当て (MOVE)』
29 ページの『グループ・データ項目への値の割り当て (MOVE)』
30 ページの『画面またはファイルからの入力の割り当て (ACCEPT)』
93 ページの『データ項目の結合 (STRING)』
95 ページの『データ項目の分割 (UNSTRING)』
30 ページの『算術結果の割り当て (MOVE または COMPUTE)』
102 ページの『データ項目の計算および置換 (INSPECT)』
179 ページの『第 10 章 国際環境でのデータの処理』

例: データ項目の初期化

以下の例は、INITIALIZE ステートメントを使用して、英数字、国別編集、 および数字編集データ項目を含めたいろいろな種類のデータ項目を初期化する方法を示しています。

INITIALIZE ステートメントは、機能の面で 1 つ以上の MOVE ステートメントと同等です。 初期化に関する関連作業を見るならば、グループ項目に対して INITIALIZE ステートメントを使用して、ある特定データ・カテゴリー内にあるすべての従属データ項目をどのように便利な方法で初期化できるかが分かります。

データ項目のブランクまたはゼロへの初期化:

```
INITIALIZE identifier-1
```

***identifier-1* PICTURE**	***identifier-1* (初期化前)**	***identifier-1* (初期化後)**
9(5)	12345	00000
X(5)	AB123	*bbbbb*[1]
N(3)	410042003100[2]	200020002000[3]
99XX9	12AB3	*bbbbb*[1]
XXBX/XX	ABbC/DE	*bbbb*/*bb*[1]
**99.9CR	1234.5CR	**00.0*bb*[1]
A(5)	ABCDE	*bbbbb*[1]
+99.99E+99	+12.34E+02	+00.00E+00

1. 記号 *b* は、ブランク・スペースを表します。
2. 国別 (UTF-16) 文字「AB1」の 16 進表記。 例では、*identifier-1* が Usage National を持っていると想定しています。
3. 国別 (UTF-16) 文字「 」(3 個のブランク・スペース) の 16 進表記。 *identifier-1* が Usage National として定義されておらず、また NSYMBOL(DBCS) が有効である場合、INITIALIZE は代わりに DBCS スペース (「2020」) を *identifier-1* に保管することに注意してください。

英数字データ項目の初期化:

```
01  ALPHANUMERIC-1    PIC X    VALUE "y".
01  ALPHANUMERIC-3    PIC X(1) VALUE "A".
. . .
    INITIALIZE ALPHANUMERIC-1
        REPLACING ALPHANUMERIC DATA BY ALPHANUMERIC-3
```

ALPHANUMERIC-3	**ALPHANUMERIC-1 (初期化前)**	**ALPHANUMERIC-1 (初期化後)**
A	y	A

英数字右揃えデータ項目の初期化:

```
01  ANJUST            PIC X(8)  VALUE SPACES  JUSTIFIED RIGHT.
01  ALPHABETIC-1      PIC A(4)  VALUE "ABCD".
. . .
    INITIALIZE ANJUST
        REPLACING ALPHANUMERIC DATA BY ALPHABETIC-1
```

ALPHABETIC-1	**ANJUST (初期化前)**	**ANJUST (初期化後)**
ABCD	*bbbbbbbb*[1]	*bbbb*ABCD[1]

1. 記号 *b* は、ブランク・スペースを表します。

英数字編集データ項目の初期化:

```
01  ALPHANUM-EDIT-1   PIC XXBX/XXX  VALUE "ABbC/DEF".
01  ALPHANUM-EDIT-3   PIC X/BB      VALUE "M/bb".
. . .
    INITIALIZE ALPHANUM-EDIT-1
        REPLACING ALPHANUMERIC-EDITED DATA BY ALPHANUM-EDIT-3
```

ALPHANUM-EDIT-3	**ALPHANUM-EDIT-1 (初期化前)**	**ALPHANUM-EDIT-1 (初期化後)**
M/*bb*[1]	AB*b*C/DEF[1]	M/*bb*/*bbb*[1]
1. 記号 *b* は、ブランク・スペースを表します。		

国別データ項目の初期化:

```
01  NATIONAL-1        PIC NN  USAGE NATIONAL  VALUE N"AB".
01  NATIONAL-3        PIC NN  USAGE NATIONAL  VALUE N"CD".
. . .
    INITIALIZE NATIONAL-1
        REPLACING NATIONAL DATA BY NATIONAL-3
    INITIALIZE NATIONAL-1 NATIONAL TO VALUE
```

NATIONAL-3	**最初の INITIALIZE の前の NATIONAL-1**	**最初の INITIALIZE の後の NATIONAL-1**	**2 番目の INITIALIZE の後の NATIONAL-1**
43004400[1]	41004200[2]	43004400[1]	41004200
1. 国別文字「CD」の 16 進表記 2. 国別文字「AB」の 16 進表記			

国別編集データ項目の初期化:

```
01  NATL-EDIT-1       PIC 0NN  USAGE NATIONAL  VALUE N"123".
01  NATL-3            PIC NNN  USAGE NATIONAL  VALUE N"456".
. . .
    INITIALIZE NATL-EDIT-1
        REPLACING NATIONAL-EDITED DATA BY NATL-3
```

NATL-3	**NATL-EDIT-1 (初期化前)**	**NATL-EDIT-1 (初期化後)**
340035003600[1]	310032003300[2]	300034003500[3]
1. 国別文字「456」の 16 進表記 2. 国別文字「123」の 16 進表記 3. 国別文字「045」の 16 進表記		

数値 (ゾーン 10 進数) データ項目の初期化:

```
01  NUMERIC-1         PIC 9(8)        VALUE 98765432.
01  NUM-INT-CMPT-3    PIC 9(7)  COMP  VALUE 1234567.
. . .
    INITIALIZE NUMERIC-1
        REPLACING NUMERIC DATA BY NUM-INT-CMPT-3
```

NUM-INT-CMPT-3	NUMERIC-1 (初期化前)	NUMERIC-1 (初期化後)
1234567	98765432	01234567

数値 (国別 10 進数) データ項目の初期化:

```
01  NAT-DEC-1         PIC 9(3)  USAGE  NATIONAL VALUE 987.
01  NUM-INT-BIN-3     PIC 9(2)  BINARY VALUE 12.
. . .
    INITIALIZE NAT-DEC-1
        REPLACING NUMERIC DATA BY NUM-INT-BIN-3
```

NUM-INT-BIN-3	NAT-DEC-1 (初期化前)	NAT-DEC-1 (初期化後)
12	390038003700[1]	300031003200[2]
1. 国別文字「987」の 16 進表記 2. 国別文字「012」の 16 進表記		

数字編集 (USAGE DISPLAY) データ項目の初期化:

```
01  NUM-EDIT-DISP-1   PIC $ZZ9V  VALUE "$127".
01  NUM-DISP-3        PIC 999V   VALUE 12.
. . .
    INITIALIZE NUM-EDIT-DISP-1
        REPLACING NUMERIC-EDITED DATA BY NUM-DISP-3
```

NUM-DISP-3	NUM-EDIT-DISP-1 (初期化前)	NUM-EDIT-DISP-1 (初期化後)
012	$127	$ 12

数字編集 (USAGE NATIONAL) データ項目の初期化:

```
01  NUM-EDIT-NATL-1   PIC $ZZ9V  NATIONAL VALUE N"$127".
01  NUM-NATL-3        PIC 999V   NATIONAL VALUE 12.
. . .
    INITIALIZE NUM-EDIT-NATL-1
        REPLACING NUMERIC-EDITED DATA BY NUM-NATL-3
```

NUM-NATL-3	NUM-EDIT-NATL-1 (初期化前)	NUM-EDIT-NATL-1 (初期化後)
300031003200[1]	2400310032003700[2]	2400200031003200[3]
1. 国別文字「012」の 16 進表記 2. 国別文字「$127」の 16 進表記 3. 国別文字「$ 12」の 16 進表記		

関連タスク

27 ページの『構造の初期化 (INITIALIZE)』
65 ページの『テーブルの初期化 (INITIALIZE)』
35 ページの『数値データの定義』

関連参照

274 ページの『NSYMBOL』

構造の初期化 (INITIALIZE)

INITIALIZE ステートメントをそのグループ項目に適用することによって、グループ項目内のすべての従属データ項目の値を初期化することができます。 ただし、グループ内のすべての項目を初期化することが必要な場合を除き、グループ全体を初期化することは非効率的です。

以下の例は、プログラムが作成するトランザクション・レコードの各フィールドをスペースおよびゼロにリセットする方法を示しています。 フィールドの値は、作成される各レコードで同一というわけではありません。 (トランザクション・レコードは、英数字グループ項目 TRANSACTION-OUT として定義されています。)

```
01  TRANSACTION-OUT.
    05  TRANSACTION-CODE        PIC X.
    05  PART-NUMBER             PIC 9(6).
    05  TRANSACTION-QUANTITY    PIC 9(5).
    05  PRICE-FIELDS.
        10  UNIT-PRICE          PIC 9(5)V9(2).
        10  DISCOUNT            PIC V9(2).
        10  SALES-PRICE         PIC 9(5)V9(2).
. . .
    INITIALIZE TRANSACTION-OUT
```

レコード	**TRANSACTION-OUT (初期化前)**	**TRANSACTION-OUT (初期化後)**
1	R00138300024000000000000000	*b*00000000000000000000000000[1]
2	R00139000048000000000000000	*b*00000000000000000000000000[1]
3	S00141000012000000000000000	*b*00000000000000000000000000[1]
4	C00138300000000004250000000	*b*00000000000000000000000000[1]
5	C00201000000000000100000000	*b*00000000000000000000000000[1]
1. 記号 *b* は、ブランク・スペースを表します。		

同様に国別グループ項目内のすべての従属データ項目の値は、そのグループ項目に INITIALIZE ステートメントを適用することにより、リセットできます。 以下の構造は、上記の構造と似ていますが、Unicode UTF-16 データを使用している点で異なります。

```
01  TRANSACTION-OUT GROUP-USAGE NATIONAL.
    05  TRANSACTION-CODE        PIC N.
    05  PART-NUMBER             PIC 9(6).
    05  TRANSACTION-QUANTITY    PIC 9(5).
    05  PRICE-FIELDS.
        10  UNIT-PRICE          PIC 9(5)V9(2).
        10  DISCOUNT            PIC V9(2).
        10  SALES-PRICE         PIC 9(5)V9(2).
. . .
    INITIALIZE TRANSACTION-OUT
```

トランザクション・レコードの直前の内容とは無関係に、上記の INITIALIZE ステートメントの実行後には次のようになります。

- TRANSACTION-CODE には NX"2000" (国別スペース) が入ります。
- TRANSACTION-OUT の残りの 27 の国別文字の各位置には、NX"3000" (国別 10 進数のゼロ) が入ります。

INITIALIZE ステートメントを使用して英数字または国別グループ・データ項目を初期化すると、そのデータ項目はグループ項目として、つまりグループ・セマンティクスで処理されます。 グループ内の基本データ項目は、上記の例に示されているように認識および処理されます。 INITIALIZE ステートメントの REPLACING 句をコーディングしない場合、次のようになります。

- SPACE は、英字、英数字、英数字編集、DBCS、カテゴリー国別、および国別編集の各受信項目用の暗黙の送信項目です。
- ZERO は、数字および数字編集受信項目用の暗黙の送信項目です。

関連概念
187 ページの『国別グループ』

関連タスク
65 ページの『テーブルの初期化 (INITIALIZE)』
191 ページの『国別グループの使用』

関連参照

INITIALIZE ステートメント (*COBOL for Linux on x86* 言語解説書)

基本データ項目への値の割り当て (MOVE)

MOVE ステートメントを使用して、基本データ項目に値を割り当てます。

以下の文は、基本データ項目 Customer-Name の内容を、基本 データ項目 Orig-Customer-Name に割り当てます。

```
Move Customer-Name to Orig-Customer-Name
```

Customer-Name が Orig-Customer-Name より長い場合は、右側で切り捨てが起こります。 Customer-Name が短い場合には、Orig-Customer-Name の右側の余分な文字位置がスペースで埋められます。

数値を含んでいるデータ項目の場合、文字データ項目の場合よりも移動が複雑になることがあります。これは、数値には表現方法が幾通りもあるためです。 文字データでは桁ごとの移動が行われますが、一般に数値では、可能な場合には 代数値が移動されます。 例えば、以下の MOVE ステートメントの場合、 Item-x には、値 3.0 (0030 で表される) が入ります。

```
01  Item-x      Pic 999v9.
. . .
    Move 3.06 to Item-x
```

英字、英数字、英数字編集、DBCS、整数、または数字編集の各データ項目を、カテゴリー国別または国別編集データ項目に移動でき、送出項目が変換されます。 国別データ項目をカテゴリー国別または国別編集のデータ項目に移動できます。 カテゴリー国別データ項目の内容に数値が含まれている場合、その項目を、数値、数字編集、外部浮動小数点、または内部浮動小数点のデータ項目に移動できます。 国別編集データ項目は、カテゴリー国別データ項目または別の国別編集データ項目にのみ移動できます。 埋め込みや切り捨てが行われることがあるので、

基本移動の全詳細については、MOVE ステートメントに関する以下の関連資料を参照してください。

以下の例は、国別データ項目に移動される、ギリシャ語の英数字データ項目を示しています。

```
. . .
01 Data-in-Unicode   Pic N(100) usage national.
01 Data-in-Greek     Pic X(100).
. . .
    Read Greek-file into Data-in-Greek
    Move Data-in-Greek to Data-in-Unicode
```

関連概念
180 ページの『Unicode および言語文字のエンコード』

関連タスク
29 ページの『グループ・データ項目への値の割り当て (MOVE)』
188 ページの『国別 (Unicode) 表現との間の変換』

関連参照
データのクラスおよびカテゴリー (*COBOL for Linux on x86* 言語解説書)
MOVE ステートメント (*COBOL for Linux on x86* 言語解説書)

グループ・データ項目への値の割り当て (MOVE)

グループ・データ項目に値を割り当てるには、MOVE ステートメントを使用してください。

国別グループ項目 (GROUP-USAGE NATIONAL 節で記述されているデータ項目) を別の国別グループ項目に移動できます。 それぞれの国別グループ項目がカテゴリー国別の基本項目であるかのように、すなわち、それぞれの項目が PIC N(*m*) (ここで、*m* はその項目の長さ (国別文字位置数) です) として記述されているかのように、コンパイラーは移動を処理します。

英数字グループ項目を、英数字グループ項目または国別グループ項目に移動できます。 国別グループ項目を英数字グループ項目に移動することもできます。 コンパイラーはこのような移動をグループ移動として実行します。すなわち、送信グループまたは受信グループ内の個々の基本項目を考慮に入れずに、また送信データ項目を変換せずに実行します。 送信および受信グループ項目内の従属データ記述の互換性が保たれるようにしてください。 実行時に破壊オーバーラップが起きたとしても移動は行われます。

CORRESPONDING 句を MOVE ステートメントにコーディングすれば、従属基本項目を、あるグループ項目から別のグループ項目の同一名の対応する従属基本項目に移動できます。

```
01  Group-X.
    02 T-Code    Pic X    Value "A".
    02 Month     Pic 99   Value 04.
    02 State     Pic XX   Value "CA".
    02 Filler    PIC X.
01  Group-N     Group-Usage National.
    02 State     Pic NN.
    02 Month     Pic 99.
    02 Filler    Pic N.
    02 Total     Pic 999.
. . .
    MOVE CORR Group-X TO Group-N
```

上記の例では、Group-N 内の State および Month は、Group-X から State および Month の国別表現の値をそれぞれ受け取ります。 Group-N 内の他のデータ項目は未変更のままです。 (受信グループ項目内の Filler 項目は、 MOVE CORRESPONDING ステートメントによっては変更されません。)

MOVE CORRESPONDING ステートメントでは、送信グループ項目および受信グループ項目はグループ項目として扱われ、基本データ項目としては扱われません。グループ・セマンティクスが適用されます。 すなわち、それぞれのグループ内の基本データ項目が認識され、結果は、対応するデータ項目のそれぞれのペアが、別個の MOVE ステートメントで参照された場合と同じになります。 データ変換は、以下の関連した解説書に明記されている MOVE ステートメントの規則に従って実行されます。 どのタイプの基本データ項目が対応するかについての詳細は、 CORRESPONDING 句に関する関連した解説書を参照してください。

関連概念
180 ページの『Unicode および言語文字のエンコード』
187 ページの『国別グループ』

関連タスク
28 ページの『基本データ項目への値の割り当て (MOVE)』
191 ページの『国別グループの使用』
188 ページの『国別 (Unicode) 表現との間の変換』

関連参照

グループ項目のクラスおよびカテゴリー (*COBOL for Linux on x86* 言語解説書)

MOVE ステートメント (*COBOL for Linux on x86* 言語解説書)
CORRESPONDING 句 (*COBOL for Linux on x86* 言語解説書)

算術結果の割り当て (MOVE または COMPUTE)

データ項目に数値を割り当てるときには、MOVE ステートメントではなく COMPUTE ステートメントを使用することを考慮してください。

```
Move w to z
Compute z = w
```

上記の例では、ほとんどの場合、2 つのステートメントは同じ効果があります。 ただし、MOVE ステートメントは割り当て時に切り捨てを行います。 DIAGTRUNC コンパイラー・オプションを使用して、数値受け取り側で切り捨てが起こる可能性のある MOVE ステートメントについてコンパイラーが警告を出すように要求することができます。

ただし、実行時に左側の有効数字が失われる場合、COMPUTE ステートメントを使用するとこの条件を検出し、それに対処することができます。 COMPUTE ステートメントの ON SIZE ERROR 句を使用すると、コンパイラーはサイズ・オーバーフロー条件を検出するコードを生成します。 条件が起こると、ON SIZE ERROR 句内のコードが実行され、z の内容は未変更のままになります。 ON SIZE ERROR 句を指定しないと、割り当て時に切り捨てが行われます。 MOVE ステートメントの ON SIZE ERROR サポートはありません。

COMPUTE ステートメントを使用して、算術式または組み込み関数の結果をデータ項目に割り当てることもできます。 次に例を示します。

```
Compute z = y + (x ** 3)
Compute x = Function Max(x y z)
```

関連参照

264 ページの『DIAGTRUNC』
組み込み関数 (*COBOL for Linux on x86* 言語解説書)

画面またはファイルからの入力の割り当て (ACCEPT)

データ項目に値を割り当てる方法の 1 つとして、画面またはファイルから値を読み取ることができます。

画面からデータを入力するには、最初にモニターを SPECIAL-NAMES 段落内の簡略名と関連付けます。 次に、 ACCEPT を使用して、画面から入力された入力行をデータ項目に割り当てます。 次に例を示します。

```
Environment Division.
Configuration Section.
Special-Names.
    Console is Names-Input.
. . .
    Accept Customer-Name From Names-Input
```

画面ではなくファイルから読み取る場合は、次のいずれかの変更を行います。

- Console を *device* に変更します (ここで、*device* は任意の有効なシステム装置 (例えば、SYSIN) です)。 例えば、次のように指定します。

  ```
  SYSIN is Names-Input
  ```

- export コマンドを使用して、環境変数 CONSOLE を有効なファイル指定に設定します。 例えば、次のように指定します。

```
export CONSOLE=/myfiles/myinput.rpt
```

  環境変数名は、使用されているシステム装置名と同じものでなければなりません。 上記の例では、システム装置は Console ですが、環境変数をシステム装置名に割り当てる方法は、すべての有効なシステム装置に対してサポートされます。 例えば、システム装置が SYSIN の場合は、ファイル指定に割り当てなければならない環境変数も SYSIN となります。

ACCEPT ステートメントは、入力行をデータ項目に割り当てます。 入力行がデータ項目より短い場合は、適切な表現のスペースがデータ項目に埋め込まれます。 画面から読み取るとき、入力行がデータ項目より長いと、残った文字は廃棄されます。 ファイルから読み取るとき、入力行がデータ項目より長いと、残った文字はファイルの次の入力行として保存されます。

ACCEPT ステートメントを使用するなら、値を英数字または国別グループ項目に割り当てたり、USAGE DISPLAY、USAGE DISPLAY-1、または USAGE NATIONAL が指定されている基本データ項目に割り当てたりすることができます。

値を USAGE NATIONAL データ項目に割り当てると、入力データは、現在のランタイム・ロケールに関連付けられているコード・ページから国別 (Unicode UTF-16) 表現に変換されます。ただし、これは、端末からの入力の場合に限られます。

入力データが他の装置からのものであるときに変換を実行させるには、NATIONAL-OF 組み込み関数を使用します。

関連概念
180 ページの『Unicode および言語文字のエンコード』

関連タスク
215 ページの『環境変数の設定』
189 ページの『英数字または DBCS から国別への変換 (NATIONAL-OF)』
383 ページの『CICS のもとでのシステム日付の取得』

関連参照
ACCEPT ステートメント (*COBOL for Linux on x86* 言語解説書)
SPECIAL-NAMES 段落 (*COBOL for Linux on x86* 言語解説書)

画面上またはファイル内での値の表示 (DISPLAY)

DISPLAY ステートメントを使用すると、データ項目の値を画面に表示したり、ファイルに書き込むことができます。

```
Display "No entry for surname '" Customer-Name "' found in the file.".
```

上記の例で、データ項目 *Customer-Name* の内容が JOHNSON の場合、ステートメントは、次のメッセージを画面に表示します。

```
No entry for surname 'JOHNSON' found in the file.
```

データを画面以外の宛先に書き込みたい場合には、UPON 句を使用してください。 例えば、次のステートメントは、SYSOUT 環境変数の値として指定した ファイルに書き込みます。

```
Display "Hello" upon sysout.
```

USAGE NATIONAL データ項目の値を表示するとき、出力データは、現在のロケールに関連付けられているコード・ページに変換されます。

関連概念
180 ページの『Unicode および言語文字のエンコード』

関連タスク

189 ページの『国別の英数字への変換 (DISPLAY-OF)』
382 ページの『CICS のもとで実行する COBOL プログラムのコーディング』

関連参照

220 ページの『ランタイム環境変数』
DISPLAY ステートメント (*COBOL for Linux on x86* 言語解説書)

組み込み関数の使用 (組み込み関数)

一部の高水準プログラム言語には組み込み関数があります。組み込み関数は、プログラム内で、定義済み属性および事前定義値を持つ変数であるかのように参照することができます。 COBOL では、これらの関数は 組み込み関数 (intrinsic functions) と呼ばれます。 これらの関数はストリングと数値を操作する機能を提供します。

組み込み関数の値は参照時に自動的に導き出されるため、関数を DATA DIVISION で定義する必要はありません。 引数として使用する非リテラル・データ項目だけを定義してください。 形象定数を引数として使用することはできません。

関数 *ID* は、COBOL 予約語 FUNCTION と、それに続く関数名 (Max など) と、それに続く関数の評価に使用される任意の引数 (x、y、z など) の組み合わせです。 (必要に応じて、関数名が REPOSITORY 段落で参照されている場合は予約語 FUNCTION を省略できます。) 例えば、強調表示された語句のグループは関数 ID です。

```
Unstring Function Upper-case(Name) Delimited By Space
    Into Fname Lname
Compute A = 1 + Function Log10(x)
Compute M = Function Max(x y z)
```

関数 ID は、関数の呼び出しと、その関数によって返されるデータ値の両方を表します。関数 ID は、実際にデータ項目を表すため、戻り値の属性を持つデータ項目が使用できるところならば、PROCEDURE DIVISION のほとんどの場所で使用することができます。

COBOL ワード function は予約語ですが、関数名は予約されていません。 このため、関数名を他のコンテキスト (データ項目の名前など) で使用することができます。 例えば、Sqrt は、組み込み関数を呼び出し、プログラム内のデータ項目を指定するために使用できます。

```
Working-Storage Section.
01  x                 Pic 99  value 2.
01  y                 Pic 99  value 4.
01  z                 Pic 99  value 0.
01  Sqrt              Pic 99  value 0.
. . .
    Compute Sqrt = 16 ** .5
    Compute z = x + Function Sqrt(y)
    . . .
```

関数 ID は、英数字、国別、数字、または整数のいずれかのタイプの値を表します。 英数字関数または国別関数の関数 ID には、サブストリング指定 (参照修飾子) を組み込むことができます。 数字組み込み関数は、それらが戻す数値のタイプに応じてさらに分類されます。

関数 MAX、MIN、DATEVAL、および UNDATE は、指定された引数の型に応じていずれかのタイプの値を戻します。

関数 DATEVAL、UNDATE、および YEARWINDOW は、ウィンドウ表示日付フィールドの操作および変換を援助するために、2000 年言語拡張として提供されています。

関数は、ネストされた関数の結果が外側の関数の引数についての要件を満たす限り、引数として他の関数を参照することができます。 例えば、Function Sqrt(5) は数値を戻します。 したがって、下記の MAX 関数への 3 つの引数はすべて、この関数についての許容される引数型である数値になります。

```
Compute x = Function Max((Function Sqrt(5)) 2.5 3.5)
```

関連タスク
79 ページの『組み込み関数を使用したテーブル項目の処理』
104 ページの『データ項目の変換 (組み込み関数)』
106 ページの『データ項目の評価 (組み込み関数)』

テーブル (配列) とポインターの使用

COBOL では、配列はテーブル と呼ばれます。 テーブルは、OCCURS 節を使用して DATA DIVISION に定義される論理的に連続するデータ項目の集合です。

ポインターは、仮想記憶アドレスを含むデータ項目です。 ポインターは、USAGE IS POINTER 節を使用して DATA DIVISION の中に明示的に定義するか、または ADDRESS OF 特殊レジスターとして暗黙的に定義します。

ポインター・データ項目を使用して以下の操作を実行することができます。

- CALL . . BY REFERENCE ステートメントを使用してプログラム間でポインター・データ項目の受け渡しを行う。
- ALLOCATE ステートメントおよび FREE ステートメントを使用して、割り振られたストレージやフリー・ストレージを向くようにポインターを設定する。
- SET ステートメントを使用してそれらを他のポインターへ移動する。
- 比較条件を使用して他のポインターと比較し、等しいかどうかを調べる。
- VALUE IS NULL を使用して、それらが無効なアドレスを含むように初期化する。

ポインター・データ項目は、以下のことを行うために使用してください。

- 限定された基底アドレッシングの実施。特に、レコード域 (OCCURS DEPENDING ON で定義されるために可変位置である) のアドレスを受け渡ししたい場合。
- チェーン・リストの処理。

関連タスク
59 ページの『テーブルの定義 (OCCURS)』
454 ページの『プロシージャー・ポインターと関数ポインターの使用』

第 3 章 数値および算術演算

一般的に、COBOL 数値データは、一連の 10 進数字の桁として表示することができます。 ただし、数値項目は、算術符号や通貨記号などの特殊な特性を持つこともできます。

算術演算を効率的に実行できるように数値データを定義、表示、および格納するには、次のようにします。

- 数値データを定義するには、PICTURE 節と、文字 9、+、-、P、S、および V を使用します。
- 数値データを表示するには、PICTURE 節と、MOVE および DISPLAY ステートメントと一緒に編集文字 (Z、コンマ、ピリオドなど) を使用します。
- 数値データの格納方法を制御するには、さまざまな形式を指定した USAGE 節を使用します。
- データ値が適切であるかどうかを妥当性検査するには、数値のクラス・テストを使用します。
- 算術を実行するには、ADD、SUBTRACT、MULTIPLY、DIVIDE、および COMPUTE ステートメントを使用します。
- 必要な通貨記号を指定するには、CURRENCY SIGN 節と適切な PICTURE 文字を使用します。

関連タスク
35 ページの『数値データの定義』
37 ページの『数値データの表示』
38 ページの『数値データの保管方法の制御』
48 ページの『非互換データの検査 (数値のクラス・テスト)』
48 ページの『算術の実行』
55 ページの『通貨記号の使用』

数値データの定義

数値項目を定義するには、数値の 10 進数の桁数を表すためにデータ記述で文字 9 を指定した PICTURE 節を使用します。 英数字データ項目用の X を使用しないでください。

例えば、以下の Count-y は数値データ項目であり、USAGE DISPLAY を持つ外部 10 進数項目 (ゾーン *10* 進数項目) です。

```
05  Count-y        Pic 9(4)  Value 25.
05  Customer-name  Pic X(20) Value "Johnson".
```

同様にして、国別文字 (UTF-16) を保持する数値データ項目を定義できます。 例えば、以下の Count-n は USAGE NATIONAL を持つ外部 10 進数データ (国別 *10* 進数項目) です。

```
05  Count-n        Pic 9(4)  Value 25  Usage National.
```

デフォルトのコンパイラー・オプションである ARITH(COMPAT) (互換モード と呼ばれる) を使用してコンパイルする場合は、PICTURE 節には最大 18 桁までコーディングすることができます。 ARITH(EXTEND) (拡張モード と呼ばれる) を使用してコンパイルする場合は、PICTURE 節には最大 31 桁までコーディングすることができます。

それ以外にコーディングできる特殊な意味を持つ文字は、次のとおりです。

P
先行ゼロまたは後続ゼロを示します。

S
正または負の符号を示します。

V
小数点を暗黙指定します。

© Copyright IBM Corp. 2021

次の例の s は、値が符号付きであることを意味します。

```
05  Price  Pic s99v99.
```

したがって、このフィールドには、正または負の値を格納することができます。 v は、暗黙の小数点の位置を示しますが、ストレージ上の位置を占めないので、項目のサイズには含まれません。 デフォルトでは s はストレージ上の位置を必要としないので、通常、s は数値項目のサイズに含まれません。

しかし、プログラムまたはデータを別のマシンに移植する予定である場合、ゾーン 10 進数データ項目用の符号をストレージ上の別個の位置としてコーディングすることができます。 次の場合、符号は 1 バイトを占めます。

```
05  Price  Pic s99V99  Sign Is Leading, Separate.
```

このようにすれば、現在使用中のマシンとは非分離符号を格納するための規約が異なるマシンを使用する場合に、予測外の結果が生じることがなくなります。

分離符号は、印刷または表示されるゾーン 10 進数データ項目にとっても望ましいものです。

分離符号は、符号付きの国別 10 進数データ項目には必要です。 符号は、以下の例のように 2 バイトのストレージを占有します。

```
05 Price Pic s99V99 Usage National Sign Is Leading, Separate.
```

PICTURE 節に内部浮動小数点データ (COMP-1 または COMP-2) を指定することはできません。 しかし、VALUE 節を使用して、内部浮動小数点リテラルの初期値を提供できます。

```
05  Compute-result  Usage Comp-2  Value 06.23E-24.
```

外部浮動小数点データについては、以下に参照されている例および数値データの形式に関する関連概念を参照してください。

41 ページの『例: 数値データおよび内部表現』

関連概念
39 ページの『数値データの形式』
531 ページの『付録 C 中間結果および算術精度』

関連タスク
37 ページの『数値データの表示』
38 ページの『数値データの保管方法の制御』
48 ページの『算術の実行』
187 ページの『国別数値データ項目の定義』

関連参照

47 ページの『ゾーンおよびパック 10 進数データのサイン表記』
193 ページの『文字データの保管』
253 ページの『ARITH』
SIGN 節 (*COBOL for Linux on x86* 言語解説書)

数値データの表示

数値項目を特定の編集記号 (小数点、コンマ、ドル記号、借方記号、貸方記号など) を付けて定義すると、項目の表示または印刷時に、項目をより見やすく理解しやすいようにすることができます。

例えば、以下のコードの Edited-price は、USAGE DISPLAY を持つ数字編集項目です。 (数字編集項目に節 USAGE IS DISPLAY を指定することができますが、これは暗黙の指定です。 これは、項目が文字形式で保管されることを意味します。)

```
05  Price          Pic     9(5)v99.
05  Edited-price   Pic  $zz,zz9.99.
. . .
Move Price To Edited-price
Display Edited-price
```

Price の内容が 0150099 (値 1,500.99 を表す) である場合は、コードを実行すると $ 1,500.99 が表示されます。 Edited-price の PICTURE 節内の z は、先行ゼロの抑止を示します。

英数字ではなく国別 (UTF-16) 文字を保持する数字編集データ項目を定義できます。 そのためには、数字編集項目を USAGE NATIONAL と定義してください。USAGE NATIONAL を持つ数字編集項目の 場合の編集記号の効果は、USAGE DISPLAY を持つ数字編集項目の場合と同様ですが、編集が国別文字で行われる点は異なります。 例えば、上記のコードで Edited-price が USAGE NATIONAL と宣言された場合、項目は国別文字を使用して編集および表示されます。

基本数値項目または数字編集項目に BLANK WHEN ZERO 節をコーディングすることにより、値ゼロが項目に保管されたとき、その項目がスペースで充てんされるようにすることができます。 例えば、以下のそれぞれの DISPLAY ステートメントの場合、ゼロではなくブランクが表示されます。

```
05 Price           Pic     9(5)v99.
05 Edited-price-D  Pic  $99,999.99
       Blank When Zero.
05 Edited-price-N  Pic  $99,999.99 Usage National
       Blank When Zero.
. . .
Move 0 to Price
Move Price to Edited-price-D
Move Price to Edited-price-N
Display Edited-price-D
Display Edited-price-N
```

算術式の中、あるいは ADD、SUBTRACT、MULTIPLY、DIVIDE、または COMPUTE ステートメントの中で、数字編集項目を送信オペランドとして使用することはできません。 (これらのステートメントのいずれかで数字編集項目が受信フィールドであるとき、あるいは MOVE ステートメントが数字編集受信フィールド、および数字編集または数値送信フィールドを持っているとき、数字編集が行われます)。 数字編集項目は、主として、数値データの表示または印刷のために使用されます。

数字編集項目は、数値項目または数字編集項目に移動することができます。 以下の例では、数字編集項目の値 (USAGE DISPLAY を持っているか USAGE NATIONAL を持っているかにかかわらず) は数値項目に移動します。

```
Move Edited-price to Price
Display Price
```

上記の最初の例のステートメントの直後にこれら 2 つのステートメントが続いている場合、Price は 0150099 (値 1,500.99 を表す) と表示されます。 Edited-price が USAGE NATIONAL を持っている場合にも、Price は 0150099 と表示されます。

数字編集項目を、英数字データ項目、英数字編集データ項目、浮動小数点データ項目、および国別データ項目に移動することもできます。 数字編集データの有効な受信項目の完全なリストについては、 MOVE ステートメントに関する関連した解説書を参照してください。

41 ページの『例: 数値データおよび内部表現』

関連タスク
31 ページの『画面上またはファイル内での値の表示 (DISPLAY)』
38 ページの『数値データの保管方法の制御』
35 ページの『数値データの定義』
48 ページの『算術の実行』
187 ページの『国別数値データ項目の定義』
188 ページの『国別 (Unicode) 表現との間の変換』

関連参照

MOVE ステートメント (*COBOL for Linux on x86* 言語解説書)
BLANK WHEN ZERO 節 (*COBOL for Linux on x86* 言語解説書)

数値データの保管方法の制御

データ記述記入項目に USAGE 節をコーディングすることによって、コンピューターが数値データを保管する方法を制御することができます。

次のような理由から、形式を制御する場合があります。

- 計算データ型を使用して実行する算術の方が、USAGE DISPLAY や USAGE NATIONAL データ型を使用して実行する算術より効率的である。
- パック 10 進数形式の方が、USAGE DISPLAY または USAGE NATIONAL データ型に比べ、1 桁当たりのストレージが少なくて済む。
- パック 10 進数形式の方が 2 進数形式の場合よりも、DISPLAY または NATIONAL 形式との変換が効率的である。
- 浮動小数点形式は、位取りが大きく変化するような算術オペランドおよび結果を格納するのに最適であり、最大数の有効数字が維持される。
- データをあるマシンから別のマシンに移すときに、データ形式を保持する必要がある。

プログラム内で使用する数値データは、COBOL で使用可能な以下の形式のいずれかです。

- 外部 10 進数 (USAGE DISPLAY または USAGE NATIONAL)
- 外部浮動小数点 (USAGE DISPLAY または USAGE NATIONAL)
- 内部 10 進数 (USAGE PACKED-DECIMAL)
- 2 進数 (USAGE BINARY)
- 固有 2 進数 (USAGE COMP-5)
- 内部浮動小数点 (USAGE COMP-1 または USAGE COMP-2)

COMP および COMP-4 は BINARY と同義であり、COMP-3 は PACKED-DECIMAL と同義です。

コンパイラーは、表示可能な数値をそれらの数値の内部表現に変換してから、それらを算術演算で使用します。 したがって、データ項目を DISPLAY や NATIONAL ではなく BINARY または PACKED-DECIMAL と定義すると、さらに効率的になることがよくあります。 次に例を示します。

```
05  Initial-count   Pic S9(4)  Usage Binary   Value 1000.
```

どの USAGE 節を使用して値の内部表現を制御するかにかかわらず、使用する PICTURE 節の規約および VALUE 節の 10 進値は同じです (ただし、 内部浮動小数点データの場合は別で、この場合、PICTURE 節を使用できません)。

41 ページの『例: 数値データおよび内部表現』

関連概念
39 ページの『数値データの形式』

46 ページの『データ形式の変換』
531 ページの『付録 C 中間結果および算術精度』

関連タスク
35 ページの『数値データの定義』
37 ページの『数値データの表示』
48 ページの『算術の実行』

関連参照

46 ページの『変換および精度』
47 ページの『ゾーンおよびパック 10 進数データのサイン表記』

数値データの形式

数値データに使用できる形式には幾つかあります。

外部 10 進数 (DISPLAY および NATIONAL) 項目

カテゴリー数値データ項目で USAGE DISPLAY が (コーディングされているため、あるいはデフォルトにより) 有効である場合、ストレージのそれぞれの位置 (バイト) は 1 つの 10 進数字を含みます。 項目は表示可能な形式で保管されます。 USAGE DISPLAY を持つ外部 10 進数項目は、ゾーン *10* 進数 データ項目と呼びます。

カテゴリー数値データ項目で USAGE NATIONAL が有効である場合、それぞれの 10 進数字ごとに 2 バイトのストレージが必要です。 項目は UTF-16 形式で保管されます。 USAGE NATIONAL を持つ外部 10 進数項目は、国別 *10* 進数 データ項目と呼ばれます。

国別 10 進数データ項目は、符号付きの場合には、SIGN SEPARATE 節が有効になっている必要があります。 ゾーン 10 進数項目のその他の規則すべてが国別 10 進数項目に適用されます。 国別 10 進数項目は、他のカテゴリー数値データ項目を使用できる場所ならどこででも使用できます。

外部 10 進数 (ゾーン 10 進数と国別 10 進数の両方) データ項目は 、プログラムとファイル、端末、またはプリンターとの間で数値をやり取りすることを主な目的としています。 外部 10 進数項目は、算術処理で、オペランドおよび受け取り側として使用することもできます。 ただし、プログラムで多数の算術計算を集中的に実行し、効率を優先させるのであれば、算術計算で使用するデータ項目には、COBOL の計算数値タイプを使用した方がよい場合もあります。

外部浮動小数点 (DISPLAY および NATIONAL) 項目

浮動小数点データ項目で USAGE DISPLAY が (コーディングされているため、あるいはデフォルトにより) 有効である場合、それぞれの PICTURE 文字位置 (使用されている場合、暗黙の小数点である v を除く) は 1 バイトのストレージを占有します。 項目は表示可能な形式で保管されます。 USAGE DISPLAY を持つ外部浮動小数点項目は、USAGE NATIONAL を持つ外部浮動小数点項目と区別する必要があるとき、本書では表示浮動小数点 データ項目と呼びます。

以下の例では、Compute-Result が暗黙的に表示浮動小数点項目として定義されています。

```
05  Compute-Result  Pic -9v9(9)E-99.
```

負符号 (-) は、仮数および指数が必ず負の数値でなければならないことを意味するものではありません。 負符号は、数値が表示されるときに、正数であれば符号がブランクとなり、負数であれば符号が負符号となることを意味しています。 正符号 (+) をコーディングすると、符号は、正数であれば正符号となり、負数であれば負符号となります。

浮動小数点データ項目で USAGE NATIONAL が有効である場合、それぞれの PICTURE 文字位置 (使用されている場合、v を除く) は 2 バイトのストレージを占有します。 項目は国別文字 (UTF-16) として保管されます。 USAGE NATIONAL を持つ外部浮動小数点項目は、 国別浮動小数点 データ項目と呼びます。

表示浮動小数点項目の既存の規則は、国別浮動小数点項目に適用されます。

以下の例では、Compute-Result-N は国別浮動小数点項目です。

```
05  Compute-Result-N  Pic -9v9(9)E-99  Usage National.
```

Compute-Result-N が表示される場合、Compute-Result について上述したように符号が表示されますが、国別文字で表示されます。

外部浮動小数点項目に VALUE 節を使用することはできません。

浮動小数点数は、外部 10 進数と同様、(コンパイラーによって) 数値の内部表現に変換しなければ、算術演算で使用することはできません。 デフォルト・オプションの ARITH (COMPAT) を使用してコンパイルした場合、外部浮動小数点数は長精度 (64 ビット) の浮動小数点形式に変換されます。 ただし、ARITH (EXTEND) を使用してコンパイルすれば、外部浮動小数点数は拡張精度 (128 ビット) の浮動小数点形式に変換されます。

2 進数 (COMP) 項目

BINARY、COMP、および COMP-4 は同義語です。 2 進数形式の数値は、2、4、または 8 バイトのストレージを占めます。 PICTURE 文節で項目が符号付きであることが指定されている場合は、左端のビットが演算符号として使用されます。

2 進数は、付随する PICTURE 記述が 4 個以下の 10 進数字であれば 2 バイトを占め、5 個から 9 個の 10 進数字であれば 4 バイトを占め、10 個から 18 個の 10 進数字であれば 8 バイトを占めます。 9 桁以上の 2 進数項目には、コンパイラーによる余分な処理が必要となります。

2 進数項目には、例えば、指標、添え字、スイッチ、および算術オペランドや結果を入れることができます。

2 進データ (BINARY、COMP、または COMP-4) の切り捨て方法を指定するには、TRUNC(STD|OPT|BIN) コンパイラー・オプションを使用してください。

固有 2 進数 (COMP-5) 項目

USAGE COMP-5 として定義したデータ項目は、ストレージ内では 2 進データとして表されます。しかし、これらは、USAGE COMP 項目とは異なり、PICTURE 節の 9 の数で暗黙指定される値に制限されるのではなく、固有 2 進数表現 (2、4、または 8 バイト) の容量までの大きさの値を含むことができます。

数値データを COMP-5 項目に移動または保管すると、COBOL PICTURE サイズ制限ではなく、2 進数フィールド・サイズで切り捨てが行われます。 COMP-5 項目を参照する場合、演算では完全な 2 進数フィールド・サイズが使用されます。

したがって、COMP-5 は、データが COBOL PICTURE 節に適合しない可能性のある非 COBOL プログラムから生じる 2 進データ項目の場合に特に有用です。

次の表は、COMP-5 データ項目に指定可能な値の範囲を示しています。

表 *3. **COMP-5** データ項目の値の範囲*

PICTURE	**ストレージ表現**	**数値**
S9(1) から S9(4)	2 進数ハーフワード (2 バイト)	-32768 から +32767
S9(5) から S9(9)	2 進数フルワード (4 バイト)	-2,147,483,648 から +2,147,483,647
S9(10) から S9(18)	2 進数ダブルワード (8 バイト)	-9,223,372,036,854,775,808 から +9,223,372,036,854,775,807
9(1) から 9(4)	2 進数ハーフワード (2 バイト)	0 から 65535
9(5) から 9(9)	2 進数フルワード (4 バイト)	0 から 4,294,967,295

表 3. ***COMP-5* データ項目の値の範囲** (続き)		
PICTURE	**ストレージ表現**	**数値**
9(10) から 9(18)	2 進数ダブルワード (8 バイト)	0 から 18,446,744,073,709,551,615

COMP-5 項目の PICTURE 節に、スケーリング (すなわち、小数部の桁数または暗黙の整数桁数) を指定することができます。 その場合は、上記にリストされた最大容量とほぼ同じ大きさをスケーリングする必要があります。 例えば、PICTURE S99V99 COMP-5 として記述したデータ項目は、ストレージ内では 2 進数ハーフワードとして表され、-327.68 から +327.67 までの範囲の値をサポートします。

VALUE 節のラージ・リテラル: COMP-5 項目の VALUE 節に指定されたリテラルは、一部の例外を除いて、固有 2 進数表現の容量までの大きさの値を含むことができます。 例外については、*COBOL for Linux on x86* 言語解説書を参照してください。

TRUNC コンパイラー・オプションの設定に関係なく、COMP-5 データ項目は、TRUNC(BIN) でコンパイルされたプログラムでは、2 進データのように動作します。

パック 10 進数 (COMP-3) 項目

PACKED-DECIMAL と COMP-3 は同義語です。 パック 10 進数項目は、PICTURE 記述でコーディングされる 2 つの 10 進数字ごとに 1 バイトのストレージを占めます。ただし、右端のバイトだけが例外で、右端のバイトには 1 つの数字と符号が入ります。 この形式が最も効率的に使用されるのは、PICTURE 記述で奇数の桁をコーディングして、左端のバイトが完全に使用されるようにするときです。 パック 10 進数項目は、算術演算の目的では固定小数点数として扱われます。

内部浮動小数点 (COMP-1 および COMP-2) 項目

COMP-1 は短精度浮動小数点形式を指し、COMP-2 は長精度浮動小数点形式を指します。これらの形式は、それぞれ 4 バイトと 8 バイトのストレージを占めます。

COMP-1 および COMP-2 データ項目は、FLOAT(NATIVE) コンパイラー・オプション (デフォルト) が有効である場合、IEEE 形式で表されます。 FLOAT(BE) が有効な場合は、COMP-1 および COMP-2 データ項目が一貫して System z®、つまり 16 進数の浮動小数点形式で表わされます。詳細については、FLOAT オプションに関する関連参照を参照してください。

関連概念
180 ページの『Unicode および言語文字のエンコード』
531 ページの『付録 C 中間結果および算術精度』

関連タスク
35 ページの『数値データの定義』
187 ページの『国別数値データ項目の定義』

関連参照
193 ページの『文字データの保管』
286 ページの『TRUNC』
269 ページの『FLOAT』
データのクラスおよびカテゴリー (*COBOL for Linux on x86* 言語解説書)
SIGN 節 (*COBOL for Linux on x86* 言語解説書)
VALUE 節 (*COBOL for Linux on x86* 言語解説書)

例: 数値データおよび内部表現

次の表は、数値項目の内部表現を示しています。

次の表は、バイナリー・データ・タイプの数値項目の内部表現を示しています。

表 *4.* ***2*** **進数値項目の内部表現**

数値タイプ	**PICTURE および USAGE 節とオプションの SIGN 節**	**値**	**内部表現**
2 進数	PIC S9999 BINARY PIC S9999 COMP PIC S9999 COMP-4	+ 1234	D2 04
		- 1234	2E FB
	PIC S9999 COMP-5	+ 12345[1]	39 30
		- 12345[1]	C7 CF
	PIC 9999 BINARY PIC 9999 COMP PIC 9999 COMP-4	1234	D2 04
	PIC 9999 COMP-5	60000[1]	60 EA

1. この例では、COMP-5 データ項目に含めることのできる値が、PICTURE 節の 9 の数によって暗黙指定された値に制限されるのではなく、固有 2 進数表現 (2、4、または 8 バイト) の容量までの大きさの値を入れることができることを示しています。

次の表は、固有データ形式の数値項目の内部表現を示しています。 CHAR(NATIVE) および FLOAT(NATIVE) コンパイラー・オプションが有効であるとします。

表 5. **固有数値項目の内部表現**

数値タイプ	**PICTURE および USAGE 節とオプションの SIGN 節**	**値**	**内部表現**
外部 10 進数	PIC S9999 DISPLAY	+ 1234	31 32 33 34
		- 1234	31 32 33 74
		1234	31 32 33 34
	PIC 9999 DISPLAY	1234	31 32 33 34
	PIC 9999 NATIONAL	1234	31 00 32 00 33 00 34 00
	PIC S9999 DISPLAY SIGN LEADING	+ 1234	31 32 33 34
		- 1234	71 32 33 34
	PIC S9999 DISPLAY SIGN LEADING SEPARATE	+ 1234	2B 31 32 33 34
		- 1234	2D 31 32 33 34
	PIC S9999 DISPLAY SIGN TRAILING SEPARATE	+ 1234	31 32 33 34 2B
		- 1234	31 32 33 34 2D
	PIC S9999 NATIONAL SIGN LEADING SEPARATE	+ 1234	2B 00 31 00 32 00 33 00 34 00
		- 1234	2D 00 31 00 32 00 33 00 34 00
	PIC S9999 NATIONAL SIGN TRAILING SEPARATE	+ 1234	31 00 32 00 33 00 34 00 2B 00
		- 1234	31 00 32 00 33 00 34 00 2D 00
内部 10 進数	PIC S9999 PACKED-DECIMAL PIC S9999 COMP-3	+ 1234	01 23 4C
		- 1234	01 23 4D
	PIC 9999 PACKED-DECIMAL PIC 9999 COMP-3	1234	01 23 4C
内部浮動小数点	COMP-1	+ 1234	00 40 9A 44
		- 1234	00 40 9A C4
	COMP-2	+ 1234	00 00 00 00 00 48 93 40
		- 1234	00 00 00 00 00 48 93 C0

表 5. **固有数値項目の内部表現** (続き)			
数値タイプ	**PICTURE** および **USAGE** 節とオプションの **SIGN** 節	**値**	**内部表現**
外部浮動小数点	PIC +9(2).9(2)E+99 DISPLAY	+ 12.34E+02	2B 31 32 2E 33 34 45 2B 30 32
		- 12.34E+02	2D 31 32 2E 33 34 45 2B 30 32
	PIC +9(2).9(2)E+99 NATIONAL	+ 12.34E+02	2B 00 31 00 32 00 2E 00 33 00 34 00 45 00 2B 00 30 00 32 00
		- 12.34E+02	2D 00 31 00 32 00 2E 00 33 00 34 00 45 00 2B 00 30 00 32 00

次の表は、IBM Z® ホスト・データ形式の数値項目の内部表現を示しています。 CHAR(EBCDIC) および FLOAT(BE) コンパイラー・オプションが有効であるとします。

表 6. ***CHAR(EBCDIC)*** **および** ***FLOAT(BE)*** **が有効であるときの数値項目の内部表現**

数値タイプ	**PICTURE** および **USAGE** 節とオプションの **SIGN** 節	値	内部表現
外部 10 進数	PIC S9999 DISPLAY	+ 1234	F1 F2 F3 C4
		- 1234	F1 F2 F3 D4
		1234	F1 F2 F3 C4
	PIC 9999 DISPLAY	1234	F1 F2 F3 F4
	PIC 9999 NATIONAL	1234	00 31 00 32 00 33 00 34
	PIC S9999 DISPLAY SIGN LEADING	+ 1234	C1 F2 F3 F4
		- 1234	D1 F2 F3 F4
	PIC S9999 DISPLAY SIGN LEADING SEPARATE	+ 1234	4E F1 F2 F3 F4
		- 1234	60 F1 F2 F3 F4
	PIC S9999 DISPLAY SIGN TRAILING SEPARATE	+ 1234	F1 F2 F3 F4 4E
		- 1234	F1 F2 F3 F4 60
	PIC S9999 NATIONAL SIGN LEADING SEPARATE	+ 1234	00 2B 00 31 00 32 00 33 00 34
		- 1234	00 2D 00 31 00 32 00 33 00 34
	PIC S9999 NATIONAL SIGN TRAILING SEPARATE	+ 1234	00 31 00 32 00 33 00 34 00 2B
		- 1234	00 31 00 32 00 33 00 34 00 2D
内部 10 進数	PIC S9999 PACKED-DECIMAL PIC S9999 COMP-3	+ 1234	01 23 4C
		- 1234	01 23 4D
	PIC 9999 PACKED-DECIMAL PIC 9999 COMP-3	1234	01 23 4C
内部浮動小数点	COMP-1	+ 1234	43 4D 20 00
		- 1234	C3 4D 20 00
	COMP-2	+ 1234	43 4D 20 00 00 00 00 00
		- 1234	C3 4D 20 00 00 00 00 00

表 6. *CHAR(EBCDIC)* および *FLOAT(BE)* が有効であるときの数値項目の内部表現 (続き)			
数値タイプ	**PICTURE** および **USAGE** 節と オプションの **SIGN** 節	**値**	**内部表現**
外部浮動小数点	PIC +9(2).9(2)E+99 DISPLAY	+ 12.34E+02	4E F1 F2 4B F3 F4 C5 4E F0 F2
		- 12.34E+02	60 F1 F2 4B F3 F4 C5 4E F0 F2
	PIC +9(2).9(2)E+99 NATIONAL	+ 12.34E+02	00 2B 00 31 00 32 00 2E 00 33 00 34 00 45 00 2B 00 30 00 32
		- 12.34E+02	00 2D 00 31 00 32 00 2E 00 33 00 34 00 45 00 2B 00 30 00 32

データ形式の変換

プログラム内のコードがデータ形式の異なる項目の相互作用を含んでいると、コンパイラーはそれらの項目を一時的に (比較および算術演算の場合) または永続的に (MOVE、COMPUTE、またはその他の算術ステートメントの受け取り側への割り当ての場合) 変換します。

変換とは、実際には、値をあるデータ項目から別のデータ項目に移動することです。 コンパイラーは、MOVE および COMPUTE ステートメントについて使用されるのと同じ規則を使用して、比較および算術の実行時に必要とされる変換を実行します。

可能であれば、コンパイラーは、直接的な 1 桁ずつの移動ではなく、数値を保持する移動を実行します。

変換には、通常、追加のストレージと処理時間が必要とされます。これは、演算が実行される前に、データが内部作業域に移動され、変換されるためです。 さらに、結果を作業域に戻し、再度変換することが必要な場合もあります。

固定小数点データ形式 (外部 10 進数、パック 10 進数、または 2 進数) 間の変換は、ターゲット・フィールドがソース・オペランドのすべての桁を含むことができれば、精度が失われることなく完了します。

固定小数点データ形式と浮動小数点データ形式 (短精度浮動小数点、長精度浮動小数点、または外部浮動小数点) 間の変換では、精度が失われる可能性があります。 このような変換は、固定小数点と浮動小数点の両方のオペランドが混在する算術計算時に起こります。

関連参照
46 ページの『変換および精度』
47 ページの『ゾーンおよびパック 10 進数データのサイン表記』

変換および精度

数値変換によっては精度が低下する場合があるほか、精度が保持されたり、丸めが行われる場合もあります。

固定小数点項目と外部浮動小数点項目は、どちらも 10 進数の特性を持つため、以下の例での固定小数点項目への参照は、特に明記しない限り、外部浮動小数点項目への参照を含みます。

コンパイラーが固定小数点形式を内部浮動小数点形式に変換するときには、基数 10 の固定小数点数は、内部で使用される数体系に変換されます。

コンパイラーが比較のために短精度形式を長精度形式に変換するときには、短精度数値の埋め込みにはゼロが使用されます。

精度が低下する変換

USAGE COMP-2 データ項目が 18 桁を超える固定小数点データ項目に移動されるときには、固定小数点データ項目は有効数字を 18 個だけ受け取り、残りの桁は 0 になります。

USAGE COMP-1 データ項目が 6 桁を超える固定小数点データ項目に移動されるときには、固定小数点データ項目は有効数字を 6 個だけ受け取り、残りの桁は 0 になります。

精度を保つ変換

6 桁以下の固定小数点データ項目が USAGE COMP-1 データ項目に移動され、その後で固定小数点データ項目に戻される場合は、元の値がリカバリーされます。

USAGE COMP-1 データ項目が 6 桁以上の固定小数点データ項目に移動され、その後で USAGE COMP-1 データ項目に戻される場合は、元の値がリカバリーされます。

15 桁以下の固定小数点データ項目が USAGE COMP-2 データ項目に移動され、その後で固定小数点データ項目に戻される場合は、元の値がリカバリーされます。

USAGE COMP-2 データ項目が 18 桁以上の固定小数点 (外部浮動小数点ではなく) データ項目に移動され、その後で USAGE COMP-2 データ項目に戻される場合は、元の値がリカバリーされます。

丸めを生じさせる変換

USAGE COMP-1 データ項目、USAGE COMP-2 データ項目、外部浮動小数点データ項目、または浮動小数点リテラルが固定小数点データ項目に移動されると、ターゲット・データ項目の低位桁で丸めが起こります。0.5 以上の小数値は切り上げられます。0.5 未満の小数値は切り捨てられます。

USAGE COMP-2 データ項目が USAGE COMP-1 データ項目に移動されると、ターゲット・データ項目の低位桁で丸めが起こります。

固定小数点データ項目の PICTURE に外部浮動小数点データ項目の PICTURE よりも多くの桁位置が含まれている場合に、固定小数点データ項目が外部浮動小数点データ項目に移動されると、ターゲット・データ項目の低位桁で丸めが起こります。

関連概念
531 ページの『付録 C 中間結果および算術精度』

ゾーンおよびパック 10 進数データのサイン表記

サイン表記は、ゾーン 10 進数データおよび内部 10 進数データの処理や相互作用に影響します。

X'*sd*' (ここで *s* はサイン表記であり、*d* は数字を表します) が与えられた場合、SIGN IS SEPARATE 節を持たない ゾーン 10 進数 (USAGE DISPLAY) データの有効なサイン表記は次のとおりです。

正:
3、C、および F

負:
7 および D

CHAR(NATIVE) コンパイラー・オプションが有効であると、内部生成される符号は、正および符号なしの場合で 3、負の場合で 7 になります。

CHAR(EBCDIC) コンパイラー・オプションが有効であると、内部生成される符号は、正の場合で C、符号なしの場合で F、負の場合で D になります。

X'*ds*' (*d* は数字を表し、*s* はサイン表記です) が与えられた場合、内部 10 進数 (USAGE PACKED-DECIMAL) データの有効なサイン表記は次のとおりです。

正:
A、C、E、および F

負:
B および D

内部生成される符号は、正および符号なしの場合で C、負の場合で D になります。

符号なし内部 10 進数のサイン表記は、COBOL for Linux とホスト COBOL とでは異なります。 ホスト COBOL は、符号なし内部 10 進数の符号として F を内部生成します。

関連参照
292 ページの『ZWB』
521 ページの『データ表現』

非互換データの検査 (数値のクラス・テスト)

コンパイラーは、データ項目に指定された値を PICTURE および USAGE 節に有効であると見なし、それらの値の妥当性検査を行いません。 データ項目を後続の処理で使用する前に、その内容が PICTURE および USAGE 節に適合していることを確認してください。

値がプログラムに渡され、それらの値に対する互換性のないデータ記述を持つ項目に割り当てられることがよくあります。 例えば、非数値データが、数値として定義されたフィールドに移動されたり、渡されたりすることがあります。また、符号付き数値が、符号なしとして定義されたフィールドに渡されることもあります。 いずれの場合も、受け取り側フィールドには無効なデータが含まれます。 項目にそれのデータ記述と互換性のない値が与えられると、PROCEDURE DIVISION 内でのその項目への参照が未定義になり、結果が予測できなくなります。

数値のクラス・テストを使用して、データ妥当性検査を行うことができます。 次に例を示します。

```
Linkage Section.
01  Count-x   Pic 999.
. . .
Procedure Division Using Count-x.
    If Count-x is numeric then display "Data is good"
```

数値のクラス・テストでは、データ項目の内容が、そのデータ項目の PICTURE および USAGE にとって有効な値のセットと照合されます。

算術の実行

いくつかの COBOL 言語機能 (COMPUTE、算術式、数値組み込み関数、日時呼び出し可能サービスを含む) のいずれかを使用して、算術を実行できます。 どれを選択するかは、特定の必要を機能が満たすかどうかによって違ってきます。

ほとんどの一般的な算術計算の場合、COMPUTE ステートメントが適切です。 数値リテラル、数値データ、または算術演算子を使用する必要がある場合は、算術式を使用できます。 数値表現が許可されている場合には、数値組み込み関数を使用すれば時間を節約できます。

関連タスク
49 ページの『COMPUTE およびその他の算術ステートメントの使用』
49 ページの『算術式の使用』
50 ページの『数字組み込み関数の使用』

COMPUTE およびその他の算術ステートメントの使用

ほとんどの算術計算では、ADD、SUBTRACT、MULTIPLY、および DIVIDE ステートメントではなく、COMPUTE ステートメントが使用されます。 いくつかの個々の算術ステートメントの代わりに、1 つの COMPUTE ステートメントをコーディングするだけでよいことがよくあります。

COMPUTE ステートメントは、算術式の結果を 1 つ以上のデータ項目に割り当てます。

```
Compute z     = a + b / c ** d - e
Compute x y z = a + b / c ** d - e
```

COMPUTE 以外の算術ステートメントを使用した算術計算の中には、いっそう直感的なものがあります。 次に例を示します。

COMPUTE	等価の算術ステートメント
`Compute Increment = Increment + 1`	`Add 1 to Increment`
`Compute Balance =` `  Balance - Overdraft`	`Subtract Overdraft from Balance`

`Compute IncrementOne =`
`  IncrementOne + 1`
`Compute IncrementTwo =`
`  IncrementTwo + 1`
`Compute IncrementThree =`
`  IncrementThree + 1`
	`Add 1 to IncrementOne,`
	`  IncrementTwo,`
	`  IncrementThree`

さらに、剰余を処理したい除算については、DIVIDE ステートメント (REMAINDER 句を指定した) を使用したい場合もあります。 REM 組み込み関数も、剰余を処理する機能を提供します。

算術計算を実行するとき、ゾーン 10 進数データ項目を使用するのと同様に、国別 10 進数データ項目をオペランドとして使用できます。 また、表示浮動小数点オペランドを使用するのと同様に、国別浮動小数点データ項目を使用することもできます。

関連概念
53 ページの『固定小数点演算と浮動小数点演算の対比』
531 ページの『付録 C 中間結果および算術精度』

関連タスク
35 ページの『数値データの定義』

算術式の使用

数値データ項目が許可されているステートメント内の多くの場所で、算術式を使用できます (ただし、どの場所でも使用できるわけではありません)。

例えば、算術式を比較条件の被比較数として使用することができます。

```
If (a + b) > (c - d + 5) Then. . .
```

算術式は、単一の数字リテラル、単一の数値データ項目、または単一の組み込み関数参照で構成することができます。 また、これらの項目のいくつかを算術演算子で結合して構成することもできます。

算術演算子は、次の優先順位に従って評価されます。

表 7. **算術演算子の評価の順序**		
演算子	**意味**	**評価の順序**
単項 + または -	代数符号	1 番目
**	指数	2 番目
/ または *	除算または乗算	3 番目
2 項 + または -	加算または減算	最後

優先順位が同じレベルの演算子は、左から右へと評価されます。ただし、演算子とともに括弧を使用して、それらが評価される順序を変更することができます。 括弧の中の式は、個々の演算子が評価される前に評価されます。 必要であるかどうかにかかわらず、括弧を使用するとプログラムが読みやすくなります。

関連概念
53 ページの『固定小数点演算と浮動小数点演算の対比』
531 ページの『付録 C 中間結果および算術精度』

数字組み込み関数の使用

数字組み込み関数は、数式を使用できる場所でのみ使用することができます。 これらの関数を使用すると、時間の節約に役立ちます。これは、これらの関数が取り扱う多種類の一般的な計算をコーディングする必要がなくなるためです。

数字組み込み関数は符号付き数値を戻し、一時数値データ項目として扱われます。

数字関数は、以下のカテゴリーに分類されます。

整数
整数を戻すもの。

浮動小数点
長精度 (64 ビット) または拡張精度 (128 ビット) の浮動小数点値を戻すもの (これは、デフォルト・オプション ARITH(COMPAT) を使用してコンパイルするか、ARITH(EXTEND) を使用してコンパイルするかによって決まります)。

混合
引数によって、整数、浮動小数点値、または 小数部の桁がある固定小数点数を戻すもの。

組み込み関数を使用すると、次の表に概説されているようなさまざまな種類の算術演算を実行することができます。

表 8. **数字組み込み関数**

数値処理	日付および時刻	金融	数学	統計
LENGTH MAX MIN NUMVAL NUMVAL-C ORD-MAX ORD-MIN	ADD-DURATION CONVERT-DATE-TIME CURRENT-DATE DATE-OF-INTEGER DATE-TO-YYYYMMDD DATEVAL DAY-OF-INTEGER DAY-TO-YYYYDDD EXTRACT-DATE-TIME FIND-DURATION INTEGER-OF-DATE INTEGER-OF-DAY UNDATE SUBTRACT-DURATION TEST-DATE-TIME WHEN-COMPILED YEAR-TO-YYYY YEARWINDOW	ANNUITY PRESENT-VALUE	ACOS ASIN ATAN COS FACTORIAL INTEGER INTEGER-PART LOG LOG10 MOD REM SIN SQRT SUM TAN	MEAN MEDIAN MIDRANGE RANDOM RANGE STANDARD-DEVIATION VARIANCE

51 ページの『例: 数字組み込み関数』

ある関数を別の関数の引数として参照することができます。 ネストされた関数は、外側の関数からは独立して評価されます (ただし、コンパイラーが混合関数を固定小数点命令と浮動小数点命令のどちらを使用して評価すべきかを判別するときは例外です)。

算術式を数字関数への引数としてネストすることもできます。 例えば、次の例には、3 つの関数引数 (a、b、および算術式 (c / d)) がありません。

```
Compute x = Function Sum(a b (c / d))
```

ALL 添え字を使用すると、あるテーブル (または配列) のすべてのエレメントを関数の引数として参照することができます。

また、整数タイプの特殊レジスターは、整数の引数が使用できるところであればどこでも引数として使用することができます。

関連概念
53 ページの『固定小数点演算と浮動小数点演算の対比』
531 ページの『付録 C 中間結果および算術精度』

関連参照

253 ページの『ARITH』

例: 数字組み込み関数

以下の例と付随する説明では、それぞれのカテゴリーごとに組み込み関数を示します。

以下の例がゾーン 10 進数データ項目を示している場合、代わりに国別 10 進数項目を使用できます。 (ただし、符号付き国別 10 進数項目の場合は、SIGN SEPARATE 節が有効でなければなりません。)

一般数値処理

2 つの価格 (ドル記号付きの英数字項目として以下に示されています) のうちの最大値を見つけ、その値を出力レコードの数値フィールドに入れ、それから出力レコードの長さを判別したいとしましょう。 そのためには、NUMVAL-C (英数字または国別リテラルあるいは英数字または国別データ項目の、数値を戻す関数)、および MAX 関数と LENGTH 関数を使用できます。

```
01  X                   Pic 9(2).
01  Price1              Pic x(8)   Value "$8000".
01  Price2              Pic x(8)   Value "$2000".
01  Output-Record.
    05  Product-Name    Pic x(20).
    05  Product-Number  Pic 9(9).
    05  Product-Price   Pic 9(6).
. . .
Procedure Division.
    Compute Product-Price =
      Function Max (Function Numval-C(Price1) Function Numval-C(Price2))
    Compute X = Function Length(Output-Record)
```

さらに、Product-Name の内容が大文字になるようにするために、次のステートメントを使用することができます。

```
Move Function Upper-case (Product-Name) to Product-Name
```

日付および時刻

次の例は、今日から 90 日後の満期日を計算する方法を示しています。 CURRENT-DATE 関数から戻される最初の 8 文字は、日付を 4 桁の年、2 桁の月、および 2 桁の日という形式 (YYYYMMDD) で表します。 この日付がその整数値に変換されます。そのあと、この値に 90 が追加され、整数が YYYYMMDD 形式に再度変換されます。

```
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
```

金融

ビジネス投資の判断では、計画された投資の利益率を評価するために、予期される将来の現金流入の現在価格を計算することがしばしば必要になります。 将来の特定の時期に受け取ることが期待される金額の現在価格は、今日特定の利率で投資された場合に、累積されてその将来の金額になるであろう金額です。

例えば、計画された $1,000 の投資で、次の 3 年間にわたり、それぞれ年 1 回の支払いで $100、$200、および $300 の支払いの流れになるとします。 次の COBOL ステートメントは、10% の利率でこれらの現金流入の現在の値を計算する方法を示しています。

```
01  Series-Amt1      Pic 9(9)V99       Value 100.
01  Series-Amt2      Pic 9(9)V99       Value 200.
01  Series-Amt3      Pic 9(9)V99       Value 300.
01  Discount-Rate    Pic S9(2)V9(6)    Value .10.
01  Todays-Value     Pic 9(9)V99.
. . .
    Compute Todays-Value =
      Function
        Present-Value(Discount-Rate Series-Amt1 Series-Amt2 Series-Amt3)
```

ANNUITY 関数は、ローンの元金および利息を返済するために必要な分割払いの支払金 (年賦金) の金額を判断することが必要とされるビジネス問題で使用できます。 一連の支払いの特徴は、各期間の長さと、期間ごとの金額および利率が一定であるということです。 次の例は、$15,000 のローンを 12% の年利で 3 年

間で返済するのに必要な月賦金額を計算する方法を示しています (36 か月払い、1 か月当たりの利率 = .12/12)。

```
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)
```

数学

次の COBOL ステートメントでは、組み込み関数をネストし、算術式を引数として使用し、前の複雑な計算を簡単に行う方法を示しています。

```
Compute Z = Function Log(Function Sqrt (2 * X + 1)) + Function Rem(X 2)
```

ここでは、組み込み関数 REM (REMAINDER 節を指定した DIVIDE ステートメントではなく) が、X を 2 で割った剰余を加数に戻します。

統計

組み込み関数を使用すると、統計情報の計算が簡単になります。 さまざまな市民税を分析していて、平均値、中央値、および範囲 (最高税額と最低税額の差) を計算したいとします。

```
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)
```

関連タスク
105 ページの『数値への変換 (NUMVAL、NUMVAL-C)』

固定小数点演算と浮動小数点演算の対比

プログラム内の算術計算では (それが算術ステートメント、組み込み関数、式、または相互にネストされたこれらの組み合わせのいずれであっても)、算術計算のコーディング方法によって、浮動小数点演算になるか、固定小数点演算になるかが決まります。

プログラム内の多くのステートメントには、算術計算が伴うことがあります。 例えば、以下のそれぞれの COBOL ステートメントには、ある種の算術計算が必要です。

- 一般算術計算

```
compute report-matrix-col = (emp-count ** .5) + 1
add report-matrix-min to report-matrix-max giving report-matrix-tot
```

- 式および関数

```
compute report-matrix-col = function sqrt(emp-count) + 1
compute whole-hours       = function integer-part((average-hours) + 1)
```

- 算術比較

```
if report-matrix-col <     function sqrt(emp-count) + 1
if whole-hours       not = function integer-part((average-hours) + 1)
```

浮動小数点計算

通常、算術計算に以下のいずれかの特性がある場合、それは浮動小数点演算で評価されます。

- オペランドまたは結果フィールドが浮動小数点である。

  オペランドは、浮動小数点リテラルとしてコーディングするか、あるいは USAGE COMP-1、USAGE COMP-2、または外部浮動小数点 (浮動小数点 PICTURE を指定した USAGE DISPLAY または USAGE NATIONAL) として定義されたデータ項目としてコーディングした場合に浮動小数点になります。

  オペランドがネストされた算術式である場合、または数字組み込み関数への参照である場合、そのオペランドは次の条件のいずれかが当てはまるとき、浮動小数点演算になります。

  – 算術式内の引数が浮動小数点になる。
  – 関数が浮動小数点関数である。
  – 関数が 1 つ以上の浮動小数点引数を持つ混合関数である。
- 指数に小数部の桁が含まれている。

  指数は、小数部の桁を含むか (小数部の桁を含むリテラルを使用する場合)、小数部の桁を含む PICTURE を項目に与えるか、あるいは結果が小数部の桁を持つ算術式または関数を使用します。

算術式または数字関数は、オペランドまたは引数 (除数および指数を除く) に小数部の桁がある場合には、小数部の桁がある結果をもたらします。

固定小数点計算

一般に、算術演算に浮動小数点に関する上記のどの特性も含まれていない場合、コンパイラーはそれを固定小数点演算で評価します。 すなわち、算術計算が固定小数点として処理されるのは、すべてのオペランドが固定小数点で、結果フィールドが固定小数点と定義されており、しかもどの指数も小数部の桁がある値を表さない場合だけです。 また、ネストされた算術式および関数参照も、固定小数点値を表さなければなりません。

算術比較 (比較条件)

関係演算子を使用して数式を比較する場合、数式 (それらがデータ項目、算術式、関数参照、またはこれらの組み合わせのいずれであっても) は、計算全体のコンテキストでは、被比較数です。 すなわち、それぞれの属性が互いの計算に影響を与える可能性があり、両方の式が固定小数点で評価されるか、または両方の式が浮動小数点で評価されることになります。 これは、簡略比較にも当てはまります (比較の中で一方の被比較数が明示的に指定されない場合でも)。 次に例を示します。

```
if (a + d) = (b + e) and c
```

このステートメントには、(a + d) = (b + e) と (a + d) = c の 2 つの比較があります。(a + d) は、2 番目の比較では明示的に指定されていませんが、その比較の被比較数です。 したがって、c の属性が (a + d) の計算に影響を与える可能性があります。

比較演算 (および比較の中にネストされた算術式の評価) は、一方の被比較数が浮動小数点値であるかまたは結果が浮動小数点値になる場合、コンパイラーによって浮動小数点演算として処理されます。

比較演算 (および比較の中にネストされた算術式の評価) は、両方の被比較数が固定小数点値であるかまたは結果が固定小数点値になる場合、コンパイラーによって固定小数点演算として処理されます。

暗黙的な比較 (関係演算子が使用されない) は、単位として処理されません。 ただし、2 つの被比較数は、浮動小数点演算または固定小数点演算での評価に関しては、個別に扱われます。 以下の例では、実際には、それぞれの属性に関係なく評価され、その後で相互に比較される 5 つの算術式があります。

```
evaluate (a + d)
    when (b + e) thru c
    when (f / g) thru (h * i)
    . . .
end-evaluate
```

55 ページの『例: 固定小数点計算および浮動小数点計算』

関連参照
539 ページの『非算術ステートメントの算術式』

例: 固定小数点計算および浮動小数点計算

次の例は、固定小数点演算と浮動小数点演算を使用して評価されるステートメントを示しています。

従業員表のデータ項目を次のように定義すると仮定します。

```
01  employee-table.
    05  emp-count         pic  9(4).
    05  employee-record occurs 1 to 1000 times
            depending on emp-count.
        10 hours           pic +9(5)ve+99.
. . .
01  report-matrix-col     pic  9(3).
01  report-matrix-min     pic  9(3).
01  report-matrix-max     pic  9(3).
01  report-matrix-tot     pic  9(3).
01  average-hours         pic  9(3)v9.
01  whole-hours           pic  9(4).
```

以下のステートメントは、浮動小数点演算を使用して評価されます。

```
compute report-matrix-col = (emp-count ** .5) + 1
compute report-matrix-col = function sqrt(emp-count) + 1
if report-matrix-tot < function sqrt(emp-count) + 1
```

以下のステートメントは、固定小数点演算を使用して評価されます。

```
add report-matrix-min to report-matrix-max giving report-matrix-tot
compute report-matrix-max =
    function max(report-matrix-max report-matrix-tot)
if whole-hours not = function integer-part((average-hours) + 1)
```

通貨記号の使用

多くのプログラムでは金融情報を処理する必要があり、それらの情報を適切な通貨記号で出力表示する必要があります。 COBOL 通貨サポート (およびご使用のプリンターやディスプレイ装置に合ったコード・ページ) を使用するなら、プログラムで幾つかの通貨記号を使用できます。

以下の記号の 1 つ以上を使用できます。

- ドル記号 ($) のような記号
- 複数文字からなる通貨記号 (USD または EUR など)
- 欧州経済通貨同盟 (EMU) によって確立されているユーロ記号

金融情報を表示するための記号を指定するには、それらの記号に関連付けられる PICTURE 文字を指定した CURRENCY SIGN 節を (CONFIGURATION SECTION の SPECIAL-NAMES 段落の中で) 使用してください。次の例で、PICTURE 文字 $ は、通貨記号として $US を使用することを示しています。

```
    Currency Sign is "$US" with Picture Symbol "$".
    . . .
77  Invoice-Amount      Pic $$,$$9.99.
. . .
    Display "Invoice amount is " Invoice-Amount.
```

この例で、Invoice-Amount に 1500.00 が含まれている場合は、次のように出力されます。

```
Invoice amount is  $US1,500.00
```

プログラム内で複数の CURRENCY SIGN 節を使用することにより、複数の通貨記号を表示することができます。

16 進リテラルを使用して通貨記号の値を表すことができます。 ソース・プログラムのデータ入力方法では、対象とする文字を簡単に入力できない場合、16 進数リテラルを使用すると役立つことがあります。 次の例は、通貨記号として使用される 16 進値 X'80' を示しています。

```
    Currency Sign X'80' with Picture Symbol 'U'.
    . . .
01  Deposit-Amount       Pic UUUUU9.99.
```

キーボード上にユーロ記号に相当する文字がない場合は、それを CURRENCY SIGN 節で 16 進値として指定する必要があります。

ユーロ記号の 16 進値は、コード・ページ 1252 (Latin 1) の場合、X'80' です。

関連参照
260 ページの『CURRENCY』
CURRENCY SIGN 節 (*COBOL for Linux on x86* 言語解説書)

例: 複数の通貨符号

次の例は、ユーロ通貨 (EUR) とスイスのフラン (CHF) の両方で値を表示する方法を示すものです。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EuroSamp.
Environment Division.
Configuration Section.
Special-Names.
    Currency Sign is "CHF "  with Picture Symbol "F"
    Currency Sign is "EUR "  with Picture Symbol "U".
Data Division.
WORKING-STORAGE SECTION.
01  Deposit-in-Euro        Pic S9999V99 Value 8000.00.
01  Deposit-in-CHF         Pic S99999V99.
01  Deposit-Report.
    02  Report-in-Franc    Pic -FFFFF9.99.
    02  Report-in-Euro     Pic -UUUUU9.99.
01  EUR-to-CHF-Conv-Rate   Pic 9V99999  Value 1.53893.
. . .
PROCEDURE DIVISION.
Report-Deposit-in-CHF-and-EUR.
    Move Deposit-in-Euro to Report-in-Euro
    Compute Deposit-in-CHF Rounded
          = Deposit-in-Euro * EUR-to-CHF-Conv-Rate
      On Size Error
        Perform Conversion-Error
      Not On Size Error
        Move Deposit-in-CHF to Report-in-Franc
        Display "Deposit in euro  = " Report-in-Euro
```

```
         Display "Deposit in franc = " Report-in-Franc
    End-Compute
    Goback.
Conversion-Error.
           Display "Conversion error from EUR to CHF"
           Display "Euro value: " Report-in-Euro.
```

上記の例は、次のような表示出力を作成します。

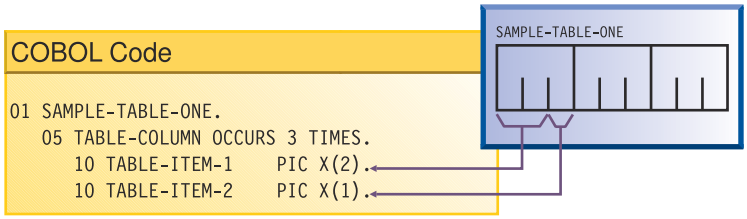
```


Deposit in euro  =  EUR 8000.00
Deposit in franc = CHF 12311.44
```

この例で使用されている交換レートは例として使われているだけです。

第 4 章 テーブルの処理

テーブル は、合計額や月平均額のような同じデータ記述を持つデータ項目の集合です。 テーブルは、テーブル名とテーブル・エレメント と呼ばれる従属項目から成ります。 テーブルは、配列に相当する COBOL 用語です。



上記の例では、SAMPLE-TABLE-ONE が、テーブルを含むグループ項目です。 TABLE-COLUMN は、3 回出現する 1 次元テーブルのテーブル・エレメントを指しています。

反復項目を DATA DIVISION に別個の連続する項目として定義するのではなく、DATA DIVISION 記入項目の OCCURS 節を使用してテーブルを定義します。 この方法には、次のような利点があります。

- コードは、項目 (テーブル・エレメント) の単位を明確に示します。
- 添え字と指標を使用してテーブル・エレメントを参照することができます。
- データ項目の反復が容易です。

テーブルは、プログラムの処理速度、特にレコードを探索するプログラムの速度を高めるうえで大切なものです。

関連概念
73 ページの『複合 OCCURS DEPENDING ON』

関連タスク
59 ページの『テーブルの定義 (OCCURS)』
61 ページの『テーブルのネスト』
62 ページの『テーブル内の項目の参照』
65 ページの『テーブルに値を入れる方法』
70 ページの『可変長テーブルの作成 (DEPENDING ON)』
76 ページの『テーブルの探索』
79 ページの『組み込み関数を使用したテーブル項目の処理』

501 ページの『テーブルの効率的処理』

テーブルの定義 (OCCURS)

テーブルをコーディングするには、テーブルにグループ名を与え、*n* 回繰り返される従属項目 (テーブル・エレメント) を定義します。

```
01  table-name.
    05 element-name OCCURS n TIMES.
    . . . (subordinate items of the table element)
```

上記の例では、table-name は、英数字グループ項目の名前です。 テーブル・エレメント定義 (OCCURS 節が組み込まれている) は、テーブルを含むグループ項目に従属しています。 OCCURS 節をレベル 01 記述で使用することはできません。

© Copyright IBM Corp. 2021

テーブルに入れるのが Unicode (UTF-16) データのみであり、テーブルを含んでいるグループ項目がほとんどの操作で基本カテゴリー国別項目と同様に振る舞うようにさせたい場合には、グループ項目に GROUP-USAGE NATIONAL 節をコーディングします。

```
01  table-nameN Group-Usage National.
    05  element-nameN OCCURS m TIMES.
        10  elementN1  Pic nn.
        10  elementN2  Pic S99 Sign Is Leading, Separate.
        . . .
```

国別グループに従属する基本項目は、明示的または暗黙的に USAGE NATIONAL として記述する必要があります。また符合付きの従属数値データ項目は、暗黙的または明示的に SIGN IS SEPARATE 節で記述されている必要があります。

2 次元から 7 次元までのテーブルを作成するには、ネストされた OCCURS 節を使用してください。

可変長テーブルを作成するには、OCCURS 節に DEPENDING ON 句をコーディングしてください。

テーブルの 1 つ以上のキー・フィールドの値に基づいて、テーブル・エレメントが昇順または降順に配列されるよう指定するには、 OCCURS 節の ASCENDING または DESCENDING KEY 句 (あるいはその両方) をコーディングしてください。 キーの名前は重要度の高い順に指定します。 キーは、クラス英字、英数字、DBCS、国別、または数値にすることができます。 (USAGE NATIONAL を持っている場合、キーはカテゴリー国別にすることができます。あるいは国別編集、数字編集、国別 10 進数、または国別浮動小数点の項目にすることもできます。)

テーブルの二分探索 (SEARCH ALL) を行うには、OCCURS 節の ASCENDING または DESCENDING KEY 句をコーディングする必要があります。形式 2 SORT ステートメントを使用して、定義済みキーに従ってテーブルを順序付けし、それにより SEARCH ALL ステートメントでテーブルを検索可能にすることができます。テーブルがキーに従って順序付けられていない場合、SEARCH ALL は予測不能な結果を返すことに注意してください。

78 ページの『例: 二分探索』

関連概念
187 ページの『国別グループ』

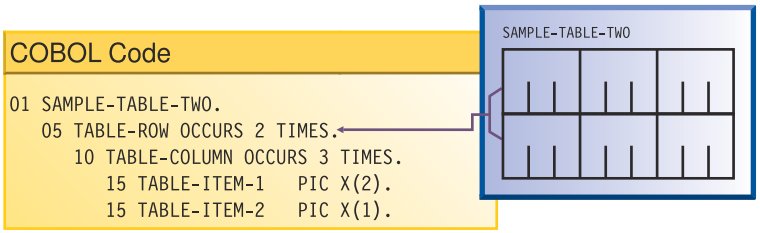
関連タスク
61 ページの『テーブルのネスト』
62 ページの『テーブル内の項目の参照』
65 ページの『テーブルに値を入れる方法』
70 ページの『可変長テーブルの作成 (DEPENDING ON)』
191 ページの『国別グループの使用』
78 ページの『二分探索 (SEARCH ALL)』
35 ページの『数値データの定義』

関連参照

OCCURS 節 (*COBOL for Linux on x86* 言語解説書)
SIGN 節 (*COBOL for Linux on x86* 言語解説書)
ASCENDING KEY および DESCENDING KEY 句
(*COBOL for Linux on x86* 言語解説書)
SORT ステートメント (*COBOL for Linux on x86* 言語解説書)

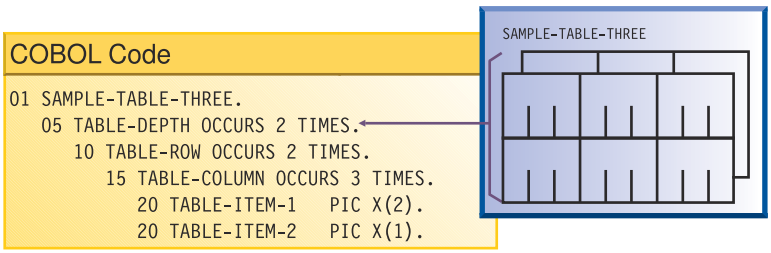
テーブルのネスト

2 次元テーブルを作成するには、ある 1 次元テーブルのそれぞれのオカレンス (出現) の中で別の 1 次元テーブルを定義します。



例えば、上の SAMPLE-TABLE-TWO の TABLE-ROW は、2 回出現する 1 次元テーブルのエレメントです。 TABLE-COLUMN は、2 次元テーブルのエレメントで、TABLE-ROW のそれぞれのオカレンスで 3 回出現します。

3 次元テーブルを作成するには、ある 1 次元テーブル (それ自体が別の 1 次元テーブルのそれぞれのオカレンスに含まれている) のそれぞれのオカレンスの中で別の 1 次元テーブルを定義します。 次に例を示します。



SAMPLE-TABLE-THREE の TABLE-DEPTH は 1 次元テーブルのエレメントで、2 回出現します。 TABLE-ROW は 2 次元テーブルのエレメントで、TABLE-DEPTH のそれぞれのオカレンスで 2 回出現します。 TABLE-COLUMN は 3 次元テーブルのエレメントで、TABLE-ROW のそれぞれのオカレンスで 3 回出現します。

2 次元テーブルでは、2 つの添え字が行番号と列番号に対応します。 3 次元テーブルでは、3 つの添え字が深さ番号、列番号、および行番号に対応します。

62 ページの『例: 添え字付け』
62 ページの『例: 指標付け』

関連タスク

59 ページの『テーブルの定義 (OCCURS)』
62 ページの『テーブル内の項目の参照』
65 ページの『テーブルに値を入れる方法』
70 ページの『可変長テーブルの作成 (DEPENDING ON)』
76 ページの『テーブルの探索』
79 ページの『組み込み関数を使用したテーブル項目の処理』
501 ページの『テーブルの効率的処理』

関連参照

OCCURS 節 (*COBOL for Linux on x86* 言語解説書)

例: 添え字付け

次の例は、リテラル添え字を使用している、SAMPLE-TABLE-THREE への有効な参照を示しています。 2 番目の例では、スペースは必須です。

```
TABLE-COLUMN (2, 2, 1)
TABLE-COLUMN (2 2 1)
```

いずれのテーブル参照でも、最初の値 (2) は TABLE-DEPTH 内の 2 番目のオカレンスを参照し、2 つ目の値 (2) は TABLE-ROW 内の 2 番目のオカレンスを参照し、3 つ目の値 (1) は TABLE-COLUMN 内の 1 番目のオカレンスを参照します。

次の SAMPLE-TABLE-TWO への参照では、変数添え字が使用されています。 SUB1 と SUB2 が、テーブルの範囲内の正の整数値を含むデータ名であれば、この参照は有効になります。

```
TABLE-COLUMN (SUB1 SUB2)
```

関連タスク
63 ページの『添え字付け』

例: 指標付け

次の例は、指標で参照されるエレメントの変位を計算する方法を示しています。

次の 3 次元テーブル SAMPLE-TABLE-FOUR を考えてみてください。

```
01  SAMPLE-TABLE-FOUR
    05 TABLE-DEPTH OCCURS 3 TIMES INDEXED BY INX-A.
       10 TABLE-ROW OCCURS 4 TIMES INDEXED BY INX-B.
          15 TABLE-COLUMN OCCURS 8 TIMES INDEXED BY INX-C  PIC X(8).
```

SAMPLE-TABLE-FOUR に対して次の相対指標付け参照をコーディングするとします。

```
TABLE-COLUMN (INX-A + 1, INX-B + 2, INX-C - 1)
```

この参照によって、TABLE-COLUMN への変位が次のように計算されます。

```
  (contents of INX-A) + (256 * 1)
+ (contents of INX-B) + (64 * 2)
+ (contents of INX-C) - (8 * 1)
```

この計算は、次のエレメント長に基づいています。

- TABLE-DEPTH のそれぞれのオカレンスは 256 バイトの長さです (4 * 8 * 8)。
- TABLE-ROW のそれぞれのオカレンスは 64 バイトの長さです (8 * 8)。
- TABLE-COLUMN のそれぞれのオカレンスは 8 バイトの長さです。

関連タスク
64 ページの『索引付け』

テーブル内の項目の参照

テーブル・エレメントは集合名を持ちますが、その中の個々の項目は固有のデータ名を持っていません。

項目を参照するには、次の 3 つの方法のいずれかを使用できます。

- テーブル・エレメントのデータ名と一緒に、そのオカレンス番号 (添え字 と呼ばれる) を括弧で囲んで使用する。 この手法は、添え字付け と呼ばれます。
- テーブル・エレメントのデータ名と一緒に、項目を位置指定するために (テーブルの先頭からの変位として) テーブルのアドレスに追加される値 (指標 と呼ばれる) を使用する。 この手法は、指標付け、または指標名を使用する添え字付けと呼ばれます。
- 添え字と指標の両方を使用する。

関連タスク
63 ページの『添え字付け』
64 ページの『索引付け』

添え字付け

可能な一番小さい添え字値は 1 であり、これはテーブル・エレメントの最初に現れるものを指します。 1 次元テーブルでは、添え字は行番号に対応します。

添え字としてはリテラルまたはデータ名を使用できます。 リテラルの添え字を持つデータ項目が固定長である場合は、コンパイラーがそのデータ項目の位置を解決します。

データ名を変数添え字として使用する場合、データ名を基本数値整数として記述する必要があります。 最も効率的な形式は、PICTURE サイズが 5 桁よりも少ない COMPUTATIONAL (COMP) です。 添え字として使用されるデータ名に添え字を付けることはできません。 アプリケーションのために生成されるコードが、実行時に変数添え字の位置を解決します。

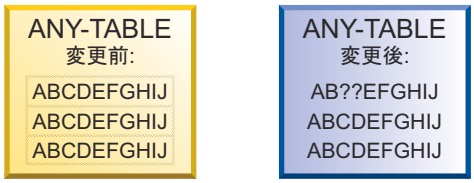
リテラルまたは変数の添え字を、指定した整数分だけ増分または減分することができます。 次に例を示します。

```
TABLE-COLUMN (SUB1 - 1, SUB2 + 3)
```

テーブル・エレメント全体ではなく、その一部を変更することができます。 これを行うには、変更するサブストリングの文字位置と長さを参照すればよいだけです。 次に例を示します。

```
01  ANY-TABLE.
    05 TABLE-ELEMENT     PIC X(10)
        OCCURS 3 TIMES   VALUE "ABCDEFGHIJ".
. . .
    MOVE "??" TO TABLE-ELEMENT (1) (3 : 2).
```

上の例の MOVE ステートメントは、ストリング「??」を、文字位置 3 から始めて 2 の長さだけ、テーブル・エレメント 1 に移動します。



62 ページの『例: 添え字付け』

関連タスク
64 ページの『索引付け』
65 ページの『テーブルに値を入れる方法』
76 ページの『テーブルの探索』
501 ページの『テーブルの効率的処理』

索引付け

指標名を識別する OCCURS 節の INDEXED BY 句を使用して、指標を作成します。

例えば、以下のコードにおける INX-A は指標名です。

```
05 TABLE-ITEM PIC X(8)
     OCCURS 10 INDEXED BY INX-A.
```

コンパイラーは、指標に含まれる値を、オカレンス番号 (添え字) から 1 引いた値にテーブル・エレメントの長さを掛けた値として計算します。 したがって、TABLE-ITEM の 5 回目のオカレンスの場合、INX-A に含まれる 2 進値は、(5 - 1) * 8、すなわち 32 です。

指標名を使用して別のテーブルを参照できるのは、両方のテーブル記述のテーブル・エレメントの数が同じであり、テーブル・エレメントが同じ長さである場合のみです。

USAGE IS INDEX 節を使用して指標データ項目を作成でき、また任意のテーブルで指標データ項目を使用できます。 例えば、以下のコードの INX-B は指標データ項目です。

```
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-ITEM (INX-A)
        . . .
    END-PERFORM.
```

上のテーブル TABLE-ITEM を全探索するのに、指標名 INX-A を使用します。 テーブルの最後のエレメントの指標を保持するには、指標データ項目 INX-B を使用します。このタイプのコーディングの利点は、テーブル・エレメントのオフセットの計算が最小限で済み、UNTIL 条件の変換が不要であることです。

SET ステートメントを使用すれば、上のステートメント SET INX-B TO INX-A のように、指標名に保管した値を指標データ項目に割り当てることができます。 例えば、レコードを可変長テーブルにロードする際に、最終レコードの指標値を、USAGE IS INDEX として定義されたデータ項目に保管できます。 その後で、現行指標値を最終レコードの指標値と比較することにより、テーブルの終わりをテストすることができます。 この手法は、テーブルを初めから終わりまで検索したり、テーブルを処理したりする場合に有用です。

例えば次のように、基本整数データ項目またはゼロ以外の整数リテラルによって指標名を増分したり、減分したりすることができます。

```
SET INX-A DOWN BY 3
```

整数は出現回数を表します。 索引に対して加算または減算される前に、指標値に変換されます。

SET、PERFORM VARYING、または SEARCH ALL ステートメントを使用して、指標名を初期化してください。 その後で、指標名を SEARCH ステートメントまたは関係条件ステートメントで使用できます。 値を変更するには、PERFORM、SEARCH、または SET ステートメントを使用してください。

物理的変位を比較するので、SEARCH および SET ステートメントでのみ、あるいは指標または他の指標データ項目との比較にのみ、指標データ項目を直接使用できます。 指標データ項目を添え字または指標として使用することはできません。

62 ページの『例: 指標付け』

関連タスク

63 ページの『添え字付け』
65 ページの『テーブルに値を入れる方法』
76 ページの『テーブルの探索』
79 ページの『組み込み関数を使用したテーブル項目の処理』
501 ページの『テーブルの効率的処理』

関連参照
INDEXED BY 句 (*COBOL for Linux on x86* 言語解説書)
INDEX 句 (*COBOL for Linux on x86* 言語解説書)
SET ステートメント (*COBOL for Linux on x86* 言語解説書)

テーブルに値を入れる方法

テーブルを定義するときに、テーブルの動的ロード、INITIALIZE ステートメントによるテーブルの初期化、または VALUE 節による値の割り当てによって、値をテーブルに入れる ことができます。

関連タスク
65 ページの『テーブルの動的なロード』
71 ページの『可変長テーブルのロード』
65 ページの『テーブルの初期化 (INITIALIZE)』
66 ページの『テーブルの定義時の値の割り当て (VALUE)』
72 ページの『可変長テーブルへの値の割り当て』

テーブルの動的なロード

テーブルの初期値がプログラムの実行のたびに異なる場合は、初期値を指定せずにテーブルを定義することができます。 代わりに、プログラムでテーブルを参照する前に、変更済みの値を動的にテーブルに読み込むことができます。

テーブルをロードするには、PERFORM ステートメントと添え字付けまたは索引付けのいずれかを使用してください。

データを読み取ってテーブルをロードするときは、データがテーブルに割り振られているスペースを超えないように確認してください。 最大項目カウントには、名前付きの値 (リテラルではなく) を使用してください。 そうすれば、テーブルをより大きくする場合に、リテラルへのすべての参照を変更する代わりに、1 つの値を変更するだけで済みます。

68 ページの『例: PERFORM と添え字付け』
69 ページの『例: PERFORM および索引付け』

関連参照
PERFORM ステートメント (*COBOL for Linux on x86* 言語解説書)

テーブルの初期化 (INITIALIZE)

1 つ以上の INITIALIZE ステートメントをコーディングすることにより、テーブルをロードできます。

例えば、以下に示す TABLE-ONE という名前のテーブルのそれぞれの基本数値データ項目に値 3 を移動するには、次のステートメントをコーディングできます。

```
INITIALIZE TABLE-ONE REPLACING NUMERIC DATA BY 3.
```

文字「X」を、TABLE-ONE のそれぞれの基本英数字データ項目に移動するには、次のステートメントをコーディングできます。

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

INITIALIZE ステートメントを使用してテーブルを初期化するとき、テーブルはグループ項目として (つまり、グループ・セマンティクスで) 処理されます。 すなわち、グループ内の基本データ項目が認識されて処理されます。 例えば、TABLE-ONE が、以下のように定義された英数字グループであるとしましょう。

```
01  TABLE-ONE.
  02  Trans-out  Occurs 20.
      05  Trans-code       Pic X    Value "R".
      05  Part-number      Pic XX   Value "13".
```

```
      05  Trans-quan       Pic 99   Value 10.
      05  Price-fields.
          10  Unit-price   Pic 99V  Value 50.
          10  Discount     Pic 99V  Value 25.
          10  Sales-Price  Pic 999  Value 375.
      . . .
      Initialize TABLE-ONE Replacing Numeric Data By 3
                                     Alphanumeric Data By "X"
```

以下の表は、上記の INITIALIZE ステートメントの実行前および実行後に 20 個の 12 バイト・エレメント Trans-out(*n*) のそれぞれが含んでいる内容を示しています。

Trans-out(*n*) (実行前)	**Trans-out(*n*) (実行後)**
R13105025375	XX*b*030303003[1]
1. 記号 *b* は、ブランク・スペースを表します。	

同様に INITIALIZE ステートメントを使用して、国別グループとして定義されたテーブルをロードできます。 例えば、上記の TABLE-ONE で GROUP-USAGE NATIONAL 節を指定した場合で、 Trans-code および Part-number の PICTURE 節に X ではなく N が指定されている場合、以下のステートメントは、上記の INITIALIZE ステートメントと同じ効果を持つことになります (ただし、TABLE-ONE のデータは代わりに UTF-16 でエンコードされます)。

```
 Initialize TABLE-ONE Replacing Numeric  Data By 3
                                National Data By N"X"
```

REPLACING NUMERIC 句は、浮動小数点データ項目も初期化します。

INITIALIZE ステートメントの REPLACING 句を同様に使用して、テーブル内の基本データ項目 ALPHABETIC、DBCS、ALPHANUMERIC-EDITED、NATIONAL-EDITED、および NUMERIC-EDITED のすべてを初期化できます。

INITIALIZE ステートメントでは、値を可変長テー ブル (つまり、OCCURS DEPENDING ON 節を使用して定義されたテーブル) に割り当てることはできません。

23 ページの『例: データ項目の初期化』

関連タスク
27 ページの『構造の初期化 (INITIALIZE)』
66 ページの『テーブルの定義時の値の割り当て (VALUE)』
72 ページの『可変長テーブルへの値の割り当て』
91 ページの『テーブルのループ処理』
20 ページの『データ項目とグループ項目の使用』
191 ページの『国別グループの使用』

関連参照

INITIALIZE ステートメント (*COBOL for Linux on x86* 言語解説書)

テーブルの定義時の値の割り当て (VALUE)

テーブルに安定値 (日や月など) が入る場合、テーブルの定義時に特定の値を設定できます。

テーブル内の静的値は、次のいずれかの方法で設定します。

- 各テーブル項目を個別に初期化する。
- テーブル全体をグループ・レベルで初期化する。
- 特定のテーブル・エレメントのすべてのオカレンスを同じ値に初期化する。

関連タスク
67 ページの『それぞれのテーブル項目の個別の初期化』

67 ページの『グループ・レベルでのテーブルの初期化』
68 ページの『ある特定テーブル・エレメントのすべての出現の初期化』
27 ページの『構造の初期化 (INITIALIZE)』

それぞれのテーブル項目の個別の初期化

テーブルが小さい場合は、VALUE 節を使用してそれぞれの項目の値を個別に設定できます。

以下のコード例に示されている手法を使用します。

1. テーブルに入れられる項目を含むレコード (以下の Error-Flag-Table など) を定義します。
2. 各項目の初期値を VALUE 節で設定します。
3. そのレコードをテーブルに入れるための REDEFINES 記入項目をコーディングします。

```
************************************************************
***          E R R O R   F L A G   T A B L E             ***
************************************************************
 01  Error-Flag-Table                  Value Spaces.
   88 No-Errors                        Value Spaces.
     05 Type-Error                     Pic X.
     05 Shift-Error                    Pic X.
     05 Home-Code-Error                Pic X.
     05 Work-Code-Error                Pic X.
     05 Name-Error                     Pic X.
     05 Initials-Error                 Pic X.
     05 Duplicate-Error                Pic X.
     05 Not-Found-Error                Pic X.
 01  Filler Redefines Error-Flag-Table.
     05 Error-Flag Occurs 8 Times
          Indexed By Flag-Index         Pic X.
```

上の例で、01 レベルの VALUE 節は、それぞれのテーブル項目を同じ値に初期化します。 代わりに、それぞれのテーブル項目に独自の VALUE 節を記述して、その項目を別個の値に初期化することもできます。

もっと大きなテーブルを初期化する場合は、MOVE、PERFORM、または INITIALIZE ステートメントを使用します。

関連タスク
27 ページの『構造の初期化 (INITIALIZE)』
72 ページの『可変長テーブルへの値の割り当て』

関連参照

REDEFINES 節 (*COBOL for Linux on x86* 言語解説書)
OCCURS 節 (*COBOL for Linux on x86* 言語解説書)

グループ・レベルでのテーブルの初期化

英数字または国別グループ・データ項目をコーディングし、VALUE 節でそれにテーブル全体の内容を割り当てます。 次に、従属データ項目で、OCCURS 節を使用して個々のテーブル項目を定義します。

以下の例の英数字グループ・データ項目 TABLE-ONE は、TABLE-TWO の 4 個のエレメントそれぞれを初期化する VALUE 節を使用しています。

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

以下の例の国別グループ・データ項目 Table-OneN は、従属データ項目 Table-TwoN の 3 個のエレメントそれぞれを初期化する VALUE 節を使用しています (エレメントのそれぞれは暗黙的に USAGE NATIONAL です)。 英数字リテラルを使用する VALUE 節 (以下に示されている) または 国別リテラルを使用する VALUE 節で国別グループ・データ項目を初期化できることに注意してください。

```
01  Table-OneN  Group-Usage National  Value "AB12CD34EF56".
    05  Table-TwoN   Occurs 3 Times   Indexed By MyI.
        10  ElementOneN  Pic nn.
        10  ElementTwoN  Pic 99.
```

Table-OneN の初期化後に、ElementOneN(1) には、NX"41004200" ('AB' の UTF-16 表現) が入り、国別 10 進数項目 ElementTwoN(1) には、NX"31003200" ('12' の UTF-16 表現) が入ります。以下同様です。

関連参照
OCCURS 節 (*COBOL for Linux on x86* 言語解説書)
GROUP-USAGE 節 (*COBOL for Linux on x86* 言語解説書)

ある特定テーブル・エレメントのすべての出現の初期化

テーブル・エレメントのデータ記述にある VALUE 節を使用して、そのエレメントのすべてのインスタンスを指定した値に初期化することができます。

```
01  T2.
    05 T-OBJ                   PIC 9   VALUE 3.
    05 T OCCURS 5 TIMES
           DEPENDING ON T-OBJ.
       10 X                    PIC XX  VALUE "AA".
       10 Y                    PIC 99  VALUE 19.
       10 Z                    PIC XX  VALUE "BB".
```

例えば、上のコードによって、すべての X エレメント (1 から 5) が AA に初期化され、すべての Y エレメント (1 から 5) が 19 に初期化され、すべての Z エレメント (1 から 5) が BB に初期化されます。 その後、T-OBJ が 3 に設定されます。

関連タスク
72 ページの『可変長テーブルへの値の割り当て』

関連参照
OCCURS 節 (*COBOL for Linux on x86* 言語解説書)

例: PERFORM と添え字付け

この例は、設定されたエラー・コードが検出されるまで、添え字付けを使用してエラー・フラグ (error-flag) テーブルを全探索します。 エラー・コードが見つかると、対応するエラー・メッセージが報告書印刷フィールドに移動されます。

```
************************************************************
***           E R R O R   F L A G   T A B L E            ***
************************************************************
 01  Error-Flag-Table                    Value Spaces.
   88 No-Errors                          Value Spaces.
     05 Type-Error                       Pic X.
     05 Shift-Error                      Pic X.
     05 Home-Code-Error                  Pic X.
     05 Work-Code-Error                  Pic X.
     05 Name-Error                       Pic X.
     05 Initials-Error                   Pic X.
     05 Duplicate-Error                  Pic X.
     05 Not-Found-Error                  Pic X.
 01  Filler Redefines Error-Flag-Table.
     05 Error-Flag Occurs 8 Times
           Indexed By Flag-Index         Pic X.
 77  Error-on                            Pic X  Value "E".
************************************************************
***         E R R O R   M E S S A G E   T A B L E        ***
************************************************************
 01  Error-Message-Table.
     05  Filler                           Pic X(25) Value
          "Transaction Type Invalid".
     05  Filler                           Pic X(25) Value
          "Shift Code Invalid".
```

```
     05  Filler                          Pic X(25) Value
          "Home Location Code Inval.".
     05  Filler                          Pic X(25) Value
          "Work Location Code Inval.".
     05  Filler                          Pic X(25) Value
          "Last Name - Blanks".
     05  Filler                          Pic X(25) Value
          "Initials - Blanks".
     05  Filler                          Pic X(25) Value
          "Duplicate Record Found".
     05  Filler                          Pic X(25) Value
          "Commuter Record Not Found".
 01  Filler Redefines Error-Message-Table.
     05  Error-Message Occurs 8 Times
            Indexed By Message-Index      Pic X(25).
 . . .
 PROCEDURE DIVISION.
     . . .
     Perform
          Varying Sub From 1 By 1
          Until No-Errors
        If Error-Flag (Sub) = Error-On
          Move Space To Error-Flag (Sub)
          Move Error-Message (Sub) To Print-Message
          Perform 260-Print-Report
        End-If
     End-Perform
     . . .
```

例: PERFORM および索引付け

この例は、設定されたエラー・コードが検出されるまで、指標付けを使用してエラー・フラグ (error-flag) テーブルを全探索します。 エラー・コードが見つかると、対応するエラー・メッセージが報告書印刷フィールドに移動されます。

```
*************************************************************
***          E R R O R   F L A G   T A B L E              ***
*************************************************************
 01  Error-Flag-Table                Value Spaces.
   88 No-Errors                       Value Spaces.
     05 Type-Error                    Pic X.
     05 Shift-Error                   Pic X.
     05 Home-Code-Error               Pic X.
     05 Work-Code-Error               Pic X.
     05 Name-Error                    Pic X.
     05 Initials-Error                Pic X.
     05 Duplicate-Error               Pic X.
     05 Not-Found-Error               Pic X.
 01  Filler Redefines Error-Flag-Table.
     05 Error-Flag Occurs 8 Times
          Indexed By Flag-Index       Pic X.
 77  Error-on                        Pic X  Value "E".
*************************************************************
***        E R R O R   M E S S A G E   T A B L E          ***
*************************************************************
 01  Error-Message-Table.
     05  Filler                          Pic X(25) Value
          "Transaction Type Invalid".
     05  Filler                          Pic X(25) Value
          "Shift Code Invalid".
     05  Filler                          Pic X(25) Value
          "Home Location Code Inval.".
     05  Filler                          Pic X(25) Value
          "Work Location Code Inval.".
     05  Filler                          Pic X(25) Value
          "Last Name - Blanks".
     05  Filler                          Pic X(25) Value
          "Initials - Blanks".
     05  Filler                          Pic X(25) Value
          "Duplicate Record Found".
     05  Filler                          Pic X(25) Value
          "Commuter Record Not Found".
 01  Filler Redefines Error-Message-Table.
     05  Error-Message Occurs 8 Times
            Indexed By Message-Index      Pic X(25).
 . . .
 PROCEDURE DIVISION.
```

```
. . .
Set Flag-Index To 1
Perform Until No-Errors
  Search Error-Flag
    When Error-Flag (Flag-Index) = Error-On
      Move Space To Error-Flag (Flag-Index)
      Set Message-Index To Flag-Index
      Move Error-Message (Message-Index) To
        Print-Message
      Perform 260-Print-Report
  End-Search
End-Perform
. . .
```

可変長テーブルの作成 (DEPENDING ON)

テーブル・エレメントが出現する回数が実行前にわからない場合には、可変長テーブルを定義してください。 そのためには、OCCURS DEPENDING ON (ODO) 節を使用します。

```
X OCCURS 1 TO 10 TIMES DEPENDING ON Y
```

上の例で、X は *ODO* サブジェクトと呼び、Y は *ODO* オブジェクトと呼びます。

可変長レコードを正しく操作するためには、次の 2 つの要因が影響します。

- レコード長を正しく計算すること

  グループ項目の可変部分の長さは、DEPENDING ON 句のオブジェクトと OCCURS 節のサブジェクトの長さとの積です。
- OCCURS DEPENDING ON 節のオブジェクトにおけるデータの PICTURE 節との適合性

  ODO オブジェクトの内容がそれの PICTURE 節と一致していない場合には、プログラムが異常終了することがあります。 ODO オブジェクトに、テーブル・エレメントの現在のオカレンス回数を正しく指定するようにしてください。

次の例は、OCCURS DEPENDING ON 節のサブジェクトとオブジェクトの両方を含むグループ項目 (REC-1) を示しています。 グループ項目の長さがどのようにして決定されるかは、それがデータを送り出しているのか、データを受け取っているのかによって異なります。

```
WORKING-STORAGE SECTION.
01  MAIN-AREA.
    03 REC-1.
        05 FIELD-1                        PIC 9.
        05 FIELD-2 OCCURS 1 TO 5 TIMES
           DEPENDING ON FIELD-1          PIC X(05).
01  REC-2.
    03 REC-2-DATA                        PIC X(50).
```

REC-1 (この場合は送り出し項目) を REC-2 に移動したい 場合、REC-1 の長さは、FIELD-1 の現行値を使用して、移動の 直前に決定されます。 FIELD-1 の内容がその PICTURE 節と一致している場合 (すなわち、FIELD-1 がゾーン 10 進数項目を含んでいる場合)、REC-1 の実際の長さに基づいて移動は続行可能です。 それ以外の場合は、結果は予測できません。 移動を開始する前に、ODO オブジェクトに正しい値が含まれていることを確認してください。

REC-1 (この場合は受け取り項目) に移動を行う場合、REC-1 の長さは、最大のオカレンス回数を使用して決定されます。 この例では、FIELD-2 の 5 回のオカレンスと FIELD-1 の合計で 26 バイトの長さになります。 この場合、REC-1 を受け取り項目として参照する前に、ODO オブジェクト (FIELD-1) を設定する必要はありません。 ただし、移動によって受信フィールドの ODO オブジェクトを有効に設定するようにするためには、送信フィールドの ODO オブジェクト (非表示) を 1 から 5 の間の有効な数値に設定しなければなりません。

しかし、REC-1 の後に可変位置グループ (複合 *ODO*) が続いているような REC-1 (この場合も受け取り項目) に移動を行う場合は、REC-1 の実際の長さは、ODO オブジェクト (FIELD-1) の現行値を使用して、移動の直前に計算されます。 次の例では、REC-1 と REC-2 は同じレコードにありますが、REC-2 は REC-1 に従属していないため、可変位置項目です。

```
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).
```

コンパイラーは、実際の長さが使用されたことを知らせるメッセージを出します。 この場合には、グループ項目を受信フィールドとして使用する前に、ODO オブジェクトの値を設定することが必要となります。

次の例は、ODO オブジェクト (下記の LOCATION-TABLE-LENGTH) がグループの外側にあるときの、可変長テーブルの定義方法を示します。

```
 DATA DIVISION.
 FILE SECTION.
 FD  LOCATION-FILE.
 01  LOCATION-RECORD.
     05  LOC-CODE                 PIC XX.
     05  LOC-DESCRIPTION          PIC X(20).
     05  FILLER                   PIC X(58).
 WORKING-STORAGE SECTION.
 01  FLAGS.
     05 LOCATION-EOF-FLAG         PIC X(5) VALUE SPACE.
        88 LOCATION-EOF              VALUE "FALSE".
 01  MISC-VALUES.
     05 LOCATION-TABLE-LENGTH     PIC 9(3) VALUE ZERO.
     05 LOCATION-TABLE-MAX        PIC 9(3) VALUE 100.
*****************************************************************
***               L O C A T I O N   T A B L E                 ***
***               FILE CONTAINS LOCATION CODES.               ***
*****************************************************************
 01  LOCATION-TABLE.
     05 LOCATION-CODE OCCURS 1 TO 100 TIMES
          DEPENDING ON LOCATION-TABLE-LENGTH   PIC X(80).
```

関連概念
73 ページの『複合 OCCURS
DEPENDING ON』

関連タスク
72 ページの『可変長テーブルへの値の割り当て』
71 ページの『可変長テーブルのロード』
75 ページの『エレメントを可変テーブルに追加する際のオーバーレイを防止する』
109 ページの『データ項目の長さの検出』

関連参照
OCCURS DEPENDING ON 節
(*COBOL for Linux on x86* 言語解説書)
可変長テーブル (*COBOL for Linux on x86* 言語解説書)

可変長テーブルのロード

do-until 構造 (TEST AFTER ループ) を使用して、可変長テーブルのロードを制御することができます。 例えば、次のコードが実行されると、LOCATION-TABLE-LENGTH には、テーブルの最後の項目の添え字が入れられます。

```
 DATA DIVISION.
```

```
 FILE SECTION.
 FD  LOCATION-FILE.
 01  LOCATION-RECORD.
     05  LOC-CODE              PIC XX.
     05  LOC-DESCRIPTION       PIC X(20).
     05  FILLER                PIC X(58).
 . . .
 WORKING-STORAGE SECTION.
 01  FLAGS.
     05 LOCATION-EOF-FLAG      PIC X(5) VALUE SPACE.
        88 LOCATION-EOF                 VALUE "YES".
 01  MISC-VALUES.
     05 LOCATION-TABLE-LENGTH  PIC 9(3) VALUE ZERO.
     05 LOCATION-TABLE-MAX     PIC 9(3) VALUE 100.
*****************************************************************
***               L O C A T I O N   T A B L E                 ***
***               FILE CONTAINS LOCATION CODES.               ***
*****************************************************************
 01  LOCATION-TABLE.
     05 LOCATION-CODE OCCURS 1 TO 100 TIMES
          DEPENDING ON LOCATION-TABLE-LENGTH   PIC X(80).
 . . .
 PROCEDURE DIVISION.
     . . .
     Perform Test After
         Varying Location-Table-Length From 1 By 1
           Until Location-EOF
           Or Location-Table-Length = Location-Table-Max
       Move Location-Record To
           Location-Code (Location-Table-Length)
       Read Location-File
           At End Set Location-EOF To True
       End-Read
     End-Perform
```

可変長テーブルへの値の割り当て

DEPENDING ON 句を持つ OCCURS 節を含んでいる、従属データ項目のある英数字または国別グループ項目に、VALUE 節をコーディングできます。 DEPENDING ON 句を含むそれぞれの従属構造は、最大オカレンス回数を使用して初期化されます。

DEPENDING ON 句を使用してテーブル全体を定義する場合、ODO (OCCURS DEPENDING ON) オブジェクトの最大定義値を使用して、全エレメントが初期化されます。

ODO オブジェクトが VALUE 節で初期化される場合、これは必然的に、ODO サブジェクトが初期化された後に初期化されます。

```
01  TABLE-THREE           VALUE "3ABCDE".
    05 X                  PIC 9.
    05 Y OCCURS 5 TIMES
          DEPENDING ON X  PIC X.
```

例えば、上記のコードで、ODO サブジェクト Y(1) は「A」に、Y(2) は「B」に、・・・Y(5) は「E」に初期化され、最後に ODO オブジェクト X が 3 に初期化されます。 TABLE-THREE への後続の参照 (DISPLAY ステートメントでの 参照など) は、X とテーブルの 最初の 3 つのエレメント (Y(1) から Y(3)) を参照します。

関連タスク
66 ページの『テーブルの定義時の値の割り当て (VALUE)』

関連参照

OCCURS DEPENDING ON 節
(*COBOL for Linux on x86* 言語解説書)

複合 OCCURS DEPENDING ON

複合 OCCURS DEPENDING ON (複合 *ODO*) にはいくつかのタイプがあります。 複合 ODO は、85 COBOL 標準 の拡張としてサポートされます。

コンパイラーによって認められる複合 ODO の基本形式は、以下のとおりです。

- 可変位置項目またはグループ: DEPENDING ON 句を指定した OCCURS 節で記述されたデータ項目の後に、非従属基本データ項目またはグループ・データ項目が続きます。
- 可変位置テーブル: DEPENDING ON 句を指定した OCCURS 節によって記述されたデータ項目の後に、OCCURS 節によって記述された非従属データ項目が続きます。
- 可変長エレメントを持つテーブル: OCCURS 節によって記述されたデータ項目に、OCCURS 節に DEPENDING ON 句を指定して記述された従属データ項目が含まれています。
- 可変長エレメントを持つテーブルの指標名。
- 可変長エレメントを持つテーブルのエレメント。

73 ページの『例: 複合 ODO』

関連タスク
74 ページの『ODO オブジェクト値を変更する際の指標エラーを防止する』
75 ページの『エレメントを可変テーブルに追加する際のオーバーレイを防止する』

関連参照
74 ページの『ODO オブジェクト値の変更の影響』
OCCURS DEPENDING ON 節
(*COBOL for Linux on x86* 言語解説書)

例: 複合 ODO

次の例は、複合 ODO が現れる場合の可能なタイプを示しています。

```
01  FIELD-A.
    02 COUNTER-1                              PIC S99.
    02 COUNTER-2                              PIC S99.
    02 TABLE-1.
       03 RECORD-1 OCCURS 1 TO 5 TIMES
                   DEPENDING ON COUNTER-1     PIC X(3).
    02 EMPLOYEE-NUMBER                        PIC X(5). (1)
    02 TABLE-2 OCCURS 5 TIMES                           (2)(3)
              INDEXED BY INDX.                          (4)
       03 TABLE-ITEM                          PIC 99.   (5)
       03 RECORD-2 OCCURS 1 TO 3 TIMES
                   DEPENDING ON COUNTER-2.
          04 DATA-NUM                         PIC S99.
```

定義: この例では、COUNTER-1 は *ODO* オブジェクト です。つまり、RECORD-1 の DEPENDING ON 節のオブジェクトです。 RECORD-1 は *ODO* サブジェクト であると言われます。 同様に、COUNTER-2 は、対応する ODO サブジェクトである RECORD-2 の ODO オブジェクトです。

上記の例で現れている複合 ODO のタイプは次のとおりです。

(1)
可変位置項目: EMPLOYEE-NUMBER は、同じレベル 01 レコード内の可変長テーブルに続いている (ただし、従属してはいない) データ項目です。

(2)
可変位置テーブル: TABLE-2 は、同じレベル 01 レコード内の可変長テーブルに続いている (ただし、従属してはいない) テーブルです。

(3)
可変長エレメントを持つテーブル: TABLE-2 は、従属データ項目 RECORD-2 を含んでいるテーブルであり、この従属データ項目の出現回数は ODO オブジェクトの内容によって異なります。

(4)
可変長エレメントを持つテーブルの指標名 INDX。

(5)
可変長エレメントを持つテーブルの エレメント TABLE-ITEM。

長さの計算方法

各レコードの可変部分の長さは、その ODO オブジェクトとその ODO サブジェクトの長さとの積です。 例えば、上記に示された複合 ODO 項目の 1 つに参照が行われるたびに、使用される際の実際の長さは、次のように計算されます。

- TABLE-1 の長さは、COUNTER-1 の内容 (RECORD-1 のオカレンスの回数) に 3 (RECORD-1 の長さ) を掛けることによって計算されます。
- TABLE-2 の長さは、COUNTER-2 の内容 (RECORD-2 のオカレンスの回数) に 2 (RECORD-2 の長さ) を掛け、TABLE-ITEM の長さを加算することによって計算されます。
- FIELD-A の長さは、COUNTER-1、COUNTER-2、TABLE-1、EMPLOYEE-NUMBER、および TABLE-2 の長さに 5 を掛けたものを加算することによって計算されます。

ODO オブジェクトの値の設定

グループ内の複合 ODO 項目を参照するには、グループ項目内のすべての ODO オブジェクトを設定しておく必要があります。 例えば、上記のコードの EMPLOYEE-NUMBER を参照するには、その前に、COUNTER-1 と COUNTER-2 を設定しておかなければなりません。ただし、EMPLOYEE-NUMBER は ODO オブジェクトに直接依存して値を得るわけではありません。

制約事項: ODO オブジェクトは可変的に配置することはできません。

ODO オブジェクト値の変更の影響

DEPENDING ON 句が指定された OCCURS 節で記述されたデータ項目の後に、同じグループ内で 1 つ以上の非従属データ項目 (複合 ODO 形式) が続いている場合に、ODO オブジェクトの値を変更すると、レコード内の複合 ODO 項目への後続の参照が影響を受けます。

例えば、次のように指定します。

- 関係のある ODO 節を含んでいるグループのサイズは、ODO オブジェクトの新しい値を反映します。
- ODO オブジェクトの新しい値に基づいて、ODO サブジェクトを含んでいるグループへの移動 (MOVE) が行われます。
- ODO 節で記述された項目に続いている非従属項目の位置は、ODO オブジェクトの新しい値の影響を受けます。 (非従属項目の内容を保持するためには、ODO オブジェクトの値が変更される前に、非従属項目を作業域に移動しておき、後でそれらを戻してください。)

ODO オブジェクトの値は、データをその ODO オブジェクトに移動するか、その ODO オブジェクトが含まれているグループに移動すると、変更される可能性があります。 また、ODO オブジェクトが READ ステートメントのターゲットであるレコードに含まれている場合にも、その値が変更されることがあります。

関連タスク
74 ページの『ODO オブジェクト値を変更する際の指標エラーを防止する』
75 ページの『エレメントを可変テーブルに追加する際のオーバーレイを防止する』

ODO オブジェクト値を変更する際の指標エラーを防止する

テーブル内の従属データ項目の ODO オブジェクトの値を変更した後に、複合 ODO 指標名 (つまり、可変長エレメントを持つテーブルの指標名) を参照する場合には、注意してください。

ODO オブジェクトの値を変更すると、テーブルの長さが変わるため、関連する複合 ODO 指標のバイト・オフセットはもう有効ではありません。 ですから次のような指標名への参照をコーディングした場合、予防措置を講じなければ、 予期しない結果が生じることになります。

- テーブルのエレメントへの参照

- SET *integer-data-item* TO *index-name* 形式の SET ステートメント (形式 1)
- SET *index-name* UP | DOWN BY *integer* 形式の SET ステートメント (形式 2)

この種のエラーを回避するためには、次の手順を行ってください。

1. 指標を整数データ項目に保存する。 (これを行うと、暗黙の変換が行われます。整数項目は、指標の オフセットに対応するテーブル・エレメント出現番号を受け取ります。)
2. ODO オブジェクトの値を変更する。
3. ただちに整数データ項目から指標を復元する。 (これを行うと、暗黙の変換が行われます。指標名は、整数項目でのテーブル・エレメント出現番号に対応するオフセットを受け取ります。 オフセットは、その時点で有効なテーブルの長さに従って計算されます。)

次のコードは、ODO オブジェクト COUNTER-2 が変更される場合の指標名の保存方法と復元方法を示しています (73 ページの『例: 複合 ODO』 を参照)。

```
 77  INTEGER-DATA-ITEM-1      PIC 99.
 . . .
     SET INDX TO 5.
*         INDX is valid at this point.
     SET INTEGER-DATA-ITEM-1 TO INDX.
*         INTEGER-DATA-ITEM-1 now has the
*         occurrence number that corresponds to INDX.
     MOVE NEW-VALUE TO COUNTER-2.
*         INDX is not valid at this point.
     SET INDX TO INTEGER-DATA-ITEM-1.
*         INDX is now valid, containing the offset
*         that corresponds to INTEGER-DATA-ITEM-1, and
*         can be used with the expected results.
```

関連参照

SET ステートメント (*COBOL for Linux on x86* 言語解説書)

エレメントを可変テーブルに追加する際のオーバーレイを防止する

同じグループ内で 1 つ以上の非従属データ項目が後に続いている可変オカレンス・テーブル内のエレメントの数を増やす場合には、注意してください。 ODO オブジェクトの値を増分し、エレメントをテーブルに追加する際に、テーブルの後に続く可変位置データ項目を誤ってオーバーレイするおそれがあります。

この種のエラーを回避するためには、次の手順を行ってください。

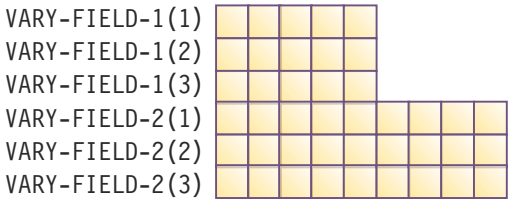
1. テーブルの後に続く可変位置データ項目を別のデータ域に保管する。
2. ODO オブジェクトの値を増分する。
3. データを新しいテーブル・エレメントに移動する (必要な場合)。
4. 可変位置データ項目を、それらを保管したデータ域から復元する。

次の例では、テーブル VARY-FIELD-1 にエレメントを追加しますが、このテーブルのエレメントの数は ODO オブジェクト CONTROL-1 に左右されます。 VARY-FIELD-1 の後には、非従属可変位置データ項目である GROUP-ITEM-1 が続いており、このエレメントがオーバーレイされる可能性があります。

```
WORKING-STORAGE SECTION.
01  VARIABLE-REC.
    05  FIELD-1                              PIC X(10).
    05  CONTROL-1                            PIC S99.
    05  CONTROL-2                            PIC S99.
    05  VARY-FIELD-1 OCCURS 1 TO 10 TIMES
          DEPENDING ON CONTROL-1             PIC X(5).
    05  GROUP-ITEM-1.
        10  VARY-FIELD-2
              OCCURS 1 TO 10 TIMES
              DEPENDING ON CONTROL-2         PIC X(9).
01  STORE-VARY-FIELD-2.
    05  GROUP-ITEM-2.
        10  VARY-FLD-2
```

```
               OCCURS 1 TO 10 TIMES
               DEPENDING ON CONTROL-2         PIC X(9).
```

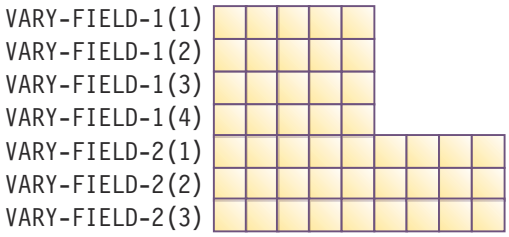
VARY-FIELD-1 の各エレメントは 5 バイトで、VARY-FIELD-2 の各エレメントは 9 バイトです。 CONTROL-1 と CONTROL-2 の両方に値 3 が含まれている場合は、VARY-FIELD-1 および VARY-FIELD-2 のストレージは次のように示すことができます。



VARY-FIELD-1 に 4 番目のエレメントを追加する場合は、次のようにコーディングすれば、VARY-FIELD-2 の最初の 5 バイトのオーバーレイを回避することができます。 (GROUP-ITEM-2 は、可変位置 GROUP-ITEM-1 の一時記憶域として働きます。)

```
MOVE GROUP-ITEM-1 TO GROUP-ITEM-2.
ADD 1 TO CONTROL-1.
MOVE five-byte-field TO
  VARY-FIELD-1 (CONTROL-1).
MOVE GROUP-ITEM-2 TO GROUP-ITEM-1.
```

VARY-FIELD-1 と VARY-FIELD-2 の更新後のストレージは次のように 示すことができます。



VARY-FIELD-1 の 4 番目のエレメントが VARY-FIELD-2 の最初のエレメントをオーバーレイしていないことに注意してください。

テーブルの探索

COBOL は、2 つのテーブルの検索手法 (逐次 および バイナリー) を提供します。

逐次探索を行うには、SEARCH と索引付けを使用します。 可変長テーブルの場合は、PERFORM と添え字付けまたは指標付けを使用することができます。

二分探索を行うには、SEARCH ALL と索引付けを使用します。

二分探索の方が、逐次探索よりも効率が相当よくなる可能性があります。 逐次探索の場合、比較の数は、*n* の次数、すなわちテーブルの項目の数です。 二分探索の場合、比較の数は、*n* の対数 (基底 2) の次数にすぎません。ただし、二分探索を行うには、テーブル項目をあらかじめソートしておく必要があります。

関連タスク
76 ページの『逐次探索 (SEARCH)』
78 ページの『二分探索 (SEARCH ALL)』

逐次探索 (SEARCH)

SEARCH ステートメントを使用して、現行指標設定値を開始点として逐次 (順次) 探索を行います。 指標設定値を変更するには、SET ステートメントを使用してください。

WHEN 句の条件は、それらが指定された順番に評価されます。

- どの条件も満たされない場合には、指標が次のテーブル・エレメントに対応するように増加され、WHEN 条件が再度評価されます。
- WHEN 条件の 1 つが満たされると、探索は終了します。 指標は条件を満たしたテーブル・エレメントを指したままになります。
- テーブル全体が探索され、どの条件も満たされなかった場合には、AT END 命令ステートメントが実行されます (存在する場合)。 AT END をコーディングしなかった場合、 制御はプログラム内の次のステートメントに渡ります。

それぞれの SEARCH ステートメントでは、テーブルの 1 つのレベル (1 つのテーブル・エレメント) だけを参照することができます。 テーブルの複数のレベルを探索するには、ネストされた SEARCH ステートメントを使用してください。 ネストされたそれぞれの SEARCH ステートメントは、END-SEARCH で区切らなければなりません。

パフォーマンス: 検出された条件がテーブル内の中間点より後にくる場合には、SET ステートメントを使用してその点より後から探索を開始するように索引を設定することによって、探索を速めることができます。また、最も頻繁に使用されるデータがテーブルの先頭になるようにテーブルを配置すると、逐次探索をより効率的に行うことができます。 テーブルが大きくて、事前にソートされている場合には、二分探索の方が効率的です。

77 ページの『例: 逐次探索』

関連参照
SEARCH ステートメント (*COBOL for Linux on x86* 言語解説書)

例: 逐次探索

次の例は、3 次元テーブルの最も内部にあるテーブルから特定のストリングを見つける方法を示しています。

テーブルの各次元には独自の指標が割り当てられています (それぞれ 1、4、および 1 に設定されています)。 最も内部のテーブル (TABLE-ENTRY3) には昇順キーがあります。

```
01  TABLE-ONE.
    05 TABLE-ENTRY1 OCCURS 10 TIMES
          INDEXED BY TE1-INDEX.
       10 TABLE-ENTRY2 OCCURS 10 TIMES
             INDEXED BY TE2-INDEX.
          15 TABLE-ENTRY3 OCCURS 5 TIMES
                ASCENDING KEY IS KEY1
                INDEXED BY TE3-INDEX.
             20 KEY1                PIC X(5).
             20 KEY2                PIC X(10).
. . .
PROCEDURE DIVISION.
    . . .
    SET TE1-INDEX TO 1
    SET TE2-INDEX TO 4
    SET TE3-INDEX TO 1
    MOVE "A1234" TO KEY1 (TE1-INDEX, TE2-INDEX, TE3-INDEX + 2)
    MOVE "AAAAAAAA00" TO KEY2 (TE1-INDEX, TE2-INDEX, TE3-INDEX + 2)
    . . .
    SEARCH TABLE-ENTRY3
      AT END
        MOVE 4 TO RETURN-CODE
      WHEN TABLE-ENTRY3(TE1-INDEX, TE2-INDEX, TE3-INDEX)
          = "A1234AAAAAAAA00"
        MOVE 0 TO RETURN-CODE
    END-SEARCH
```

実行後の値

```
TE1-INDEX = 1
TE2-INDEX = 4
TE3-INDEX points to the TABLE-ENTRY3 item
```

```
          that equals "A1234AAAAAAAA00"
RETURN-CODE = 0
```

二分探索 (SEARCH ALL)

SEARCH ALL を使用して二分探索を行う場合、開始する前に指標を設定する必要はありません。 指標は常に、OCCURS 節内の最初の指標名と関連付けられるものです。 この指標は、探索の効率を最大にするために、実行中に変わります。

SEARCH ALL ステートメントを使用してテーブルを探索するには、テーブルで OCCURS 節の ASCENDING または DESCENDING KEY 句 (あるいはその両方) を指定する必要があり、また ASCENDING および DESCENDING KEY 句で指定されたキーに基づいて既に順序付けられている必要があります。形式 2 SORT ステートメントを使用して、定義済みキーに従ってテーブルを順序付けし、それにより SEARCH ALL ステートメントでテーブルを検索可能にすることができます。 テーブルがキーに従って順序付けられていない場合、SEARCH ALL は予測不能な結果を返すことに注意してください。

SEARCH ALL ステートメントの WHEN 句では、テーブルの ASCENDING または DESCENDING KEY 句で指定されているキーをテストできますが、前に現れている すべてのキー (ある場合) をテストする必要があります。 テストは等価条件でなければならず、WHEN 句ではキー (テーブルに関連した最初の指標名で添え字が付けられている) かまたはキーに関連した条件名を指定する必要があります。 WHEN 条件は、論理連結語として AND のみを使用した複数の単純条件から形成した複合条件であっても構いません。

それぞれのキーとその比較の対象はデータ項目の比較規則に従って、互換性がなければなりません。 ただし、キーを国別リテラルまたは ID と比較する場合、キーは国別データ項目にする必要があることに注意してください。

78 ページの『例: 二分探索』

関連タスク
59 ページの『テーブルの定義 (OCCURS)』

関連参照

SEARCH ステートメント (*COBOL for Linux on x86* 言語解説書)
一般比較条件 (*COBOL for Linux on x86* 言語解説書)

例: 二分探索

次の例は、テーブルの二分探索をコーディングする方法を示しています。

テーブルにそれぞれが 40 バイトの 90 個のエレメントが含まれており、3 つのキーがあるとします。 1 次キーと 2 次キー (KEY-1 と KEY-2) は昇順ですが、最も重要でないキー (KEY-3) は降順です。

```
01  TABLE-A.
    05 TABLE-ENTRY OCCURS 90 TIMES
           ASCENDING KEY-1, KEY-2
           DESCENDING KEY-3
           INDEXED BY INDX-1.
       10 PART-1       PIC 99.
       10 KEY-1        PIC 9(5).
       10 PART-2       PIC 9(6).
       10 KEY-2        PIC 9(4).
       10 PART-3       PIC 9(18).
       10 KEY-3        PIC 9(5).
```

このテーブルは、次のステートメントを使用して探索することができます。

```
SEARCH ALL TABLE-ENTRY
  AT END
    PERFORM NOENTRY
  WHEN KEY-1 (INDX-1) = VALUE-1 AND
       KEY-2 (INDX-1) = VALUE-2 AND
       KEY-3 (INDX-1) = VALUE-3
```

```
    MOVE PART-1 (INDX-1) TO OUTPUT-AREA
END-SEARCH
```

3 つのキーのそれぞれが比較対象の値 (それぞれ VALUE-1、VALUE-2 、および VALUE-3) と等しい項目が検出された場合、その項目の PART-1 は OUTPUT-AREA へ移動されます。 TABLE-A 内のどの項目でも一致するキーが検出されない場合には、NOENTRY ルーチンが実行されます。

テーブルのソート

形式 2 SORT ステートメントを使用してテーブルをソートできます。 これは 2002 COBOL 標準の一部です。

形式 2 SORT ステートメントは、指定されたテーブル・キーに従ってテーブル・エレメントをソートします。 これは特に、SEARCH ALL で使用されるテーブルに役立ちます。 ソートのキーはテーブル定義の一部として指定でき、これは SEARCH ALL ステートメントでも使用できます。 また、ソートのキーは SORT ステートメントの一部として指定することもできます。 これは、テーブル定義で指定されたものとは異なるキーを使用してテーブルをソートしたい場合や、テーブルでキーが指定されていない場合でも可能です。

形式 2 SORT ステートメントでは、形式 1 SORT ステートメントのように入出力プロシージャーを使用する必要はありません。

指定されたキーに基づいてテーブルをソートする例を以下に示します。

```
WORKING-STORAGE SECTION.
01 GROUP-ITEM.
    05 TABL OCCURS 10 TIMES
       10 ELEM-ITEM1 PIC X.
       10 ELEM-ITEM2 PIC X.
       10 ELEM-ITEM3 PIC X.
...
PROCEDURE DIVISION.
    ...
    SORT TABL DESCENDING ELEM-ITEM2 ELEM-ITEM3.
    IF TABL (1)...
```

関連参照
SORT ステートメント (*COBOL for Linux on x86* 言語解説書)
518 ページの『形式 2 SORT ステートメントを使用したテーブルのソート』

組み込み関数を使用したテーブル項目の処理

組み込み関数を使用して、英字、英数字、国別、または数値のテーブル項目を処理できます。 (DBCS データ項目は NATIONAL-OF 組み込み関数でのみ処理できます)。 テーブル項目のデータ記述は、関数の引数要件と互換性があるようにする必要があります。

個々のデータを関数引数として参照するには、添え字または指標を使用してください。 例えば、 Table-One が 3 x 3 の数値項目の配列であるとすると、次のようなステートメントを使用して中間エレメントの平方根を求めることができます。

```
Compute X = Function Sqrt(Table-One(2,2))
```

テーブル内のデータを繰り返し処理することが必要な場合もあります。 複数の引数を受け入れる組み込み関数の場合、添え字 ALL を使用して、テーブル内またはテーブルの単一次元内のすべての項目を参照することができます。 反復は自動的に処理されるため、コードがより短く、単純になります。

複数の引数を受け入れる関数の場合は、スカラーと配列引数を混在させることができます。

```
Compute Table-Median = Function Median(Arg1 Table-One(ALL))
```

80 ページの『例: 組み込み関数を使用したテーブルの処理』

関連タスク
32 ページの『組み込み関数の使用 (組み込み関数)』
104 ページの『データ項目の変換 (組み込み関数)』
106 ページの『データ項目の評価 (組み込み関数)』

関連参照

組み込み関数 (*COBOL for Linux on x86* 言語解説書)

例: 組み込み関数を使用したテーブルの処理

以下の例は、ALL 添え字を使用してテーブル内のエレメントの一部または全部に組み込み関数を適用する方法を示しています。

Table-Two が 2 x 3 x 2 の配列であるとすると、次のステートメントは、エレメント Table-Two(1,3,1)、Table-Two(1,3,2)、Table-Two(2,3,1)、および Table-Two(2,3,2) の値を合計します。

```
Compute Table-Sum = FUNCTION SUM (Table-Two(ALL, 3, ALL))
```

次の例は、全従業員のさまざまな給与値を計算します。従業員の給与は Employee-Table にエンコードされています。

```
01  Employee-Table.
    05 Emp-Count      Pic s9(4) usage binary.
    05 Emp-Record     Occurs 1 to 500 times
                        depending on Emp-Count.
       10 Emp-Name    Pic x(20).
       10 Emp-Idme    Pic 9(9).
       10 Emp-Salary  Pic 9(7)v99.
. . .
Procedure Division.
    Compute Max-Salary    = Function Max(Emp-Salary(ALL))
    Compute I             = Function Ord-Max(Emp-Salary(ALL))
    Compute Avg-Salary    = Function Mean(Emp-Salary(ALL))
    Compute Salary-Range  = Function Range(Emp-Salary(ALL))
    Compute Total-Payroll = Function Sum(Emp-Salary(ALL))
```

第 5 章 プログラム・アクションの選択と反復

COBOL 制御言語を使用すると、論理テストの結果に基づいてプログラム・アクションを選択すること、プログラムおよびデータの選択された部分を繰り返すこと、および 1 つのグループとして実行すべきステートメントを識別することができます。

これらの制御には、IF、EVALUATE、および PERFORM ステートメントと、スイッチおよびフラグの使用が含まれます。

関連タスク
81 ページの『プログラム・アクションの選択』
89 ページの『プログラム・アクションの繰り返し』

プログラム・アクションの選択

1 つ以上のデータ項目のテストされた値に基づいて、さまざまなプログラム・アクションに備えることができます。

COBOL の IF および EVALUATE ステートメントは、条件式によって 1 つ以上のデータ項目をテストします。

関連タスク
81 ページの『アクションの選択項目のコーディング』
85 ページの『条件式のコーディング』

関連参照

IF ステートメント (*COBOL for Linux on x86* 言語解説書)
EVALUATE ステートメント (*COBOL for Linux on x86* 言語解説書)

アクションの選択項目のコーディング

IF . . . ELSE は、2 つの処理アクション間での選択項目をコーディングする場合に使用します。(THEN という語はオプションです。) 3 つ以上の可能なアクションからいずれかを選択するようにするには、EVALUATE ステートメントを使用します。

```
IF condition-p
  statement-1
ELSE
  statement-2
END-IF
```

2 つの処理選択項目の一方がアクションを取らない場合は、IF ステートメントに ELSE を指定してもしなくてもかまいません。 ELSE 節はオプションであるため、IF ステートメントは次のようにコーディングすることができます。

```
IF condition-q
  statement-1
END-IF
```

このコーディングは、簡単な場合に適しています。 ロジックが複雑になった場合は、おそらく、ELSE 節を使用する必要があります。 例えば、処理選択項目の 1 つだけに対応するアクションがある、ネストされた IF ステートメントが あるとします。 その場合は、次のように、ELSE 節と CONTINUE ステートメントを使用して IF ステートメントの NULL ブランチをコーディングすることができます。

```
IF condition-q
  statement-1
```

© Copyright IBM Corp. 2021

```
ELSE
  CONTINUE
END-IF
```

EVALUATE ステートメントは IF ステートメントの拡張形式であり、これを使用すると、IF ステートメントのネスト (論理エラーやデバッグ問題の一般的原因となる) を避けることができます。

関連タスク
82 ページの『ネストされた IF ステートメントの使用』
83 ページの『EVALUATE ステートメントの使用』
85 ページの『条件式のコーディング』

ネストされた IF ステートメントの使用

IF ステートメントが、その可能な分岐の 1 つとして別の IF ステートメントを含んでいるとき、これらの IF ステートメントはネストされている といいます。 論理上は、ネストされた IF ステートメントの深さに制限はありません。

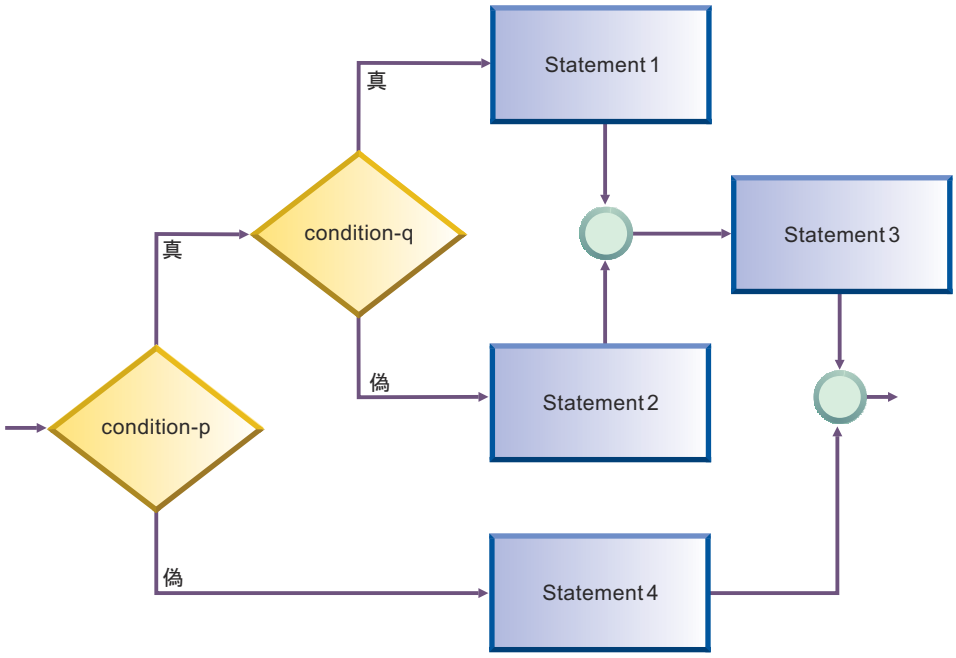
ただし、ネストされた IF ステートメントを多用しないでください。 明示範囲終了符号および字下げが役に立つとはいえ、ロジックをたどるのが困難になる可能性があります。 プログラムが 2 つを超える値について変数をテストしなければならない場合には、おそらく EVALUATE を使用する方が適切です。

以下に、ネストされた IF ステートメントの疑似コードを示します。

```
IF condition-p
  IF condition-q
    statement-1
  ELSE
    statement-2
  END-IF
  statement-3
ELSE
  statement-4
END-IF
```

上記の疑似コードでは、IF ステートメントと順次構造が外側の IF の 1 つの分岐にネストされています。このような構造では、ネストされた IF を閉じる END-IF が非常に重要になります。 ピリオドは外側の IF 構造も終了させるため、ピリオドではなく END-IF を使用してください。

次の図は、上記の疑似コードの論理構造を示しています。



関連タスク
81 ページの『アクションの選択項目のコーディング』

関連参照
明示範囲終了符号 (*COBOL for Linux on x86* 言語解説書)

EVALUATE ステートメントの使用

ネストされた一連の IF ステートメントではなく、EVALUATE ステートメントを使用して、いくつかの条件をテストし、かつ それぞれについて異なるアクションを指定できます。 したがって、EVALUATE ステートメントを使用すると、ケース構造 またはデシジョン・テーブルをインプリメントすることができます。

また以下の例に示されているように、EVALUATE ステートメントを使用して、複数の条件で同じ処理が行われるようにすることもできます。

84 ページの『例: THRU 句を使用した EVALUATE』
84 ページの『例: 複数の WHEN 句を使用する EVALUATE』

EVALUATE ステートメントでは、WHEN 句の前のオペランドは選択サブジェクト と呼ばれ、WHEN 句の中のオペランドは選択サブジェクト と呼ばれます。 選択サブジェクトは、ID、リテラル、条件式、あるいはワード TRUE または FALSE にすることができます。 選択オブジェクトは、ID、リテラル、条件式、算術式、あるいはワード TRUE、FALSE、または ANY にすることができます。

複数の選択サブジェクトを ALSO 句で分離することができます。 複数の選択オブジェクトも ALSO 句で分離することができます。 それぞれの選択オブジェクト・セット内の選択オブジェクトの数は、次の例に示されているように、選択サブジェクトの数と等しくする必要があります。

85 ページの『例: 複数の条件をテストする EVALUATE』

選択オブジェクト内に現れる ID、リテラル、または算術式は、選択サブジェクト・セット内の対応するオペランドと比較できるよう、有効なオペランドにする必要があります。 選択オブジェクト内に現れる条件あるいはワード TRUE または FALSE は、選択サブジェクト・セット内の条件式あるいはワード TRUE または FALSE に対応していなければなりません。 (選択オブジェクトとしてワード ANY を使用すると、任意のタイプの選択サブジェクトに対応付けることができます。)

EVALUATE ステートメントの実行は、次のいずれかの条件が発生すると終了します。

- 選択された WHEN 句に関連付けられているステートメントが実行された。
- WHEN OTHER 句に関連付けられているステートメントが実行された。
- いずれの WHEN 条件も満たされない。

WHEN 句は、ソース・プログラムに現れている順序どおりにテストされます。 ですから、最高のパフォーマンスが得られるようにこれらの句を順序付ける必要があります。 最初に、適合する可能性が最も高い選択オブジェクトを含んでいる WHEN 句をコーディングし、その後、次に可能性の高いものという順にコーディングしてください。 例外は WHEN OTHER 句です。これは最後に置かなければなりません。

関連タスク
81 ページの『アクションの選択項目のコーディング』

関連参照

EVALUATE ステートメント (*COBOL for Linux on x86* 言語解説書)
一般比較条件 (*COBOL for Linux on x86* 言語解説書)

例: *THRU* 句を使用した *EVALUATE*

この例は、THRU 句をコーディングすることにより、いくつかの条件をある範囲の値でコーディングして同じ処理が行われるようにする方法を示しています。 THRU 句内のオペランドは、同じクラスにする必要があります。

この例では、CARPOOL-SIZE は選択サブジェクト であり、1、 2、および 3 THRU 6 は選択オブジェクトです。

```
EVALUATE CARPOOL-SIZE
  WHEN 1
    MOVE "SINGLE" TO PRINT-CARPOOL-STATUS
  WHEN 2
    MOVE "COUPLE" TO PRINT-CARPOOL-STATUS
  WHEN 3 THRU 6
    MOVE "SMALL GROUP" TO PRINT-CARPOOL STATUS
  WHEN OTHER
    MOVE "BIG GROUP" TO PRINT-CARPOOL STATUS
END-EVALUATE
```

次のネストされた IF ステートメントも同じロジックを表します。

```
IF CARPOOL-SIZE = 1 THEN
  MOVE "SINGLE" TO PRINT-CARPOOL-STATUS
ELSE
  IF CARPOOL-SIZE = 2 THEN
    MOVE "COUPLE" TO PRINT-CARPOOL-STATUS
  ELSE
    IF CARPOOL-SIZE >= 3 and CARPOOL-SIZE <= 6 THEN
      MOVE "SMALL GROUP" TO PRINT-CARPOOL-STATUS
    ELSE
      MOVE "BIG GROUP" TO PRINT-CARPOOL-STATUS
    END-IF
  END-IF
END-IF
```

例: 複数の *WHEN* 句を使用する *EVALUATE*

次の例は、いくつかの条件で同じ処置が行われるようにしなければならない場合は複数の WHEN 句をコーディングできることを示しています。 この方法は、THRU 句のみを使用する場合と比べてさらに柔軟性があります。条件が、ある範囲の値に評価される必要もなく、また同じクラスを持つ必要もないからです。

```
EVALUATE MARITAL-CODE
  WHEN "M"
    ADD 2 TO PEOPLE-COUNT
  WHEN "S"
  WHEN "D"
  WHEN "W"
    ADD 1 TO PEOPLE-COUNT
END-EVALUATE
```

次のネストされた IF ステートメントも同じロジックを表します。

```
IF MARITAL-CODE = "M" THEN
  ADD 2 TO PEOPLE-COUNT
ELSE
  IF MARITAL-CODE = "S" OR
     MARITAL-CODE = "D" OR
     MARITAL-CODE = "W" THEN
       ADD 1 TO PEOPLE-COUNT
  END-IF
END-IF
```

例: 複数の条件をテストする *EVALUATE*

この例は、ALSO 句を使用して、2 つの選択サブジェクトを分離し (True ALSO True)、それぞれの選択オブジェクト・セット内の対応する 2 つの選択オブジェクトを分離する (例えば、When A + B < 10 Also C = 10) 方法を示しています。

WHEN 句の中の選択オブジェクトはどちらも、関連した処置が実行される前に、TRUE、TRUE 条件を満たす必要があります。 両方のオブジェクトが TRUE に評価されなければ、次の WHEN 句が処理されます。

```
Identification Division.
  Program-ID. MiniEval.
Environment Division.
  Configuration Section.
Data Division.
  Working-Storage Section.
  01   Age             Pic  999.
  01   Sex             Pic  X.
  01   Description     Pic  X(15).
  01   A               Pic  999.
  01   B               Pic  9999.
  01   C               Pic  9999.
  01   D               Pic  9999.
  01   E               Pic  99999.
  01   F               Pic  999999.
Procedure Division.
  PN01.
    Evaluate True Also True
      When Age < 13 Also Sex = "M"
        Move "Young Boy" To Description
      When Age < 13 Also Sex = "F"
        Move "Young Girl" To Description
      When Age > 12 And Age < 20 Also Sex = "M"
        Move "Teenage Boy" To Description
      When Age > 12 And Age < 20 Also Sex = "F"
        Move "Teenage Girl" To Description
      When Age > 19 Also Sex = "M"
        Move "Adult Man" To Description
      When Age > 19 Also Sex = "F"
        Move "Adult Woman" To Description
      When Other
        Move "Invalid Data" To Description
    End-Evaluate
    Evaluate True Also True
      When A + B < 10 Also C = 10
        Move "Case 1" To Description
      When A + B > 50 Also C = ( D + E ) / F
        Move "Case 2" To Description
      When Other
        Move "Case Other" To Description
    End-Evaluate
    Stop Run.
```

条件式のコーディング

IF および EVALUATE ステートメントを 使用して、条件式の真理値に従って実行されるプログラム・アクションをコーディングすることができます。

以下の条件を指定できます。

- 次のような比較条件
  - 数値比較
  - 英数字比較
  - DBCS 比較
  - 国別比較
- クラス条件 (データ項目が次の条件に当てはまるかどうかのテストなど)
  - IS NUMERIC
  - IS ALPHABETIC

  - IS ALPHABETIC-LOWER
  - IS ALPHABETIC-UPPER
  - IS DBCS
  - IS KANJI
- 条件名条件 (定義した条件変数の値のテスト)
- 符号条件 (数値オペランドが IS POSITIVE、NEGATIVE、または ZERO の条件に当てはまるかどうかのテスト)
- 切り替え状況条件 (SPECIAL-NAMES 段落で名前を付けた UPSI スイッチの状況のテスト)
- 次のような複合条件
  - 否定条件。NOT (A IS EQUAL TO B) など
  - 複合条件 (論理演算子 AND または OR を組み合わせた条件)

関連概念
86 ページの『スイッチおよびフラグ』

関連タスク
87 ページの『スイッチおよびフラグの定義』
88 ページの『スイッチとフラグのリセット』
48 ページの『非互換データの検査 (数値のクラス・テスト)』
194 ページの『国別 (UTF-16) データの比較』
200 ページの『有効な DBCS 文字に関するテスト』

関連参照
302 ページの『UPSI』
一般比較条件 (*COBOL for Linux on x86* 言語解説書)
クラス条件 (*COBOL for Linux on x86* 言語解説書)
条件名項目の規則 (*COBOL for Linux on x86* 言語解説書)
符号条件 (*COBOL for Linux on x86* 言語解説書)
複合条件 (*COBOL for Linux on x86* 言語解説書)

スイッチおよびフラグ

プログラム中のいくつかの決定は、データ項目の値が真か偽か、オンかオフか、はいかいいえかに基づきます。スイッチとして働くレベル 88 項目に意味のある名前 (条件名) を付けて定義して、これらの両方向決定を制御してください。

その他のプログラム決定は、データ項目の特定の値または値の範囲に依存します。 フィールドにオンまたはオフ以外の値を与えるために条件名を使用するときには、そのフィールドはフラグ と呼ばれるのが普通です。

フラグおよびスイッチを使用すると、コードの変更が容易になります。 条件の値を変更する必要がある場合は、そのレベル 88 条件名の値を変更するだけで済みます。

例えば、特定の給与範囲についてフィールドをテストするために、プログラムが条件名を使用するとします。 別の給与範囲を検査するようにプログラムを変更しなければならない場合は、DATA DIVISION 内の条件名の値を変更するだけで済みます。 PROCEDURE DIVISION で変更を行う必要はありません。

関連タスク
87 ページの『スイッチおよびフラグの定義』
88 ページの『スイッチとフラグのリセット』

スイッチおよびフラグの定義

DATA DIVISION では、スイッチまたはフラグとして機能するレベル 88 項目を定義して、それらに分かりやすい名前を与えます。

フラグを持つ 2 つを超える値をテストするには、複数のレベル 88 項目を使用することによって、フィールドに複数の条件名を割り当ててください。

意味のある条件名が選択されており、かつそれらに割り当てられた値が論理値に関連付けられているならば、プログラムを読むときコードを追跡するのが容易になります。

87 ページの『例: スイッチ』
87 ページの『例: フラグ』

例: スイッチ

以下の例は、レベル 88 項目を使用してプログラム内のさまざまな 2 進値 (オン/オフ) 条件をテストする方法を示しています。

例えば、Transaction-File という名前の入力ファイルのファイル終了 (EOF) 条件をテストするには、データ定義を以下のように記述できます。

```
WORKING-STORAGE SECTION.
01  Switches.
    05  Transaction-EOF-Switch  Pic X  value space.
        88  Transaction-EOF     value "y".
```

レベル 88 記述では、Transaction-EOF-Switch の値が「y」の場合に Transaction-EOF という名前の条件が有効になることが指定されています。PROCEDURE DIVISION 内で Transaction-EOF を参照することは、Transaction-EOF-Switch = "y" をテストすることと同じ条件を表します。 例えば、次のステートメントによって報告書が印刷されるのは、Transaction-EOF-Switch が 'y' に設定されている場合に限られます。

```
If Transaction-EOF Then
    Perform Print-Report-Summary-Lines
End-if
```

例: フラグ

以下の例は、EVALUATE ステートメントと一緒にいくつかのレベル 88 項目を使用して、プログラム内のいくつかある条件のうちどれが真であるかを判別する方法を示しています。

例えば、メイン・ファイルを更新するプログラムを考えてみましょう。更新内容は、トランザクション・ファイルから読み取られます。 ファイル内のレコードには、3 つの機能 (追加、変更、または削除) のうちどれを実行するかを指示するフィールドが入っています。 入力ファイルのレコード記述で、レベル 88 項目を使用して機能コード用のフィールドをコーディングします。

```
01  Transaction-Input Record
    05  Transaction-Type         Pic X.
        88  Add-Transaction      Value "A".
        88  Change-Transaction   Value "C".
        88  Delete-Transaction   Value "D".
```

これらの条件名をテストしてどの機能が実行されるかを判別するための、 PROCEDURE DIVISION 内のコードは、次のようになります。

```
Evaluate True
  When Add-Transaction
    Perform Add-Main-Record-Paragraph
  When Change-Transaction
    Perform Update-Existing-Record-Paragraph
```

```
  When Delete-Transaction
    Perform Delete-Main-Record-Paragraph
End-Evaluate
```

スイッチとフラグのリセット

プログラムの随所で、スイッチまたはフラグを、それらのデータ記述における元の値にリセットすることが必要になる場合があります。 そのためには、SET ステートメントを使用するか、データ項目をスイッチまたはフラグに移動するように定義します。

SET *condition-name* TO TRUE ステートメントを使用すると、スイッチまたはフラグは、データ記述の中で割り当てられた元の値に設定されます。 複数の値を持つレベル 88 項目の場合、SET *condition-name* TO TRUE は、最初の値 (次の例では A) を割り当てます。

```
88 Record-is-Active Value "A" "O" "S"
```

SET ステートメントと意味のある条件名を使用すれば、他のプログラマーにもコードが容易に追跡できるようになります。

88 ページの『例: スイッチをオンに設定する』
89 ページの『例: スイッチをオフに設定する』

例: スイッチをオンに設定する

条件名値を条件変数に移行する SET ステートメントをコーディングすることでスイッチをオンにする方法を以下に例示します。

例えば、次の例にある SET ステートメントは、ステートメント `Move "y" to Transaction-EOF-Switch` をコーディングした場合と同じ働きをします。

```
01  Switches
    05  Transaction-EOF-Switch   Pic X  Value space.
        88  Transaction-EOF             Value "y".
. . .
Procedure Division.
000-Do-Main-Logic.
    Perform 100-Initialize-Paragraph
    Read Update-Transaction-File
      At End Set Transaction-EOF to True
    End-Read
```

次の例では、入力レコードのトランザクション・コードに基づいて、出力レコード内のフィールドに値を割り当てる方法を示します。

```
01  Input-Record.
    05  Transaction-Type         Pic X(9).
01  Data-Record-Out.
    05  Data-Record-Type         Pic X.
        88 Record-Is-Active           Value "A".
        88 Record-Is-Suspended        Value "S".
        88 Record-Is-Deleted          Value "D".
    05  Key-Field                Pic X(5).
. . .
Procedure Division.
    Evaluate Transaction-Type of Input-Record
      When "ACTIVE"
        Set Record-Is-Active to TRUE
      When "SUSPENDED"
        Set Record-Is-Suspended to TRUE
      When "DELETED"
        Set Record-Is-Deleted to TRUE
    End-Evaluate
```

例: スイッチをオフに設定する

条件名値を条件変数に移行する MOVE ステートメントをコーディングすることでスイッチをオフにする方法を以下に例示します。

例えば、次のコードのように、SWITCH-OFF というデータ項目を使用してオン/オフ・スイッチをオフに設定できます。そうすると、ファイルの終わりに達していないことを示すようにスイッチがリセットされます。

```
01  Switches
    05  Transaction-EOF-Switch       Pic X  Value space.
        88  Transaction-EOF                 Value "y".
01  SWITCH-OFF                       Pic X  Value "n".
. . .
Procedure Division.
    . . .
    Move SWITCH-OFF to Transaction-EOF-Switch
```

プログラム・アクションの繰り返し

PERFORM ステートメントを使用すると、指定された回数だけ同じコードを繰り返すか (つまりループ)、判断の結果に基づいてループすることができます。

また、PERFORM ステートメントを使用すると、段落を実行し、その後で次の実行可能ステートメントに暗黙的に制御権を戻すようにすることもできます。 実際には、この PERFORM ステートメントは、プログラムの異なる多くの部分から入ることができる閉じたサブルーチンをコーディングするための手段です。

PERFORM ステートメントはインラインまたはライン外にすることができます。

関連タスク
89 ページの『インラインまたはライン外 PERFORM の選択』
90 ページの『ループのコーディング』
91 ページの『テーブルのループ処理』
91 ページの『複数の段落またはセクションの実行』

関連参照
PERFORM ステートメント (*COBOL for Linux on x86* 言語解説書)

インラインまたはライン外 PERFORM の選択

インライン PERFORM は、プログラムの通常フローで実行される命令ステートメントです。ライン外 PERFORM は、指定された段落への分岐およびその段落からの暗黙の戻りを引き起こします。

インラインまたはライン外のいずれの PERFORM ステートメントをコーディングするかを決定するには、以下の点を考慮してください。

- PERFORM ステートメントを複数の場所で使用しますか。

  プログラム内のいくつかの場所で同じコード部分を使用したい場合、ライン 外 PERFORM を使用してください。
- どちらのステートメントの配置が読みやすいですか。

  実行するコードが短い場合は、インライン PERFORM の方が読みやすくなります。 ただし、コードがいくつもの画面にわたる場合は、ライン外 PERFORM を使用した方が、プログラムのロジック・フローは分かりやすくなります。 (ただし、構造化プログラミングの各段落は 1 つの論理機能を実行するようにする必要があります。)
- 効率性を優先させますか。

  インライン PERFORM の場合は、ライン外 PERFORM で発生する分岐のオーバーヘッドが避けられます。しかし、ライン外 PERFORM コーディングでもコード最適化を利用できるので、効率性を過度に重要視する必要はありません。

1974 COBOL 標準では、PERFORM ステートメントはライン外であり、このため、別の手順への分岐と暗黙の戻りが必要になります。実行された手順が、プログラムのそれ以降の順次フローの中にある場合は、ロジック・フローの中でもう一度実行されます。この追加の実行を回避するためには、手順を通常の順次フローの外側 (例えば、GOBACK の後) に置くか、または手順のそばに分岐をコーディングしてください。

インライン PERFORM のサブジェクトは、命令ステートメントです。 したがって、インライン PERFORM 内のステートメント (命令ステートメント以外) は、明示範囲終了符号を付けてコーディングしなければなりません。

90 ページの『例: インライン PERFORM ステートメント』

例: インライン PERFORM ステートメント

この例は、必須の範囲終了符号と必須の END-PERFORM 句を持つインライン PERFORM ステートメントの構造を示しています。

```
   Perform 100-Initialize-Paragraph
* The following statement is an inline PERFORM:
   Perform Until Transaction-EOF
      Read Update-Transaction-File Into WS-Transaction-Record
         At End
            Set Transaction-EOF To True
         Not At End
            Perform 200-Edit-Update-Transaction
            If No-Errors
               Perform 300-Update-Commuter-Record
            Else
               Perform 400-Print-Transaction-Errors
* End-If is a required scope terminator
            End-If
            Perform 410-Re-Initialize-Fields
* End-Read is a required scope terminator
      End-Read
   End-Perform
```

ループのコーディング

PERFORM . . . TIMES ステートメントは、手順を指定された回数だけ実行する場合に使用します。

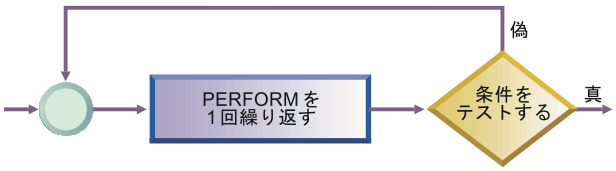
```
PERFORM 010-PROCESS-ONE-MONTH 12 TIMES
INSPECT . . .
```

上記の例では、制御が PERFORM ステートメントに達すると、手順 010-PROCESS-ONE-MONTH のコードが 12 回実行されてから、制御が INSPECT ステートメントに移ります。

PERFORM . . . UNTIL ステートメントは、選択した条件が満たされるまで手順を実行する場合に使用します。 以下のいずれかの形式を使用することができます。

```
PERFORM . . . WITH TEST AFTER  . . . . UNTIL . . .
PERFORM . . . [WITH TEST BEFORE] . . . UNTIL . . .
```

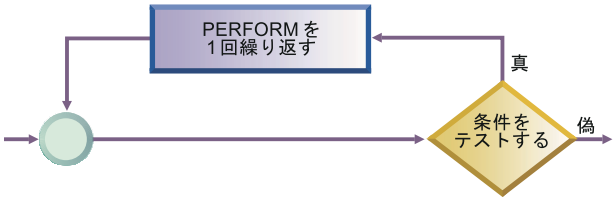
PERFORM . . . WITH TEST AFTER . . . UNTIL ステートメントは、手順を少なくとも 1 回実行し、その後テストしてからそれ以降を実行したい場合に使用します。このステートメントは、do-until 構造と同等です。



次の例では、暗黙の WITH TEST BEFORE 句によって do-while 構造が提供されます。

```
PERFORM  010-PROCESS-ONE-MONTH
  UNTIL MONTH GREATER THAN 12
INSPECT . . .
```

制御が PERFORM ステートメントに達すると、条件 MONTH GREATER THAN 12 がテストされます。 条件が満たされると、制御が INSPECT ステートメントに移ります。 条件が満たされない場合には、010-PROCESS-ONE-MONTH が実行され、条件が再度テストされます。 このサイクルは、条件が真になるまで継続されます。 (プログラムを読みやすくするために、WITH TEST BEFORE 節をコーディングすることが必要な場合もあります。)



テーブルのループ処理

PERFORM . . . VARYING ステートメントを使用してテーブルを初期化することができます。 この形式の PERFORM ステートメントでは、条件が満たされるまで変数が増加または減少され、テストされます。

そのあと、PERFORM ステートメントを使用して、テーブルを操作するループを制御することができます。 以下のいずれかの形式を使用することができます。

```
PERFORM . . . WITH TEST AFTER  . . . . VARYING . . . UNTIL . . .
PERFORM . . . [WITH TEST BEFORE] . . . VARYING . . . UNTIL . . .
```

以下のコードのセクションは、テーブル全体をループ処理して無効データがないか検査する例を示しています。

```
PERFORM TEST AFTER VARYING WS-DATA-IX
      FROM 1 BY 1 UNTIL WS-DATA-IX = 12
   IF WS-DATA (WS-DATA-IX) EQUALS SPACES
      SET SERIOUS-ERROR TO TRUE
      DISPLAY ELEMENT-NUM-MSG5
   END-IF
END-PERFORM
INSPECT . . .
```

上記の PERFORM ステートメントに制御が達すると、WS-DATA-IX は 1 に設定され、PERFORM ステートメントが実行されます。 その後、条件 WS-DATA-IX = 12 がテストされます。 条件が真である場合には、制御が INSPECT ステートメントに渡ります。 条件が偽である場合、WS-DATA-IX が 1 だけ増やされて、PERFORM ステートメントが実行され、条件が再度テストされます。 この実行とテストのサイクルは、WS-DATA-IX が 12 になるまで継続されます。

上記のループは、項目 WS-DATA の 12 個のフィールドに関する入力検査を制御します。 アプリケーションでは空のフィールドは許可されません。ですから、コードのセクションはループし、必要に応じてエラー・メッセージを発行します。

複数の段落またはセクションの実行

構造化プログラミングでは、通常、単一の段落を実行します。 ただし、PERFORM . . . THRU ステートメントをコーディングすれば、 段落グループ、単一セクション、またはセクション・グループを実行できます。

PERFORM . . . THRU ステートメントを使用するときには、段落 EXIT ステートメントをコーディングして、一連の段落のエンドポイントを明確に示してください。

関連タスク
79 ページの『組み込み関数を使用したテーブル項目の処理』

関連参照
EXIT PERFORM または EXIT PERFORM CYCLE ステートメント
(*COBOL for Linux on x86* 言語解説書)
EXIT PARAGRAPH または EXIT SECTION ステートメント
(*COBOL for Linux on x86* 言語解説書)

第 6 章 ストリングの処理

COBOL は、ストリング・データ項目に対して多種類の操作を実行するための言語構造体を提供します。

例えば、次のようなことが可能です。

- データ項目の結合または分割。
- ヌル終了ストリングの操作 (文字のカウントや移動など)。
- 通常の位置 (必要があれば、および長さ) によるサブストリングへの参照。
- データ項目の計算および置換 (データ項目内に特定文字が現れた回数のカウントなど)。
- データ項目の変換 (大文字または小文字への変更など)。
- データ項目の評価 (データ項目の長さの判別など)。

関連タスク
93 ページの『データ項目の結合 (STRING)』
95 ページの『データ項目の分割 (UNSTRING)』
98 ページの『ヌル終了ストリングの取り扱い』
99 ページの『データ項目のサブストリングの参照』
102 ページの『データ項目の計算および置換 (INSPECT)』
104 ページの『データ項目の変換 (組み込み関数)』
106 ページの『データ項目の評価 (組み込み関数)』
179 ページの『第 10 章 国際環境でのデータの処理』

データ項目の結合 (STRING)

STRING ステートメントは、いくつかのデータ項目またはリテラルの、すべてまたは一部を 1 つのデータ項目に結合する場合に使用します。 1 つの STRING ステートメントを複数の MOVE ステートメントの代わりに使用できます。

STRING ステートメントは、示された順序でデータを受信データ項目に転送します。 STRING ステートメントでは、以下のものも指定します。

- 送信フィールド・セットごとの区切り文字。検出されると、これにより送信フィールドの転送は停止されます (DELIMITED BY 句)
- (オプション) すべての送信データが処理される前に受信フィールドが満杯になっている場合に実行する処置 (ON OVERFLOW 句)
- (オプション) データの転送先である受信フィールド内の左端文字位置を示す、整数データ項目 (WITH POINTER 句)

受信データ項目を編集項目にしてはなりません。また表示浮動小数点項目や国別浮動小数点項目にしてもなりません。 受信データ項目が何を持っているかによって次のような違いが生じます。

- USAGE DISPLAY を持っている場合、ステートメント内のそれぞれの ID は (POINTER ID を除いて) USAGE DISPLAY を持っている必要があり、ステートメント内のそれぞれのリテラルは英数字でなければなりません。
- USAGE NATIONAL を持っている場合、ステートメント内のそれぞれの ID は (POINTER ID を除いて) USAGE NATIONAL を持っている必要があり、ステートメント内のそれぞれのリテラルは国別でなければなりません。
- USAGE DISPLAY-1 を持っている場合、ステートメント内のそれぞれの ID は (POINTER ID を除いて) USAGE DISPLAY-1 を持っている必要があり、ステートメント内のそれぞれのリテラルは DBCS でなければなりません。

STRING ステートメントによってデータが書き込まれる、受信フィールドの特定部分のみが変更されます。

94 ページの『例: STRING ステートメント』

© Copyright IBM Corp. 2021

関連タスク
167 ページの『ストリングの結合および分割におけるエラーの処理』

関連参照

STRING ステートメント (*COBOL for Linux on x86* 言語解説書)

例: STRING ステートメント

次の例は、レコードから情報を選択して出力行にフォーマットする STRING ステートメントを示しています。

FILE SECTION は以下のレコードを定義します。

```
01  RCD-01.
    05  CUST-INFO.
        10  CUST-NAME    PIC X(15).
        10  CUST-ADDR    PIC X(35).
    05  BILL-INFO.
        10  INV-NO       PIC X(6).
        10  INV-AMT      PIC $$,$$$.99.
        10  AMT-PAID     PIC $$,$$$.99.
        10  DATE-PAID    PIC X(8).
        10  BAL-DUE      PIC $$,$$$.99.
        10  DATE-DUE     PIC X(8).
```

WORKING-STORAGE SECTION は以下の各フィールドを定義します。

```
77  RPT-LINE             PIC X(120).
77  LINE-POS             PIC S9(3).
77  LINE-NO              PIC 9(5) VALUE 1.
77  DEC-POINT            PIC X VALUE ".".
```

レコード RCD-01 には、以下の情報が含まれています (記号 *b* はブランク・スペースを示します)。

```
J.B.bSMITHbbbbb
444bSPRINGbST.,bCHICAGO,bILL.bbbbbb
A14275
$4,736.85
$2,400.00
09/22/76
$2,336.85
10/22/76
```

PROCEDURE DIVISION では、以下の設定値は STRING ステートメントの前にきます。

- RPT-LINE は SPACES に設定されます。
- LINE-POS (POINTER フィールドとして使用されるデータ項目) は 4 に設定されます。

STRING ステートメントを以下に示します。

```
STRING
   LINE-NO SPACE CUST-INFO INV-NO SPACE DATE-DUE SPACE
      DELIMITED BY SIZE
   BAL-DUE
      DELIMITED BY DEC-POINT
   INTO RPT-LINE
   WITH POINTER LINE-POS.
```

STRING ステートメントが実行される前、POINTER フィールドの LINE-POS は値 4 を持っているので、データは 受信フィールド RPT-LINE に移動されるとき、文字位置 4 から開始されます。位置 1 から 3 の文字は未変更のままです。

DELIMITED BY SIZE を指定している送信項目は、その全体が受信フィールドに移動されます。 BAL-DUE は DEC-POINT で区切られているので、受信フィールドへの BAL-DUE の移動は、小数点 (DEC-POINT の値) が検出されると停止します。

STRING の結果

STRING ステートメントが実行されると、次の表に示されるように、項目は RPT-LINE に移動されます。

項目	位置
LINE-NO	4 - 8
スペース	9
CUST-INFO	10 - 59
INV-NO	60 - 65
スペース	66
DATE-DUE	67 - 74
スペース	75
BAL-DUE の小数点より前の部分	76 - 81

STRING ステートメントの実行後、LINE-POS の値は 82 であり、RPT-LINE は以下に示す値を持ちます。

```
Column
4    10                                        60     67      76
|    |                                         |      |       |
00001 J.B. SMITH     444 SPRING ST., CHICAGO, ILL.     A14275 10/22/76 $2,336
```

データ項目の分割 (UNSTRING)

UNSTRING ステートメントは、送信フィールドを複数の受信フィールドに分割するために使用します。 1 つの UNSTRING ステートメントを複数の MOVE ステートメントの代わりに使用できます。

UNSTRING ステートメントでは、以下のものを指定することができます。

- 区切り文字。区切り文字の 1 つが送信フィールドで検出されると、現行受信フィールドは受け取りを停止し、次のフィールド (ある場合) が受け取りを開始します (DELIMITED BY 句)
- 区切り文字用のフィールド。送信フィールドで区切り文字が検出されると、現行受信フィールドは受け取りを停止します (DELIMITER IN 句)
- 現行受信フィールドに入れられた文字の数を保管する整数データ項目 (COUNT IN 句)
- UNSTRING 処理が開始される送信フィールド内の左端文字位置を示す、整数データ項目 (WITH POINTER 句)
- 操作対象の受信フィールドの数の計算値を保管する、整数データ項目 (TALLYING IN 句)
- 送信データ項目の最後に達する前にすべての受信フィールドが満杯になった場合に実行する処置 (ON OVERFLOW 句)

送信データ項目および DELIMITED BY 句の区切り文字は、カテゴリー英字、英数字、英数字編集、DBCS、国別、または国別編集にする必要があります。

受信データ項目は、カテゴリー英字、英数字、数値、DBCS、または国別にすることができます。 数値の受信データ項目は、ゾーン 10 進数または国別 10 進数にする必要があります。 受信データ項目が何を持っているかによって次のような違いが生じます。

- USAGE DISPLAY を持っている場合、送信項目およびステートメント内のそれぞれの区切り文字項目は USAGE DISPLAY を持っている必要があり、ステートメント内のそれぞれのリテラルは英数字でなければなりません。
- USAGE NATIONAL を持っている場合、送信項目およびステートメント内のそれぞれの区切り文字項目は USAGE NATIONAL を持っている必要があり、ステートメント内のそれぞれのリテラルは国別でなければなりません
- USAGE DISPLAY-1 を持っている場合、送信項目およびステートメント内のそれぞれの区切り文字項目は USAGE DISPLAY-1 を持っている必要があり、ステートメント内のそれぞれのリテラルは DBCS でなければなりません

96 ページの『例: UNSTRING ステートメント』

関連概念
180 ページの『Unicode および言語文字のエンコード』

関連タスク
167 ページの『ストリングの結合および分割におけるエラーの処理』

関連参照

UNSTRING ステートメント (*COBOL for Linux on x86* 言語解説書)
データのクラスおよびカテゴリー (*COBOL for Linux on x86* 言語解説書)

例: UNSTRING ステートメント

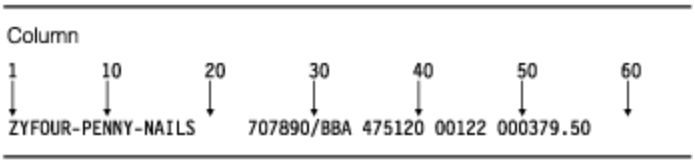
次の例は、選択した情報を入力レコードから転送する UNSTRING ステートメントを示しています。 情報の中には、印刷用に編成されているものもあれば、その後の処理用に編成されているものもあります。

FILE SECTION は以下のレコードを定義します。

```
*  Record to be acted on by the UNSTRING statement:
 01  INV-RCD.
     05  CONTROL-CHARS               PIC XX.
     05  ITEM-INDENT                 PIC X(20).
     05  FILLER                      PIC X.
     05  INV-CODE                    PIC X(10).
     05  FILLER                      PIC X.
     05  NO-UNITS                    PIC 9(6).
     05  FILLER                      PIC X.
     05  PRICE-PER-M                 PIC 99999.
     05  FILLER                      PIC X.
     05  RTL-AMT                     PIC 9(6).99.
*
*  UNSTRING receiving field for printed output:
 01  DISPLAY-REC.
     05  INV-NO                      PIC X(6).
     05  FILLER                      PIC X VALUE SPACE.
     05  ITEM-NAME                   PIC X(20).
     05  FILLER                      PIC X VALUE SPACE.
     05  DISPLAY-DOLS                PIC 9(6).
*
*  UNSTRING receiving field for further processing:
 01  WORK-REC.
     05  M-UNITS                     PIC 9(6).
     05  FIELD-A                     PIC 9(6).
     05  WK-PRICE REDEFINES FIELD-A  PIC 9999V99.
     05  INV-CLASS                   PIC X(3).
*
*  UNSTRING statement control fields:
 77  DBY-1                           PIC X.
 77  CTR-1                           PIC S9(3).
 77  CTR-2                           PIC S9(3).
 77  CTR-3                           PIC S9(3).
 77  CTR-4                           PIC S9(3).
 77  DLTR-1                          PIC X.
 77  DLTR-2                          PIC X.
 77  CHAR-CT                         PIC S9(3).
 77  FLDS-FILLED                     PIC S9(3).
```

PROCEDURE DIVISION では、以下の設定値は UNSTRING ステートメントの前にきます。

- ピリオド (.) は、DBY-1 内では区切り文字として配置されます。
- CHAR-CT (POINTER フィールド) は 3 に設定します。
- 値ゼロ (0) を FLDS-FILLED (TALLYING フィールド) に入れます。
- データは読み取られてレコード INV-RCD に入れられます。このレコードのフォーマットを以下のとおりです。



UNSTRING ステートメントを以下に示します。

```
* Move subfields of INV-RCD to the subfields of DISPLAY-REC
* and WORK-REC:
     UNSTRING INV-RCD
       DELIMITED BY ALL SPACES  OR "/"  OR DBY-1
       INTO ITEM-NAME    COUNT IN CTR-1
            INV-NO       DELIMITER IN DLTR-1  COUNT IN CTR-2
            INV-CLASS
            M-UNITS      COUNT IN CTR-3
            FIELD-A
            DISPLAY-DOLS DELIMITER IN DLTR-2  COUNT IN CTR-4
       WITH POINTER CHAR-CT
       TALLYING IN  FLDS-FILLED
       ON OVERFLOW  GO TO UNSTRING-COMPLETE.
```

UNSTRING ステートメントの実行前には POINTER フィールドである CHAR-CT の値は 3 であるので、INV-RCD 内の CONTROL-CHARS フィールドの 2 つの文字位置は無視されます。

UNSTRING の結果

この UNSTRING ステートメントを実行すると、以下のステップで処理が行われます。

1. INV-RCD の桁 3 から 18 (FOUR-PENNY-NAILS) が ITEM-NAME に入れられ、区域内で左寄せされ、未使用の 4 つの文字位置にスペースが埋め込まれます。 値 16 が CTR-1 に入れられます。
2. ALL SPACES が区切り文字としてコーディングされているので、桁 19 から 23 の 5 つの連続スペース文字は 1 つの区切り文字とみなされます。
3. 桁 24 から 29 (707890) が INV-NO に入れられます。 区切り文字のスラッシュ (/) が DLTR-1 に入れられ、値 6 が CTR-2 に入れられます。
4. 桁 31 から 33 (BBA) が INV-CLASS に入れられます。 区切り文字は SPACE ですが、区切り文字の受取域としてフィールドが定義されていないので、桁 34 のスペースは迂回されます。
5. 桁 35 から 40 (475120) が M-UNITS に入れられます。 値 6 が CTR-3 に入れられます。 区切り文字は SPACE ですが、区切り文字の受取域としてフィールドが定義されていないので、桁 41 のスペースは迂回されます。
6. 桁 42 から 46 (00122) が FIELD-A に入れられ、領域内で右寄せされます。 高位桁位置にはゼロ (0) が埋め込まれます。 区切り文字は SPACE ですが、区切り文字の受取域としてフィールドが定義されていないので、桁 47 のスペースは迂回されます。
7. 桁 48 から 53 (000379) が DISPLAY-DOLS に入れられます。 DBY-1 内のピリオド (.) 区切り文字が DLTR-2 に入れられ、値 6 が CTR-4 に入れられます。
8. すべての受信フィールドに対して操作が行われたが、INV-RCD の 2 文字が検査されなかったので、ON OVERFLOW ステートメントが実行されます。 UNSTRING ステートメントの実行は完了です。

UNSTRING ステートメントの実行後、フィールドには以下の値が入っています。

フィールド	値
DISPLAY-REC	707890 FOUR-PENNY-NAILS 000379
WORK-REC	475120000122BBA
CHAR-CT (POINTER フィールド)	55
FLDS-FILLED (TALLYING フィールド)	6

ヌル終了ストリングの取り扱い

さまざまな仕組みを使用して、ヌル終了ストリング (例えば C プログラムとの間で受け渡されるストリング) を構成し、操作することができます。

例えば、次のようなことが可能です。

- ヌル終了リテラル定数 (Z". . . ")。
- INSPECT ステートメントを使用して、ヌル終了ストリング内の文字数をカウントする。

```
MOVE 0 TO char-count
INSPECT source-field TALLYING char-count
                     FOR CHARACTERS
                     BEFORE X"00"
```

- UNSTRING ステートメントを使用して、ヌル終了ストリング内の文字をターゲット・フィールドに移動し、文字カウントを得る。

```
WORKING-STORAGE SECTION.
01  source-field          PIC X(1001).
01  char-count     COMP-5 PIC 9(4).
01  target-area.
    02 individual-char OCCURS 1 TO 1000 TIMES DEPENDING ON char-count
                          PIC X.
. . .
PROCEDURE DIVISION.
    UNSTRING source-field DELIMITED BY X"00"
                          INTO target-area
                          COUNT IN char-count
      ON OVERFLOW
        DISPLAY "source not null terminated or target too short"
    END-UNSTRING
```

- SEARCH ステートメントを使用して、後続ヌルまたはスペース文字を見つける。 検査するストリングを単一文字からなるテーブルとして定義してください。
- ループのフィールドの各文字を検査する (PERFORM)。 フィールドの各文字は、source-field (I:1) のような参照修飾子を使用して検査することができます。

98 ページの『例: ヌル終了ストリング』

関連タスク
451 ページの『ヌル終了ストリングの処理』

関連参照

英数字リテラル (*COBOL for Linux on x86* 言語解説書)

例: ヌル終了ストリング

以下の例は、ヌル終了ストリングを処理できるいくつかの方法を示しています。

```
01  L pic X(20) value z'ab'.
01  M pic X(20) value z'cd'.
01  N pic X(20).
```

```
 01  N-Length pic 99 value zero.
 01  Y pic X(13) value 'Hello, World!'.
  . . .
* Display null-terminated string:
     Inspect N tallying N-length
       for characters before initial x'00'
     Display 'N: ' N(1:N-Length) ' Length: ' N-Length
     . . .
* Move null-terminated string to alphanumeric, strip null:
     Unstring N delimited by X'00' into X
     . . .
* Create null-terminated string:
     String Y     delimited by size
            X'00' delimited by size
            into N.
     . . .
* Concatenate two null-terminated strings to produce another:
     String L     delimited by x'00'
            M     delimited by x'00'
            X'00' delimited by size
            into N.
```

データ項目のサブストリングの参照

参照修飾子を使用することにより、USAGE DISPLAY、DISPLAY-1、または NATIONAL を持つデータ項目のサブストリングを参照します。 参照修飾子を使用すると、組み込み関数によって戻される英数字または国別文字ストリングのサブストリングを参照することもできます。

以下の例は、参照修飾子を使用して、Customer-Record という名前のデータ項目の 20 文字のサブストリングを参照する方法を示しています。

```
Move Customer-Record(1:20) to Orig-Customer-Name
```

データ項目の直後に括弧で囲んだ参照修飾子をコーディングします。 例に示されるように、参照修飾子は、次の順で、コロンで分離された 2 つの値を含むことができます。

1. サブストリングを開始したい文字の序数位置 (左からの)
2. (オプション) 必要なサブストリングの長さ (文字位置)

USAGE DISPLAY を持つ項目の参照修飾子の位置および長さは、1 バイト文字で表されます。 USAGE DISPLAY-1 または NATIONAL を持つ項目の参照修飾子の位置および長さは、それぞれ DBCS 文字位置および国別文字位置で表されます。

参照修飾子の長さを省略すると (先頭文字の序数位置とその後のコロンのみをコーディングすると)、サブストリングは項目の最後まで延長されます。 可能であれば、より単純でエラーになりにくいコーディング手法として、長さを省略してください。

参照修飾子を使用すると、英数字グループ、英数字編集データ項目、数字編集データ項目、表示浮動小数点データ項目、およびゾーン 10 進数データ項目を含め、USAGE DISPLAY データ項目のサブストリングを参照できます。 これらのデータ項目のいずれかを参照修飾した場合、結果はカテゴリー英数字になります。英字データ項目を参照修飾した場合、結果はカテゴリー英字になります。

参照修飾子を使用すると、国別グループ、国別編集データ項目、数字編集データ項目、国別浮動小数点データ項目、および国別 10 進数データ項目を含め、 USAGE NATIONAL データ項目のサブストリングを参照することができます。 これらのデータ項目のいずれかを参照修飾した場合、結果はカテゴリー国別になります。 例えば、次のように国別 10 進数データ項目を定義するとしましょう。

```
01  NATL-DEC-ITEM  Usage National  Pic 999  Value 123.
```

NATL-DEC-ITEM はカテゴリー数値なので、NATL-DEC-ITEM を算術式で使用できます。 しかし、NATL-DEC-ITEM(2:1) (国別文字 2、16 進数表記では NX"3200") はカテゴリー国別なので、これを算術式で使用することはできません。

参照修飾子を使用すると、可変長項目を含め、テーブル項目のサブストリングを参照することができます。テーブル記入項目のサブストリングを参照するには、参照修飾子の前に添え字式をコーディングします。例えば、PRODUCT-TABLE は、正しくコーディングされた 文字ストリング・テーブルであると想定しましょう。 D をテーブル内の 2 番目のストリングの 4 文字目に移動するために、次のステートメントをコーディングできます。

```
MOVE 'D' to PRODUCT-TABLE (2), (4:1)
```

参照修飾子の中の 2 つの値の一方または両方を、変数または算術式として コーディングできます。

101 ページの『例: 参照修飾子としての演算式』

数字関数 ID は、算術式を使用できる場所ならどこでも使用できるので、左端文字位置または長さ (あるいはその両方) として、数字関数 ID を参照修飾子の中でコーディングできます。

102 ページの『例: 参照修飾子としての組み込み関数』

参照修飾子の中のそれぞれの数値は少なくとも 1 の値でなければなりません。 サブストリングの最後を越えて参照することがないよう、2 つの数値の合計が、 データ項目の全長を 2 文字位置以上超えるようなことがあってはなりません。

左端の文字位置または長さ値が固定小数点の非整数の場合には、整数を作成するために切り捨てが行われます。 浮動小数点の非整数の場合には、整数を作成するための丸めが行われます。

以下のオプションを使用すると、範囲外の参照修飾子が検出され、実行時メッセージによって違反が示されます。

- SSRANGE コンパイラー・オプション
- CHECK ランタイム・オプション

関連概念
100 ページの『参照修飾子』
180 ページの『Unicode および言語文字のエンコード』

関連タスク
62 ページの『テーブル内の項目の参照』

関連参照
284 ページの『SSRANGE』
参照変更 (*COBOL for Linux on x86* 言語解説書)
関数定義 (*COBOL for Linux on x86* 言語解説書)

参照修飾子

参照修飾子を使用すると、データ項目のサブストリングを容易に参照できます。

例えば、システムから現在の時刻を取り出して、その値を拡張形式で表示したいとします。 現在の時刻は、ACCEPT ステートメントを使用して取り出すことができます。このステートメントは、次の形式で、時、分、秒、および 100 分の 1 秒を戻します。

```
HHMMSSss
```

しかし、現在の時刻を次の形式で表示したいとします。

```
HH:MM:SS
```

参照修飾子を使用しない場合は、両方の形式についてのデータ項目を定義しなければなりません。 さらに、1 つの形式を別の形式に変換するためのコードも書く必要があります。

参照修飾子を使用する場合は、TIME エレメントを記述するサブフィールドに名前を指定する必要はありません。 必要なデータ定義は、システムによって戻される時刻用のデータ定義だけです。 例えば、次のように指定します。

```
01  REFMOD-TIME-ITEM    PIC X(8).
```

次のコードは、時刻値を取り出して、拡張します。

```
     ACCEPT REFMOD-TIME-ITEM FROM TIME.
     DISPLAY "CURRENT TIME IS: "
* Retrieve the portion of the time value that corresponds to
*   the number of hours:
       REFMOD-TIME-ITEM (1:2)
       ":"
* Retrieve the portion of the time value that corresponds to
*   the number of minutes:
       REFMOD-TIME-ITEM (3:2)
       ":"
* Retrieve the portion of the time value that corresponds to
*   the number of seconds:
       REFMOD-TIME-ITEM (5:2)
```

101 ページの『例: 参照修飾子としての演算式』
102 ページの『例: 参照修飾子としての組み込み関数』

関連タスク
30 ページの『画面またはファイルからの入力の割り当て (ACCEPT)』
99 ページの『データ項目のサブストリングの参照』
180 ページの『COBOL での国別データ (Unicode) の使用』

関連参照
参照変更 (*COBOL for Linux on x86* 言語解説書)

例: 参照修飾子としての演算式

あるフィールドに右揃えされたいくつかの文字が入っている場合に、それらの文字を別のフィールドに移動し、右ではなく左に揃えたいとします。 これは、参照修飾子と INSPECT ステートメントを使用して行うことができます。

プログラムに次のデータが入っているとします。

```
01  LEFTY     PIC X(30).
01  RIGHTY    PIC X(30)  JUSTIFIED RIGHT.
01  I          PIC 9(9)   USAGE BINARY.
```

プログラムは、先行するスペースの数をカウントし、参照修飾子内の算術式を使用して、右揃えされた文字を別のフィールドに移動し、左寄せします。

```
MOVE SPACES TO LEFTY
MOVE ZERO TO I
INSPECT RIGHTY
   TALLYING I FOR LEADING SPACE.
IF I IS LESS THAN LENGTH OF RIGHTY THEN
   MOVE RIGHTY ( I + 1 : LENGTH OF RIGHTY - I ) TO LEFTY
END-IF
```

MOVE ステートメントは、RIGHTY の文字を、I + 1 で計算された位置から、LENGTH OF RIGHTY - I で計算された長さだけ、フィールド LEFTY に移動します。

例: 参照修飾子としての組み込み関数

コンパイル時にサブストリングの左端位置またはその長さを知らない場合、参照修飾子の中で組み込み関数を使用できます。

例えば、以下のコード・フラグメントにより、Customer-Record のサブストリングがデータ項目 WS-name に移動されます。 サブストリングは、実行時に決定されます。

```
05  WS-name        Pic x(20).
05  Left-posn      Pic 99.
05  I              Pic 99.
. . .
Move Customer-Record(Function Min(Left-posn I):Function Length(WS-name)) to WS-name
```

整数関数を使わなければならない位置で非整数関数を使用したい場合、INTEGER または INTEGER-PART 関数を使用して結果を整数に変換できます。 例えば、次のように指定します。

```
Move Customer-Record(Function Integer(Function Sqrt(I)): ) to WS-name
```

関連参照
INTEGER (*COBOL for Linux on x86* 言語解説書)
INTEGER-PART (*COBOL for Linux on x86* 言語解説書)

データ項目の計算および置換 (INSPECT)

INSPECT ステートメントを使用して、データ項目内の文字または文字グループを検査し、必要に応じてそれらを置換します。

INSPECT ステートメントを使用して以下のタスクを行います。

- データ項目内に特定文字が現れる回数をカウント します (TALLYING 句)。
- データ項目またはデータ項目内の選択部分に、指定された文字 (スペース、アスタリスク、またはゼロなど) を充てんします (REPLACING 句)。
- データ項目内に特定文字または文字ストリングがあればそれらすべてを、指定された置換文字に変換します (CONVERTING 句)。

検査する項目として、以下のデータ項目の 1 つを指定できます。

- USAGE DISPLAY、USAGE DISPLAY-1、または USAGE NATIONAL として明示的または暗黙的に記述された基本項目
- 英数字グループ項目または国別グループ項目

検査する項目に何が指定されているかによって次のような違いが生じます。

- USAGE DISPLAY が指定されている場合、ステートメント内のそれぞれの ID は (TALLYING カウント・フィールドを除いて) USAGE DISPLAY が指定されている必要があり、ステートメント内のそれぞれの リテラルは英数字でなければなりません。
- USAGE NATIONAL が指定されている場合、ステートメント内のそれぞれの ID は (TALLYING カウント・フィールドを除いて) USAGE NATIONAL が指定されている必要があり、ステートメント内のそれぞれのリテラルは国別でなければなりません。
- USAGE DISPLAY-1 が指定されている場合、ステートメント内のそれぞれの ID は (TALLYING カウント・フィールドを除いて) USAGE DISPLAY-1 が指定されている必要があり、ステートメント内のそれぞれのリテラルは DBCS リテラルでなけれ ばなりません。

103 ページの『例: INSPECT ステートメント』

関連概念
180 ページの『Unicode および言語文字のエンコード』

関連参照

例: INSPECT ステートメント

以下の例では、文字を検査して置き換える INSPECT ステートメントの使用法をいくつか示します。

次の例では、INSPECT ステートメントは、データ項目 DATA-2 内の文字を検査して置き換えます。 データ項目内に現れる先行ゼロ (0) の数は、累算されて COUNTR に入れられます。 文字 C の最初のインスタンスの後に続く文字 A の最初のインスタンスは、文字 2 に置き換えられます。

```
77  COUNTR            PIC 9   VALUE ZERO.
01  DATA-2            PIC X(11).
. . .
    INSPECT DATA-2
      TALLYING COUNTR FOR LEADING "0"
      REPLACING FIRST "A" BY "2" AFTER INITIAL "C"
```

DATA-2 (実行前)	**COUNTR (実行後)**	**DATA-2 (実行後)**
00ACADEMY00	2	00AC2DEMY00
0000ALABAMA	4	0000ALABAMA
CHATHAM0000	0	CH2THAM0000

次の例では、INSPECT ステートメントは、データ項目 DATA-3 内の文字を検査して置き換えます。 引用符 (") の最初のインスタンスより前にある各文字は、文字 0 で置き換えられます。

```
77  COUNTR            PIC 9   VALUE ZERO.
01  DATA-3            PIC X(8).
. . .
    INSPECT DATA-3
      REPLACING CHARACTERS BY ZEROS BEFORE INITIAL QUOTE
```

DATA-3 (実行前)	**COUNTR (実行後)**	**DATA-3 (実行後)**
456"ABEL	0	000"ABEL
ANDES"12	0	00000"12
"TWAS BR	0	"TWAS BR

次の例では、AFTER 句と BEFORE 句を指定した INSPECT CONVERTING を使用して、データ項目 DATA-4 内の文字を検査して置き換えます。 文字 / の最初のインスタンスより後にあり、文字 ? (もしあれば) の最初のインスタンスより前にあるすべての文字が、小文字から大文字に変換されます。

```
01  DATA-4            PIC X(11).
. . .
    INSPECT DATA-4
      CONVERTING
         "abcdefghijklmnopqrstuvwxyz" TO
         "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
      AFTER INITIAL "/"
      BEFORE INITIAL"?"
```

DATA-4 (実行前)	**DATA-4 (実行後)**
a/five/?six	a/FIVE/?six
r/Rexx/RRRr	r/REXX/RRRR

DATA-4 (実行前)	DATA-4 (実行後)
zfour?inspe	zfour?inspe

データ項目の変換 (組み込み関数)

組み込み関数を使用して、文字ストリング・データ項目を他のいくつかのフォーマットに (例: 大文字または小文字に、逆順に、数字に、あるコード・ページから別のコード・ページに) 変換できます。

NATIONAL-OF および DISPLAY-OF 組み込み関数を使用して、国別 (Unicode) ストリングとの間で変換を行うことができます。

INSPECT ステートメントを使用して文字を変換することもできます。

103 ページの『例: INSPECT ステートメント』

関連タスク
104 ページの『大/小文字の変更 (UPPER-CASE、LOWER-CASE)』
104 ページの『逆順への変換 (REVERSE)』
105 ページの『数値への変換 (NUMVAL、NUMVAL-C)』
106 ページの『あるコード・ページから別のコード・ページへの変換』

大/小文字の変更 (UPPER-CASE、LOWER-CASE)

UPPER-CASE および LOWER-CASE 組み込み関数を使用すれば、英数字、英字、または国別ストリングの大/小文字を容易に変更できます。

```
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
```

上記のコードは、次のメッセージをシステムの論理出力装置に表示します。

```
Hello World!
HELLO WORLD!
hello world!
HELLO WORLD!
```

DISPLAY ステートメントは、Item-1 の実際の内容は変更せず、文字の表示方法にのみ影響を与えます。しかし、MOVE ステートメントでは、Item-2 の内容が大文字に置き換わります。

変換では、現在のロケールで定義された大/小文字マッピングを使用します。 関数の結果の長さは、引数の長さとは異なることがあります。

関連タスク
30 ページの『画面またはファイルからの入力の割り当て (ACCEPT)』
31 ページの『画面上またはファイル内での値の表示 (DISPLAY)』

逆順への変換 (REVERSE)

REVERSE 組み込み関数を使用して、ストリング内の文字の順序を反転させることができます。

```
Move Function Reverse(Orig-cust-name) To Orig-cust-name
```

例えば、上記ステートメントは、Orig-cust-name の中の文字の順序を逆にします。 開始値が JOHNSON*bbb* であれば、 ステートメントの実行後の値は *bbb*NOSNHOJ になります (*b* はブランク・スペースを表します)。

関連概念
180 ページの『Unicode および言語文字のエンコード』

数値への変換 (NUMVAL、NUMVAL-C)

NUMVAL、NUMVAL-C、および 関数は、文字ストリング (英数字または国別リテラル、あるいはクラス英数字またはクラス国別データ項目) を数値に変換します。これらの関数を使用して、数値的に処理できるよう、フリー・フォーマット文字表現の数値を数値形式に変換します。

NUMVAL-C は、上の例で示されているように、引数に通貨記号またはコンマ (またはその両方) が含まれているときに使用します。 文字ストリングの前または後ろに代数符号を入れることができ、その符号が処理されます。 引数は、デフォルト・オプション ARITH(COMPAT) (互換モード) でコンパイルするときは 18 桁を超えてはならず、ARITH(EXTEND) (拡張モード) でコンパイルするときは 31 桁を超えてはなりません (桁数には編集記号は含みません)。

NUMVAL、NUMVAL-C、および は、互換モードでは長精度 (64 ビット) 浮動小数点値を戻し、拡張モードでは拡張精度 (128 ビット) 浮動小数点値を戻します。これらの関数への参照は、数値データ項目への参照を表します。

最大でも、正確に長精度浮動小数点に変換できるのは 15 桁の 10 進数字までです (変換および精度に関する以下の関連参照で説明されています)。 NUMVAL、NUMVAL-C、または の引数が 15 桁を超える場合、ARITH(EXTEND) コンパイラー・オプションを指定して、引数の値を正確に表現できる拡張精度関数結果が戻されるようにすることをお勧めします。

NUMVAL、NUMVAL-C、または を使用する場合は、数値データを固定形式で静的に定義したり、入力データを正確な方法で入力したりする必要はありません。例えば、次のように入力される数値を定義するとします。

```
01  X   Pic S999V99  leading sign is separate.
. . .
    Accept X from Console
```

アプリケーションのユーザーは、PICTURE 節で定義されているとおりに正確に数値を入力しなければなりません。 次に例を示します。

```
+001.23
-300.00
```

しかし、NUMVAL 関数を使用する場合は、次のようにコーディングすることができます。

```
01  A   Pic x(10).
01  B   Pic S999V99.
. . .
    Accept A from Console
    Compute B = Function Numval(A)
```

入力は次のようにすることができます。

```
1.23
-300
```

関連概念
39 ページの『数値データの形式』

46 ページの『データ形式の変換』
180 ページの『Unicode および言語文字のエンコード』

関連タスク
188 ページの『国別 (Unicode) 表現との間の変換』

関連参照
46 ページの『変換および精度』
253 ページの『ARITH』

あるコード・ページから別のコード・ページへの変換

DISPLAY-OF 組み込み関数と NATIONAL-OF 組み込み関数をネストすると、任意のコード・ページを別の任意のコード・ページに簡単に変換できます。

例えば、次のコードでは、EBCDIC のストリングを ASCII のストリングに変換しています。

```
  77  EBCDIC-CCSID PIC 9(4) BINARY VALUE 1140.
  77  ASCII-CCSID  PIC 9(4) BINARY VALUE 819.
  77  Input-EBCDIC PIC X(80).
  77  ASCII-Output PIC X(80).
  . . .
* Convert EBCDIC to ASCII
      Move Function Display-of
          (Function National-of (Input-EBCDIC EBCDIC-CCSID),
              ASCII-CCSID)
      to ASCII-output
```

関連概念
180 ページの『Unicode および言語文字のエンコード』

関連タスク
188 ページの『国別 (Unicode) 表現との間の変換』

データ項目の評価 (組み込み関数)

組み込み関数を使用して、照合シーケンス内の文字の序数位置を判別したり、 一連のものから最大項目または最小項目を検出したり、データ項目の長さを検出したり、あるいはプログラムがコンパイルされた時間を判別したりできます。

以下の組み込み関数を使用します。

- CHAR および ORD。これらは、プログラム内で使用される照合シーケンスを基準にして、整数および単一の英字または英数字を評価するためのものです。
- MAX、MIN、ORD-MAX、および ORD-MIN。これらは、USAGE NATIONAL データ項目を含む一連のデータ項目から最大項目または最小項目を検出するためのものです。
- LENGTH。 これはデータ項目 (USAGE NATIONAL データ項目
- WHEN-COMPILED。これは、プログラムがコンパイルされた日時を調べるためのものです。

関連概念
180 ページの『Unicode および言語文字のエンコード』

関連タスク
107 ページの『照合シーケンスに関する単一文字の評価』
107 ページの『最大または最小データ項目の検出』
109 ページの『データ項目の長さの検出』
110 ページの『コンパイルの日付の検出』

照合シーケンスに関する単一文字の評価

照合シーケンス内のある特定の文字 (英字または英数字) の序数位置を検出するには、引数として文字を持つ ORD 関数を使用します。 ORD はその序数位置を表す整数を戻します。

以下のように、データ項目の 1 文字のサブストリングを ORD への引数として使用することができます。

```
IF Function Ord(Customer-record(1:1)) IS > 194 THEN . . .
```

ある文字の照合シーケンスにおける序数位置がわかっていて、それに対応する文字を検索する場合には、CHAR でその整数の序数位置を引数として使用することができます。 CHAR は、要求された文字を返します。 次に例を示します。

```
INITIALIZE Customer-Name REPLACING ALPHABETIC BY Function Char(65)
```

関連参照

CHAR (*COBOL for Linux on x86* 言語解説書)
ORD (*COBOL for Linux on x86* 言語解説書)

最大または最小データ項目の検出

2 つ以上の英数字、英字、または国別データ項目のうち、どれが最大値を持つかを判別する場合には、MAX または ORD-MAX 組み込み関数を使用します。 どの項目が最小値を持つのかを判別するには、MIN または ORD-MIN を使用します。 これらの関数は、照合シーケンスに従って評価します。

数値項目 (USAGE NATIONAL のあるものを含む) を比較するには、MAX、ORD-MAX、MIN、または ORD-MIN を使用します。 これらの組み込み関数を使用すると、引数の代数値が比較されます。

MAX および MIN 関数は、指定された引数の 1 つの内容を返します。 例えば、プログラムに以下のデータ定義が含まれているとします。

```
05  Arg1   Pic x(10)  Value "THOMASSON ".
05  Arg2   Pic x(10)  Value "THOMAS    ".
05  Arg3   Pic x(10)  Value "VALLEJO   ".
```

次のステートメントは、VALLEJO*bbb* を Customer-record の最初の 10 文字位置に割り当てます (ここで *b* はブランク・スペースを表します)。

```
Move Function Max(Arg1 Arg2 Arg3) To Customer-record(1:10)
```

代わりに MIN を使用すると、THOMAS*bbbb* が割り当てられます。

ORD-MAX および ORD-MIN 関数は、提供した引数のリストの中で最大値または最小値を持つ引数の (左からカウントした) 序数位置を表す整数を返します。 上の例で ORD-MAX 関数を使用した場合、数字関数への参照が有効な場所にないため、コンパイラーはエラー・メッセージを出します。上の例と同じ引数を使用して、次のように ORD-MAX を使用できます。

```
Compute x = Function Ord-max(Arg1 Arg2 Arg3)
```

上記のステートメントは、前の例と同じ引数が使用された場合、整数 3 を x に割り当てます。 代わりに ORD-MIN を使用した場合には、整数 2 が返されます。 Arg1、Arg2、および Arg3 が配列 (テーブル) の連続エレメントであったなら、上の例はもっと現実的なものになると思われます。

任意の引数に国別項目を指定する場合、すべての引数をクラス国別と指定する必要があります。

関連タスク
48 ページの『算術の実行』

79 ページの『組み込み関数を使用したテーブル項目の処理』
108 ページの『英数字または国別関数によって戻される可変結果』

関連参照
MAX (*COBOL for Linux on x86* 言語解説書)
MIN (*COBOL for Linux on x86* 言語解説書)
ORD-MAX (*COBOL for Linux on x86* 言語解説書)
ORD-MIN (*COBOL for Linux on x86* 言語解説書)

英数字または国別関数によって戻される可変結果

英数字または国別関数の結果は、関数引数によって、長さや値が異なることがあります。

次の例では、R3 に移動されるデータの量および COMPUTE ステートメントの結果は、R1 および R2 の値とサイズによって異なります。

```
01  R1    Pic x(10) value "e".
01  R2    Pic x(05) value "f".
01  R3    Pic x(20) value spaces.
01  L     Pic 99.
. . .
    Move Function Max(R1 R2) to R3
    Compute L = Function Length(Function Max(R1 R2))
```

このコードの結果は次のようになります。

- R2 は R1 より大きいものと評価されます。
- ストリング 'f*bbbb*' が R3 に移動されます (ここで *b* はブランク・スペースを表します)。 (R3 内の残りの文字位置にはスペースが埋められます。)
- L は値 5 と評価されます。

R1 が「e」ではなく「g」を含んでいたなら、コードの結果は以下のようになります。

- R1 は、R2 より大きいと評価されます。
- ストリング 'g*bbbbbbbbb*' が R3 に移動されます。 (R3 内の残りの文字位置にはスペースが埋められます。)
- 値 10 が L に割り当てられます。

プログラムが関数引数として国別データを使用する場合、同様に関数結果の長さおよび値が変化します。例えば、以下のコードは上のフラグメントと等しいですが、英数字データではなく国別データを使用します。

```
01  R1    Pic n(10) national value "e".
01  R2    Pic n(05) national value "f".
01  R3    Pic n(20) national value spaces.
01  L     Pic 99    national.
. . .
    Move Function Max(R1 R2) to R3
    Compute L = Function Length(Function Max(R1 R2))
```

このコードの結果は以下のようになります。これらは、国別文字についてのものであることを除けば、最初の結果のセットと似ています。

- R2 は R1 より大きいものと評価されます。
- ストリング NX"6600 2000 2000 2000 2000" (国別文字 'f*bbbb*' と同等のもの。ここで、*b* はブランク・スペース) は、ここでは読みやすくするためスペースが挿入された 16 進表記で示されており、R3 に移動されます。 R3 内の空白文字位置には、国別スペースが埋め込まれます。
- L は値 5 (R2 の国別文字位置の長さ) と評価されます。

英数字または国別の関数からの可変長出力を取り扱うことがあります。 それに応じてプログラムを設計しなければなりません。 例えば、書き込むレコードによって長さが異なる可能性があるときには、可変長レコード・ファイルの使用を考えることが必要になります。

```
File Section.
FD  Output-File Recording Mode V.
01  Short-Customer-Record  Pic X(50).
01  Long-Customer-Record   Pic X(70).
Working-Storage Section.
01  R1    Pic x(50).
01  R2    Pic x(70).
. . .
    If R1 > R2
      Write Short-Customer-Record from R1
    Else
      Write Long-Customer-Record from R2
    End-if
```

関連タスク
107 ページの『最大または最小データ項目の検出』
48 ページの『算術の実行』

関連参照
MAX (*COBOL for Linux on x86* 言語解説書)

データ項目の長さの検出

LENGTH 関数を多くのコンテキスト (テーブルおよび数値データを含む) で使用して、項目の長さを判別することができます。 例えば、LENGTH 関数を呼び出して、英数字または国別リテラルの長さ、あるいは DBCS 以外の不特定タイプのデータ項目の長さを判別できます。

LENGTH 組み込み関数

LENGTH 関数は、国別項目 (リテラル、あるいは USAGE NATIONAL を持つ任意の項目 (国別グループ項目を含む)) の長さを、引数の長さ (国別文字位置の数) に等しい整数として戻します。 これは、他の任意のデータ項目の長さを、引数の長さ (英数字位置の数) に等しい整数として戻します。

次の COBOL ステートメントは、データ項目を、顧客名を入れるレコードのフィールドに移動する例を示しています。

```
Move Customer-name To Customer-record(1:Function Length(Customer-name))
```

LENGTH OF 特殊レジスター

LENGTH OF 特殊レジスターを使用することもできます。この特殊レジスターは国別データの場合でも、長さをバイト単位で戻します。 Function Length(Customer-name) または LENGTH OF Customer-name のどちらをコーディングしても、英数字項目の場合は同じ結果 (Customer-name のバイト単位の長さ) が戻されます。

LENGTH 関数は、算術式が許可される場所でのみ使用できます。しかし、LENGTH OF 特殊レジスターはさまざまなコンテキストで使用することができます。 例えば、 整数引数を受け入れる組み込み関数に対する引数として LENGTH OF 特殊レジスターを使用できます。 (組み込み関数を LENGTH OF 特殊レジスターに対するオペランドとして使用することはできません。) LENGTH OF 特殊レジスターは、CALL ステートメントのパラメーターとして使用することもできます。

関連タスク
48 ページの『算術の実行』
70 ページの『可変長テーブルの作成 (DEPENDING ON)』
79 ページの『組み込み関数を使用したテーブル項目の処理』

関連参照
252 ページの『ADDR』

LENGTH (*COBOL for Linux on x86* 言語解説書)
LENGTH OF (*COBOL for Linux on x86* 言語解説書)

コンパイルの日付の検出

WHEN-COMPILED 組み込み関数を使用して、プログラムがいつコンパイルされたのかを知ることができます。 21 文字の結果は、コンパイル時の 4 桁の年、月、日、および時刻 (時間、分、秒、および百分の 1 秒)、およびグリニッジ標準時との差 (時間と分) を示します。

最初の 16 桁の形式は次のとおりです。

```
YYYYMMDDhhmmsshh
```

代わりに WHEN-COMPILED 特殊レジスターを使用して、次のフォーマットで、コンパイルの日時を知ることもできます。

```
MM/DD/YYhh.mm.ss
```

WHEN-COMPILED 特殊レジスターは、2 桁の年のみをサポートし、秒の小数部を扱いません。 この特殊レジスターは、MOVE ステートメントの送信フィールドとしてのみ使用できます。

関連参照
WHEN-COMPILED (*COBOL for Linux on x86* 言語解説書)

第 7 章 ファイルの処理

ファイルからのデータ読み取りとファイルへのデータ書き込みは、大半の COBOL プログラムで重要となります。 プログラムは情報を検索し、それを要求どおりに処理した後、結果を書き込みます。

ただし、処理を行う前に、各ファイルを識別してそれらの物理構造を記述し、順次、相対、 索引付き、行順次のいずれでファイルが編成されているかを示す必要があります。 ファイルを識別するには、ファイルとそのファイル・システムに名前を付ける必要があります。 また、処理が正しく行われたことを検証するために、後で検査が可能なファイル状況フィールドを設定することも可能です。

ファイルの処理で実行できる主なタスクでは、まずファイルをオープンして読み取り、(ファイル編成およびアクセスのタイプに応じて) レコードの追加、置換、または削除を行います。

関連概念
111 ページの『ファイルの概念と用語』
117 ページの『ファイル・システム』
124 ページの『世代別データ・グループ』

関連タスク
113 ページの『ファイルの識別』
121 ページの『ファイル編成およびアクセス・モードの指定』
131 ページの『ファイルの連結』
132 ページの『オプション・ファイルのオープン』
133 ページの『ファイル状況フィールドの設定』
133 ページの『ファイル構造の詳細記述』
134 ページの『ファイルの入出力ステートメントのコーディング』
143 ページの『Db2 ファイルの使用』
146 ページの『QSAM ファイルの使用』
146 ページの『SFS ファイルの使用』

ファイルの概念と用語

ファイルに関する COBOL for Linux の情報では、以下の概念と用語が使用されています。

システム・ファイル名
ハード・ディスクまたはその他の外部メディア上のファイルの名前。 システム・ファイル名は、固有性を確実にするために、パスまたはその他の接頭部で修飾されることがあります。 ファイルは、特定のファイル・システム内に存在します。

ファイル・システムには通常、ファイルを管理するためのコマンドが提供されています。 次の例は、`ls` コマンドを使用して STL ファイル・システム内のファイル `Transaction.log` に関する詳細を出力する方法と、システムの応答を示しています。

```
> ls -l Transaction.log
-rw-r--r--    1 cobdev   cobdev     6144 May 27 17:29 Transaction.log
```

内部ファイル名
`FILE SECTION` のファイル記述項目で `FD` キーワードの後に指定されるユーザー定義語。プログラム内でファイルを参照するために使用されます。

以下の例では、`LOG-FILE` は内部ファイル名です。

```
Data division.
File section.
FD  LOG-FILE.
01  LOG-FILE-RECORD.
```

© Copyright IBM Corp. 2021

プログラムは、OPEN、CLOSE、READ、WRITE、および START などの入出力ステートメントを使用して、内部ファイルに対して操作を行います。 この用語が示すように、内部ファイル名は、プログラムの外では意味を持ちません。

ASSIGN 節は、下に説明するように、内部ファイル名をシステム・ファイル名と関連付ける手段です。

ファイル・システム ID
ファイルが保管されていて、ファイルへのアクセスが経由されるファイル・システムを指定する 3 文字のストリング。

外部ファイル名
内部ファイル名と、それに関連付けられるシステム・ファイル名の間を仲介する働きをする名前。 外部ファイル名は、プログラム外でも可視であり、通常、プログラムの実行前に *file-information* (オプションのファイル・システム ID にシステム・ファイル名が続くもの) に設定される環境変数の名前として使用されます。

(外部ファイル名は､外部ファイル (FD 項目で EXTERNAL キーワードにより定義されるファイル) の名前とは区別されます｡)

ASSIGN 文節は、FILE-CONTROL 段落で指定され、内部ファイル名をシステム・ファイル名に関連付けます。 ASSIGN 文節には次の 3 つの基本形式があります。

- ```
  SELECT internal-file-name ASSIGN TO user-defined-word
  ```
- ```
  SELECT internal-file-name ASSIGN TO 'literal'
  ```
- ```
  SELECT internal-file-name ASSIGN USING data-name
  . . .
      MOVE file-information TO data-name
  ```

user-defined word および *literal* はそれぞれ、最大で 3 つのコンポーネントをハイフンで分離して構成します。 左から右に、次のコンポーネントがあります。

1. (オプション) コメント
2. (オプション) ファイル・システム ID
3. ユーザー定義言が指定された場合は外部ファイル名、リテラルが指定された場合はシステム・ファイル名

file-information は、最大で 2 つのコンポーネントをハイフンで分離して構成します。 左から右に、次のコンポーネントがあります。

1. (オプション) ファイル・システム ID
2. システム・ファイル名

関連概念
117 ページの『ファイル・システム』

関連タスク
113 ページの『ファイルの識別』

関連参照
ASSIGN 節 (*COBOL for Linux on x86* 言語解説書)
  ```

ファイルの識別

ファイルを識別するには、FILE-CONTROL 段落で SELECT および ASSIGN 文節を使用して、COBOL プログラム内部で使用されるファイル名を、対応するシステム・ファイル名に関連付けます。

この指定の簡単な形式は次のとおりです。

```
SELECT internalFilename ASSIGN TO fileSystemID-externalFilename
```

internalFilename は、ファイルを参照するためにプログラム内部で使用される名前を指定します。 内部名は通常、外部ファイル名またはシステム・ファイル名と異なります。

ASSIGN 文節には、アクセスしたいファイルの外部名 (*externalFilename*) を指定します。またオプションで、ファイルが存在する、またはファイルが作成されるファイル・システム (*fileSystemID*) を指定します。

fileSystemID をコーディングしてファイル・システムを識別する場合は、以下のいずれかの値を使用します。

Db2
Db2 リレーショナル・データベース・ファイル・システム

LSQ
行順次ファイル・システム

QSAM
待機順次アクセス方式ファイル・システム

RSD
レコード順次区切りファイル・システム

SdU
SMARTdata ユーティリティー・ファイル・システム

SFS
CICS Structured File Server ファイル・システム

STL
標準言語ファイル・システム

VSA
仮想記憶アクセス方式。SFS または STL ファイル・システムを暗黙に示します。

システム・ファイル名の最初 (一番左) の部分が の値に一致する (`/.:/cics/sfs` で始まる) 場合は、SFS が暗黙的に示されます。一致しない場合、VSA は STL ファイル・システムを暗黙に示します。

LINE SEQUENTIAL ファイルの場合、LSQ (行順次ファイル・システム) を指定またはデフォルトで使用できます。

INDEXED、RELATIVE、および SEQUENTIAL ファイルの場合、Db2、SdU、SFS、または STL を指定できます。 SEQUENTIAL ファイルの場合、RSD または QSAM を選択しても有効です。

特定のファイルにファイル・システムを指定しない場合、ファイル・システムは、優先順位に関する関連参照で説明されている優先順位に従って決定されます。

内部ファイル名をシステム・ファイル名に関連付けるには、以下に説明するように、ASSIGN 文節で以下の項目のいずれかを使用します。

- ユーザー定義語
- リテラル
- データ名

ユーザー定義語使用によるファイルの識別

ユーザー定義語を使用して内部ファイル名をシステム・ファイル名に関連付けるには、以下の形式で SELECT 文節と ASSIGN 文節をコーディングします。

```
SELECT internalFilename ASSIGN TO userDefinedWord
```

内部ファイル名のシステム・ファイル名への関連付けは、実行時に完了します。 次の例は、環境変数 TRANLOG を使用して、内部ファイル名 LOG-FILE を STL ファイル・システムのファイル Transaction.log に関連付ける方法を示しています。

```
SELECT LOG-FILE ASSIGN TO TRANLOG
. . .
export TRANLOG=STL-Transaction.log
```

LOG-FILE の OPEN ステートメントの実行時に環境変数 TRANLOG が設定されていない場合、またはヌル・ストリングに設定されている場合、LOG-FILE は、デフォルトのファイル・システムにある TRANLOG という名前のファイルに関連付けられます。

リテラル使用によるファイルの識別

リテラルを使用して内部ファイル名をシステム・ファイル名に関連付けるには、以下の形式で SELECT 文節と ASSIGN 文節をコーディングします。

```
SELECT internalFilename ASSIGN TO 'fileSystemID-systemFilename'
```

リテラルには、システム・ファイル名およびオプションでファイル・システムを指定します。

次の例は、内部ファイル名 myFile を SFS ファイル・システムのサーバー *sfsServer* にあるファイル extFile に関連付ける方法を示しています。

```
SELECT myFile ASSIGN TO 'SFS-/.:/cics/sfs/sfsServer/extFile'
```

リテラルは明示的にファイル・システムとシステム・ファイル名を指定するため、ファイルの関連付けはコンパイル時に解決されます。

ASSIGN 文節のコーディングについて詳しくは、適切な関連参照を参照してください。

データ名使用によるファイルの識別

データ名を使用して内部ファイル名をシステム・ファイル名に関連付けるには、以下の形式で SELECT 文節と ASSIGN 文節をコーディングします。

```
SELECT internalFilename ASSIGN USING dataName
```

ファイルが処理される前に、ファイル・システムとファイル名の情報を変数 *dataName* に転記します。

次の例は、内部ファイル名 myFile を SdU ファイル・システムのファイル FebPayroll に関連付ける方法を示しています。

```
SELECT myFile ASSIGN USING fileData
. . .
01 fileData PIC X(50).
. . .
    MOVE 'SdU-FebPayroll' TO fileData
    OPEN INPUT myFile
```

関連付けは、実行時に無条件で解決されます。

関連概念
117 ページの『ファイル・システム』

122 ページの『行順次ファイル編成』
124 ページの『世代別データ・グループ』

関連タスク
8 ページの『オペレーティング・システムに対するファイルの識別 (ASSIGN)』
115 ページの『Db2 ファイルの識別』
115 ページの『SFS ファイルの識別』
131 ページの『ファイルの連結』

関連参照
116 ページの『ファイル・システム決定の優先順位』
220 ページの『ランタイム環境変数』
ASSIGN 節 (*COBOL for Linux on x86* 言語解説書)

Db2 ファイルの識別

Db2 ファイル・システムのファイルを識別するには、ファイル・システム ID DB2 を指定するかまたはデフォルトで使用します。

システム・ファイル名には、基になる Db2 表のスキーマを含める必要があります。スキーマは、システム・ファイル名の接頭部として直接指定します。

CICS TXSeries または CICS TX との相互協調処理には、スキーマ名 CICS を使用します。例えば、TRANS という名前の Db2 ファイルをスキーマ CICS で環境変数 EXTFILENAME に関連付けるには、次のコマンドを使用できます。

```
export EXTFILENAME=DB2-CICS.TRANS
```

または、より柔軟性を高めるには、ファイル・システムとシステム・ファイル名を別々に指定します。

```
export COBRTOPT=FILESYS=DB2
export EXTFILENAME=CICS.TRANS
```

DB2® ファイルの使用について詳しくは、以下の適切な関連タスクを参照してください。

関連タスク
113 ページの『ファイルの識別』
143 ページの『Db2 ファイルの使用』
172 ページの『ファイル・システム状況コードの使用』
215 ページの『環境変数の設定』

関連参照
118 ページの『Db2 ファイル・システム』
300 ページの『FILESYS』

SFS ファイルの識別

SFS ファイル・システムのファイルを識別するには、ファイル・システム ID SFS を指定するかまたはデフォルトで使用します。

システム・ファイル名は、先頭が接頭部 SFS- で、その後に SFS サーバー名とファイル名が続いていなければなりません。ファイルが複数の SFS サーバー上にある場合は、システム・ファイル名を指定できます。次の例は、INVENTORY という名前のシステム・ファイルが sfsServer という名前の SFS サーバー上にあることを示しています。

```
export EXTFN=SFS-/.:/cics/sfs/sfsServer/INVENTORY
```

環境変数 CICS_TK_SFS_SERVER を必要な SFS サーバーに設定した場合は、 システム・ファイル名に対して、完全修飾名を使用する代わりに省略表現の指定を使用できます。 システム・ファイル名には、

CICS_TK_SFS_SERVER の値が接頭部として追加され、その後ろにスラッシュが続いて、完全修飾されたシステム・ファイル名が作成されます。 次に例を示します。

```
export CICS_TK_SFS_SERVER=/.:/cics/sfs/sfsServer
export EXTFN=SFS-INVENTORY
```

次の export コマンドは、代替索引を 2 つ持つ索引付き SFS ファイルを識別するように環境変数 MYFILE を設定する方法を示す、より複雑な例です。

```
export MYFILE="/.:/cics/sfs/sfsServer/mySFSfil(¥
/.:/cics/sfs/sfsServer/mySFSfil;myaltfil1,¥
/.:/cics/sfs/sfsServer/mySFSfil;myaltfil2)"
```

このコマンドは、以下の情報を提供します。

- /.:/cics/sfs/*sfsServer* は、CICS サーバーの完全修飾名です。
- *mySFSfil* はベース SFS ファイルです。
- /.:/cics/sfs/*sfsServer*/*mySFSfil* は、完全修飾ベース・システム・ファイル名です。
- *myaltfil1* および *myaltfil2* は、代替索引ファイルです。

各代替索引ファイルの場合、ファイル名は、その完全修飾ベース・システム・ファイル名と、その後にセミコロン (;) および代替索引ファイル名が続く形式にする必要があります (/.:/cics/sfs/*sfsServer*/*mySFSfil*;*myaltfil1*)。

export コマンドでは、代替索引ファイルの指定と指定の間のコンマは必須です。

関連タスク
113 ページの『ファイルの識別』
146 ページの『SFS ファイルの使用』
172 ページの『ファイル・システム状況コードの使用』

関連参照
120 ページの『SFS ファイル・システム』

ファイル・システム決定の優先順位

特定の SEQUENTIAL、INDEXED、または RELATIVE ファイルに適用されるファイル・システムは、次の優先順位 (一番上から一番下) に従って決定されます。

1. *assignment-name* ランタイム環境変数、または ASSIGN 文節にコーディングされた USING データ項目の値によって指定されたファイル・システム
2. ASSIGN 文節にコーディングされたリテラルまたはユーザー定義語の右から 2 番目のコンポーネントによって指定されたファイル・システム。ただし、そのコンポーネントが少なくとも 3 文字である場合 (および、ASSIGN 文節に関する資料に記述されたその他の基準を満たしている場合)
3. FILESYS ランタイム・オプションで指定されたデフォルトのファイル・システム (COBRTOPT ランタイム環境変数で指定)

上記の方法でファイル・システムが決定できない場合、システム・ファイル名の一番左の部分が /.:/cics/sfs で始まる場合は SFS ファイル・システムが、一致しない場合は STL ファイル・システムがデフォルトで選択されます。

関連概念
117 ページの『ファイル・システム』

関連タスク
8 ページの『オペレーティング・システムに対するファイルの識別 (ASSIGN)』
113 ページの『ファイルの識別』

関連参照

220 ページの『ランタイム環境変数』
300 ページの『FILESYS』
ASSIGN 節 (*COBOL for Linux on x86* 言語解説書)

ファイル・システム

順次、相対、索引付き、または行順次編成のレコード単位ファイルへは、ファイル・システム を通してアクセスします。

COBOL for Linux は、順次ファイル、相対ファイル、および索引付きファイルについて以下のファイル・システムをサポートします。

Db2 (Db2 リレーショナル・データベース) ファイル・システム
バッチ COBOL プログラムが、Db2 に保管される CICS ファイルを 作成したり、そのファイルにアクセスしたりできます。

SdU (SMARTdata ユーティリティー) ファイル・システム
SdU ファイル・システム内のファイルは、PL/I プログラムと共有できます。

SFS (CICS 構造化ファイル・サーバー) ファイル・システム
CICS によって使用されるファイル・システムの 1 つ。 CICS SFS は、CICS の一部として提供されます。 SFS ファイルは PL/I プログラムと共用できる。

STL (標準言語) ファイル・システム
ローカル・ファイルの基本機能を提供します。

COBOL for Linux は、順次ファイルについて以下のファイル・システムをサポートします。

QSAM (待機順次アクセス方式) ファイル・システム
COBOL プログラムが、FTP を使用してメインフレームから Linux に転送された QSAM ファイルにアクセスできます。

RSD (レコード順次区切り) ファイル・システム
COBOL プログラムが、他の言語で書かれたプログラムとデータを共有できます。 RSD ファイルは、固定長または可変長レコードを持つ順次のみですが、レコード内のあらゆる COBOL データ型に対応しています。 レコード内のテキスト・データは、大半のファイル・エディターで編集することができます。

特定の順次、相対、または索引付きファイルに使用するファイル・システムは、いくつかの方法で指定できます。 詳細については、ファイル・システム決定の優先順位に関する関連参照を参照してください。

行順次編成のレコード単位ファイルには、LSQ (行順次) ファイル・システムからのみアクセスできます。

Db2 ファイルは、DB2 コマンド行ユーティリティーで管理します。SFS ファイルは、`sfsadmin` コマンド行ユーティリティーで管理します。その他のファイルはすべて、行順次の Linux ファイル・システムに存在し、`cp`、`ls`、`mv`、および `rm` などの標準 Linux コマンドで管理します。(ただし、SdU ファイルには `cp` または `mv` コマンドを使用しないでください。SdU ファイルは内部で相互に参照しあう複数のコンポーネント・ファイルで構成されているためです。)

どのファイル・システムでも、COBOL ステートメントを使用して COBOL ファイルの読み取りと書き込みを行うことができます。 大半のプログラムは、どのファイル・システムでも同じように動作します。 ただし、あるファイル・システムで書き込まれたファイルを、別のファイル・システムで読み取ることはできません。

関連概念
122 ページの『行順次ファイル編成』

関連タスク
8 ページの『オペレーティング・システムに対するファイルの識別 (ASSIGN)』
113 ページの『ファイルの識別』

関連参照
116 ページの『ファイル・システム決定の優先順位』
118 ページの『Db2 ファイル・システム』

119 ページの『QSAM ファイル・システム』
119 ページの『RSD ファイル・システム』
120 ページの『SdU ファイル・システム』
120 ページの『SFS ファイル・システム』
121 ページの『STL ファイル・システム』

529 ページの『付録 B IBM Z ホスト・データ形式についての考慮事項』

Db2 ファイル・システム

Db2 ファイル・システムは、順次ファイル、索引付きファイル、および相対ファイルをサポートします。Db2 ファイル・システムは CICS TXSeries または CICS TX との拡張相互協調処理を提供しており、Db2 に保管されている CICS ESDS、KSDS、および RRDS ファイルにバッチ COBOL プログラムがアクセスできるようにします。

Db2 ファイル・システムの実装により、各 COBOL 操作がデータベースにコミットされ、トランザクションまたはその他のデータベースのセマンティクスが COBOL プログラムから不可視になります。

Db2 データベース管理システム (DBMS) はバックアップ、圧縮、暗号化、およびユーティリティー機能を提供しており、また、Db2 ユーザーに慣れ親しんだ保守および管理プロトコルを提供しています。

db2 コマンド行ユーティリティーは、Db2 ファイルの管理機能を提供します。例えば、次のように db2 describe コマンドを使用して、Db2 ファイル・システム内にある CICS.Transaction.log というファイルについての詳細を出力することができます。

```
> db2 describe table CICS.¥"Transaction.log¥"
```

```
                        Data type                     Column
Column name             schema    Data type name      Length   Scale Nulls
----------------------- --------- ------------------- -------- ----- ----
RBA                     SYSIBM    CHARACTER                8      0 No
F1                      SYSIBM    CHARACTER               41      0 No
F2                      SYSIBM    VARCHAR                 29      0 No
```

db2 ユーティリティーによって提供される機能の詳細については、db2 コマンドを入力してください。

Db2 ファイル・システムは、非階層型です。

制約事項:

- 特定のプログラムは、1 つのデータベース内のみの Db2 ファイルを使用できます。
- 複数のスレッドとともに使用する場合、Db2 ファイル・システムは安全ではありません。

CICS TXSeries または CICS TX との相互協調処理:

CICS TXSeries または CICS TX との相互協調処理を行う場合、Db2 ファイルには以下の追加要件があります。

- Db2 表のスキーマ名は、CICS である必要があります。以下の項目のいずれかで、完全修飾名を指定します。
  - ASSIGN TO リテラル。例えば、ASSIGN TO 'CICS.MYFILE'
  - 環境変数。例えば、export ENVAR=CICS.MYFILE
  - ASSIGN USING のデータ名値。例えば、MOVE 'CICS.MYFILE' TO fileData

  スキーマの後ろの残りのファイル名は大文字である必要があります。
- 固定長ファイルの最大レコード長は次のとおりです。
  - 索引付き (KSDS): 4005 バイト
  - 相対 (RRDS): 4001 バイト

  - 順次 (ESDS): 4001 バイト
- 固定長ファイルの最大レコード長よりも大きいレコード長を持つファイルは、可変長として定義する必要があります。
- 最大レコード長は 32,767 バイトです。
- Db2 ファイルが COBOL プログラムによって作成される場合、ランタイム・オプション FILEMODE(SMALL) が有効になっている必要があります。

関連概念
121 ページの『ファイル編成およびアクセス・モード』

関連タスク
115 ページの『Db2 ファイルの識別』
143 ページの『Db2 ファイルの使用』

関連参照

ファイル・タイプへの CLOSE ステートメントの効果 (*COBOL for Linux on x86* 言語解説書)

コンパイラー限界値 (*COBOL for Linux on x86* 言語解説書)
DB2 データベース: 管理の概念および構成リファレンス (SQL 制限)

QSAM ファイル・システム

QSAM (待機順次アクセス方式) ファイル・システムは、固定、可変、およびスパンの各レコードをサポートします。 QSAM ファイル・システムを使用すると、メインフレームから Linux に転送した QSAM ファイルに直接アクセスできます。QSAM ファイルはレコード内のすべての COBOL データ・タイプをサポートします。

FTP にオプション binary および quote site rdw を指定したものを使用して、QSAM ファイルをメインフレームから入手することができます。 ファイルに EBCDIC 文字データが含まれている場合、文字データを EBCDIC として読み書きするために、Linux COBOL プログラムを -host を指定してコンパイルします。 QSAM ファイルが既に存在している場合は、同じファイルをメインフレームにアップロードできます。 このファイルが存在しない場合は、正しいファイル属性を使用して、このファイルを作成する必要があります。

注 : QSAM ファイル・システムは、以前のいくつかの PTF 資料では RAW ファイル・システムと呼ばれていました。

関連概念
121 ページの『ファイル編成およびアクセス・モード』

関連タスク
146 ページの『QSAM ファイルの使用』

RSD ファイル・システム

RSD (レコード順次区切り) ファイル・システムは、固定長または可変長レコードを持つ順次ファイルに対応しています。 RSD ファイルは、ブラウズ、編集、コピー、削除、印刷などの標準的なシステム・ファイル・ユーティリティー機能を使用して処理することができます。

RSD ファイルは優れたパフォーマンスを提供します。 また、Linux システムと Windows ベースのシステムとの間でファイルを容易に移植でき、異なる言語で書かれたプログラムやアプリケーションの間でファイルを共用することができます。

RSD ファイルは、固定長または可変長レコード内のあらゆる COBOL データ型に対応しています。 書き込まれた各レコードの後には、改行制御文字が続きます。

関連概念
121 ページの『ファイル編成およびアクセス・モード』

SdU ファイル・システム

SdU (SMARTdata ユーティリティー) ファイル・システムは、順次ファイル、索引付きファイル、および相対ファイルをサポートします。

OPEN OUTPUT ステートメントと WRITE ステートメントを使用して SdU ファイルを作成する 場合、に複数のファイルが作成されます。

- ファイル属性は、接尾部が .DDMEA となっているファイルに保管されます。
- 1 次索引は、ファイル名の先頭に文字 @ が付けられたファイルに保管されます。
- 代替索引は、ファイル名が接尾部 .@00 や .@01 などで終わるファイルに保管されます。

SdU ファイルは、内部で相互に参照しあう複数のファイルで構成されているため、SdU ファイルのコピー、移動、または名前変更を行わないでください。 SdU ファイルを削除 (rm コマンドを使用して) する際は、必ずその SdU ファイルのコンポーネント・ファイルもすべて削除するようにしてください。コンポーネント・ファイルの一部は、ピリオド (.) で始まるため、非表示になっています。 例えば、現行ディレクトリーにある Log123 という名前の SdU ファイルのコンポーネントをすべてリストするには、次のコマンドを使用します。

```
ls -l *Log123* .*Log123*
```

SdU ファイル・システムは 85 COBOL 標準 に準拠しています。

制約事項:

- SdU ファイルへは、COBOL プログラム以外からアクセスしてはいけません。これは、SdU ファイルのメタデータが、上記での説明のとおり、ファイルとは別に保管されるためです。
- 複数のスレッドとともに使用する場合、SdU ファイル・システムは安全ではありません。

関連概念
121 ページの『ファイル編成およびアクセス・モード』

関連参照

VSAM ファイル・システム 応答メッセージ

SFS ファイル・システム

CICS SFS (Structured File Server) ファイル・システムは、順次 (入力順)、相対、および索引付き (クラスター) の 3 タイプのファイル編成をサポートするレコード単位のファイル・システムです。SFS ファイル・システムには、ファイルへの順次、ランダム、または動的アクセスに必要な、基本的な機能が備わっています。

SFS ファイルは、読み取り、書き込み、再書き込み、および削除など、標準のファイル操作を使用して処理できます。

各 SFS ファイルには、ファイル内でのレコードの物理的順序を定義する内部 1 次索引が 1 つあり、任意の数の副次索引を格納することができます。副次索引は、レコードにアクセスできる代替シーケンスを提供します。

SFS ファイル内のすべてのデータは、SFS サーバーによって管理されます。 SFS には、システム・ツール、sfsadmin を備えています。これは、ファイルおよび索引の作成や SFS サーバー上で使用可能なボリュームの判別などの管理機能を、コマンド行インターフェースを使用して実行するためのツールです。 詳しくは、関連参照にある CICS の資料を参照してください。

SFS ファイル・システムは、非階層型です。 つまり、SFS ファイルの識別時にサーバー名の後に指定できるのは、ディレクトリー名ではなく、個々のファイル名のみです。

SFS ファイルへの COBOL アクセスは非トランザクション・アクセスです。SFS ファイルに対する各操作は、*atomic*、つまり全体として実行されるか全く実行されないかのいずれかです。 SFS システム障害が発

生すると、COBOL アプリケーションが完了したファイルの操作の結果が SFS ファイルに反映されない場合があります。

SFS ファイル・システムは 85 COBOL 標準 に準拠しています。

SFS ファイル・システムを使用すると、PL/I プログラムと共用するファイルの読み取りと書き込みを容易に行うことができます。

制約事項:

- 複数のスレッドとともに使用する場合、SFS ファイル・システムは安全ではありません。
- 64 ビット COBOL for Linux プログラムを使用して SFS ファイルを処理することはできません。

関連概念
121 ページの『ファイル編成およびアクセス・モード』

関連タスク
113 ページの『ファイルの識別』
115 ページの『SFS ファイルの識別』
146 ページの『SFS ファイルの使用』
149 ページの『SFS パフォーマンスの向上』

関連参照

CICS TXSeries の資料

STL ファイル・システム

STL ファイル・システム (標準言語ファイル・システム) は、順次ファイル、索引付きファイル、および相対ファイルに対応しています。 これは、ファイルにアクセスするための基本のファイル機能を提供します。

STL ファイル・システムは 85 COBOL 標準 に準拠しており、パフォーマンスが高く、Linux システムと Windows ベースのシステムの間で容易に移植できることも特長です。

関連概念
121 ページの『ファイル編成およびアクセス・モード』

ファイル編成およびアクセス・モードの指定

FILE-CONTROL 段落では、以下に示すように、 ファイルの物理構造とアクセス・モードを定義する必要があります。

```
FILE-CONTROL.
    SELECT file ASSIGN TO FileSystemID-Filename
    ORGANIZATION IS org ACCESS MODE IS access.
```

org に対しては、SEQUENTIAL (デフォルト)、LINE SEQUENTIAL、INDEXED、 RELATIVE のいずれかを選択することができます。

access に対しては、SEQUENTIAL (デフォルト)、RANDOM、DYNAMIC> のいずれかを選択することができます。

順次ファイルおよび行順次ファイルは、順次にアクセスする必要があります。 索引付きファイルと相対ファイルの場合は 3 つのアクセス・モードをすべて使用することができます。

ファイル編成およびアクセス・モード

ファイル編成は、順次、行順次、索引付き、相対のいずれかの形式で行うことができます。 アクセス・モードは、ファイルの編成方法ではなく、COBOL でのファイルの読み取り方法と書き込み方法を定義します。

プログラムの設計時には、ファイル編成とアクセス・モードを決定する必要があります。

次の表は、COBOL ファイルのファイル編成とアクセス・モードをまとめたものです。

表 9. **ファイル編成およびアクセス・モード**			
ファイル編成	**レコードの順序**	**レコードは削除または置換可能か**	**アクセス・モード**
順次	レコードが書き込まれた順序	レコードは削除できないが、そのスペースを同じ長さのレコードに再利用することが可能	順次のみ
行順次	レコードが書き込まれた順序	レコードは削除できないが、そのスペースを同じ長さのレコードに再利用することが可能	順次のみ
索引付き	キー・フィールドによる照合シーケンス	はい	順次、ランダム、または動的
相対	相対レコード番号の順序	はい	順次、ランダム、または動的

ファイル管理システムは、入出力装置からの入出力要求とレコード検索を処理します。

関連概念
122 ページの『順次ファイルの編成』
122 ページの『行順次ファイル編成』
123 ページの『索引付きファイル編成』
123 ページの『相対ファイル編成』
123 ページの『順次アクセス』
124 ページの『ランダム・アクセス』
124 ページの『動的アクセス』

関連タスク
121 ページの『ファイル編成およびアクセス・モードの指定』

順次ファイルの編成

順次ファイルに含まれるレコードは、それらが入力された順序で編成されています。 レコードの順序は固定されます。

順次ファイル内のレコードの読み取りや書き込みは、順次形式でのみ行うことができます。

順次ファイルにレコードを挿入したら、そのレコードの長さを調整したり、削除することはできません。ただし、長さが変わらなければ、レコードを更新 (REWRITE) することはできます。 新しいレコードはファイルの終わりに追加されます。

ファイル内のレコードの順序が重要でない場合は、レコード数の多少にかかわらず、順次編成を選択することをお勧めします。 順次出力は、レポートを印刷する際にも便利です。

関連概念
123 ページの『順次アクセス』

関連参照
137 ページの『順次ファイルに有効な COBOL ステートメント』

行順次ファイル編成

行順次ファイルは順次ファイルと似ていますが、レコード内にデータとして文字しか含めることができない点が異なります。 行順次ファイルは、オペレーティング・システムのネイティブ・バイト・ストリーム・ファイルでサポートされています。

ADVANCING 句を持つ WRITE ステートメントを使用して作成される 行順次ファイルは、プリンターまたはディスクに送信することができます。

関連概念
122 ページの『順次ファイルの編成』

関連タスク
113 ページの『ファイルの識別』

関連参照
137 ページの『行順次ファイルに有効な COBOL ステートメント』

索引付きファイル編成

索引付きファイルには、レコード・キーによって順序指定されたレコードが含まれています。 レコード・キーは、レコードを固有に識別し、それがアクセスされる順序を他のレコードとの関連で判別します。

各レコードには、レコード・キーを含むフィールドがあります。 レコードのレコード・キーは、例えば、従業員番号や送り状番号にすることができます。

また、索引付きファイルでは、代替索引 (レコードの別の論理配置を使用してファイルにアクセスできるレコード・キー) を使用することも可能です。 例えば、従業員番号ではなく、従業員の部門を介してファイルにアクセスすることができます。

索引付きファイルに対して使用できるレコード伝送 (アクセス) モードは、順次モード、ランダム・モード、または動的モードです。 索引付きファイルの順次読み取りまたは書き込みの順序は、キー値の順序になります。

EBCDIC についての考慮事項: 照合シーケンスが変更された場合と同様に、索引付きファイルがローカル EBCDIC ファイルの場合は、EBCDIC キーが COBOL プログラムの外部にあるものとして認識されません。 例えば、外部ソート・プログラムは、EBCDIC にも対応していない限り、期待どおりの順序でレコードをソートすることはありません。

関連参照
138 ページの『索引付きファイルおよび相対ファイルに有効な COBOL ステートメント』

相対ファイル編成

相対レコード・ファイルには、相対キー、すなわちファイルの先頭から相対したレコードの位置を表す レコード番号によって順序指定されたレコードが含まれています。

例えば、ファイルの最初のレコードの相対レコード番号は 1 で、10 番目のレコードの相対レコード番号は 10 というようになっています。 レコードは、固定長でも可変長でも構いません。

相対ファイルのレコード伝送モードは、順次、ランダム、または動的です。 相対ファイルの順次読み取りまたは書き込みの順序は、相対レコード番号の順序です。

関連参照
138 ページの『索引付きファイルおよび相対ファイルに有効な COBOL ステートメント』

順次アクセス

順次アクセスの場合は、FILE-CONTROL 段落で ACCESS IS SEQUENTIAL をコーディングします。

索引付きファイルのレコードは、ファイル位置標識の現在位置から始まって、選択されたキー・フィールド (基本または代替) の順にアクセスされます。

相対ファイルのレコードは、相対レコード番号の順にアクセスされます。

関連概念
124 ページの『ランダム・アクセス』
124 ページの『動的アクセス』

関連参照
136 ページの『ファイル位置標識』

ランダム・アクセス

ランダム・アクセスの場合は、FILE-CONTROL 段落で ACCESS IS RANDOM をコーディングします。

索引付きファイルのレコードは、キー・フィールドに入れられた値 (基本、代替、相対) に従ってアクセスされます。 代替索引は 1 つ以上存在する可能性があります。

相対ファイルのレコードは、相対キーに入れられた値に従ってアクセスされます。

関連概念
123 ページの『順次アクセス』
124 ページの『動的アクセス』

動的アクセス

動的アクセスの場合は、FILE-CONTROL 段落で ACCESS IS DYNAMIC をコーディングします。

動的アクセスでは、同じプログラム内での順次アクセスとランダム・アクセスの混在がサポートされています。 動的アクセスでは、順次処理とランダム処理の両方を実行する 1 つの COBOL ファイル定義を使用することで、あるレコードへは順次にアクセスし、別のレコードへはキーによってアクセスすることができます。

例えば、従業員レコードの索引付きファイルがあり、従業員の時間給がレコード・キーを形成しているものとします。 また、プログラムでは、時間給が $12.00 から $18.00 の従業員と、時間給が $25.00 以上の従業員について処理するものとします。 この情報にアクセスするには、1200 のキーに基づいて (ランダム検索 READ を使用して) 最初のレコードをランダムに検索します。 次に、給与フィールドが 1800 を超えるまで、(READ NEXT を使用して) 順次読み取りを行います。 今度は 2500 のキーに基づいて、ランダム読み取りに切り替えます。 このランダム読み取りの後、ファイルの終わりに到達するまで、順次読み取りを行います。

関連概念
123 ページの『順次アクセス』
124 ページの『ランダム・アクセス』

世代別データ・グループ

世代別データ・グループ *(GDG)* は、複数の関連ファイルの発生順の集合です。 GDG は、関連データの複数バージョンの処理を簡素化します。

GDG 内の各ファイルは、世代別データ・セット *(GDS)* または 世代 と呼ばれます。 (本書では、世代別データ・セットは 生成ファイル と呼びます。 ワークステーション上のファイル という用語は、ホスト上のデータ・セット という用語と同じです。)

GDG 内で、世代は ORGANIZATION、レコード形式、およびレコード長を含む、類似または非類似属性 (like or unlike attributes) を持つことができます。 グループ内のすべての世代で、属性の一貫性があり、編成が順次編成であれば、これらの世代を単一ファイルとして取り出せます。

関連ファイルのグループ化には利点があります。 以下に例を示します。

- グループ内のファイルを共通名で参照できる。
- グループ内のファイルが世代順に保持される。
- 期限切れのファイルを自動的に破棄できる。

GDG 内の世代は、その経過時間を示す、順次順序の相対名および絶対名を持ちます。

生成ファイルの相対名は、グループ名の後ろに整数が括弧内に続く形式です。 例えば、グループ名が hlq.PAY の場合は、以下のようになります。

- hlq.PAY(0) は、現行世代を表します。
- hlq.PAY(-1) は、1 つ前の世代を表します。
- hlq.PAY(+1) は、新世代を追加することを指定します。

生成ファイルの絶対名には、世代番号とバージョン番号が含まれます。 例えば、グループ名が hlq.PAY の場合は、以下のようになります。

- hlq.PAY.g0005v00 は、生成ファイル 5、バージョン 0 を表します。
- hlq.PAY.g0006v00 は、生成ファイル 6、バージョン 0 を表します。

絶対名および相対名の形式について詳しくは、関連タスクを参照してください。

世代順は通常、ファイルがグループに追加された順序と同じですが、必ずしもそうとは限りません。 絶対名および相対名を使用して生成ファイルを追加した方法によっては、グループ内の予期しない位置に世代が挿入されることがあります。 詳細については、生成ファイルの挿入と折り返しに関する関連参照を参照してください。

GDG はすべての COBOL for Linux ファイル・システムでサポートされます。

制約事項 : GDG には、代替索引を持つ SFS 索引付きファイルも、またはファイル名に代替索引のリストを必要とする SdU 索引付きファイルも含めることはできません。 この制限は、括弧で囲んだ代替索引リストの構文と、同じく括弧で囲んだ式を必要とする GDG 相対名の構文との間のあいまいさが原因です。

世代別データ・グループの作成および初期化については、適切な関連タスクを参照してください。

世代別グループを削除、再構築、クリーンアップ、変更、またはリストするには、また、グループ内の世代を追加または削除するには、gdgmgr ユーティリティーを使用します。 gdgmgr 機能の要約を参照するには、コマンド gdgmgr -h を発行します。 gdgmgr ユーティリティーの詳細については、man ページを参照してください。

関連タスク
125 ページの『世代別データ・グループの作成』
127 ページの『世代別データ・グループの使用』

関連参照
128 ページの『生成ファイルの名前形式』
129 ページの『生成ファイルの挿入と折り返し』
130 ページの『世代別データ・グループの限界処理 (limit processing)』
523 ページの『ファイル指定』

世代別データ・グループの作成

世代別データ・グループ (GDG) を作成するには、まず、-c フラグを指定して gdgmgr コマンドを使用して GDG カタログを作成します。 (*GDG* カタログは、Linux ネイティブ・ファイル・システム内のバイナリー・ファイルです。GDG カタログ名の形式は、*gdgBaseName*.catalogue です。)

次に、通常はファイルを作成する COBOL プログラムを実行して、その GDG に生成ファイルを入れます。

Linux LSQ ファイル・システムの場合

LSQ、RSD、SdU、または STL ファイル・システムで GDG カタログを作成するには、-c フラグを指定して gdgmgr コマンドを使用します。 例えば、カタログ ./myGroups/transactionGroup.catalogue を作成するには、次のコマンドを発行します。

```
gdgmgr -c ./myGroups/transactionGroup
```

カタログは、デフォルトで現行作業ディレクトリー (./) に作成されます。 上記に示すように、オプションで、gdgmgr コマンドでカタログ名の前にパス名を付けることができます。 カタログと生成ファイルは、同じディレクトリーに作成される必要があります。

SFS ファイル・システムの場合

SFS ファイル・システムで GDG カタログを作成するには、-c フラグを指定して gdgmgr コマンドを使用します。 SFS ファイル名は、完全修飾形式または短縮形式のいずれかで指定できます。 例えば、次のコマンドでは、完全修飾の SFS 名を使用して GDG カタログを作成します。

```
gdgmgr -c /.:/cics/sfs/sfsServer/baseName
```

代わりに、最初に環境変数 CICS_TK_SFS_SERVER を設定してから、-F フラグで SFS ファイル・システムも指定して gdgmgr コマンドを発行すると、短縮形式の SFS ファイル名を指定できます。 次に例を示します。

```
export CICS_TK_SFS_SERVER=/.:/cics/sfs/sfsServer
gdgmgr -F SFS -c baseName
```

SFS グループのデフォルトの GDG ホーム・ディレクトリー (~/gdg) を指定変更するには、環境変数 gdg_home を設定します。 例えば、次のコマンドでは、GDG カタログ ~/groups/forSFS/sfs/*sfsServer*/myGroup.catalogue が作成されます。

```
export gdg_home=~/groups/forSFS
gdgmgr -c /.:/cics/sfs/sfsServer/myGroup
```

特定のグループ内の SFS 生成ファイルはすべて、同じ SFS サーバー上にある必要があります。

Db2 ファイル・システムの場合

Db2 ファイル・システムで GDG カタログを作成するには、以下のステップを実行します。

1. 使用する Db2 インスタンスのプロファイルを実行して、Db2 環境を初期化します。
2. 環境変数 DB2DBDFT を、グループのデータベースに設定します。
3. Db2 ファイル・システムを指定する -F フラグと、-c フラグを指定して gdgmgr コマンドを使用します。必須カタログ名にスキーマを直接指定します。

例えば、次のコマンドでは、カタログ ~/gdg/db2/db2inst1/*database*/cics.dbGroup.catalogue が作成されます。

```
. /home/db2inst1/sqllib/db2profile
export DB2DBDFT=database
gdgmgr -F DB2 -c cics.dbGroup
```

Db2 ファイルの GDG カタログのホーム・ディレクトリーは、環境変数 $gdg_home から取得されるか、あるいはデフォルトで ~/gdg になります。

世代別データ・グループ内の生成ファイルはすべて、1 つのスキーマ下の 1 つのデータベース内にある必要があります。

関連概念
117 ページの『ファイル・システム』
124 ページの『世代別データ・グループ』

関連タスク
115 ページの『Db2 ファイルの識別』
115 ページの『SFS ファイルの識別』
127 ページの『世代別データ・グループの使用』

関連参照
130 ページの『世代別データ・グループの限界処理 (limit processing)』

世代別データ・グループの使用

世代別データ・グループを使用するには、世代別データ・グループの参照方法、およびグループ内の生成ファイルの絶対名および相対名の形成方法を理解する必要があります。

グループ名の後ろにオプションで括弧内にアスタリスクを付けて、グループ全体を参照します (例 abc.SALES(*))。 どちらの形式でも、グループ内のすべてのファイルを示します。ファイルは、現行世代を先頭にして世代順で連結されます。 特定の世代を参照してグループに追加するには、グループ名の後ろに絶対参照または相対参照のいずれかを使用します。

絶対名の場合

絶対参照を使用してグループの特定の世代を指定するには、グループ名の後ろに絶対世代番号およびバージョン番号を使用します。 例えば、グループ名が abc.SALES の場合は、以下のようになります。

- abc.SALES.g0001v00 は、生成ファイル 1、バージョン 0 を参照します。
- abc.SALES.g0002v00 は、生成ファイル 2、バージョン 0 を参照します。

相対名の場合

相対参照を使用してグループの特定の世代を指定するには、グループ名の後ろに整数を括弧内に入れて使用します。 例えば、グループ名が abc.SALES の場合は、以下のようになります。

- abc.SALES(0) は、現行世代を参照します。
- abc.SALES(-1) は、1 つ前の世代を参照します。
- abc.SALES(+1) は、新世代を追加することを指定します。

絶対名および相対名について詳しくは、生成ファイルの名前形式に関する関連参照を参照してください。

絶対参照または相対参照が構文的に有効な場合、ランタイムは、適切な名前の世代別データ・グループのカタログの存在を確認して、意図した世代データ参照が世代別データ・グループまたは生成ファイルを参照するかどうか判断します。 カタログが見つからない場合、参照は、世代別データ・グループまたは生成ファイルの名前としてではなく、通常のファイル ID として扱われます。

符号なしまたは負の相対参照は、同等の絶対ファイル ID に解決されます。同等の ID の判断には、カタログが調べられます。 解決対象の相対参照に対応する絶対名がカタログ内にない場合、その参照は通常のファイル ID として扱われます。

符号付きの正の相対参照は通常、グループに新しい世代を追加することを表します。 増分は現行 (ゼロ番目の) 世代の世代番号に追加されて、新規世代の番号が形成されます。 符号付きの正の相対参照は、同等の絶対名を指定する代わりの手段です。

世代の折り返し (Generation wrapping)

増分と現行世代番号の合計が 9999 よりも大きい場合、新しい世代番号は折り返して (9999 の減算) 形成されます。 例については、生成ファイルの挿入と折り返しに関する関連参照を参照してください。

グループにその新規番号を持つ世代が含まれておらず、以下の条件のいずれかが真の場合、新規世代はグループ内の適切な順序位置に追加されます。

- ファイルが OPTIONAL ではなく、オープン・モードが OUTPUT である。
- ファイルが OPTIONAL で、オープン・モードが I-O または EXTEND である。

グループにその新規番号を持つ世代が既に含まれている場合は、オープン・モードが INPUT でないと、既存の世代が再利用され、上書きされることがあります。

関連概念
124 ページの『世代別データ・グループ』

関連タスク
125 ページの『世代別データ・グループの作成』

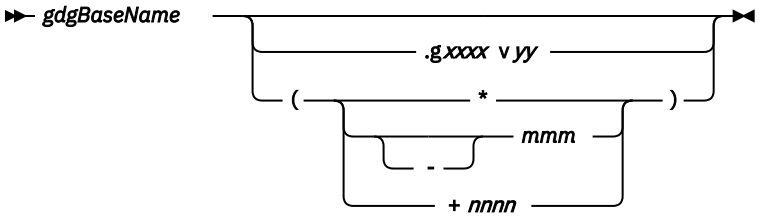
関連参照
128 ページの『生成ファイルの名前形式』

130 ページの『世代別データ・グループの限界処理 (limit processing)』
129 ページの『生成ファイルの挿入と折り返し』
523 ページの『ファイル指定』

生成ファイルの名前形式

次の構文は、世代別データ・グループ (GDG) および生成ファイルへの絶対参照および相対参照の形成方法を示しています。

GDG および GDS の構文



gdgBaseName
世代別データ・グループおよび GDG カタログのベース名 (Linux ネイティブ・ファイル・システム内のバイナリー・ファイル *gdgBaseName*.catalogue)。

gdgBaseName には任意の有効なファイル名を使用できます。 ただし、例えば my(+1)Base のような絶対参照または相対参照のようなベース名を指定することは避けてください。

.gxxxxvyy
絶対世代およびバージョン番号。この組み合わせで特定の世代を識別します。ここで、

- *xxxx* は、0001 から 9999 までの符号なしの 4 桁の 10 進数です。
- *yy* は 00 です。 00 のみがサポートされているバージョン番号です。 ゼロ以外のバージョン番号は無視されます。

したがって、絶対名の形式は *gdgBaseName*.g*xxxx*v*yy* です。

****
グループ全体を指定する相対接尾部。 グループ内のすべてのファイルが、現行世代を先頭にして世代順に連結されます。

グループ全体への参照の形式は、*gdgBaseName*(*) または、*gdgBaseName* です。

mmm
0 から 999 までの相対世代番号。特定の世代を識別します。 相対世代番号では、現行世代を (0)、前の世代 (古い世代) を (1) または (-1) のように参照します。 負の符号はオプションです。

存在しない世代への参照はエラーになります。

したがって、既存の世代の相対名の形式は、*gdgBaseName*(*mmm*) または *gdgBaseName*(*-mmm*) です。

+nnnn
1 から 9999 までの正の世代番号。通常は、世代が作成されて GDG に追加されることを識別します。

1 より大きい番号を指定すると、世代番号がスキップされます。 例えば、世代が 1 つしか存在しない場合に、*gdgBaseName*(+3) を指定すると、2 世代がスキップされます。

参照の解決先がグループ内の既存ファイルである場合、*gdgBaseName*(+1234) のような参照を入力用にオープンするのはエラーではありません。

関連概念
124 ページの『世代別データ・グループ』

関連タスク
127 ページの『世代別データ・グループの使用』

関連参照
129 ページの『生成ファイルの挿入と折り返し』
523 ページの『ファイル指定』

生成ファイルの挿入と折り返し

GDG 内の世代は通常、現行世代の後ろに挿入されて、新しい現行世代になります。

増分と現行世代番号の合計が 9999 よりも大きい場合、折り返し (9999 の減算) が行われて、新しい世代番号が現行世代番号よりも小さくなります。 この場合、新しい世代は現行世代の前に挿入され、グループ内で前の (古い) 世代になる場合があります。

次のような初期グループを考えてみます。

```
0: base.g0001v00
```

コロンの前の番号はエポック番号と呼ばれ、この番号により挿入が予測どおりに行われ、世代番号に限界が設定されます。

通常、新しい世代は、折り返しが行われた後に新しい世代番号によって示される位置でグループ内に挿入され、GDG 内のエポック番号は変わりません。

以下の例はそれぞれ、最も古い世代が最初にリストされ、最新の現行世代が最後にリストされています。 イタリックは、グループに追加された新規世代を示します。

上記の初期グループの例を考えてみます。 base(+1) を指定すると、現行世代番号 (0001) が 1 増加します。 この結果、グループには次のように新しい世代 0002 が含まれることになります。

```
0: base.g0001v00
0: base.g0002v00
```

次に base(+4) を指定すると、現行世代番号 (0002) が 4 増加します。 この結果、グループには次のように新しい世代 (0006) が含まれることになります。

```
0: base.g0001v00
0: base.g0002v00
0: base.g0006v00
```

次に、base(+9997) を指定すると、現行世代番号 (0006) が 9997 増加します。 結果の世代番号 (10003) は 9999 より大きくなるため、折り返されて 0004 になります。 結果のグループでは、この新しい世代 (0004) は、次のように現行世代 (0006) の前に挿入されます。

```
0: base.g0001v00
0: base.g0002v00
0: base.g0004v00
0: base.g0006v00
```

この最後の挿入後に、base(+9997) を入力でオープンするのは誤りではありません。これは、base(+9997) への参照が、既存ファイル base.g0004v00 を示し、グループの構造には何も変化を与えないためです。

通常、新しい世代のエポック番号は、現行世代のエポック番号と同じです。 ただし、次に示す 2 つのケースでは、エポック番号が調整されて、新しい世代の挿入位置が少し明確でなくなります。

- 現行世代番号が 9000 以上で、新しい世代番号が 1000 未満の場合、折り返しが行われます。 ただし、新しい世代のエポック番号が増えて、新しい世代の世代番号が現行世代より小さいにも関わらず、新しい世代が現行世代の後に挿入されます。

  例えば、次の初期グループを考えてみます。

```
0: base.g1000v00
0: base.g9000v00
```

base(+1499) を指定すると、現行世代番号 (9000) が 1499 増加します。 結果の世代番号 (10499) は 9999 より大きくなるため、折り返されて 0500 になります。 結果のグループで、新しい世代 (0500) は、大きいエポック番号が与えられて、次のようにグループ内で最も小さい世代番号を持つにも関わらず、新しい現行世代になります。

```
0: base.g1000v00
0: base.g9000v00
1: base.g0500v00
```

- 現行世代番号が 1000 未満で、新しい世代番号が 9000 以上の場合、現行世代のエポック番号が既にゼロでなかった限り、新しい世代のエポック番号は減少します。 既にゼロだった場合、既存の世代のエポック番号が増加して、新しい世代は、世代番号が大きいにも関わらず、すべての世代の前に挿入されます。

  例えば、次の初期グループを考えてみます。

```
0: base.g0001v00
0: base.g0999v00
```

base(+8501) を指定すると、現行世代 (0999) が 8501 増加します。 結果の世代番号 (9500) は 9999 未満であるため、折り返しは行われません。 結果のグループには、新しい世代 (9500) が含まれますが、エポック番号は 0 です。 既存の世代のエポック番号は、1 に増加します。 結果として、新しい世代が、次のようにグループ内で最も古い世代になります。

```
0: base.g9500v00
1: base.g0001v00
1: base.g0999v00
```

関連概念
124 ページの『世代別データ・グループ』

関連タスク
127 ページの『世代別データ・グループの使用』

関連参照
128 ページの『生成ファイルの名前形式』
523 ページの『ファイル指定』

世代別データ・グループの限界処理 (limit processing)

新しい世代が世代別データ・グループに追加されると (通常は OPEN ステートメントの結果として) いつでも、限界処理 (limit processing) が行われます。 また、gdgmgr ユーティリティーの -C (クリーンアップ) オプションを使用して、手動で限界処理 (limit processing) を行うこともできます。

グループの empty オプションが有効な場合、有効期限が切れたファイルはすべて限界処理 (limit processing) によりグループから削除されます。 グループの empty オプションが有効でない場合は、追加が行われると、最も古い世代から最も新しい世代の順で、グループが限界を満たすまで有効期限が切れたファイルが削除されます。

生成ファイルが期限切れとみなされるのは、現在のシステム日付とファイル作成日の差異が、グループの days プロパティーで指定された日数を超えた場合です。

グループから削除された生成ファイルは、グループの scratch オプションが有効な場合、ファイル・システムからも削除されます。

Db2 および SFS ファイルのグループでは、-D *days* オプションがサポートされておらず、グループの days プロパティーは強制的にゼロになります。このため、これらのファイルの経過日数は、限界処理 (limit processing) 時の要因にはなりません。

131 ページの『例: 限界処理』

関連概念
124 ページの『世代別データ・グループ』

関連タスク
125 ページの『世代別データ・グループの作成』
127 ページの『世代別データ・グループの使用』

関連参照
523 ページの『ファイル指定』

例: 限界処理

次の例は、限界処理 (limit processing) が世代別データ・グループの内容に影響を与える様子を示しています。

限界が 3 世代、days オプションが 5 で設定され、noempty オプションが設定されている世代別データ・グループ audit があるとします。この世代別データ・グループは、次の 4 つの新規メンバー生成ファイルで作成されています。

```
0: audit.g0001v00
0: audit.g0003v00
0: audit.g0005v00
0: audit.g0007v00
```

このグループが作成された際はどのファイルも有効期限を過ぎていないため、グループは 3 世代の限界を超えていても許されます。 ただし、7 日後には、既存の世代の有効期限がすべて切れています。 したがって、ここで新しい世代が追加されると、追加後に限界に準拠するように最も古い期限切れの世代が 2 つ削除されます。

通常、世代の追加はプログラムの実行によって行われますが、次の例では、別の方法で世代を追加する方法と、その結果のグループの内容を示しています。

```
gdgmgr -a "audit(+2)" -1
```

```
0: audit.g0005v00
0: audit.g0007v00
0: audit.g0009v00
```

元のグループで代わりに empty オプションが指定されている場合、追加後のグループの内容は次のように生成ファイルが 1 つのみになります。

```
0: audit.g0009v00
```

ファイルの連結

COBOL for Linux では、個々のファイル ID をコロン (:) で区切ることにより、複数のファイルを連結できます。

例えば、次の export コマンドは、MYFILE 環境変数を STL-/home/*myUserID*/file1、続けて STL-/home/*myUserID*/file2 に設定します。

```
export MYFILE='STL-/home/myUserID/file1:STL-/home/myUserID/file2'
```

上記の export コマンドと、COBOL 内部ファイル名と MYFILE 環境変数を関連付ける SELECT および ASSIGN 文節を使用すると、COBOL プログラムでこの 2 つのファイルが単一のファイルとして扱われるようになります。

```
SELECT concatfile ASSIGN TO MYFILE
```

連結がサポートされるのは、連結に使用する *assignment-name* が環境変数 (上記のとおり)、リテラル、または USING データ名の場合です。

ファイル ID の連結に割り当てられる COBOL 内部ファイルは、以下の基準を満たしている必要があります。

- ORGANIZATION は SEQUENTIAL または LINE SEQUENTIAL です。
- ACCESS MODE は SEQUENTIAL です。
- ファイルの OPEN ステートメントのモードが INPUT です。

どのファイル・システムにあるファイルも連結することができます。 特定の連結で、任意の、またはすべてのファイル ID についてファイル・システム ID を指定できます。 ただし、連結内のファイル ID はすべて、同一のファイル・システムを指定しているか、または実行時にデフォルトで同一のファイル・システムに設定される必要があります。 また、ファイルはすべて属性が整合している必要があります。

GDG: 世代別データ・グループ (GDG) 全体を連結することも、1 つ以上のグループから個別の生成ファイルを連結することもできます。 連結で GDG を指定すると、最初に最新の生成ファイルが読み取られ、次に 2 番目に最新のファイルが読み取られ、といったように読み取りが続き、最後にグループ内の最も古い世代の生成ファイルが読み取られます。 新しく定義した、空の GDG を連結に含めるのは誤りです。

連結内の個々のファイル ID の検証は、対応する COBOL ファイルが開かれる時まで行われません。 ファイルが開かれるときに、連結内の最初のファイル ID が解決されて、オープンが試行されます。

OPEN が成功すると、連結内の最初のファイルが最後のレコードに達するまで読み込まれます。 次の READ ステートメントで、連結内の次のファイル ID が解決されてオープンされます。

OPEN ステートメントがオプションの COBOL ファイルで実行された場合に、連結内のファイルがすべて利用不可の場合は、OPEN が成功してファイル状況キーは 05 に設定されます。 最初の READ 操作はファイルの終わりを返し、AT END 条件が存在します。

関連概念
124 ページの『世代別データ・グループ』

関連タスク
113 ページの『ファイルの識別』
170 ページの『ファイルの終わり条件 (AT END) の使用』

関連参照

ASSIGN 節 (*COBOL for Linux on x86* 言語解説書)
ファイル連結 (*COBOL for Linux on x86* 言語解説書)

オプション・ファイルのオープン

存在しないファイルのオープンや読み取りをプログラムが試みると、通常はエラーが発生します。

ただし、存在しないファイルのオープンが意味をなす場合もあります。 このような場合は、次のように SELECT 文節にキーワード OPTIONAL をコーディングします。

```
SELECT OPTIONAL file ASSIGN TO filename
```

ファイルが使用可能であるのか、またはオプションであるのかによって、OPEN の処理、ファイルの作成、および結果のファイル状況キーに影響があります。 例えば、EXTEND、I-O、または INPUT モードで、存在しない OPTIONAL 以外のファイルを開くと、結果は OPEN エラーになり、ファイル状況 35 が返されま

す。 ただし、ファイルが OPTIONAL である場合、同じ OPEN ステートメントでファイル状況 05 が返され、オープン・モード EXTEND および I-O ではファイルが作成されます。

関連タスク
168 ページの『入出力操作でのエラーの処理』
170 ページの『ファイル状況キーの使用』

関連参照

ファイル状況キー (*COBOL for Linux on x86* 言語解説書)
OPEN ステートメントに関する注意事項 (*COBOL for Linux on x86* 言語解説書)

ファイル状況フィールドの設定

ファイル状況キーは、FILE-CONTROL 段落の FILE STATUS 文節および DATA DIVISION のデータ定義を使用して設定します。

```
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    . . .
    FILE STATUS IS file-status
WORKING-STORAGE SECTION.
01  file-status  PIC 99.
```

ファイル状況キー *file-status* は、2 文字のカテゴリー英数字またはカテゴリー国別項目として、あるいは 2 桁の ゾーン 10 進数 (USAGE DISPLAY) (上記に示すとおり) または 国別 10 進数 (USAGE NATIONAL) 項目として指定します。

制限: FILE STATUS 文節で参照されるデータ項目を可変的な場所に置くことはできません。例えば、可変長テーブルの後に置くことはできません。

関連タスク
170 ページの『ファイル状況キーの使用』

関連参照
FILE STATUS 節 (*COBOL for Linux on x86* 言語解説書)

ファイル構造の詳細記述

DATA DIVISION の FILE SECTION では、 キーワード FD と、FILE-CONTROL 段落において 対応する SELECT 文節で使用した同じファイル名を使用して、ファイル記述を開始します。

```
DATA DIVISION.
FILE SECTION.
FD  filename
01  recordname
    nn . . . fieldlength & type
    nn . . . fieldlength & type
    . . .
```

上記の例では、*filename* は、OPEN、READ、 および CLOSE ステートメントでも使用されるファイル名です。

recordname は、WRITE および REWRITE ステートメントで使用されるレコードの名前です。 1 つのファイルに対して複数のレコードを指定することができます。

fieldlength はフィールドの論理長です。*type* はフィールドの形式を指定します。 この方法でレコード記述項目をレベル 01 以上に分ける場合、各エレメントはレコードのフィールドに対して正確にマップしなければなりません。

関連参照

データ関係 (*COBOL for Linux on x86* 言語解説書)
レベル番号 (*COBOL for Linux on x86* 言語解説書)
PICTURE 節 (*COBOL for Linux on x86* 言語解説書)
USAGE 節 (*COBOL for Linux on x86* 言語解説書)

ファイルの入出力ステートメントのコーディング

ENVIRONMENT DIVISION および DATA DIVISION でファイルの識別と記述を行った後、プログラムの PROCEDURE DIVISION でファイル・レコードを処理することができます。

COBOL プログラムは、使用するファイルのタイプ (順次、行順次、索引付き、相対のいずれか) に従ってコーディングします。 入力および出力のコーディングを行う際の一般的なフォーマット (後述の例を参照) には、ファイルのオープン、読み取り、情報の書き込み、およびクローズが含まれます。

134 ページの『例: COBOL でのファイルのコーディング』

関連タスク
113 ページの『ファイルの識別』
121 ページの『ファイル編成およびアクセス・モードの指定』
132 ページの『オプション・ファイルのオープン』
133 ページの『ファイル状況フィールドの設定』
133 ページの『ファイル構造の詳細記述』
136 ページの『ファイルのオープン』
139 ページの『ファイルからのレコードの読み取り』
140 ページの『ファイルへのレコードの追加』
141 ページの『ファイル内のレコードの置換』
141 ページの『ファイルからのレコードの削除』

関連参照

136 ページの『ファイル位置標識』
140 ページの『ファイルへのレコード書き込み時に使用するステートメント』
142 ページの『ファイルの更新に使用する PROCEDURE DIVISION ステートメント』

例: COBOL でのファイルのコーディング

入出力コーディングの一般形式を、以下の例に示します。 コードの後、ユーザーが指定した情報 (例内の小文字のテキスト) について説明します。

```
IDENTIFICATION DIVISION.
. . .
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT filename ASSIGN TO assignment-name  (1) (2)
    ORGANIZATION IS org ACCESS MODE IS access  (3) (4)
    FILE STATUS IS file-status  (5)
    . . .
DATA DIVISION.
FILE SECTION.
FD  filename
01  recordname  (6)
    nn . . . fieldlength & type  (7) (8)
    nn . . . fieldlength & type
    . . .
WORKING-STORAGE SECTION.
01  file-status    PIC 99.
    . . .
PROCEDURE DIVISION.
    OPEN iomode filename   (9)
    . . .
    READ filename
    . . .
    WRITE recordname
    . . .
```

```
    CLOSE filename
  STOP RUN.
```

(1) *filename*
任意の有効な COBOL 名。 SELECT 文節および FD 項目、 および OPEN、READ、START、DELETE、 および CLOSE の各ステートメントでは、同じファイル名を使用する必要があります。 この名前は、必ずしもシステム・ファイル名でなくても構いません。 各ファイルには、独自の SELECT 節、FD 項目、および入出力ステートメントが必要です。 WRITE および REWRITE では、ファイルに対して定義されたレコードを指定します。

(2) *assignment-name*
ASSIGN TO *assignment-name* をコーディングして、ターゲット・ファイル・システム ID とシステム・ファイル名を直接指定するか、または環境変数を使用して値を間接的に設定することができます。

OPEN の際にシステム・ファイル名を識別したい場合は、ASSIGN USING *data-name* を指定できます。当該ファイルに対して OPEN ステートメントが実行されるときには、*data-name* の値が使用されます。また、オプションで、ハイフンを分離文字として使用してシステム・ファイル名の前にファイル・システム ID を付けることも可能です。

次の例は、inventory-file を MOVE ステートメントにより動的にファイル /user/inventory/parts と関連付ける方法を示しています。

```
SELECT inventory-file ASSIGN USING a-file . . .
. . .
FD inventory-file . . .
. . .
77 a-file PIC X(25) VALUE SPACES.
. . .
    MOVE "/user/inventory/parts" TO a-file
    OPEN INPUT inventory-file
```

次の例は、inventory-file を索引付き CICS SFS ファイル parts と動的に関連付ける方法と、代替索引ファイル altpart1 および altpart2 を CICS サーバーの完全修飾名 (この例では /.:/cics/sfs) と関連付ける方法を示しています。

```
SELECT inventory-file ASSIGN USING a-file . . .
    ORGANIZATION IS INDEXED
    ACCESS MODE IS DYNAMIC
    RECORD KEY IS FILESYSFILE-KEY
    ALTERNATE RECORD KEY IS ALTKEY1
    ALTERNATE RECORD KEY IS ALTKEY2. . . .
. . .
FILE SECTION.
FD inventory-file . . .
. . .
WORKING-STORAGE SECTION.
01 a-file  PIC X(80). . .
. . .
    MOVE "/.:/cics/sfs/parts(/.:/cics/sfs/parts;altpart1,/.:/
       cics/sfs/parts;altpart2)" TO a-file
    OPEN INPUT inventory-file
```

(3) *org*
編成 (SEQUENTIAL、LINE SEQUENTIAL、INDEXED、または RELATIVE) を示します。 この文節を省略した場合は、デフォルトの ORGANIZATION SEQUENTIAL が使用されます。

(4) *access*
アクセス・モード (SEQUENTIAL、RANDOM、または DYNAMIC) を示します。 この文節を省略した場合は、デフォルトの ACCESS SEQUENTIAL が使用されます。

(5) *file-status*
COBOL ファイル状況キー。 ファイル状況キーを、2 文字の英数字または 国別データ項目として、あるいは 2 桁のゾーン 10 進数または国別 10 進数項目として指定してください。

(6) ***recordname***
WRITE および REWRITE ステートメントで使用されるレコードの名前。 1 つのファイルに対して複数のレコードを指定することができます。

(7) ***fieldlength***
フィールドの論理長。

(8) ***type***
ファイルのレコード形式と一致していなければなりません。 レコード記述項目をレベル 01 以上に分ける場合は、各エレメントはレコードのフィールドに対して正確にマップします。

(9) ***iomode***
オープン・モードを指定します。 ファイルから読み取りだけを行う場合は、INPUT をコーディングします。 ファイルへの書き込みだけを行う場合は、OUTPUT (新規ファイルのオープンまたは既存ファイル上での書き込み) または EXTEND (ファイル末尾へのレコードの追加) をコーディングします。 読み取りと書き込みの両方を行う場合は、I-O をコーディングします。

制約事項 : 行順次ファイルの場合、I-O は OPEN ステートメントの有効なモードではありません。

ファイル位置標識

ファイル位置標識は、順次 COBOL 要求の場合にアクセスされる次のレコードを示します。

ファイル位置標識は、プログラマーがプログラム内に明示的に設定するものではありません。 この標識は、成功した OPEN、START、READ、READ NEXT、および READ PREVIOUS ステートメントによって設定されます。 その後の READ、READ NEXT、または READ PREVIOUS 要求は、設定されたファイル位置標識の位置を使用した後、それを更新します。

ファイル位置標識は、出力ステートメント WRITE、REWRITE、または DELETE によって使用されたり、影響を受けたりすることはありません。 ファイル位置標識は、ランダム処理では意味がありません。

ファイルのオープン

WRITE、START、 READ、REWRITE、 または DELETE ステートメントを使用してファイル内のレコードを処理するためには、 前もってそのファイルを OPEN ステートメントを使用してオープンしておかなければなりません。

```
PROCEDURE DIVISION.
    . . .
    OPEN iomode filename
```

上記の例では、*iomode* はオープン・モードを指定します。 ファイルから読み取りだけを行う場合は、オープン・モードに対して INPUT をコーディングします。 ファイルへの書き込みだけを行う場合は、 オープン・モードに対して OUTPUT (新規ファイルのオープンまたは 既存ファイル上での書き込み) または EXTEND (ファイル末尾へのレコードの追加) を コーディングします。

すでにレコードが入っているファイルをオープンするには、OPEN INPUT、OPEN I-O (行順次ファイルの場合は無効)、または OPEN EXTEND を使用してください。

レコードが格納されている SdU ファイルまたは SFS ファイルに対して OPEN OUTPUT とコーディングすると、COBOL ランタイムがそのファイルを削除し、その後で COBOL で提供される属性を持つファイルを作成します。 SdU ファイルまたは SFS ファイルを削除したくない場合は、代わりに OPEN I-O とコーディングしてファイルをオープンします。

順次ファイル、行順次ファイル、または相対ファイルを EXTEND としてオープンする場合には、ファイル内の最後の既存レコードの後に、追加されたレコードが置かれます。 索引付きファイルを EXTEND としてオープンする場合には、追加するそれぞれのレコードが、ファイル内の最高位レコードよりも大きいレコード・キーを持っていなければなりません。

関連概念
121 ページの『ファイル編成およびアクセス・モード』

関連タスク
132 ページの『オプション・ファイルのオープン』

関連参照
137 ページの『順次ファイルに有効な COBOL ステートメント』
137 ページの『行順次ファイルに有効な COBOL ステートメント』
138 ページの『索引付きファイルおよび相対ファイルに有効な COBOL ステートメント』

OPEN ステートメント (*COBOL for Linux on x86* 言語解説書)

順次ファイルに有効な COBOL ステートメント

次の表に、順次ファイルに対して使用可能な入出力ステートメントの組み合わせを示します。「X」のマークは、当該列の最上部に記載されているオープン・モードで、当該ステートメントが使用できることを示しています。

表 *10.* **順次ファイルに有効な *COBOL* ステートメント**

アクセス・モード	COBOL ステートメント	OPEN INPUT	OPEN OUTPUT	OPEN I-O	OPEN EXTEND
順次	OPEN	X	X	X	X
	WRITE		X		X
	START				
	READ	X		X	
	REWRITE			X	
	DELETE				
	CLOSE	X	X	X	X

関連概念
122 ページの『順次ファイルの編成』
123 ページの『順次アクセス』

行順次ファイルに有効な COBOL ステートメント

次の表に、行順次ファイルに対して使用可能な入出力ステートメントの組み合わせを示します。「X」のマークは、当該列の最上部に記載されているオープン・モードで、当該ステートメントが使用できることを示しています。

表 *11.* **行順次ファイルに有効な *COBOL* ステートメント**

アクセス・モード	COBOL ステートメント	OPEN INPUT	OPEN OUTPUT	OPEN I-O	OPEN EXTEND
順次	OPEN	X	X		X
	WRITE		X		X
	START				
	READ	X			
	REWRITE				
	DELETE				
	CLOSE	X	X		X

関連概念
122 ページの『行順次ファイル編成』
123 ページの『順次アクセス』

索引付きファイルおよび相対ファイルに有効な COBOL ステートメント

次の表に、索引付きファイルおよび相対ファイルに対して使用可能な入出力ステートメントの組み合わせを示します。「X」のマークは、当該列の最上部に記載されているオープン・モードで、当該ステートメントが使用できることを示しています。

表 *12.* **索引付きファイルおよび相対ファイルに有効な *COBOL* ステートメント**

アクセス・モード	COBOL ステートメント	OPEN INPUT	OPEN OUTPUT	OPEN I-O	OPEN EXTEND
順次	OPEN	X	X	X	X
	WRITE		X		X
	START	X		X	
	READ	X		X	
	REWRITE			X	
	DELETE			X	
	CLOSE	X	X	X	X
ランダム	OPEN	X	X	X	
	WRITE		X	X	
	START				
	READ	X		X	
	REWRITE			X	
	DELETE			X	
	CLOSE	X	X	X	
動的	OPEN	X	X	X	
	WRITE		X	X	
	START	X		X	
	READ	X		X	
	REWRITE			X	
	DELETE			X	
	CLOSE	X	X	X	

関連概念
123 ページの『索引付きファイル編成』
123 ページの『相対ファイル編成』
123 ページの『順次アクセス』
124 ページの『ランダム・アクセス』
124 ページの『動的アクセス』

ファイルからのレコードの読み取り

READ ステートメントを使用して、ファイルからレコードを取り出します。 レコードを読み取るには、OPEN INPUT または OPEN I-O (OPEN I-O は行順次ファイルでは無効) を使用してファイルをオープンする必要があります。 各 READ の後で、ファイル状況キーを検査してください。

順次ファイルおよび行順次ファイルの中のレコードは、それらが書き込まれたシーケンスでしか検索することができません。

索引付きおよび相対レコード・ファイルの中のレコードは、順次 (索引付きファイルの場合はキーの昇順に従い、相対ファイルの場合は相対レコード位置に従う)、ランダム、または動的に検索することができます。

動的アクセスを使用する場合は、レコードの順次読み取りと、特定のレコードの直接読み取りを切り替えることができます。 順次検索の場合は、READ NEXT と READ PREVIOUS をコーディングします。(キーによる) ランダム検索の場合は、READ を使用します。

特定のレコードから順次に読み取るには、READ NEXT または READ PREVIOUS ステートメントの前に START ステートメントを使用して、ファイル位置指示子が特定のレコードを指すように設定します。 START の後に READ NEXT をコーディングすると、次のレコードが読み取られ、ファイル位置標識は次のレコードにリセットされます。 START の後に READ PREVIOUS をコーディングすると、前のレコードが読み取られ、ファイル位置標識は前のレコードにリセットされます。 ファイル位置標識は、START を使用してランダムに移動することができますが、すべての読み取りはそのポイントから順次に行われることになります。

レコードの順次読み取りを続行することも、START を使用してファイル位置標識を移動することも可能です。 次に例を示します。

```
START file-name KEY IS EQUAL TO ALTERNATE-RECORD-KEY
```

重複が存在する代替索引に基づいて、索引付きファイルに対して直接 READ が実行されると、その代替キー値を持つファイル (基本クラスター) の最初のレコードのみが検索されます。 同じ代替キーを持つレコードのそれぞれを検索するためには、一連の READ NEXT ステートメントが必要です。 同じ代替キーを持つ読み取り対象レコードがまだほかにある場合は、ファイル状況値 02 が戻されます。 そのキー値を持つ最後のレコードが読み取られると、値 00 が戻されます。

関連概念
123 ページの『順次アクセス』
124 ページの『ランダム・アクセス』
124 ページの『動的アクセス』
121 ページの『ファイル編成およびアクセス・モード』

関連タスク
136 ページの『ファイルのオープン』
170 ページの『ファイル状況キーの使用』

関連参照
136 ページの『ファイル位置標識』
FILE STATUS 節 (*COBOL for Linux on x86* 言語解説書)

ファイルへのレコード書き込み時に使用するステートメント

次の表に、ファイルの作成または拡張時に使用できる COBOL ステートメントを示します。

表 *13.* **ファイルへのレコード書き込み時に使用するステートメント**

除算	順次	行順次	索引付き	相対
ENVIRONMENT	SELECT			
	ASSIGN			
	ORGANIZATION IS SEQUENTIAL	ORGANIZATION IS LINE SEQUENTIAL	ORGANIZATION IS INDEXED	ORGANIZATION IS RELATIVE
	n/a		RECORD KEY	RELATIVE KEY
			ALTERNATE RECORD KEY	
	FILE STATUS			
	ACCESS MODE			
DATA	FD 記入項目			
PROCEDURE	OPEN OUTPUT			
	OPEN EXTEND			
	WRITE			
	CLOSE			

関連概念
121 ページの『ファイル編成およびアクセス・モード』

関連タスク
121 ページの『ファイル編成およびアクセス・モードの指定』
136 ページの『ファイルのオープン』
133 ページの『ファイル状況フィールドの設定』
140 ページの『ファイルへのレコードの追加』

関連参照

142 ページの『ファイルの更新に使用する PROCEDURE DIVISION ステートメント』

ファイルへのレコードの追加

COBOL の WRITE ステートメントは、既存のレコードを置き換えずに、ファイルにレコードを追加します。追加されるレコードは、ファイルの定義時に設定された最大レコード・サイズを超えてはなりません。 それぞれの WRITE ステートメントの後で、ファイル状況キーを検査してください。

レコードを順次に追加する場合: OUTPUT または EXTEND として オープンされたファイルの終わりにレコードを順次に追加するには、ACCESS IS SEQUENTIAL を使用し、WRITE ステートメントをコーディングしてください。

順次ファイルおよび行順次ファイルは、必ず順次に書き込まれます。

索引付きファイルの場合は、昇順キー配列で新規のレコードを追加しなければなりません。 ファイルが EXTEND としてオープンされている場合、追加されるレコードのレコード・キーは、ファイルがオープンされた時点のファイルの最高位基本レコード・キーよりも大きくなければなりません。

相対ファイルの場合、レコードは正しい順序になっていなければなりません。 SELECT 文節に RELATIVE KEY データ項目をコーディングすると、書き込まれるレコードの相対レコード番号がそのデータ項目に入れられます。

レコードをランダムまたは動的に追加する場合: ACCESS IS RANDOM または ACCESS IS DYNAMIC をコーディングしているレコードを索引付きファイルに書き込むとき、 レコードは任意の順序で書き込むことができます。

関連概念
121 ページの『ファイル編成およびアクセス・モード』

関連タスク
121 ページの『ファイル編成およびアクセス・モードの指定』
170 ページの『ファイル状況キーの使用』

関連参照
140 ページの『ファイルへのレコード書き込み時に使用するステートメント』
142 ページの『ファイルの更新に使用する PROCEDURE DIVISION ステートメント』

FILE STATUS 節 (*COBOL for Linux on x86* 言語解説書)

ファイル内のレコードの置換

ファイル内のレコードを置換するには、ファイルを I-O で開いた場合は REWRITE を使用します。 ファイルが I-O 以外で開かれた場合、レコードは置換されず、状況キーが 49 に設定されます。

それぞれの REWRITE ステートメントの後で、ファイル状況キーを検査してください。

順次ファイルの場合、置換レコードの長さは元のレコードの長さと同じでなければなりません。 索引ファイルまたは可変長相対ファイルの場合、置換するレコードの長さを変更することができます。

レコードをランダムまたは動的に置換するには、まずレコードを READ する必要はありません。 そうではなく、次のように置換対象のレコードを検索します。

- 索引付きファイルの場合、レコード・キーを RECORD KEY データ項目に移動してから、REWRITE を使用します。
- 相対ファイルの場合、相対レコード番号を RELATIVE KEY データ項目に移動してから、REWRITE を使用します。

関連概念
121 ページの『ファイル編成およびアクセス・モード』

関連タスク
136 ページの『ファイルのオープン』
170 ページの『ファイル状況キーの使用』

関連参照

FILE STATUS 節 (*COBOL for Linux on x86* 言語解説書)

ファイルからのレコードの削除

既存のレコードを索引付きまたは相対ファイルから削除するには、ファイルを I-O として開き、DELETE ステートメントを使用します。 順次または行順次ファイルには DELETE を使用できません。

ACCESS IS SEQUENTIAL の場合は、削除するレコードを COBOL プログラムがまず読み取る必要があります。 DELETE ステートメントは、 読み取られたレコードを除去します。 DELETE の前の READ が成功しなかった場合には、削除は行われず、ファイル状況キーが 92 に設定されます。

ACCESS IS RANDOM または ACCESS IS DYNAMIC の場合は、削除するレコードを COBOL プログラムが読み取る必要はありません。 レコードを削除するには、レコードのキーを RECORD KEY データ項目に移動してから、DELETE を発行します。

それぞれの DELETE ステートメントの後で、ファイル状況キーを検査してください。

関連概念
121 ページの『ファイル編成およびアクセス・モード』

関連タスク
136 ページの『ファイルのオープン』
139 ページの『ファイルからのレコードの読み取り』
170 ページの『ファイル状況キーの使用』

関連参照
FILE STATUS 節 (*COBOL for Linux on x86* 言語解説書)

ファイルの更新に使用する PROCEDURE DIVISION ステートメント

次の表に、順次ファイル、行順次ファイル、索引付きファイル、および相対ファイルの PROCEDURE DIVISION で使用できるステートメントを示します。

表 *14.* **ファイルの更新に使用する *PROCEDURE DIVISION* ステートメント**

アクセス方式	順次	行順次	索引付き	相対
ACCESS IS SEQUENTIAL	OPEN EXTEND WRITE CLOSE または OPEN I-O READ REWRITE CLOSE	OPEN EXTEND WRITE CLOSE	OPEN EXTEND WRITE CLOSE または OPEN I-O READ REWRITE DELETE CLOSE	OPEN EXTEND WRITE CLOSE または OPEN I-O READ REWRITE DELETE CLOSE
ACCESS IS RANDOM	適用されない	適用されない	OPEN I-O READ WRITE REWRITE DELETE CLOSE	OPEN I-O READ WRITE REWRITE DELETE CLOSE
ACCESS IS DYNAMIC (順次処理)	適用されない	適用されない	OPEN I-O READ NEXT READ PREVIOUS START CLOSE	OPEN I-O READ NEXT READ PREVIOUS START CLOSE

表 14. **ファイルの更新に使用する *PROCEDURE DIVISION* ステートメント** (続き)				
アクセス方式	**順次**	**行順次**	**索引付き**	**相対**
ACCESS IS DYNAMIC (ランダム処理)	適用されない	適用されない	OPEN I-O READ WRITE REWRITE DELETE CLOSE	OPEN I-O READ WRITE REWRITE DELETE CLOSE

関連概念
121 ページの『ファイル編成およびアクセス・モード』

関連タスク
136 ページの『ファイルのオープン』
139 ページの『ファイルからのレコードの読み取り』
140 ページの『ファイルへのレコードの追加』
141 ページの『ファイル内のレコードの置換』
141 ページの『ファイルからのレコードの削除』

関連参照

140 ページの『ファイルへのレコード書き込み時に使用するステートメント』

Db2 ファイルの使用

Linux で実行される COBOL アプリケーションから Db2 ファイルにアクセスするには、アプリケーションのコンパイルおよびリンクに関するガイドラインや、Db2 ファイル・システム、Db2 インスタンスおよびデータベース、および Db2 ファイルを識別するためのガイドラインに従う必要があります。

1. cob2 コマンドを使用して、アプリケーション内の COBOL プログラムをコンパイルおよびリンクします。
2. 使用する Db2 インスタンスのプロファイルを実行して、Db2 環境を初期化します。

   例えば、インスタンス db2inst1 を使用するために次のコマンドを発行する場合があります。

   ```
   . /home/db2inst1/sqllib/db2profile
   ```

3. Db2 インスタンスが実行されていること、およびアクセス先のデータベースに接続可能なことを確認してください。

   次の例は、データベース db2cob に接続するために使用できる db2 コマンドと、システムの応答を示しています。

   ```
   > db2 connect to db2cob
           Database Connection Information
           Database server        = DB2/Linux64 9.7.0
           SQL authorization ID   = MYUID
           Local database alias   = DB2COB
   ```

   制約事項 : COBOL アプリケーションは、一度に 1 つのデータベースおよび Db2 インスタンス内のみの Db2 ファイルにアクセスできます。
4. 環境変数 DB2DBDFT を、対象データベースに設定します。 以下に、その例を示します。

   ```
   export DB2DBDFT=db2cob
   ```

5. Db2 を使用するファイルには、Db2 ファイル・システムを指定します (FILESYS ランタイム・オプションの値として、または割り当て名値に直接指定)。スキーマ名を含め、完全修飾 Db2 表名を使用します。

   例えば、次のコマンドにより、トランザクション・ファイル TRANFILE が、Db2 ファイル・システムのシステム・ファイル名 TEST.TRANS に割り当て完了されます。

```
export TRANFILE=DB2-TEST.TRANS
```

Db2 ファイルの作成:

Db2 ファイルは、次のいくつかの方法のいずれかで作成できます。

- COBOL プログラム内で OPEN ステートメントを使用
- CICS TXSeries または CICS TX の cicsddt ユーティリティーを使用

  cicsddt コマンドについて詳しくは、以下の関連参照に記載されている CICS TXSeries または CICS TX の資料を参照してください。
- db2 create コマンドを使用

例えば、次のコマンド・シーケンスを使用すると、スキーマ CICS で EXAMPLE という名前の相対ファイルを作成できます。

```
db2 create table cics.example¥
"(rba char(4) not null for bit data, f1 varchar(80) not null for bit data)"
db2 create unique index cics.example0 on cics.example¥
"(rba) disallow reverse scans"
db2 create unique index cics.example0@ on cics.example¥
"(rba desc) disallow reverse scans"
```

結果の表は、db2 describe コマンドを発行すると表示できます。 以下に、その例を示します。

```
> db2 describe table cics.example

                        Data type                      Column
Column name             schema    Data type name       Length     Scale Nulls
----------------------- --------- -------------------- ---------- ----- ------
RBA                     SYSIBM    CHARACTER                     4     0 No
F1                      SYSIBM    VARCHAR                      80     0 No

  2 record(s) selected.
```

ファイル CICS.EXAMPLE は、COBOL の FILE SECTION での定義で 0 から 79 の間の最小レコード長と互換性のある可変長レコードを持ちます。

db2 ユーティリティーによって提供される機能の詳細については、db2 コマンドを入力してください。

TXSeries または CICS TX Knowledge Center で Db2 ファイルを 使用する場合に適用される追加要件については、Db2 ファイル・システムに関する関連参照を参照してください。

関連タスク
113 ページの『ファイルの識別』
115 ページの『Db2 ファイルの識別』
145 ページの『同一プログラム内での Db2 ファイルと SQL ステートメントの使用』
225 ページの『コマンド行からのコンパイル』
377 ページの『第 17 章 Db2 環境用のプログラミング』

関連参照
118 ページの『Db2 ファイル・システム』
216 ページの『コンパイラー環境変数とランタイム環境変数』
300 ページの『FILESYS』
TXSeries for Multiplatforms の資料
IBM CICS TX on Cloud の資料

同一プログラム内での Db2 ファイルと SQL ステートメントの使用

同一プログラム内で EXEC SQL ステートメントと Db2 ファイルの入出力を使用するには、次に説明するとおり、いくつかの重要事項を理解しておく必要があります。

- 両機能 (EXEC SQL ステートメントと Db2 ファイル入出力) とも、同じ Db2 接続を使用します。
- Db2 ファイル・システムを使用する各 COBOL 入出力更新操作は、すぐにデータベースにコミットされます。
- 既存の Db2 接続が使用可能な場合、Db2 ファイル・システムはその接続を使用します。

  接続が使用不可の場合、Db2 ファイル・システムは、DB2DBDFT 環境変数の値で参照されるデータベースに接続を確立します。

EXEC SQL ステートメントと Db2 ファイルの入出力では、同じデータベースを使用できます。または、接続を明示的に制御する場合は、別のデータベースを使用できます。

同じデータベースを使用する場合

同一プログラム内の EXEC SQL ステートメントと Db2 ファイル入出力で同じデータベースを使用する方が、別々のデータベースを使用するよりも簡単です。ただし、簡単ではあっても、以下のように、この構成を注意深く扱う必要があります。

- 異常を避けるため、Db2 ファイル・システムによる既存の接続の使用に依存しないでください。どちらの種類 (Db2 ファイル入出力、または EXEC SQL) のデータベースへのアクセスが先に行われるかに関係なく一貫性のある結果を得るために、EXEC SQL ステートメントが使用するデータベースと同じデータベースに環境変数 DB2DBDFT を設定します。
- Db2 ファイルの更新操作は、データベースへの保留中の処理をすべてコミットします。したがって、保留中の EXEC SQL 更新はすべて、Db2 ファイルの入出力操作を開始する前にロールバックまたはコミットしてください。

  入力用に Db2 ファイルをオープンしてファイルを読み取ってもデータベースのコミットは行われませんが、この動作に依存しないことをお勧めします。

別々のデータベースを使用する場合

同一プログラム内の EXEC SQL ステートメントと Db2 ファイル入出力で別々のデータベースを使用するには、次の例に示すように、データベース接続を明示的に制御する必要があります。

データベース db2pli で EXEC SQL ステートメントを使用し、環境変数 DB2DBDFT を設定してデータベース db2cob で Db2 ファイル入出力を行うとします。

```
export DB2DBDFT=db2cob
```

次の例では、(疑似コードを示す不等号括弧で) 示されている手順シーケンスを行っても、インライン・コメントに説明されているとおり、目的とするデータベースは正しく使用されません。

```
<DB2 file I/O>                          *> Uses database db2cob (from DB2DBDFT)
EXEC SQL CONNECT TO db2pli END-EXEC     *> Switches to database db2pli
<Other EXEC SQL operations>             *> Use database db2pli
<DB2 file I/O>                          *> Uses the existing connection--and
. . .                                   *>   thus database db2pli--incorrectly!
```

目的のデータベースにアクセスするには、まず EXEC SQL ステートメントで使用されたデータベースを切断してから Db2 ファイルの入出力操作を行います。次に、環境変数 DB2DBDFT の値に依存するか、または Db2 ファイル入出力に使用するデータベースへ明示的に接続します。

次の手順シーケンスは、DB2DBDFT に依存して目的の接続を正しく行う様子を示しています。

```
<DB2 file I/O>                          *> Uses database db2cob (from DB2DBDFT)
EXEC SQL CONNECT TO db2pli END-EXEC     *> Switches to database db2pli
<Other EXEC SQL operations>             *> Use database db2pli
* Commit or roll back pending operations
* here, because the following statement
* unconditionally commits pending work:
```

```
EXEC SQL CONNECT RESET END-EXEC       *> Disconnect from database db2pli
<DB2 file I/O>                        *> Uses database db2cob (from DB2DBDFT)
. . .
```

関連タスク
378 ページの『SQL ステートメントのコーディング』

関連参照
216 ページの『コンパイラー環境変数とランタイム環境変数』

QSAM ファイルの使用

QSAM (待機順次アクセス方式) ファイルは、キーなしのファイルであり、レコードは入力順に配置されます。

プログラムでは、これらのファイルを順次でのみ処理することができ、READ ステートメントを使用して、レコードをファイル内に存在しているのと同じ順序で検索します 各レコードは先行レコードの後に置かれます。 プログラムで QSAM ファイルを処理するには、次のような COBOL 言語ステートメントを使用します。

- ENVIRONMENT DIVISION および DATA DIVISION 内で QSAM ファイルを識別および記述する
- PROCEDURE DIVISION 内でこれらのファイル内のレコードを処理する

レコードの作成後は、ファイル内でレコードの長さや位置を変えることはできず、また削除することもできません。

関連タスク
113 ページの『ファイルの識別』

関連参照
119 ページの『QSAM ファイル・システム』
300 ページの『FILESYS』

SFS ファイルの使用

Linux で実行される COBOL アプリケーションから CICS SFS ファイルにアクセスするには、コンパイルおよびリンクに関するガイドラインや、ファイル・システム、CICS SFS サーバー、および SFS ファイルを識別するためのガイドラインに従う必要があります。

1. cob2 コマンドを使用して、アプリケーション内の COBOL プログラムをコンパイルおよびリンクします。
2. アプリケーションがアクセスする CICS SFS サーバーが稼働中であることを確認します。
3. (オプション) アプリケーションで 1 つ以上の SFS ファイルを作成し、sfs_SSFS_SERVER 以外の名前を持つ SFS データ・ボリューム上にファイルを割り当てたい場合は、次の名前のいずれかか両方を指定することができます。
   - SFS ファイルが作成される SFS データ・ボリュームの名前。 このためには、ランタイム環境変数 CICS_SFS_INDEX_VOLUME に値を割り当てます。SFS サーバーに対してデータ・ボリュームが定義されている必要があります。 SFS サーバーに対してどのデータ・ボリュームが使用可能かわからない場合は、コマンド sfsadmin list lvols を発行します。

     デフォルトでは、SFS ファイルは sfs_SSFS_SERVER という名前のデータ・ボリューム上に作成されます。
   - 代替索引ファイルが存在する場合は、それが作成される SFS データ・ボリュームの名前。 このためには、ランタイム環境変数 CICS_SFS_INDEX_VOLUME に値を割り当てます。SFS サーバーに対してデータ・ボリュームが定義されている必要があります。

     デフォルトでは、代替索引ファイルは対応するベース・ファイルと同じボリューム上に作成されます。
4. 各 SFS ファイルを識別します。

- ランタイム・オプション FILESYS を次のように設定して、デフォルトのファイル・システムを SFS に設定します。

  ```
  export COBRTOPT=FILESYS=SFS
  ```

  または、各 SFS ファイルの export コマンドで、下に示すように、ファイル・システム ID SFS とハイフン (-) をファイル名と SFS サーバー名の前に付けます。
- CICS SFS サーバー名はファイル名の前に付ける必要があります。
- 代替索引ファイル名はすべて、基本ファイル名で始まり、その後セミコロン (;) と代替索引名を付ける必要があります。

例えば、/.:/cics/sfs/*sfsServer* が CICS SFS サーバーで、 SFS04A が代替索引 SFS04A1 を含む SFS ファイルの場合、SFS04A を識別するには次の export コマンドを発行します。

```
export SFS04AEV="SFS-/.:/cics/sfs/sfsServer/SFS04A(/.:/cics/sfs/sfsServer/SFS04A;SFS04A1)"
```

SFS サーバーおよびファイルの完全修飾名について詳しくは、CICS SFS ファイルの識別に関する関連タスクを参照してください。

147 ページの『例: SFS ファイルへのアクセス』

関連タスク
113 ページの『ファイルの識別』
115 ページの『SFS ファイルの識別』
149 ページの『SFS パフォーマンスの向上』

関連参照
120 ページの『SFS ファイル・システム』
220 ページの『ランタイム環境変数』
300 ページの『FILESYS』

例: SFS ファイルへのアクセス

下記の例は、コーディング可能な COBOL ファイルの記述を示しています。さらに、2 つの SFS ファイルを作成してそれらにアクセスするために発行できる sfsadmin コマンドと export コマンドも示しています。

SFS04 は、代替索引を含まない索引付きファイルです。 SFS04A は、1 つの代替索引 SFS04A1 を含む索引付きファイルです。

COBOL ファイル記述

```
. . .
Environment division.
Input-output section.
File-control.
    select SFS04-file
        assign to SFS04EV
        access dynamic
        organization is indexed
        record key is SFS04-rec-num
        file status is SFS04-status.

    select SFS04A-file
        assign to SFS04AEV
        access dynamic
        organization is indexed
        record key is SFS04A-rec-num
        alternate record key is SFS04A-date-time
        file status is SFS04A-status.
```

```
Data division.
File section.
FD  SFS04-file.
  01 SFS04-record.
     05 SFS04-rec-num              pic x(10).
     05 SFS04-rec-data             pic x(70).
FD  SFS04A-file.
  01 SFS04A-record.
     05 SFS04A-rec-num             pic x(10).
     05 SFS04A-date-time.
        07 SFS04A-date-yyyymmdd    pic 9(8).
        07 SFS04A-time-hhmmsshh    pic 9(8).
        07 SFS04A-date-time-counter pic 9(8).
     05 SFS04A-rec-data            pic x(1000).
```

sfsadmin コマンド

sfsadmin create clusteredfile コマンドを発行して それぞれの索引付きファイルを作成し、sfsadmin add index コマンドを発行して代替索引を追加します。

```
sfsadmin create clusteredfile SFS04 2 ¥
PrimaryKey byteArray 10 ¥
DATA byteArray 70 ¥
primaryIndex -unique 1 PrimaryKey sfsVolume
#
sfsadmin create clusteredfile SFS04A 3 ¥
PrimaryKey byteArray 10 ¥
AltKey1 byteArray 24 ¥
DATA byteArray 1000 ¥
primaryIndex -unique 1 PrimaryKey sfsVolume
#
sfsadmin add index SFS04A SFS04A1 1 AltKey1
```

上記の最初の sfsadmin create clusteredfile コマンドで示されているように、次の項目を指定する必要があります。

- 新規索引付きファイルの名前 (この例では SFS04)
- レコード当たりのフィールド数 (2)
- 各フィールドの記述 (PrimaryKey および DATA で、それぞれがバイト配列)
- 1 次索引の名前 (primaryIndex)
- -unique オプション
- 1 次索引内のフィールド数 (1)
- 1 次索引内のフィールドの名前 (PrimaryKey)
- ファイルの保管先のデータ・ボリュームの名前 (*sfsVolume*)

CICS SFS では、デフォルトでクラスター・ファイルの 1 次索引に重複キーを使用できます。ただし、COBOL では 1 次索引のキー値がファイル内で固有でなければならないため、 上記のように -unique オプションを指定する必要があります。

上記の sfsadmin add index コマンドで示されているように、次の項目を指定する必要があります。

- 代替索引の追加先のファイルの名前 (この例では SFS04A)
- 新規索引の名前 (SFS04A1)
- 新規索引でキーとして使用するフィールドの数 (1)
- 新規索引内のフィールドの名前 (AltKey1)

コマンド sfsadmin create clusteredfile および sfsadmin add index の構文について詳しくは、関連参照を参照してください。

export コマンド

SFS ファイルを処理するプログラムを実行する前に、以下の export コマンドを発行して、ファイルにアクセスする CICS SFS サーバー (*sfsServer*) へのパス (/.:/cics/sfs)、およびファイルを保管するデータ・ボリューム (*sfsVolume*) を指定します。

```
# Set environment variables required by the SFS file system
# for SFS files:

export CICS_SFS_DATA_VOLUME=sfsVolume
export CICS_SFS_INDEX_VOLUME=sfsVolume

# Set SFS as the default file system:
export COBRTOPT=FILESYS=SFS

# Enable use of a short-form SFS specification:
export CICS_TK_SFS_SERVER=/.:/cics/sfs/sfsServer

# Set the environment variable to access SFS file SFS04
# (an example of using a short-form SFS specification):
export SFS04EV=SFS04

# Set the environment variable to access SFS
# file SFS04A and the alternate index SFS04A1:
```

```
export SFS04AEV="/.:/cics/sfs/sfsServer/SFS04A(/.:/cics/sfs/sfsServer/
SFS04A;SFS04A1)"
```

関連参照
CICS TXSeries の資料

SFS パフォーマンスの向上

SFS ファイルにアクセスするアプリケーションのパフォーマンスを向上させる方法が 2 つあります。 1 つは、クライアント・マシンでクライアント側のキャッシュを使用するという方法です。 もう 1 つは、SFS ファイルに変更内容を保存する頻度を減らすという方法です。

関連タスク
115 ページの『SFS ファイルの識別』
149 ページの『クライアント側のキャッシュの使用可能化』
150 ページの『変更の保存回数の削減』
CICS TXSeries 資料 の SFS のパフォーマンスの向上

関連参照

120 ページの『SFS ファイル・システム』

クライアント側のキャッシュの使用可能化

デフォルトでは、SFS ファイル内のレコードは SFS サーバーとの間で読み書きされ、レコードにアクセスするたびにリモート・プロシージャー・コール (RPC) が必要です。 ただし、クライアント側のキャッシュを有効にすると、レコードへのアクセスに必要な時間が短くなるため、パフォーマンスを向上させることができます。

クライアント側のキャッシュを使用している場合、レコードはクライアント・マシンのローカル・メモリー (キャッシュ) に 保管され、1 回の RPC でサーバーに送信 (フラッシュ) されます。 読み取りキャッシュと挿入キャッシュという 2 種類のキャッシュのいずれか、または両方とも指定することができます。

- 読み取りキャッシュを使用可能にすると、ファイルの最初の順次読み取りにより、現行レコードおよび隣接レコードの数がサーバーから読み取られ、読み取りキャッシュ内に入れられます。 以降の順次読み取り、更新、および削除は、サーバー上ではなく読み取りキャッシュ内で行われます。

- 挿入キャッシュを使用可能にすると、挿入操作 (読み取り、更新、または削除ではなく) は、サーバー上ではなく挿入キャッシュ内で行われます。

アプリケーション内のすべての SFS ファイルに対してクライアント側のキャッシュを使用可能にするには、アプリケーションを実行する前に CICS_VSAM_CACHE 環境変数を設定します。 CICS_VSAM_CACHE の設定を示す構文図については、ランタイム環境変数に関する関連参照を参照してください。

キャッシュ・サイズとして単一の値をコーディングして、読み取りキャッシュと挿入キャッシュに同じページ数を使用するよう指定したり、値をコロン (:) で区切って読み取りおよび挿入キャッシュに対して異なる値をコーディングすることができます。 サイズの単位はページ数として表現します。読み取りキャッシュ、挿入キャッシュ、またはその両方のサイズとしてゼロをコーディングすると、そのタイプのキャッシュは使用不可になります。 例えば、次のコマンドは、アプリケーション内の各 SFS ファイルの読み取りキャッシュのサイズを 16 ページに設定し、挿入キャッシュのサイズを 64 ページに設定します。

```
export CICS_VSAM_CACHE=16:64
```

また、クライアント側のキャッシュをより柔軟なものにするために、次のフラグのいずれかまたは両方を指定することもできます。

- 非コミット・データ (新規または変更済みだが、サーバーに送信されなかったレコードで、ダーティー・レコードとも言う) の読み取りを許可するには、ALLOW_DIRTY_READS を指定します。

  このフラグは、アクセスするファイルがロックされている必要があるという読み取りキャッシュの制限を取り除きます。
- 任意の挿入のキャッシュを許可するには、INSERTS_DESPITE_UNIQUE_INDICES を指定します。

  このフラグは、クラスター・ファイルのすべてのアクティブな索引と、入力順および相対ファイルのアクティブな代替索引が、重複を許可している必要があるという、挿入キャッシュの制限を取り除きます。

例えば、次のコマンドは最大の柔軟性を可能にします。

```
export CICS_VSAM_CACHE=16:64:ALLOW_DIRTY_READS,INSERTS_DESPITE_UNIQUE_INDICES
```

特定のファイルのそれぞれにクライアント側のキャッシュを設定するには、キャッシュ方針を変更したい各ファイルの OPEN ステートメントの前に、CICS_VSAM_CACHE を設定する putenv() 呼び出しをコーディングします。 プログラムの中では、putenv() 呼び出しで行う環境変数の設定は、 `export` コマンドで行う環境変数の設定に優先します。

224 ページの『例: 環境変数の設定とアクセス』

関連タスク
215 ページの『環境変数の設定』

関連参照
120 ページの『SFS ファイル・システム』
220 ページの『ランタイム環境変数』

変更の保存回数の削減

通常、RPC は、クライアント側のキャッシュを使用しない (つまり、SFS の強制操作 機能が使用可能になっている) SFS ファイルに対して 書き込み操作または更新操作を行うたびに実行されます。 入出力操作の結果として発生するファイルの変更はすべて、制御がアプリケーションに返される前にディスクにコミットされます。

この動作は、ファイルが閉じられるまで SFS ファイル上の入出力操作の結果がディスクにコミットされないように変更することができます (つまり、遅延書き込み 方針を使用します)。 SFS ファイルに対する変更が、毎回即時に保存されるわけでなければ、アプリケーションを高速で実行することができます。

アプリケーション内のすべての SFS ファイルに対するデフォルトのコミット動作を変更するには、アプリケーションを実行する前に、CICS_VSAM_AUTO_FLUSH 環境変数を OFF に設定します。

```
export CICS_VSAM_AUTO_FLUSH=OFF
```

特定のファイルのそれぞれにフラッシュ値を設定するには、フラッシュ値を変更したい各ファイルの OPEN ステートメントの前に、CICS_VSAM_AUTO_FLUSH を設定する putenv() 呼び出しをコーディングします。プログラムの中では、putenv() 呼び出しで行う環境変数の設定は、 `export` コマンドで行う環境変数の設定に優先します。

224 ページの『例: 環境変数の設定とアクセス』

SFS ファイルに対してクライアント側のキャッシュが有効な場合 (つまり、環境変数 CICS_VSAM_CACHE がゼロ以外の有効な値に設定されている場合)、そのファイルの CICS_VSAM_AUTO_FLUSH は無視されます。そのファイルに対して強制操作は使用できません。

関連タスク
215 ページの『環境変数の設定』

関連参照

120 ページの『SFS ファイル・システム』
220 ページの『ランタイム環境変数』

第 8 章 ファイルのソートおよびマージ

SORT または MERGE ステートメントを使用すると、レコードを特定のシーケンスで並べることができます。同じ COBOL プログラムの中に SORT ステートメントと MERGE ステートメントを混在させることができます。

SORT ステートメント
(ファイルまたは内部プロシージャーから) 順序付けられていない入力を受け入れ、要求されたシーケンスで出力を (ファイルまたは内部プロシージャーに) 作成します。 ソートの前に、レコードを追加、削除、または変更することができます。

MERGE ステートメント
2 つ以上の順序付けられたファイルからのレコードを比較し、それらを順序正しく結合します。 マージの前に、レコードを追加、削除、または変更することができます。

プログラムにいくつのソート操作およびマージ操作を含めても構いません。 また、同じ操作を何度も実行しても構いませんし、異なる操作を実行しても構いません。 ただし、1 つの操作が終了してからでなければ、別の操作を開始することはできません。

ソートまたはマージで行う手順は一般的に次のようになります。

1. ソートまたはマージに使用するソート・ファイルまたはマージ・ファイルを記述する。
2. ソートまたはマージする入力を記述する。レコードをソート前に処理したい場合には、入力プロシージャーをコーディングしてください。
3. ソートまたはマージからの出力を記述する。レコードをソートまたはマージした後に処理したい場合には、出力プロシージャーをコーディングしてください。
4. ソートまたはマージを要求する。
5. ソートまたはマージ操作が成功したかどうかを判別する。

関連概念
153 ページの『ソートおよびマージ・プロセス』

関連タスク
154 ページの『ソートまたはマージ・ファイルの記述』
154 ページの『ソートまたはマージへの入力の記述』
156 ページの『ソートまたはマージからの出力の記述』
158 ページの『ソートまたはマージの要求』
161 ページの『ソートまたはマージの成否の判断』
164 ページの『ソートまたはマージ操作の途中停止』

関連参照
SORT ステートメント (*COBOL for Linux on x86* 言語解説書)
MERGE ステートメント (*COBOL for Linux on x86* 言語解説書)

ソートおよびマージ・プロセス

ファイルのソート時に、ファイル内のレコードはすべて、それぞれのレコード内の 1 つ以上のフィールドの内容 (キー) に従って順序付けられます。 レコードは、各キーの昇順または降順にソートすることができます。

複数のキーがある場合は、レコードはまず最初の (基本) キーの内容に従ってソートされ、次に 2 番目のキーの内容に従ってソートされる、というようになります。

ファイルをソートするには、COBOL SORT ステートメントを使用します。

複数のファイルのマージ時には (これらのファイルはソート済みでなければなりません)、レコードは、各レコード内の 1 つ以上のキーの内容に従って結合され、順序付けされます。 レコードは、各キーの昇順ま

© Copyright IBM Corp. 2021

たは降順に順序付けすることができます。 ソートの場合と同様、レコードはまず最初の (基本) キーの内容に従って順序付けされ、次に 2 番目のキーの内容に従って順序付けされる、というようになります。

MERGE . . . USING を使用して、順序付けられた 1 つのファイルとして結合したい複数のファイルの名前を指定します。マージ操作では、入力ファイルのレコード内のキーを比較し、順序付けられたレコードを 1 つずつ、出力プロシージャーの RETURN ステートメントに、または GIVING 句で指定されたファイルに渡します。

関連タスク
159 ページの『ソートまたはマージ基準の設定』

関連参照
SORT ステートメント (*COBOL for Linux on x86* 言語解説書)
MERGE ステートメント (*COBOL for Linux on x86* 言語解説書)

ソートまたはマージ・ファイルの記述

ソートまたはマージに使用するソート・ファイルを記述してください。 WORKING-STORAGE または LOCAL-STORAGE からのデータ項目のみをソートまたはマージする場合でも、SELECT 節および SD 項目が必要です。

次のようにコーディングします。

1. ENVIRONMENT DIVISION の FILE-CONTROL 段落に 1 つ以上の SELECT 節をコーディングし、ソート・ファイルの名前を指定します。 例えば、次のように指定します。

```
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT Sort-Work-1 ASSIGN TO SortFile.
```

   Sort-Work-1 は、プログラム内のファイルの名前です。 この名前を使用してファイルを参照してください。
2. そのソート・ファイルを、DATA DIVISION の FILE SECTION の SD 記入項目で記述します。 それぞれの SD 記入項目がレコード記述を含んでいなければなりません。 次に例を示します。

```
DATA DIVISION.
FILE SECTION.
SD  Sort-Work-1
    RECORD CONTAINS 100 CHARACTERS.
01  SORT-WORK-1-AREA.
    05  SORT-KEY-1   PIC  X(10).
    05  SORT-KEY-2   PIC  X(10).
    05  FILLER       PIC  X(80).
```

SD 記入項目で記述するファイルは、ソートまたはマージ操作に使用される作業ファイルです。 このファイルに入力または出力操作を実行することはできません。

関連参照
10 ページの『FILE SECTION 記入項目』

ソートまたはマージへの入力の記述

ソートまたはマージ用の入力ファイルは、以下の手順に従って記述してください。

1. ENVIRONMENT DIVISION の FILE-CONTROL 段落に 1 つ以上の SELECT 節をコーディングし、入力ファイルの名前を指定します。 例えば、次のように指定します。

```
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
```

```
FILE-CONTROL.
    SELECT Input-File ASSIGN TO InFile.
```

Input-File は、プログラム内のファイルの名前です。 この名前を使用してファイルを参照してください。

2. その入力ファイル (マージの場合は複数のファイル) を、DATA DIVISION の FILE SECTION の FD 記入項目で記述します。 例えば、次のように指定します。

```
DATA DIVISION.
FILE SECTION.
FD  Input-File
    RECORD CONTAINS 100 CHARACTERS.
01  Input-Record   PIC X(100).
```

関連タスク
156 ページの『入力プロシージャーのコーディング』
158 ページの『ソートまたはマージの要求』

関連参照

10 ページの『FILE SECTION 記入項目』

例: SORT 用のソート・ファイルおよび入力ファイルの記述

次の例は、ソート作業ファイルおよび入力ファイルを記述するのに必要な ENVIRONMENT DIVISION および DATA DIVISION の記入項目を示しています。

```
 ID Division.
 Program-ID. SmplSort.
 Environment Division.
 Input-Output Section.
 File-Control.
*
* Assign name for a working file is treated as documentation.
*
     Select Sort-Work-1 Assign To SortFile.
     Select Sort-Work-2 Assign To SortFile.
     Select Input-File  Assign To InFile.
 . . .
 Data Division.
 File Section.
 SD  Sort-Work-1
     Record Contains 100 Characters.
 01  Sort-Work-1-Area.
     05  Sort-Key-1    Pic  X(10).
     05  Sort-Key-2    Pic  X(10).
     05  Filler        Pic  X(80).
 SD  Sort-Work-2
     Record Contains 30 Characters.
 01  Sort-Work-2-Area.
     05  Sort-Key      Pic  X(5).
     05  Filler        Pic  X(25).
 FD  Input-File
     Record Contains 100 Characters.
 01  Input-Record      Pic  X(100).
 . . .
 Working-Storage Section.
 01  EOS-Sw            Pic  X.
 01  Filler.
     05  Table-Entry Occurs 100 Times
             Indexed By X1    Pic X(30).
     . . .
```

関連タスク
158 ページの『ソートまたはマージの要求』

入力プロシージャーのコーディング

入力ファイルのレコードを、それらがソート・プログラムに解放される前に処理する場合には、SORT ステートメントの INPUT PROCEDURE 句を使用してください。

入力プロシージャーを使用して、以下のことを行うことができます。

- データ項目を WORKING-STORAGE または LOCAL-STORAGE からソート・ファイルに解放する。
- プログラム内の別な場所で既に読み取られているレコードを解放する。
- 入力レコードからレコードを読み取り、それらを選択または処理し、それらをソート・ファイルに解放する

それぞれの入力プロシージャーは、段落またはセクションのいずれかで構成されなければなりません。 例えば、WORKING-STORAGE または LOCAL-STORAGE の表からのレコードをソート・ファイル SORT-WORK-2 に解放するには、次のようにコーディングすることができます。

```
    SORT SORT-WORK-2
      ON ASCENDING KEY SORT-KEY
      INPUT PROCEDURE 600-SORT3-INPUT-PROC
    . . .
600-SORT3-INPUT-PROC SECTION.
    PERFORM WITH TEST AFTER
      VARYING X1 FROM 1 BY 1 UNTIL X1 = 100
      RELEASE SORT-WORK-2-AREA FROM TABLE-ENTRY (X1)
    END-PERFORM.
```

レコードをソート・プログラムに転送するためには、すべての入力プロシージャーに少なくとも 1 つの RELEASE または RELEASE FROM ステートメントが含まれていなければなりません。 例えば、X から A を解放するには、次のようにコーディングできます。

```
MOVE X TO A.
RELEASE A.
```

あるいは、次のようにコーディングできます。

```
RELEASE A FROM X.
```

次の表では、RELEASE ステートメントと RELEASE FROM ステートメントを比較しています。

RELEASE	RELEASE FROM

MOVE EXT-RECORD
  TO SORT-EXT-RECORD
PERFORM RELEASE-SORT-RECORD
. . .
RELEASE-SORT-RECORD.
  RELEASE SORT-RECORD
	PERFORM RELEASE-SORT-RECORD
	. . .
	RELEASE-SORT-RECORD.
	  RELEASE SORT-RECORD
	  FROM SORT-EXT-RECORD

関連参照
157 ページの『入出力プロシージャーに関する制約事項』
RELEASE ステートメント (*COBOL for Linux on x86* 言語解説書)

ソートまたはマージからの出力の記述

ソートまたはマージからの出力がファイルである場合は、以下の手順に従ってファイルを記述しなければなりません。

1. ENVIRONMENT DIVISION の FILE-CONTROL 段落に SELECT 節をコーディングし、出力ファイルの名前を指定します。 例えば、次のように指定します。

```
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT Output-File ASSIGN TO OutFile.
```

Output-File は、プログラム内のファイルの名前です。 この名前を使用してファイルを参照してください。

2. その出力ファイル (マージの場合は複数のファイル) を、DATA DIVISION の FILE SECTION の FD 記入項目で記述します。 例えば、次のように指定します。

```
DATA DIVISION.
FILE SECTION.
FD  Output-File
    RECORD CONTAINS 100 CHARACTERS.
01  Output-Record   PIC X(100).
```

関連タスク
157 ページの『出力プロシージャーのコーディング』
158 ページの『ソートまたはマージの要求』

関連参照

10 ページの『FILE SECTION 記入項目』

出力プロシージャーのコーディング

ソート済みのレコードをソート作業ファイルから別のファイルに書きこむ前に、それらの選択、編集、またはそれ以外の変更を行うには、SORT ステートメントの OUTPUT PROCEDURE 句を使用してください。

それぞれの出力プロシージャーは、セクションまたは段落のいずれかで構成されなければなりません。 また、出力プロシージャーには、以下の両方を含めなければなりません。

- 少なくとも 1 つ以上の RETURN ステートメントまたは INTO 句を含んだ 1 つの RETURN ステートメント
- レコードの処理に必要なステートメント。レコードは、RETURN ステートメントによって一度に 1 つずつ使用可能になります。

RETURN ステートメントによって、ソート済みの各レコードが出力プロシージャーから使用可能になります。 (ソート・ファイルに対する RETURN ステートメントは、入力ファイルに対する READ ステートメントに似ています。)

RETURN ステートメントとともに AT END および END-RETURN 句を使用することができます。 AT END 句の命令ステートメントは、ソート・ファイルからすべてのレコードが戻された後で実行されます。 END-RETURN 明示範囲終了符号は、RETURN ステートメントの有効範囲を区切る役割をします。

RETURN ではなく RETURN INTO を使用すると、レコードは WORKING-STORAGE、LOCAL-STORAGE、または出力域に戻されます。

関連参照
157 ページの『入出力プロシージャーに関する制約事項』
RETURN ステートメント (*COBOL for Linux on x86* 言語解説書)

入出力プロシージャーに関する制約事項

SORT によって呼び出されるそれぞれの入力または出力プロシージャー、および MERGE によって呼び出される各出力プロシージャーには、いくつかの制約事項が適用されます。

以下の制約事項に従ってください。

- プロシージャーに SORT または MERGE ステートメントを含めてはなりません。

- プロシージャーの中で ALTER、GO TO、および PERFORM ステートメントを使用することによって、入力または出力プロシージャーの外側にあるプロシージャー名を参照することができます。 しかし、GO TO または PERFORM ステートメントの後で、その入力または出力プロシージャーに制御権を戻さなければなりません。
- PROCEDURE DIVISION のその他の部分に、入力または出力プロシージャーの内部への制御権の移動を記述してはなりません (ただし、宣言セクションからの制御権の戻りは例外です)。
- 入力または出力プロシージャーの中から、プログラムを呼び出すことができます。 ただし、呼び出されるプログラムから SORT または MERGE ステートメントを出すことはできません。呼び出されるプログラムは呼び出し元に戻る必要があります。
- SORT または MERGE 操作時には、SD データ項目が使用されます。 出力プロシージャーの中で、最初の RETURN が実行される前に、このデータ項目を使用してはなりません。 最初の RETURN ステートメントの前に、データをこのレコード域に移動すると、戻される最初のレコードが上書きされます。

関連タスク
156 ページの『入力プロシージャーのコーディング』
157 ページの『出力プロシージャーのコーディング』

ソートまたはマージの要求

事前処理を行わずに 1 つの入力ファイル (MERGE の場合は複数のファイル) からレコードを読み取るには、SORT . . . USING または MERGE . . . USING と、SELECT 節で宣言した入力ファイルの名前を使用します。

ソート済みまたはマージ済みレコードを、これ以上処理せずに、ソート・プログラムまたはマージ・プログラムから別のファイルへ転送するには、 SORT . . . GIVING または MERGE . . . GIVING、 および SELECT 節で宣言された出力ファイルの名前を使用してください。次に例を示します。

```
SORT Sort-Work-1
    ON ASCENDING KEY Sort-Key-1
    USING Input-File
    GIVING Output-File.
```

SORT . . . USING または MERGE . . . USING の場合、コンパイラーは入力プロシージャーを生成します。このプロシージャーは、ファイルを開き、レコードを読み取り、レコードをソート/マージ・プログラムに解放し、ファイルを閉じます。SORT または MERGE ステートメントが実行を開始するとき、ファイルが開いた状態にしないでください。 SORT . . . GIVING または MERGE . . . GIVING の場合、コンパイラーは出力プロシージャーを生成します。このプロシージャーは、ファイルを開き、レコードを返し、レコードを書き込み、ファイルを閉じます。SORT または MERGE ステートメントが実行を開始するとき、ファイルが開いた状態にしないでください。

155 ページの『例: SORT 用のソート・ファイルおよび入力ファイルの記述』

ソート・レコードがソートされる前にソート・レコードに対して入力プロシージャーが実行されるようにする場合は、SORT . . . INPUT PROCEDURE を使用します。ソート済みレコードに対して出力プロシージャーが実行されるようにする場合は、SORT . . . OUTPUT PROCEDURE を使用します。次に例を示します。

```
SORT Sort-Work-1
    ON ASCENDING KEY Sort-Key-1
    INPUT PROCEDURE EditInputRecords
    OUTPUT PROCEDURE FormatData.
```

160 ページの『例: 入出力プロシージャーを使用したソート』

制約事項: MERGE ステートメントで入力プロシージャーを使用することはできません。マージ操作への入力ソースは、既にソート済みのファイルの集合でなければなりません。しかし、マージ済みレコードに対

して出力プロシージャーが実行されるようにする場合は、MERGE . . . OUTPUT PROCEDURE を使用します。次に例を示します。

```
MERGE Merge-Work
    ON ASCENDING KEY Merge-Key
    USING Input-File-1 Input-File-2 Input-File-3
    OUTPUT PROCEDURE ProcessOutput.
```

FILE SECTION では、SD 記入項目の *Merge-Work*、および FD 記入項目の入力ファイルを定義する必要があります。

関連参照
SORT ステートメント (*COBOL for Linux on x86* 言語解説書)
MERGE ステートメント (*COBOL for Linux on x86* 言語解説書)

ソートまたはマージ基準の設定

ソートまたはマージ基準を設定するには、操作の実行対象のキーを定義します。

以下の手順を 実行します。

1. ソートまたはマージするファイルのレコード記述の中で、キー (複数の場合もある) を定義します。

   制約事項: キーは可変位置にすることはできません。
2. SORT または MERGE ステートメントの中で、ASCENDING 句または DESCENDING KEY 句 (あるいはその両方) をコーディングすることにより、 順序付けに使用するキー・フィールドを指定してください。 複数のキーをコーディングする場合、一部を昇順にし、残りを降順にすることができます。

   キーの名前は重要度の高い順に指定します。 左端のキーが基本キーです。 次のキーが 2 次キー、というようになります。

SORT および MERGE キーは、クラス英字、英数字、国別 (コンパイラー・オプション NCOLLSEQ(BIN) が有効化されている場合)、数値 (ただし、USAGE NATIONAL の数値ではない) にすることができます。 USAGE NATIONAL を持っている場合、キーはカテゴリー国別にするか、あるいは国別編集または数字編集データ項目にすることができます。 キーを、国別 10 進数データ項目にしたり、国別浮動小数点データ項目にすることはできません。

国別キーの照合順序は、キーのバイナリーの順序によって決まります。 キーとして国別データ項目を指定する場合は、SORT または MERGE ステートメントの COLLATING SEQUENCE 句はいずれもそのキーに適用されません。

同じ COBOL プログラムの中に SORT ステートメントと MERGE ステートメントを混在させることができます。 プログラムはいくつのソート操作およびマージ操作でも実行することができます。 ただし、1 つの操作が終了してからでなければ、次の操作を開始することはできません。

関連タスク
207 ページの『ロケール付きの照合シーケンスの制御』

関連参照
274 ページの『NCOLLSEQ』
SORT ステートメント (*COBOL for Linux on x86* 言語解説書)
MERGE ステートメント (*COBOL for Linux on x86* 言語解説書)

代替照合シーケンスの選択

レコードのソートまたはマージは、1 バイト文字キーで指定した照合シーケンスで行うことができます。 OBJECT-COMPUTER 段落で PROGRAM COLLATING SEQUENCE 節をコーディングしない限り、デフォルトの照合シーケンスは、 コンパイル時に有効化されているロケール設定で指定された照合シーケンスです。

デフォルト・シーケンスをオーバーライドするには、SORT または MERGE ステートメントの COLLATING SEQUENCE 句を使用してください。 プログラムの SORT または MERGE ステートメントごとに異なる照合シーケンスを使用することができます。

PROGRAM COLLATING SEQUENCE 節および COLLATING SEQUENCE 句は、クラス英字または英数字のキーにのみ適用されます。 COLLATING SEQUENCE 句は、1 バイト ASCII コード・ページが有効化されている場合にのみ有効です。

関連タスク
6 ページの『照合シーケンスの指定』
207 ページの『ロケール付きの照合シーケンスの制御』
159 ページの『ソートまたはマージ基準の設定』

関連参照
OBJECT-COMPUTER 段落 (*COBOL for Linux on x86* 言語解説書)
SORT ステートメント (*COBOL for Linux on x86* 言語解説書)
データのクラスおよびカテゴリー (*COBOL for Linux on x86* 言語解説書)

例: 入出力プロシージャーを使用したソート

以下は、SORT ステートメントにおける入出力プロシージャーの使用例です。この例では、基本キー (SORT-GRID-LOCATION) と 2 次キー (SORT-SHIFT) を SORT ステートメントで使用する前に、それらのキーをどのように定義できるかも示します。

```
DATA DIVISION.
. . .
SD  SORT-FILE
    RECORD CONTAINS 115 CHARACTERS
    DATA RECORD SORT-RECORD.
01  SORT-RECORD.
    05  SORT-KEY.
        10  SORT-SHIFT              PIC X(1).
        10  SORT-GRID-LOCATION      PIC X(2).
        10  SORT-REPORT             PIC X(3).
    05  SORT-EXT-RECORD.
        10  SORT-EXT-EMPLOYEE-NUM   PIC X(6).
        10  SORT-EXT-NAME           PIC X(30).
        10  FILLER                  PIC X(73).
. . .
WORKING-STORAGE SECTION.
01  TAB1.
    05 TAB-ENTRY OCCURS 10 TIMES
           INDEXED BY TAB-INDX.
        10  WS-SHIFT                PIC X(1).
        10  WS-GRID-LOCATION        PIC X(2).
        10  WS-REPORT               PIC X(3).
        10  WS-EXT-EMPLOYEE-NUM     PIC X(6).
        10  WS-EXT-NAME             PIC X(30).
        10  FILLER                  PIC X(73).
. . .
PROCEDURE DIVISION.
    . . .
    SORT SORT-FILE
        ON ASCENDING KEY SORT-GRID-LOCATION SORT-SHIFT
        INPUT PROCEDURE 600-SORT3-INPUT
        OUTPUT PROCEDURE 700-SORT3-OUTPUT.
    . . .
600-SORT3-INPUT.
    PERFORM VARYING TAB-INDX FROM 1 BY 1 UNTIL TAB-INDX > 10
        RELEASE SORT-RECORD FROM TAB-ENTRY(TAB-INDX)
    END-PERFORM.
. . .
700-SORT3-OUTPUT.
    PERFORM VARYING TAB-INDX FROM 1 BY 1 UNTIL TAB-INDX > 10
        RETURN SORT-FILE INTO TAB-ENTRY(TAB-INDX)
            AT END DISPLAY 'Out Of Records In SORT File'
        END-RETURN
    END-PERFORM.
```

関連タスク
158 ページの『ソートまたはマージの要求』

ソートまたはマージの成否の判断

SORT または MERGE ステートメントは、各ソートまたはマージ操作の終了後に、0 (正常終了) または 16 (異常終了) のいずれかの完了コードを返します。 完了コードは、SORT-RETURN 特殊レジスターに保管されます。

それぞれの SORT または MERGE ステートメントの後で、正常終了かどうかをテストしなければなりません。次に例を示します。

```
    SORT SORT-WORK-2
        ON ASCENDING KEY SORT-KEY
        INPUT PROCEDURE IS 600-SORT3-INPUT-PROC
        OUTPUT PROCEDURE IS 700-SORT3-OUTPUT-PROC.
    IF SORT-RETURN NOT=0
        DISPLAY "SORT ENDED ABNORMALLY. SORT-RETURN = " SORT-RETURN.
    . . .
600-SORT3-INPUT-PROC SECTION.
    . . .
700-SORT3-OUTPUT-PROC SECTION.
    . . .
```

プログラムの中で SORT-RETURN をまったく参照しないと、COBOL ランタイムで完了コードがテストされます。 16 の場合、COBOL は、ランタイム診断メッセージを出し、実行単位 (マルチスレッド化環境では、スレッド) を終了します。 診断メッセージには、ソートまたはマージのエラー番号が入っています。これは、問題の原因の決定に役立たせることができます。

SORT または MERGE ステートメントの 1 つ以上 (しかし、必ずしも全部ではない) について SORT-RETURN をテストすると、 COBOL ランタイムで完了コードが検査されません。 ただし、SORT または MERGE ステートメントの後に、iwzGetSortErrno サービスを呼び出すことによって、例えば以下のように、ソートまたはマージのエラー番号を取得することができます。

```
77  sortErrno    PIC 9(9)   COMP-5.
. . .
    CALL 'iwzGetSortErrno'  USING  sortErrno
. . .
```

エラー番号とその意味のリストについては、以下に示す関連参照を参照してください。

関連参照

161 ページの『ソートおよびマージ・エラー番号』

ソートおよびマージ・エラー番号

プログラム内で SORT-RETURN を参照しない場合で、 ソースまたはマージ操作からの完了コードが 16 である場合、COBOL for Linux はランタイム診断メッセージを発行します。 これには、以下の表に示すゼロ以外のエラー番号のいずれかが含まれています。

表 *15.* **ソートおよびマージ・エラー番号**

エラー番号	説明
0	エラーなし
1	レコードの順序が間違っています
2	同等鍵付きレコードが検出されました
3	複数の main 関数が指定されました (内部エラー)

表 *15.* **ソートおよびマージ・エラー番号** *(続き)*	
エラー番号	**説明**
4	パラメーター・ファイルにエラーがあります
5	パラメーター・ファイルをオープンできませんでした
6	オペランドがオプションから欠落しています
7	オペランドが拡張オプションから欠落しています
8	オプション内のオペランドが無効です
9	拡張オプション内のオペランドが無効です
10	無効なオプションが指定されました
11	無効な拡張オプションが指定されました
12	無効な一時ディレクトリーが指定されました
13	無効なファイル名が指定されました
14	無効なフィールドが指定されました
15	フィールドがレコード内にありません
16	フィールドがレコード内で短すぎます
17	SELECT の指定に構文エラーがあります
18	無効な定数が SELECT で指定されました
19	SELECT 内の定数とデータ型の間の比較が無効です
20	SELECT 内の 2 つのデータ型の間の比較が無効です
21	形式の指定に構文エラーがあります
22	再フォーマットの指定に構文エラーがあります
23	無効な定数が再フォーマットの指定で指定されました
24	合計の指定に構文エラーがあります
25	フラグが複数回指定されました
26	指定された出力が多すぎます
27	入力ソースが指定されませんでした
28	出力宛先が指定されませんでした
29	無効な修飾子が指定されました
30	合計は許可されません
31	レコードが短すぎます
32	レコードが長すぎます
33	無効なパックまたはゾーン・フィールドが検出されました
34	ファイルに読み取りエラーがあります
35	ファイルに書き込みエラーがあります
36	入力ファイルをオープンできません
37	メッセージ・ファイルをオープンできません

表 *15.* **ソートおよびマージ・エラー番号** *(続き)*	
エラー番号	**説明**
38	SdU または SFS ファイル・エラー
39	ターゲット・バッファーのスペースが不十分です
40	一時ディスク・スペースが足りません
41	出力ファイルのスペースが足りません
42	予期しないシグナルがトラップされました
43	エラーが入力出口から戻されました
44	エラーが出力出口から戻されました
45	予期しないデータが出力ユーザー出口から戻されました
46	無効なバイトを使用した値が入力出口から戻されました
47	無効なバイトを使用した値が出力出口から戻されました
48	SMARTsort がアクティブではありません
49	実行を継続するためのストレージが不十分です
50	パラメーター・ファイルが大きすぎます
51	単一引用符が一致しません
52	引用符が一致しません
53	競合オプションが指定されました
54	レコード内の長さフィールドが無効です
55	レコード内の最終フィールドが無効です
56	必要なレコード形式が指定されませんでした
57	出力ファイルをオープンできません
58	一時ファイルをオープンできません
59	ファイル編成が無効です
60	指定されたファイル編成のユーザー出口がサポートされていません
61	ロケールがシステムに認識されていません
62	レコードに無効なマルチバイト文字があります
63	ファイルは SdU でも SFS でもありませんでした
64	S0RT に指定されたキーが、索引付き出力ファイルの定義に使用できません
65	SdU または SFS ファイルのレコード長が正しくありませんでした
66	SMARTsort オプションのファイル作成が失敗しました
67	完全修飾された、非相対パス名を作業ディレクトリーとして指定する必要があります
68	必須オプションを指定する必要があります
69	パス名が無効です
79	一時ファイルの最大数に達しました
501	関数が無効です

表 15. **ソートおよびマージ・エラー番号** (続き)	
エラー番号	**説明**
502	レコード・タイプが無効です
503	レコード長が無効です
504	タイプ長エラー
505	タイプが無効です
506	キー数が一致しません
507	タイプが長すぎます
508	キー・オフセットが無効です
509	昇順または降順キーが無効です
510	オーバーラップ・キーが無効です
511	キーが定義されませんでした。
512	入力ファイルが指定されませんでした
513	出力ファイルが指定されませんでした
514	入力ファイルのタイプが混在しています
515	出力ファイルのタイプが混在しています
516	入力作業バッファーが無効です
517	出力作業バッファーが無効です
518	COBOL 入力の入出力エラー
519	COBOL 出力の入出力エラー
520	関数がサポートされていません
521	無効なキー
522	無効な USING ファイル
523	無効な GIVING ファイル
524	作業ディレクトリーが提供されませんでした
525	作業ディレクトリーが存在しません
526	ソート共通が割り振られませんでした
527	ソート共通用のストレージがありません
528	バイナリー・バッファーが割り振られませんでした
529	行順次ファイル・バッファーが割り振られませんでした
530	ワークスペースの割り振りが失敗しました
531	FCB の割り振りが失敗しました

ソートまたはマージ操作の途中停止

ソートまたはマージ操作を停止するには、整数 16 を SORT-RETURN 特殊レジスターに移動させます。

次のいずれかの方法で、レジスターに 16 を移動してください。

- 入力または出力プロシージャーの中で MOVE を使用する。

  ソートまたはマージ処理は、次の RELEASE または RETURN ステートメントが実行された直後に停止されます。
- USING または GIVING ファイルの処理中に入る宣言セクションの中でこのレジスターをリセットする。

  ソートまたはマージ処理は、宣言セクションから出たときに停止されます。

制御は、その後、SORT または MERGE ステートメントの次のステートメントに戻ります。

第 9 章 エラーの処理

起こり得るシステムまたは実行時の問題を予期するコードをプログラムに入れてください。 そのようなコードを含めない場合、出力データまたはファイルが破損しても、ユーザーは問題が発生していることに気付くことさえない可能性があります。

エラー処理コードでは、状態の処理、メッセージの発行、プログラムの停止などのアクションを取ることができます。 例えば、データ入力エラーまたはご使用のシステムで定義されたエラーに対して、独自のエラー検出ルーチンを作成することができます。 どのようなイベントであっても、警告メッセージをコーディングするのはよいことです。

COBOL for Linux には、エラー状態を予測して訂正するのに役に立つ特殊なエレメントがいくつか含まれています。

- STRING および UNSTRING 操作の ON OVERFLOW
- 算術演算の ON SIZE ERROR
- 入出力エラーを処理するためのエレメント
- CALL ステートメントの ON EXCEPTION または ON OVERFLOW

関連タスク
167 ページの『ストリングの結合および分割におけるエラーの処理』
168 ページの『算術演算でのエラーの処理』
168 ページの『入出力操作でのエラーの処理』
175 ページの『プログラム呼び出し時のエラーの処理』

ストリングの結合および分割におけるエラーの処理

ストリングの結合または分割中に、STRING または UNSTRING に使用されるポインターは、受信フィールドの範囲外になる可能性があります。 オーバーフロー条件が存在する可能性はありますが、COBOL ではオーバーフローの発生を許可しません。

その代わりに、STRING 操作や UNSTRING 操作は完了せず、受信フィールドは未変更のままとなり、制御は次の順次ステートメントに移動します。 STRING または UNSTRING ステートメントの ON OVERFLOW 句をコーディングしないと、操作未完了の通知が出されません。

次のステートメントを考えてください。

```
String Item-1 space Item-2 delimited by Item-3
    into Item-4
    with pointer String-ptr
    on overflow
        Display "A string overflow occurred"
End-String
```

以下に、ステートメントの実行前と実行後のデータ値を示します。

データ項目	PICTURE	実行前の値	実行後の値
Item-1	X(5)	AAAAA	AAAAA
Item-2	X(5)	EEEAA	EEEAA
Item-3	X(2)	EA	EA
Item-4	X(8)	*bbbbbbbb*[1]	*bbbbbbbb*[1]
String-ptr	9(2)	0	0
1. 記号 *b* は、ブランク・スペースを表します。			

© Copyright IBM Corp. 2021

String-ptr の値は (0) で、受信フィールドには達しないため、オーバーフロー条件が発生し、STRING 操作は完了しません (String-ptr が 9 より大きい場合にも、オーバーフローが起こります)。 ON OVERFLOW が指定されていなかった場合は、Item-4 の内容が未変更のままであったことについて通知されません。

算術演算でのエラーの処理

算術演算の結果が、それらを入れる固定小数点フィールドより大きかったり、0 除算が試みられたりすることがあります。 いずれの場合も、ADD、SUBTRACT、MULTIPLY、DIVIDE、または COMPUTE ステートメントの後の ON SIZE ERROR 節でその状況を処理することができます。

ON SIZE ERROR が固定小数点オーバーフローおよび 10 進オーバーフローで正常に動作するようにするためには、TRAP(ON) ランタイム・オプションを指定する必要があります。

以下の場合は、ON SIZE ERROR 節の命令ステートメントが実行され、結果フィールドは変更されません。

- 固定小数点オーバーフロー
- 0 による除算
- 0 の 0 乗
- 0 の負数乗
- 負数の分数乗

例: 0 による除算の検査

次の例は、ゼロ除算が発生した場合にプログラムが通知メッセージを出すように ON SIZE ERROR 命令ステートメントをコーディングする方法を示しています。

```
DIVIDE-TOTAL-COST.
    DIVIDE TOTAL-COST BY NUMBER-PURCHASED
        GIVING ANSWER
        ON SIZE ERROR
          DISPLAY "ERROR IN DIVIDE-TOTAL-COST PARAGRAPH"
          DISPLAY "SPENT " TOTAL-COST, " FOR " NUMBER-PURCHASED
          PERFORM FINISH
    END-DIVIDE
    . . .
    FINISH.
    STOP RUN.
```

ゼロ除算が発生すると、プログラムはメッセージを作成し、プログラム実行を停止します。

入出力操作でのエラーの処理

入力または出力操作が失敗しても、COBOL が自動的に訂正処置をとることはありません。 重大エラーではない入出力エラー後にプログラムの実行を継続するかどうかを選択してください。

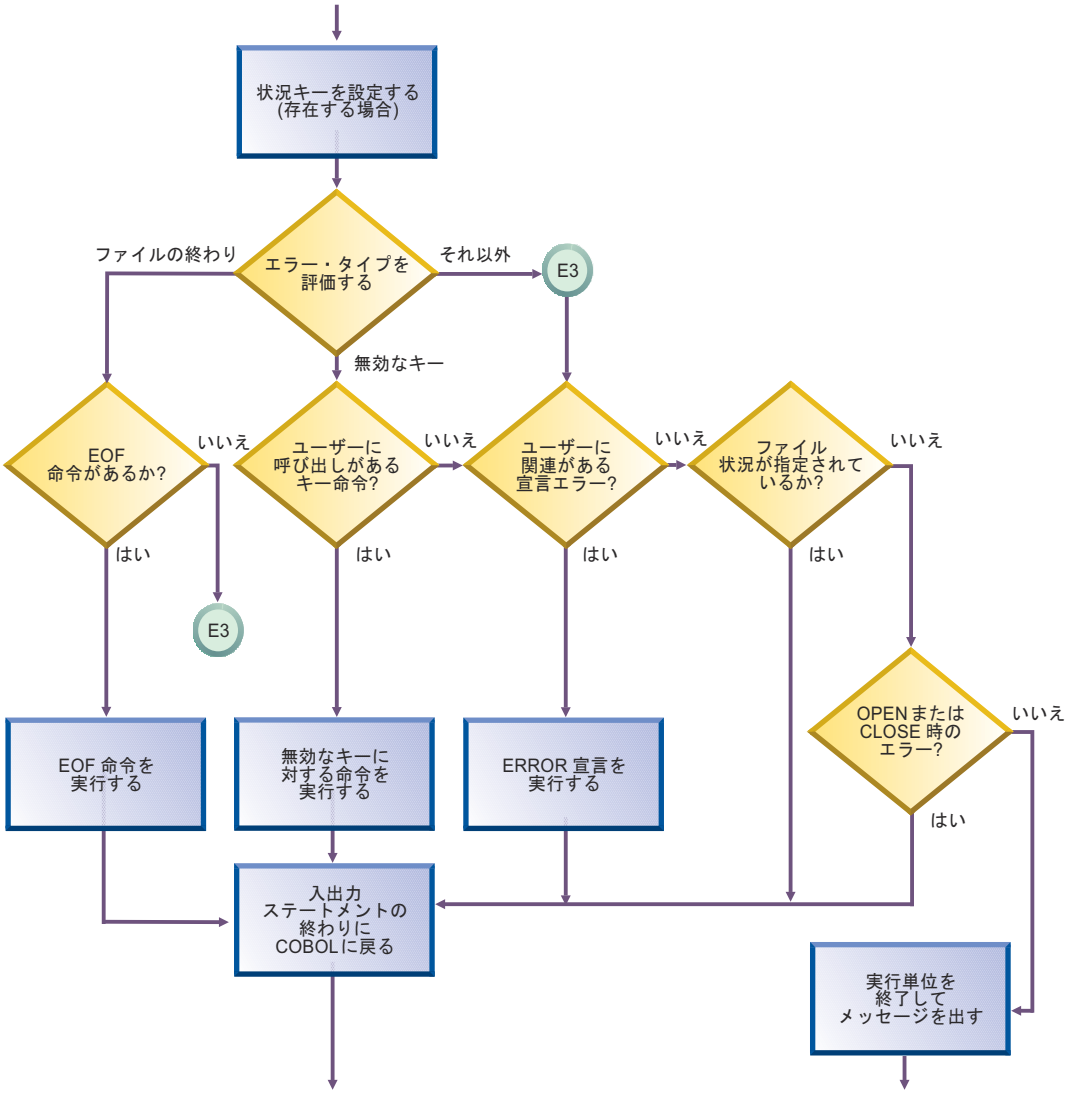
特定の入力または出力の状態またはエラーの代行受信および処理には、以下のいずれかの技法を使用することができます。

- ファイル終わり条件 (AT END)
- ERROR 宣言
- FILE STATUS 節およびファイル状況キー
- ファイル・システム状況コード
- READ または WRITE ステートメント内の命令ステートメント句
- INVALID KEY 句

プログラムを継続させる場合には、適切なエラー・リカバリー手順もコーディングする必要があります。 例えば、ファイル状況キーの値を検査する手順をコーディングすることができます。 入力または出力エラ

ーをこれらのいずれかの方法で処理しなかったときは、 COBOL ランタイム・メッセージが書き込まれ、実行単位が終了します。

次の図は、ファイル・システムの入力または出力エラーの後のロジック・フローを示しています。



関連タスク

132 ページの『オプション・ファイルのオープン』
170 ページの『ファイルの終わり条件 (AT END) の使用』
170 ページの『ERROR 宣言のコーディング』
170 ページの『ファイル状況キーの使用』
172 ページの『ファイル・システム状況コードの使用』
174 ページの『INVALID KEY 句のコーディング』

関連参照

ファイル状況キー (*COBOL for Linux on x86* 言語解説書)

ファイルの終わり条件 (AT END) の使用

READ ステートメントの AT END 句をコーディングすると、エラーまたは正常な条件をプログラムの設計に従って処理することができます。 ファイルの終わりで、AT END 句が実行されます。 AT END 句をコーディングしないと、対応する ERROR 宣言が実行されます。

多くの設計では、ファイルの終わりまでの順次読み取りが意図的に行われており、AT END 条件が予期されます。 例えば、メイン・ファイルを更新するために、トランザクションが入っているファイルを処理していると想定します。

```
PERFORM UNTIL TRANSACTION-EOF = "TRUE"
  READ UPDATE-TRANSACTION-FILE INTO WS-TRANSACTION-RECORD
    AT END
      DISPLAY "END OF TRANSACTION UPDATE FILE REACHED"
      MOVE "TRUE" TO TRANSACTION-EOF
  END READ
  . . .
END-PERFORM
```

NOT AT END 句が実行されるのは、READ ステートメントが正常に完了した場合だけです。 ファイルの終わり以外の何らかの条件のために READ 操作が失敗すると、AT END 句も NOT AT END 句も実行されません。 その代わりに、関連する宣言型プロシージャーを実行した後で、READ ステートメントの終わりに制御が渡されます。

AT END 句または EXCEPTION 宣言型プロシージャーのいずれもコーディングせずに、代わりにファイルの状況キー節をコーディングすることもできます。 その場合は、ファイルの終わり条件を検出した入力ステートメントまたは出力ステートメントの後の次の順次命令に、制御が渡されます。 その場所に、適切な操作を実行するコードを記述します。

関連参照
AT END 句 (*COBOL for Linux on x86* 言語解説書)

ERROR 宣言のコーディング

プログラムの実行時に入力または出力エラーが発生した場合に制御が与えられる ERROR 宣言型プロシージャーを 1 つ以上コーディングすることができます。 そのようなプロシージャーをコーディングしないと、入力または出力エラーの発生後に、ジョブが取り消されるか、異常終了します。

このようなプロシージャーをそれぞれ PROCEDURE DIVISION の宣言セクションに入れます。 以下のものをコーディングすることができます。

- プログラム全体用の単一の共通プロシージャー
- 各ファイル・オープン・モードのプロシージャー (INPUT、OUTPUT、I-O、または EXTEND)
- それぞれのファイルごとの個々のプロシージャー

ERROR 宣言プロシージャーでは、訂正処置の試行、操作の再試行、実行の継続または終了などをコーディングすることができます。 (ただし、ブロック化ファイルの処理を継続する場合、エラーが発生したレコードより後ろにあるブロック内の残りのレコードが失われることがあります。) エラーについてさらに詳しい分析を行う場合は、ERROR 宣言型プロシージャーとファイル状況キーを組み合わせて使用することができます。

関連参照
EXCEPTION/ERROR 宣言 (*COBOL for Linux on x86* 言語解説書)

ファイル状況キーの使用

それぞれの入力または出力ステートメントがファイルに対して実行された後、システムはファイル状況キーの 2 つの桁位置の値を更新します。 一般に、最初の桁がゼロの場合、操作が正常に行われたことを表し、両方の桁がゼロの場合、異常がなかったことを意味します。

ファイル状況キーは、次のようにコーディングして設定してください。

- FILE-CONTROL 段落の FILE STATUS 節:

```
FILE STATUS IS data-name-1
```

- DATA DIVISION (WORKING-STORAGE、LOCAL-STORAGE、または LINKAGE SECTION) のデータ定義 (一例として):

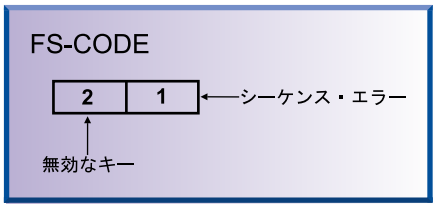
```
WORKING-STORAGE SECTION.
01  data-name-1  PIC 9(2)  USAGE NATIONAL.
```

ファイル状況キー *data-name-1* を、2 文字のカテゴリー英数字またはカテゴリー国別項目として、あるいは 2 桁のゾーン 10 進数または国別 10 進数項目として指定してください。 この *data-name-1* を可変位置にすることはできません。

プログラムはファイル状況キーを検査して、エラーが発生したかどうか、また発生した場合にはどんなタイプのエラーが発生したかを発見できます。 例えば、FILE STATUS 節が

```
FILE STATUS IS FS-CODE
```

FS-CODE は、次のような状況に関する情報を保持するために、COBOL によって使用されます。



各ファイルについて、以下の規則に従ってください。

- ファイルごとに異なるファイル状況キーを定義します。

  すると、アプリケーション論理エラーやディスク・エラーのような、ファイル入力または出力例外の原因を判別することができます。
- それぞれの入力または出力要求の後で、ファイル状況キーを検査します。

  ファイル状況キーが 0 以外の値を含んでいる場合、プログラムはエラー・メッセージを発生するか、またはその値に基づいてアクションを実行できます。

  ファイル状況キー・コードをリセットする必要はありません。ファイル状況キー・コードは各入出力が試みられた後で設定されます。

ファイル状況キーに加えて、2 番目の ID を FILE STATUS 節にコーディングして、ファイル・システムの入力または出力要求に関するさらに詳細な情報を取得できます。 詳細については、ファイル・システム状況コードに関する関連タスクを参照してください。

ファイル状況キーは単独でも、INVALID KEY 句と一緒でも使用でき、EXCEPTION または ERROR 宣言を補足するためにも使用できます。 このようにファイル状況キーを使用すると、それぞれの入力または出力操作の結果に関する正確な情報が得られます。

172 ページの『例: ファイル状況キー』
173 ページの『例: ファイル・システム状況コードの検査』

関連タスク

133 ページの『ファイル状況フィールドの設定』
172 ページの『ファイル・システム状況コードの使用』
174 ページの『INVALID KEY 句のコーディング』
304 ページの『入出力エラーの検出および処理』

関連参照
FILE STATUS 節 (*COBOL for Linux on x86* 言語解説書)
ファイル状況キー (*COBOL for Linux on x86* 言語解説書)

例: ファイル状況キー

次の例は、ファイルを開いた後にファイル状況キーの簡単な検査を行う方法を示しています。

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  SIMCHK.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT MAINFILE ASSIGN TO AS-MAINA
    FILE STATUS IS MAINFILE-CHECK-KEY
    . . .
DATA DIVISION.
. . .
WORKING-STORAGE SECTION.
01  MAINFILE-CHECK-KEY       PIC X(2).
. . .
PROCEDURE DIVISION.
    OPEN INPUT MAINFILE
    IF MAINFILE-CHECK-KEY NOT = "00"
        DISPLAY "Nonzero file status returned from OPEN " MAINFILE-CHECK-KEY
    . . .
```

ファイル・システム状況コードの使用

入力または出力要求の結果を正確に特定する上で、2 桁の ファイル状況キーは一般的過ぎることがよくあります。 FILE STATUS 文節に 2 番目のデータ項目をコーディングすることにより、Db2、LSQ、QSAM、RSD、SdU、SFS、および STL ファイル・システム要求に関する詳細情報を取得することができます。

```
FILE STATUS IS data-name-1 data-name-8
```

上記の例では、データ項目 *data-name-1* は 2 桁の COBOL ファイル状況キーを指定します。 これは、2 文字のカテゴリー英数字またはカテゴリー国別項目、あるいは 2 桁のゾーン 10 進数または国別 10 進数項目でなければなりません。 データ項目 *data-name-8* は、COBOL ファイル状況キーが 0 でない場合に ファイル・システムの状況コードが入れられるデータ項目を指定します。 *data-name-8* の長さは 6 バイト以上で、 英数字項目でなければなりません。

LSQ、QSAM、RSD、SFS、STL、 および SdU ファイル: LSQ、QSAM、RSD、SFS、STL、および SdU ファイル・システム入出力要求に 関しては、*data-name-8* の長さが 6 バイトの場合、 ファイル状況コードが含まれます。 *data-name-8* が 6 バイト長を超える場合は詳細情報付きのメッセージも含まれます。

```
01  my-file-status-2.
    02 exception-return-value PIC 9(6).
    02 additional-info        PIC X(100).
```

exception-return-value には、FILE STATUS で 記されたエラーをさらに詳細化できる値が含まれます。 additional-info には、exception-return-value で記されたエラーの詳細診断情報が含まれます。 この情報は、問題の診断に役立ちます。

Db2 ファイル: Db2 ファイル・システムの入出力要求の場合、*data-name-8* をグループ項目として定義します。次に例を示します。

```
01  FileStatus2.
    02 FS2-SQLCODE  PICTURE S9(9) COMP.
    02 FS2-SQLSTATE PICTURE X(5).
```

FS2-SQLCODE および FS2-SQLSTATE の実行時値は、以前に完了した操作の SQL フィードバック情報を表します。

173 ページの『例: ファイル・システム状況コードの検査』

関連タスク
428 ページの『言語エレメントによる違いの修正』

関連参照
118 ページの『Db2 ファイル・システム』
119 ページの『QSAM ファイル・システム』
120 ページの『SdU ファイル・システム』
120 ページの『SFS ファイル・システム』
121 ページの『STL ファイル・システム』
FILE STATUS 節 (*COBOL for Linux on x86* 言語解説書)
ファイル状況キー (*COBOL for Linux on x86* 言語解説書)

例: ファイル・システム状況コードの検査

次の例は、索引付きファイルを 5 番目のレコードから読み取り、 それぞれの入力または出力要求の後で、ファイル状況キーを検査します。 ファイル状況キーがゼロ以外の場合に、ファイル状況コードが表示されます。

さらに、以下に、処理中のファイルに 6 つのレコードが入っていたことを想定した場合の、このプログラムからの出力を図示しています。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EXAMPLE.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT FILESYSFILE ASSIGN TO FILESYSFILE
    ORGANIZATION IS INDEXED
    ACCESS DYNAMIC
    RECORD KEY IS FILESYSFILE-KEY
    FILE STATUS IS FS-CODE, FILESYS-CODE.
DATA DIVISION.
FILE SECTION.
FD  FILESYSFILE
    RECORD  30.
01  FILESYSFILE-REC.
    10 FILESYSFILE-KEY          PIC X(6).
    10 FILLER                   PIC X(24).
WORKING-STORAGE SECTION.
01  RETURN-STATUS.
    05 FS-CODE                  PIC XX.
    05 FILESYS-CODE             PIC X(6).
PROCEDURE DIVISION.
    OPEN  INPUT FILESYSFILE.
    DISPLAY "OPEN INPUT FILESYSFILE FS-CODE: " FS-CODE.

    IF FS-CODE NOT = "00"
       PERFORM FILESYS-CODE-DISPLAY
       STOP RUN
    END-IF.

    MOVE "000005" TO FILESYSFILE-KEY.
    START FILESYSFILE KEY IS EQUAL TO FILESYSFILE-KEY.
    DISPLAY "START FILESYSFILE KEY="  FILESYSFILE-KEY
            " FS-CODE: "  FS-CODE.

    IF FS-CODE NOT = "00"
       PERFORM FILESYS-CODE-DISPLAY
    END-IF.

    IF FS-CODE = "00"
       PERFORM READ-NEXT UNTIL FS-CODE NOT = "00"
    END-IF.

    CLOSE FILESYSFILE.
    STOP RUN.

READ-NEXT.
    READ FILESYSFILE NEXT.
    DISPLAY "READ NEXT FILESYSFILE FS-CODE: " FS-CODE.
```

```
    IF FS-CODE NOT = "00"
       PERFORM FILESYS-CODE-DISPLAY
    END-IF.
    DISPLAY FILESYSFILE-REC.

FILESYS-CODE-DISPLAY.
    DISPLAY "FILESYS-CODE ==>", FILESYS-CODE.
```

INVALID KEY 句のコーディング

INVALID KEY 句は、索引付きファイルおよび相対ファイルの、READ、START、WRITE、REWRITE、および DELETE ステートメントに組み込むことができます。 INVALID KEY 句に制御が与えられるのは、誤った索引キーのために入力エラーまたは出力エラーが発生した場合です。

状況キーを評価し、特定の INVALID KEY 条件を判別するには、INVALID KEY 句と一緒に FILE STATUS 節を使用してください。

INVALID KEY 句は、いくつかの点で ERROR 宣言とは異なります。 INVALID KEY 句:

- 限られたタイプのエラーのみで作動します。 ERROR 宣言は、すべての形式をカバーします。
- 入出力ステートメントで直接コーディングされます。 ERROR 宣言は、別途にコーディングされます。
- 単一の入出力操作で固有です。 ERROR 宣言はより一般的です。

INVALID KEY 条件を引き起こすステートメントで INVALID KEY をコーディングすると、制御は INVALID KEY 命令ステートメントに転送されます。 コーディングした ERROR 宣言は実行されません。

NOT INVALID KEY 句をコーディングした場合、ステートメントが正常に完了したときにのみ実行されます。 INVALID KEY 以外の条件のために操作が失敗すると、INVALID KEY 句も NOT INVALID KEY 句も実行されません。 代わりに、関連した ERROR 宣言をプログラムが実行した後、制御はステートメントの最後に渡されます。

174 ページの『例: FILE STATUS および INVALID KEY』

例: FILE STATUS および INVALID KEY

次の例は、ファイル状況コードおよび INVALID KEY 句を使用して、入力または出力ステートメントが失敗した理由をもっと明確に判別する方法を示しています。

メイン顧客レコードを含んでいるファイルがあり、トランザクション更新ファイルの情報が反映されるように、それらのレコードの一部を更新する必要があると想定しましょう。プログラムは、各トランザクション・レコードを読み取り、メイン・ファイルの中の対応するレコードを見つけ、必要な更新を行います。どちらのファイルのレコードにもそれぞれ顧客番号用のフィールドがあり、メイン・ファイルの中の各レコードには固有の顧客番号があります。

顧客レコードのメイン・ファイル用の FILE-CONTROL 記入項目には、索引編成を定義するステートメント、ランダム・アクセス、基本レコード・キーとして MAIN-CUSTOMER-NUMBER、ファイル状況キーとして CUSTOMER-FILE-STATUS が含まれています。

```
.
.  (read the update transaction record)
.
MOVE "TRUE" TO TRANSACTION-MATCH
MOVE UPDATE-CUSTOMER-NUMBER TO MAIN-CUSTOMER-NUMBER
READ MAIN-CUSTOMER-FILE INTO WS-CUSTOMER-RECORD
  INVALID KEY
    DISPLAY "MAIN CUSTOMER RECORD NOT FOUND"
    DISPLAY "FILE STATUS CODE IS: " CUSTOMER-FILE-STATUS
    MOVE "FALSE" TO TRANSACTION-MATCH
END-READ
```

プログラム呼び出し時のエラーの処理

プログラムが、別々にコンパイルされたプログラムを動的に呼び出すとき、呼び出されるプログラムが使用できないことがあります。 例えば、システムがストレージ不足だったり、プログラム・オブジェクトを見つけることができない場合です。CALL ステートメントに ON EXCEPTION 句も ON OVERFLOW 句もない場合、アプリケーションは異常終了する可能性があります。

一連のステートメントを実行してユーザー定義のエラー処理を行う場合は、ON EXCEPTION 句を使用します。 例えば、以下のフラグメントでは、 プログラム REPORTA が利用不可である場合、制御は ON EXCEPTION 句に渡されます。

```
MOVE "REPORTA" TO REPORT-PROG
CALL REPORT-PROG
  ON EXCEPTION
    DISPLAY "Program REPORTA not available, using REPORTB."
    MOVE "REPORTB" TO REPORT-PROG
    CALL REPORT-PROG
    END-CALL
END-CALL
```

ON EXCEPTION 句は、初期ロードでの呼び出し先プログラムの可用性に対してのみ適用されます。 呼び出し先プログラムがロードされたが、何かしらの理由 (初期設定など) で失敗した場合、ON EXCEPTION 句は実行されません。

第 2 部 各国語環境に合わせたプログラムの対応

© Copyright IBM Corp. 2021

第 10 章 国際環境でのデータの処理

COBOL for Linux は、実行時に国別文字データとして Unicode UTF-16 を サポートします。 UTF-16 は、固定幅の Unicode エンコード方式で、プレーン・テキストをエンコードするための一貫性のある効率的な方法を提供します。UTF-16 を使用すると、さまざまな国の言語で動作するソフトウェアを開発できます。

以下の COBOL 機能を使用して、国別データおよびそのようなデータの文化的に依存した照合順序を処理するプログラムのコーディングおよびコンパイルを行います。

- データ型およびリテラル:
  - 文字データ型。USAGE NATIONAL 節や、カテゴリー国別、国別編集、または数字編集のデータを定義する PICTURE 節で定義します。
  - 数値データ型。USAGE NATIONAL 節や、数値データ項目 (国別 *10* 進数項目) または外部浮動小数点データ項目 (国別浮動小数点項目) を定義する PICTURE 節で定義します。
  - リテラル接頭部 N または NX で指定される国別リテラル
  - 形象定数 ALL *national-literal*
  - 形象定数 QUOTE、SPACE、HIGH-VALUE、LOW-VALUE、または ZERO。これらは、国別文字コンテキストで使用されるときには国別文字 (UTF-16) 値を持ちます。
- COBOL ステートメント。COBOL ステートメントおよび国別データに関する以下の関連参照に示されています。
- 組み込み関数
  - NATIONAL-OF は、英数字または 2 バイト文字セット (DBCS) 文字ストリングを USAGE NATIONAL (UTF-16) に変換します。
  - DISPLAY-OF は、国別文字ストリングを選択されたコード・ページ (EBCDIC、ASCII、EUC、または UTF-8) の USAGE DISPLAY に変換します。
  - その他の組み込み関数は、組み込み関数および国別データに関する以下の関連参照に示されています。
- GROUP-USAGE NATIONAL 節。USAGE NATIONAL データ項目のみを含み、ほとんどの操作でカテゴリー国別基本項目と同様に振る舞う、グループを定義するためのものです。
- コンパイラー・オプション:
  - NSYMBOL は、リテラル内の N 記号および PICTURE 節に対して国別処理と DBCS 処理のどちらを使用するかを制御します。
  - NCOLLSEQ。国別オペランドの比較のために、照合シーケンスを指定します。

英数字または DBCS データ項目から国別表現への暗黙変換を利用することもできます。 ユーザーがこれらの項目を国別データ項目へ移動させるとき、またはこれらの項目を国別データ項目と比較するとき、コンパイラーは (ほとんどの場合に) この変換を 実行します。

関連概念
180 ページの『Unicode および言語文字のエンコード』
187 ページの『国別グループ』

関連タスク
180 ページの『COBOL での国別データ (Unicode) の使用』
188 ページの『国別 (Unicode) 表現との間の変換』
197 ページの『UTF-16 (国別) データ・タイプを使用して UTF-8 データを処理』
197 ページの『中国語 GB 18030 データの処理』
194 ページの『国別 (UTF-16) データの比較』
198 ページの『DBCS サポート用のコーディング』
201 ページの『第 11 章 ロケールの設定』

© Copyright IBM Corp. 2021

関連参照
182 ページの『COBOL ステートメントと国別データ』
185 ページの『組み込み関数と国別データ』
274 ページの『NCOLLSEQ』
274 ページの『NSYMBOL』
データのクラスおよびカテゴリー (*COBOL for Linux on x86* 言語解説書)
データ・カテゴリーおよび PICTURE 規則 (*COBOL for Linux on x86* 言語解説書)
MOVE ステートメント (*COBOL for Linux on x86* 言語解説書)
一般比較条件 (*COBOL for Linux on x86* 言語解説書)

Unicode および言語文字のエンコード

COBOL for Linux では、Unicode の基本的な実行時サポートを提供しています。 Unicode では、全世界で一般的に使用されている 文字や記号をすべて網羅する数万文字の取り扱いが可能となります。

文字セットは、定義された文字のセットですが、コード化表現と関連してはいません。 コード化文字セット (本書ではコード・ページ とも呼んでいます) は、セットの文字をそのコード化表現に関係付ける 明確な規則セットです。 各コード・ページには名前があり、文字セットを表現するための記号を設定した一種のテーブルとなっています。それぞれの記号は、固有のビット・パターン、すなわちコード・ポイント を持ちます。 コード・ページにはそれぞれ、コード化文字セット *ID* (*CCSID*) があり、1 から 65,536 までの値をとります。

Unicode には、*Unicode Transformation Format (UTF)* と呼ばれるいくつかのエンコード・スキーム (UTF-8、UTF-16、および UTF-32 など) があります。 COBOL for Linux では、国別リテラルおよび USAGE NATIONAL を持つデータ項目の表現として、 リトル・エンディアン・フォーマットの UTF-16 (CCSID 1200) を使用します。

UTF-8 は、ASCII インバリアント文字 a から z、A から Z、0 から 9、および特殊文字 (' @ , . + - = / * ( ) など) を、ASCII で表現される場合と同様に表します。 UTF-16 は、これらの文字を NX'*nn*00', として表します (ここで、X'*nn*' は ASCII での文字表現です)。

例えば、ストリング「ABC」は、UTF-16 では NX'410042004300' として表されます。UTF-8 では、「ABC」は X'414243' として表されます。

1 つ以上のエンコード・ユニット を使用して、コード化文字セットから文字を表します。 UTF-16 の場合、エンコード・ユニットは 2 バイトのストレージを使用します。 任意の EBCDIC、ASCII、または EUC コード・ページで定義された文字はいずれも、国別データ表現に変換されたときに 1 つの UTF-16 エンコード・ユニットで表現されます。

クロスプラットフォームに関する考慮事項: Enterprise COBOL for z/OS と COBOL for AIX® は、国別データでビッグ・エンディアン・フォーマットの UTF-16 をサポートします。デフォルトで COBOL for Linux は、国別データでリトル・エンディアン・フォーマットの UTF-16 をサポートします。UTF-16BE 表現でエンコードされた Unicode データを別のプラットフォームから COBOL for Linux へ移植する場合、そのデータをリトル・エンディアン・フォーマットの UTF-16 に変換してデータを国別データとして処理するか、UTF16 コンパイラー・オプションを使用してコンパイラーが UTF-16 エンディアンを処理する方法を変更する必要があります。このような変換は、COBOL for Linux では、NATIONAL-OF 組み込み関数を使用して行うことができます。

関連タスク
188 ページの『国別 (Unicode) 表現との間の変換』

関連参照
193 ページの『文字データの保管』
204 ページの『サポートされるロケールおよびコード・ページ』
文字セットとコード・ページ (*COBOL for Linux on x86* 言語解説書)

COBOL での国別データ (Unicode) の使用

COBOL for Linux では、いくつかの方法で国別 (UTF-16) データを指定できます。

次の国別データ型が使用可能です。

- 国別データ項目 (カテゴリー国別、国別編集、および数字編集)
- 国別リテラル
- 国別文字としての形象定数
- 数値データ項目 (国別 10 進数および国別浮動小数点)

加えて、明示的または暗黙的に USAGE NATIONAL を持つデータ項目のみを含んでおり、ほとんどの操作でカテゴリー国別基本項目と同様に振る舞う、国別グループを定義できます。

これらの宣言は、必要とされるストレージ量に影響します。

関連概念
180 ページの『Unicode および言語文字のエンコード』
187 ページの『国別グループ』

関連タスク
181 ページの『国別データ項目の定義』
182 ページの『国別リテラルの使用』
186 ページの『国別文字形象定数の使用』
187 ページの『国別数値データ項目の定義』
191 ページの『国別グループの使用』
188 ページの『国別 (Unicode) 表現との間の変換』
194 ページの『国別 (UTF-16) データの比較』

関連参照

193 ページの『文字データの保管』
データのクラスおよびカテゴリー (*COBOL for Linux on x86* 言語解説書)

国別データ項目の定義

国別 (UTF-16) 文字ストリングを保持する国別データ項目 を、USAGE NATIONAL 節で定義します。

以下のカテゴリーの国別データ項目を定義できます。

- 国別
- 国別編集
- 数字編集

カテゴリー国別データ項目を定義するには、1 つ以上の PICTURE 記号 N のみを含む PICTURE 節をコーディングしてください。

国別編集データ項目を定義するには、以下のそれぞれの記号の少なくとも 1 つを含む PICTURE 節をコーディングしてください。

- 記号 N
- 単純追加編集記号 B、0、または /

クラス国別の数字編集データ項目を定義するには、数字編集項目を定義する PICTURE 節 (例えば、-$999.99) をコーディングしてください。 また、USAGE NATIONAL 節をコーディングしてください。 USAGE NATIONAL を持つ数字編集データ項目は、 USAGE DISPLAY を持つ数字編集項目を使用するのと同様に使用できます。

また、PICTURE 節により数値として定義された基本項目に BLANK WHEN ZERO 節をコーディングすれば、データ項目を数字編集として定義することもできます。

PICTURE 節をコーディングしたが、1 つ以上の PICTURE 記号 N のみを含むデータ項目用に USAGE 節をコーディングしなかった場合、コンパイラー・オプション NSYMBOL(NATIONAL) を使用して、そうした項目が国別データ項目 (DBCS 項目ではなく) として取り扱われるようにしてください。

関連タスク
37 ページの『数値データの表示』

関連参照
274 ページの『NSYMBOL』
BLANK WHEN ZERO 節 (*COBOL for Linux on x86* 言語解説書)

国別リテラルの使用

国別リテラルを指定するには、接頭部文字 N を使用し、オプション NSYMBOL(NATIONAL) を指定してコンパイルします。

次のいずれかの表記を使用できます。

- N"*character-data*"
- N'*character-data*'

オプション NSYMBOL(DBCS) を指定してコンパイルする場合、リテラル接頭部文字 N は国別リテラルではなく DBCS リテラルを指定します。

国別リテラルを 16 進値として指定するには、接頭部 NX を使用します。 次のいずれかの表記を使用できます。

- NX"*hexadecimal-digits*"
- NX'*hexadecimal-digits*'

次の MOVE ステートメントのそれぞれは、国別データ項目 Y を文字「AB」の UTF-16 値に設定します。

```
01 Y pic NN usage national.
. . .
    Move NX"41004200" to Y
    Move N"AB"        to Y
    Move "AB"         to Y
```

国別リテラルを必要とするコンテキストで英数字 16 進数リテラルを使用しないでください。そのような使用法は誤解を招きやすくなります。 例えば、次のステートメントの場合も、UTF-16 文字「AB」(16 進ビット・パターン 4142 ではない) が、 Y に移動されます (Y は、USAGE NATIONAL で定義されます)。

```
Move X"4142" to Y
```

国別リテラルは、SPECIAL-NAMES 段落で使用したり、プログラム名として使用したりすることはできません。 国別リテラルは、METHOD-ID 段落のオブジェクト指向メソッドを指定したり、INVOKE ステートメント内のメソッド名を指定したりするのに使用できます。

SOSI コンパイラー・オプションを使用して、国別リテラル内のシフトアウトおよびシフトインの処理方法を制御します。

関連タスク
21 ページの『リテラルの使用』

関連参照
274 ページの『NSYMBOL』
280 ページの『SOSI』
国別リテラル (*COBOL for Linux on x86* 言語解説書)

COBOL ステートメントと国別データ

PROCEDURE DIVISION および以下の表に示すコンパイラー指示ステートメントで、国別データを使用できます。

表 16. ***COBOL*** **ステートメントと国別データ**

COBOL ステートメント	**国別にできるもの**	**説明**	**詳細の参照先**
ACCEPT	*ID-1*、 *ID-2*	*identifier-1* がランタイム・ロケールで指示されたコード・ページから変換されるのは、入力が端末からのものである場合のみです。	30 ページの『画面またはファイルからの入力の割り当て (ACCEPT)』
ADD	ID はすべて USAGE NATIONAL を持つ数値項目にすることができます。 *identifier-3* (GIVING) は、 USAGE NATIONAL を持つ数字編集にすることができます。		49 ページの『COMPUTE およびその他の算術ステートメントの使用』
CALL	*identifier-2*、*identifier-3*、*identifier-4*、*identifier-5*; *literal-2*、*literal-3*		447 ページの『データの受け渡し』
COMPUTE	*identifier-1* は、USAGE NATIONAL を持つ数値または数字編集 にすることができます。 *arithmetic-expression* は、 USAGE NATIONAL を持つ数値項目を含むことができます。		49 ページの『COMPUTE およびその他の算術ステートメントの使用』
COPY . . . REPLACING	REPLACING 句の *operand-1*、 *operand-2*		293 ページの『第 14 章 コンパイラー指示ステートメント』
DISPLAY	*identifier-1*	*identifier-1* は、現在のロケールに関連付けられているコード・ページに変換されます。	31 ページの『画面上またはファイル内での値の表示 (DISPLAY)』
DIVIDE	ID はすべて USAGE NATIONAL を持つ数値項目にすることができます。 *identifier-3* (GIVING) および *identifier-4* (REMAINDER) は、USAGE NATIONAL を持つ数字編集にすることができます。		49 ページの『COMPUTE およびその他の算術ステートメントの使用』
INITIALIZE	*identifier-1*。REPLACING 句の *identifier-2* または *literal-1*。	REPLACING NATIONAL または REPLACING NATIONAL-EDITED を指定する場合、*identifier-2* または *literal-1* は、*identifier-1* への移動における送信オペランドとして有効でなければなりません。	23 ページの『例: データ項目の初期化』

表 16. ***COBOL*** **ステートメントと国別データ** (続き)

COBOL ステートメント	**国別にできるもの**	**説明**	**詳細の参照先**
INSPECT	ID はすべてリテラルです。 (TALLYING 整数データ項目である *identifier-2* は USAGE NATIONAL を持つことができます。)	これらのいずれか (TALLYING ID である *identifier-2* を除く) が USAGE NATIONAL を持っている場合、すべてが国別でなければなりません。	102 ページの『データ項目の計算および置換 (INSPECT)』
MERGE	マージ・キー、NCOLLSEQ(BIN) を指定した場合	COLLATING SEQUENCE 句は適用されません。	159 ページの『ソートまたはマージ基準の設定』
MOVE	送り出し側と受け取り側の両方、または受け取り側のみ	有効な MOVE オペランドについては、暗黙変換が実行されます。	28 ページの『基本データ項目への値の割り当て (MOVE)』 29 ページの『グループ・データ項目への値の割り当て (MOVE)』
MULTIPLY	ID はすべて USAGE NATIONAL を持つ数値項目にすることができます。 *identifier-3* (GIVING) は、 USAGE NATIONAL を持つ数字編集にすることができます。		49 ページの『COMPUTE およびその他の算術ステートメントの使用』
SEARCH ALL (二分探索)	キー・データ項目とその比較対象の両方	キー・データ項目とその比較対象は、比較規則に従って互換性がなければなりません。 比較対象がクラス国別である場合、キーもそうでなければなりません。	78 ページの『二分探索 (SEARCH ALL)』
SORT	ソート・キー、NCOLLSEQ(BIN) を指定した場合	COLLATING SEQUENCE 句は適用されません。	159 ページの『ソートまたはマージ基準の設定』
STRING	ID はすべてリテラルです。 (POINTER 整数データ項目である *identifier-4* は USAGE NATIONAL を持つことができます。)	*identifier-3* (受信データ項目) が国別である場合、すべての ID およびリテラル (POINTER ID である *identifier-4* を除く) は国別でなければなりません。	93 ページの『データ項目の結合 (STRING)』
SUBTRACT	ID はすべて USAGE NATIONAL を持つ数値項目にすることができます。 *identifier-3* (GIVING) は、 USAGE NATIONAL を持つ数字編集にすることができます。		49 ページの『COMPUTE およびその他の算術ステートメントの使用』

表 16. *COBOL* ステートメントと国別データ (続き)			
COBOL ステートメント	**国別にできるもの**	**説明**	**詳細の参照先**
UNSTRING	ID はすべてリテラルです。 (*identifier-6* および *identifier-7* (それぞれ COUNT および TALLYING 整数データ項目) は USAGE NATIONAL を持つことができます。)	*identifier-4* (受信データ項目) が USAGE NATIONAL を持っている場合、送信データ項目およびそれぞれの区切り文字は USAGE NATIONAL を持っている必要があり、それぞれのリテラルは国別でなければなりません。	95 ページの『データ項目の分割 (UNSTRING)』
XML GENERATE	*identifier-1* (生成された XML 文書)、*identifier-2* (ソース・フィールド)、*identifier-4* または *literal-4* (名前空間 ID)、*identifier-5* または *literal-5* (名前空間接頭部)		409 ページの『第 20 章 XML 出力の生成』
XML PARSE	*identifier-1* (XML 文書)	XML-NTEXT 特殊レジスターには、構文解析時に国別文字文書フラグメントが入ります。	391 ページの『第 19 章 XML 入力の処理』

関連タスク
35 ページの『数値データの定義』
37 ページの『数値データの表示』
180 ページの『COBOL での国別データ (Unicode) の使用』
194 ページの『国別 (UTF-16) データの比較』

関連参照
274 ページの『NCOLLSEQ』
データのクラスおよびカテゴリー (*COBOL for Linux on x86* 言語解説書)

組み込み関数と国別データ

以下の表に示す組み込み関数で、クラス国別の引数を使用できます。

表 17. **組み込み関数と国別文字データ**		
組み込み関数	**関数型**	**詳細の参照先**
DISPLAY-OF	英数字	189 ページの『国別の英数字への変換 (DISPLAY-OF)』
LENGTH	整数	109 ページの『データ項目の長さの検出』
LOWER-CASE、UPPER-CASE	国別	104 ページの『大/小文字の変更 (UPPER-CASE、LOWER-CASE)』
NUMVAL、NUMVAL-C、	数値	105 ページの『数値への変換 (NUMVAL、NUMVAL-C)』
MAX、MIN	国別	107 ページの『最大または最小データ項目の検出』
ORD-MAX、ORD-MIN	整数	107 ページの『最大または最小データ項目の検出』
REVERSE	英数字または国別文字	104 ページの『逆順への変換 (REVERSE)』

ゾーン 10 進数引数が許可されていれば、国別 10 進数引数を使用できます。 表示浮動小数点引数が許可されていれば、国別浮動小数点引数を使用できます。 (整数または数値引数を取ることのできる組み込み関数の完全なリストについては、引数に関する以下の関連参照を参照してください。)

関連タスク
35 ページの『数値データの定義』
180 ページの『COBOL での国別データ (Unicode) の使用』

関連参照
引数 (*COBOL for Linux on x86* 言語解説書)
データのクラスおよびカテゴリー (*COBOL for Linux on x86* 言語解説書)
組み込み関数 (*COBOL for Linux on x86* 言語解説書)

国別文字形象定数の使用

国別文字を必要とするコンテキストでは、形象定数の ALL *national-literal* を使用できます。 ALL 国別リテラル は、国別リテラルを構成する連続したエンコード・ユニットの連結によって生成される、ストリングの全部または一部を表します。

国別文字を必要とするコンテキスト (MOVE ステートメント、暗黙移動、または、国別オペランドを持つ比較条件など) では、形象定数 QUOTE、SPACE、HIGH-VALUE、LOW-VALUE、または ZERO を使用できます。 こうしたコンテキストでは、形象定数は国別文字 (UTF-16) 値を表します。

国別文字を必要とするコンテキストで形象定数 HIGH-VALUE を使用すると、その値は NX'FFFF' です。 国別文字を必要とするコンテキストで LOW-VALUE を使用すると、その値は NX'0000' です。 NCOLLSEQ(BIN) コンパイラー・オプションが有効化されている場合にのみ、国別文字が必要なコンテキストで、 HIGH-VALUE または LOW-VALUE を使用できます。

制約事項: HIGH-VALUE または HIGH-VALUE から割り当てられた値は、あるデータ表現から別のデータ表現への値の変換 (例えば、USAGE DISPLAY と USAGE NATIONAL の間の変換、または CHAR(EBCDIC) コンパイラー・オプションが有効な場合の ASCII と EBCDIC の間の変換) が起こるような仕方で使用してはなりません。X'FF' (EBCDIC 照合シーケンスが使用されているときの、英数字コンテキストでの HIGH-VALUE の値) は有効な EBCDIC または ASCII 文字を表しませんし、NX'FFFF' は有効な国別文字を表しません。 このような値を別の表現に変換すると、置換文字が使用されることになります (X'FF' でも NX'FFFF' でもなくなります)。 次の例を見てください。

```
01 natl-data  PIC NN  Usage National.
01 alph-data  PIC XX.
. . .
    MOVE HIGH-VALUE TO natl-data, alph-data
    IF natl-data = alph-data. . .
```

上の IF ステートメントは、オペランドのそれぞれが HIGH-VALUE に設定された場合であっても、偽と評価されます。 基本英数字オペランドが国別オペランドと比較される前に、英数字オペランドは、一時国別データ項目に移動させられたかのように扱われ、英数字文字は対応する国別文字に変換されます。 しかし、X'FF' が UTF-16 に変換される場合、UTF-16 項目は置換文字値を取得するので、NX'FFFF' と比較して等しいとはみなされません。

関連タスク
188 ページの『国別 (Unicode) 表現との間の変換』
194 ページの『国別 (UTF-16) データの比較』

関連参照
256 ページの『CHAR』
274 ページの『NCOLLSEQ』
形象定数 (*COBOL for Linux on x86* 言語解説書)
DISPLAY-OF (*COBOL for Linux on x86* 言語解説書)

国別数値データ項目の定義

国別文字 (UTF-16) で表される数値データを保持するデータ項目を、USAGE NATIONAL 節で定義します。国別 10 進数項目および国別浮動小数点項目を定義できます。

国別 10 進数項目を定義するには、記号 9、P、S、および V のみを含む PICTURE 節をコーディングしてください。PICTURE 節が S を含んでいる場合、その項目で SIGN IS SEPARATE 節が有効でなければなりません。

国別浮動小数点項目を定義するには、浮動小数点項目を定義する PICTURE 節をコーディングしてください (例えば、+99999.9E-99)。

国別 10 進数項目は、ゾーン 10 進数項目と同じように使用できます。 国別浮動小数点項目は、表示浮動小数点項目と同じように使用できます。

関連タスク
35 ページの『数値データの定義』
37 ページの『数値データの表示』

関連参照

SIGN 節 (*COBOL for Linux on x86* 言語解説書)

国別グループ

GROUP-USAGE NATIONAL 節で明示的または暗黙的に指定される国別グループは、USAGE NATIONAL を持つデータ項目のみを含みます。 ほとんどの場合、国別グループ項目は、PIC N(*m*) (ここで、*m* はグループ内の国別 (UTF-16) 文字の数です) として記述されたカテゴリー国別基本項目として再定義されているかのように処理されます。

ただし、国別グループに対する操作の中には (英数字グループに対する一部の操作の場合と同様に)、グループ・セマンティクスが適用されるものがあります。 そのような操作 (例えば、MOVE CORRESPONDING や INITIALIZE) は、国別グループ内の基本項目を認識または処理します。

可能な場合、USAGE NATIONAL 項目を含んでいる英数字グループではなく、国別グループを使用してください。 国別グループでの国別データの処理の場合、英数字グループ内の国別データの処理と比較して、いくつかの利点があります。

- 国別グループを、USAGE NATIONAL を持つもっと長いデータ項目に移動させると、受信項目に 国別文字が埋め込まれます。 これに対して、国別文字を含む英数字グループを、国別文字を含むもっと長い英数字グループに移動させると、埋め込みには英数字スペースが使用されます。 その結果、データ項目の取り扱いを誤ることがあります。
- 国別グループを、USAGE NATIONAL を持つもっと 短いデータ項目に移動させると、国別グループは国別文字境界で切り捨てられます。 これに対して、国別文字を含む英数字グループを、国別文字を含むもっと短い英数字グループに移動させると、国別文字の 2 バイト間で切り捨てが起こります。
- 国別グループを国別編集または数字編集項目に移動させると、グループの内容が編集されます。 これに対し、英数字グループを編集項目に移動させた場合、編集は行われません。
- 国別グループを、STRING、UNSTRING、または INSPECT ステートメントのオペランドとして使用した場合、次のようになります。
  - グループの内容は、1 バイト文字としてではなく、国別文字として処理されます。
  - TALLYING および POINTER オペランドは、国別文字の論理レベルで作動します。
  - 国別グループ・オペランドは、他の国別オペランド・タイプの混じり合ったものと一緒にサポートされます。

  これに対し、これらのコンテキストで国別文字を含む英数字グループを使用した場合、文字はバイトごとに処理されます。 結果として、取り扱いが無効になったり、データの破壊が起こることがあります。

USAGE NATIONAL グループ: グループ項目では、グループ内のそれぞれの基本データ項目の USAGE の便利な省略表現として、グループ・レベルで USAGE NATIONAL 節を指定できます。 ただし、このようなグループは国別グループではなく、英数字グループであり、多数の操作 (移動や比較など) において USAGE DISPLAY の基本データ項目のように振る舞います (ただし、データの編集や変換は行われません)。

関連タスク
29 ページの『グループ・データ項目への値の割り当て (MOVE)』
93 ページの『データ項目の結合 (STRING)』
95 ページの『データ項目の分割 (UNSTRING)』
102 ページの『データ項目の計算および置換 (INSPECT)』
191 ページの『国別グループの使用』

関連参照
GROUP-USAGE 節 (*COBOL for Linux on x86* 言語解説書)

国別 (Unicode) 表現との間の変換

暗黙的または明示的にデータ項目を国別 (UTF-16) 表現に変換できます。

MOVE ステートメントを使用すれば、暗黙的に、英字、英数字、DBCS、または整数データを国別データに変換できます。暗黙変換は、英数字データ項目を USAGE NATIONAL 付きデータ項目と比較する IF ステートメントなど、 他の COBOL ステートメントでも行われます。

組み込み関数 NATIONAL-OF および DISPLAY-OF をそれぞれ使用して、明示的に国別データ項目に変換したり、国別データ項目から変換したりすることができます。 これらの組み込み関数を使用すると、データ項目で有効にされるコード・ページとは異なるコード・ページを変換用に指定することができます。

関連タスク
188 ページの『英数字、DBCS、および整数から国別への変換 (MOVE)』
189 ページの『英数字または DBCS から国別への変換 (NATIONAL-OF)』
189 ページの『国別の英数字への変換 (DISPLAY-OF)』
190 ページの『デフォルト・コード・ページのオーバーライド』
194 ページの『国別 (UTF-16) データの比較』
201 ページの『第 11 章 ロケールの設定』

英数字、DBCS、および整数から国別への変換 (MOVE)

MOVE ステートメントを使用して、データを国別表現に暗黙的に変換できます。

次の種類のデータをカテゴリー国別または国別編集データ項目に移動させることができ、そのようにしてデータを国別表現に変換できます。

- 英字
- 英数字
- 英数字編集
- DBCS
- USAGE DISPLAY の整数
- USAGE DISPLAY の数字編集

同様に次の種類のデータを、USAGE NATIONAL を持つ数字編集データ項目に移動させることができます。

- 英数字
- 表示浮動小数点 (USAGE DISPLAY の浮動小数点)
- USAGE DISPLAY の数字編集
- USAGE DISPLAY の整数

国別データへの移動に関する完全な規則については、MOVE ステートメントに関する関連参照を参照してください。

例えば、以下の MOVE ステートメントは、英数字 リテラル "AB" を国別データ項目 UTF16-Data に移動させます。

```
01  UTF16-Data  Pic N(2) Usage National.
    . . .
    Move "AB" to UTF16-Data
```

上記の MOVE ステートメントの実行後、UTF16-Data には、英数字「NX'41004200'」の国別表現である NX'00410042' が入ります。

USAGE NATIONAL を持つ受信データ項目で埋め込みが必要な場合、デフォルトの UTF-16 スペース 文字 (NX'2000') が使用されます。 切り捨てが必要な場合、それは国別文字位置の境界で行われます。

関連タスク
28 ページの『基本データ項目への値の割り当て (MOVE)』
29 ページの『グループ・データ項目への値の割り当て (MOVE)』
37 ページの『数値データの表示』
198 ページの『DBCS サポート用のコーディング』

関連参照

MOVE ステートメント (*COBOL for Linux on x86* 言語解説書)

英数字または DBCS から国別への変換 (NATIONAL-OF)

英字、英数字、または DBCS データを国別データ項目に変換するには、NATIONAL-OF 組み込み関数を使用してください。データ項目に対して有効になっているコード・ページとは異なるコード・ページでソースがエンコードされている場合は、ソース・コード・ページを 2 番目の引数として指定してください。

190 ページの『例: 国別データとの間の変換』

関連タスク
197 ページの『UTF-16 (国別) データ・タイプを使用して UTF-8 データを処理』
197 ページの『中国語 GB 18030 データの処理』
200 ページの『DBCS データを含む英数字データ項目の処理』

関連参照

NATIONAL-OF (*COBOL for Linux on x86* 言語解説書)

国別の英数字への変換 (DISPLAY-OF)

2 番目の引数として指定されたコード・ページで表現される英数字 (USAGE DISPLAY) 文字ストリングへ国別データを変換するには、DISPLAY-OF 組み込み関数を使用してください。

2 番目の引数を省略した場合、出力のコード・ページは、 実行時のロケールによって決定されます。

1 バイト文字セット (SBCS) 文字と DBCS 文字を結合した EBCDIC または ASCII コード・ページを指定すると、戻されるストリング は SBCS 文字と DBCS 文字の混合になることがあります。 関数で有効なコード・ページが EBCDIC コード・ページである場合、DBCS サブストリングはシフトイン文字とシフトアウト文字で区切られています。

190 ページの『例: 国別データとの間の変換』

関連概念
201 ページの『アクティブ・ロケール』

関連タスク
197 ページの『UTF-16 (国別) データ・タイプを使用して UTF-8 データを処理』
197 ページの『中国語 GB 18030 データの処理』

関連参照

DISPLAY-OF (*COBOL for Linux on x86* 言語解説書)

デフォルト・コード・ページのオーバーライド

状況によっては、 実行時に有効なコード・ページ とは異なるコード・ページにデータを変換しなければならない場合や、 そうしたコード・ページからデータを変換しなければならない場合があります。 そうするには、コード・ページを明示的に指定した変換関数を使用して項目を変換します。

DISPLAY-OF 組み込み関数の引数としてコード・ページを指定した場合に、 そのコード・ページが、実行時に有効なコード・ページとは異なる場合、暗黙的変換を伴う操作 (国別データ項目への割り当てまたは国別データ項目との比較など) で、関数結果を使用しないでください。 このような操作は、実行時のコード・ページが使用されることを想定しています。

例: 国別データとの間の変換

次の例は、国別 (UTF-16) データ項目との間での変換に、NATIONAL-OF および DISPLAY-OF 組み込み関数ならびに MOVE ステートメントを使用する方法を示しています。 また、複数のコード・ページでエンコードされたストリングに対して操作を行うときの明示的変換の必要性も示しています。

```
* . . .
 01  Data-in-Unicode          pic N(100) usage national.
 01  Data-in-Greek            pic X(100).
 01  other-data-in-US-English pic X(12) value "PRICE in $ =".
* . . .
     Read Greek-file into Data-in-Greek
     Move function National-of(Data-in-Greek, "ISO8859-7")
         to Data-in-Unicode
* . . . process Data-in-Unicode here . . .
     Move function Display-of(Data-in-Unicode, "ISO8859-7")
         to Data-in-Greek
     Write Greek-record from Data-in-Greek
```

上記の例は、入力コード・ページが指定されているので正しく機能します。 Data-in-Greek は、ISO8859-7 (Ascii ギリシャ語) で表されるデータとして変換されます。しかし、以下のステートメントの場合、 項目内の文字すべてが、たまたまギリシャ語と英語の両方のコード・ページで表現が同じであるものでなければ、変換が誤ったものになります。

```
Move Data-in-Greek to Data-in-Unicode
```

有効なロケールが en_US.ISO8859-1 であるとすると、上記の MOVE ステートメントは、コード・ページ ISO8859-1 から UTF-16LE への変換に基づいて、Data-in-Greek を Unicode に変換します。 Data-in-Greek は ISO8859-7 でエンコードされるので、この変換では期待される結果が得られません。

ロケールを el_GR.ISO8859-7 に設定した場合は (つまり、プログラムが ASCII データをギリシャ語で処理する場合は)、同じ例を次のようにコーディングすることができます。

```
* . . .
 01  Data-in-Unicode pic N(100) usage national.
 01  Data-in-Greek   pic X(100).
* . . .
     Read Greek-file into Data-in-Greek
* . . . process Data-in-Greek here ...
* . . . or do the following (if need to process data in Unicode):
     Move Data-in-Greek to Data-in-Unicode
* . . . process Data-in-Unicode
```

```
     Move function Display-of(Data-in-Unicode) to Data-in-Greek
     Write Greek-record from Data-in-Greek
```

関連タスク
201 ページの『第 11 章 ロケールの設定』

国別グループの使用

グループ・データ項目を国別グループとして定義するには、グループ・レベルで項目の GROUP-USAGE NATIONAL 節をコーディングしてください。 グループは、明示的または暗黙的に USAGE NATIONAL を持つデータ項目のみを含むことができます。

以下のデータ記述項目は、レベル 01 グループとその従属グループが国別グループ項目であることを指定します。

```
01  Nat-Group-1    GROUP-USAGE NATIONAL.
    02  Group-1.
        04  Month    PIC 99.
        04  DayOf    PIC 99.
        04  Year     PIC 9999.
    02  Group-2    GROUP-USAGE NATIONAL.
        04  Amount   PIC 9(4).99  USAGE NATIONAL.
```

上の例で、Nat-Group-1 は国別グループであり、その従属グループ Group-1 および Group-2 も国別グループです。 Group-1 に関して GROUP-USAGE NATIONAL 節が暗黙指定され、Group-1 の従属項目に関して USAGE NATIONAL が暗黙指定されています。 Month、DayOf、および Year は国別 10 進数項目であり、Amount は USAGE NATIONAL を持つ数字編集項目です。

英数字グループ内の国別グループは、次の例のようにして従属させることができます。

```
01  Alpha-Group-1.
    02  Group-1.
        04  Month   PIC 99.
        04  DayOf   PIC 99.
        04  Year    PIC 9999.
    02  Group-2   GROUP-USAGE NATIONAL.
        04  Amount  PIC 9(4).99.
```

上の例で、Alpha-Group-1 および Group-1 は英数字グループであり、Group-1 内の従属項目に関して USAGE DISPLAY が暗黙指定されています。 (Alpha-Group-1 が USAGE NATIONAL をグループ・レベルで指定した場合、Group-1 の従属項目のそれぞれについて USAGE NATIONAL が暗黙指定されることになります。 しかし、Alpha-Group-1 および Group-1 は (国別グループではなく) 英数字グループになり、移動や比較などの操作時に英数字グループと同様の振る舞いを示します。) Group-2 は国別グループであり、数字編集項目 Amount に関して USAGE NATIONAL が暗黙指定されています。

国別グループ内で英数字グループを従属させることはできません。 国別グループ内の基本項目はすべて明示的または暗黙的に USAGE NATIONAL として記述されている必要があり、国別グループ内の グループ項目はすべて明示的または暗黙的に GROUP-USAGE NATIONAL として記述されている必要があります。

関連概念
187 ページの『国別グループ』

関連タスク
192 ページの『国別グループを基本項目として使用』
192 ページの『国別グループをグループ項目として使用』

関連参照

GROUP-USAGE 節 (*COBOL for Linux on x86* 言語解説書)

国別グループを基本項目として使用

ほとんどの場合、国別グループは基本データ項目であるかのように使用できます。

次の例で、国別グループ項目 Group-1 は 国別編集項目 Edited-date へ移動されます。 Group-1 は移動時に基本データ項目として扱われるので、受信データ項目で編集が行われます。 移動後の Edited-date の値は、国別文字で 06/23/2010 になります。

```
01  Edited-date  PIC NN/NN/NNNN  USAGE NATIONAL.
01  Group-1    GROUP-USAGE NATIONAL.
    02 Month     PIC 99   VALUE 06.
    02 DayOf     PIC 99   VALUE 23.
    02 Year      PIC 9999 VALUE 2010.
    . . .
    MOVE Group-1 to Edited-date.
```

Group-1 が代わりに英数字グループであり、その中で従属項目のそれぞれが USAGE NATIONAL を持っているとした場合 (それぞれの基本項目ごとに USAGE NATIONAL 節で明示的に指定されるか、あるいはグループ・レベルで USAGE NATIONAL 節で暗黙的に指定されている場合)、基本移動ではなくグループ移動が行われます。 移動時には編集も変換も行われません。 移動後の Edited-date の最初の 8 つの文字位置の値は、国別文字で 06232010 になり、残りの 2 つの文字位置の値は 4 バイトの英数字スペースになります。

関連タスク
29 ページの『グループ・データ項目への値の割り当て (MOVE)』
196 ページの『国別データ・オペランドと英数字グループ・オペランドの比較』
192 ページの『国別グループをグループ項目として使用』

関連参照

MOVE ステートメント (*COBOL for Linux on x86* 言語解説書)

国別グループをグループ項目として使用

国別グループを使用するようなことがある場合、それはグループ・セマンティクスで処理されます。つまり、グループ内の基本項目は認識または処理されます。

次の例で、国別グループ項目 Group-OneN に作用する INITIALIZE ステートメントにより、国別文字の値 15 はグループ内の数値項目にのみ移動されます。

```
01  Group-OneN    Group-Usage National.
    05  Trans-codeN     Pic N   Value "A".
    05  Part-numberN    Pic NN  Value "XX".
    05  Trans-quanN     Pic 99  Value 10.
    . . .
    Initialize Group-OneN Replacing Numeric Data By 15
```

上の Group-OneN の Trans-quanN のみが数値なので、Trans-quanN のみが値 15 を受け取ります。 その他の従属項目は未変更です。

以下の表は、国別グループがグループ・セマンティクスで処理されるケースを要約したものです。

表 *18.* **グループ・セマンティクスで処理される国別グループ項目**

言語 機能	国別グループ項目の 使用法	説明
ADD、SUBTRACT、または MOVE ステートメントの CORRESPONDING 句	CORRESPONDING 句の規則に従って、グループとして処理する国別グループ項目を指定してください。	国別グループ内の基本項目は、英数字グループ内の USAGE NATIONAL を持つ基本項目と同様に処理されます。

表 *18.* **グループ・セマンティクスで処理される国別グループ項目** *(続き)*		
言語 機能	**国別グループ項目の 使用法**	**説明**
INITIALIZE ステートメント	INITIALIZE ステートメントの規則に従って、グループとして処理する国別グループを指定してください。	国別グループ内の基本項目は、英数字グループ内の USAGE NATIONAL を持つ基本項目と同様に初期化されます。
名前の 修飾	国別グループ項目の名前を使用して、国別グループ内の基本データ項目の名前および従属グループ項目の名前を修飾してください。	英数字グループの場合と同じ修飾の規則に従ってください。
RENAMES 節の THROUGH 句	THROUGH 句で国別グループ項目を指定するには、英数字グループ項目の場合と同じ規則を使用してください。	結果は英数字グループ項目です。
XML GENERATE ステートメントの FROM 句	XML GENERATE ステートメントの規則に従って、グループとして処理する国別グループ項目を FROM 句で指定してください。	国別グループ内の基本項目は、英数字グループ内の USAGE NATIONAL を持つ基本項目と同様に処理されます。

関連タスク
27 ページの『構造の初期化 (INITIALIZE)』
65 ページの『テーブルの初期化 (INITIALIZE)』
28 ページの『基本データ項目への値の割り当て (MOVE)』
29 ページの『グループ・データ項目への値の割り当て (MOVE)』
109 ページの『データ項目の長さの検出』
409 ページの『XML 出力の生成』

関連参照

修飾 (*COBOL for Linux on x86* 言語解説書)
RENAMES 節 (*COBOL for Linux on x86* 言語解説書)

文字データの保管

以下のテーブルを使用して英数字 (DISPLAY)、DBCS (DISPLAY-1)、および Unicode (NATIONAL) のエンコード方式を比較し、ストレージの使用法について計画を立ててください。

表 *19.* **英数字データ、*DBCS* データ、および国別データのエンコード方式およびサイズ**			
特性	**DISPLAY**	**DISPLAY-1**	**NATIONAL**
文字エンコード・ユニット	1 バイト	2 バイト	2 バイト
コード・ページ	ASCII、EUC、 UTF-8、または EBCDIC[3]	ASCII DBCS または EBCDIC DBCS[3]	UTF-16LE[1]
図形文字当たりのエンコード・ユニット数	1	1	1 または 2[2]
図形文字当たりのバイト数	1 バイト	2 バイト	2 または 4 バイト

表 19. **英数字データ、*DBCS* データ、および国別データのエンコード方式およびサイズ** (続き)			
特性	**DISPLAY**	**DISPLAY-1**	**NATIONAL**
1. ソース・プログラム内の国別リテラルは、実行時に使用できるように UTF-16 に変換されます。 2. 大部分の文字は 1 つのエンコード・ユニットを使用して UTF-16 で表現されます。特に次の文字は、文字ごとに単一の UTF-16 エンコード・ユニットを使用して表現されます。 • COBOL 文字 A から Z、a から z、0 から 9、スペース、+ -*/= $,;."()> <:' • EBCDIC、ASCII、または EUC コード・ページから変換されるすべての文字 3. ロケールに応じて、CHAR(NATIVE) または CHAR(EBCDIC) オプション、および EBCDIC_CODEPAGE 環境変数の設定			

関連概念
180 ページの『Unicode および言語文字のエンコード』

関連タスク
202 ページの『文字データのコード・ページの指定』

関連参照
256 ページの『CHAR』

国別 (UTF-16) データの比較

国別 (UTF-16) データ、すなわち USAGE NATIONAL を持つデータ項目 (クラス国別かクラス数値かにかかわらず) および国別リテラルを、比較条件の他の種類のデータと明示的または暗黙的に比較することができます。

以下のステートメントで、国別データを使用する条件式をコード化できます。

- EVALUATE
- IF
- INSPECT
- PERFORM
- SEARCH
- STRING
- UNSTRING

国別データ項目と他のデータ項目の比較について詳しくは、関連参照を参照してください。

関連タスク
194 ページの『2 つのクラス国別オペランドの比較』
195 ページの『クラス国別オペランドとクラス数値オペランドの比較』
196 ページの『国別数値オペランドと他の数値オペランドの比較』
196 ページの『国別と他の文字ストリング・オペランドとの比較』
196 ページの『国別データ・オペランドと英数字グループ・オペランドの比較』

関連参照
関連条件 (*COBOL for Linux on x86* 言語解説書)
一般比較条件 (*COBOL for Linux on x86* 言語解説書)
国別比較 (*COBOL for Linux on x86* 言語解説書)
グループ比較 (*COBOL for Linux on x86* 言語解説書)

2 つのクラス国別オペランドの比較

クラス国別の 2 つのオペランドの文字値を比較できます。

一方 (または両方) のオペランドは、次の項目タイプのいずれかにすることができます。

- 国別グループ
- カテゴリー国別または国別編集の基本データ項目
- USAGE NATIONAL を持つ数字編集データ項目

オペランドの 1 つは、代わりに国別リテラルまたは国別組み込み関数にすることができます。

以下のように、NCOLLSEQ コンパイラー・オプションを使用して、実行する比較のタイプを決定します。

NCOLLSEQ(BINARY)
同じ長さの 2 つのクラス国別オペランドを比較する場合、対応する文字のすべてのペアが等しいならば、それらは等しいと判別されます。 対応する文字の対に等しくないものがある場合は、等しくない最初の文字のペアの 2 進値を比較することによって、より大きい 2 進値を持つオペランドが判別されます。

長さの異なるオペランドを比較する場合、短いほうのオペランドは、 長いほうのオペランドの長さの位置までその右側にデフォルト の UTF-16 スペース文字 (NX'2000') が埋め込まれているものとして扱われます。

NCOLLSEQ(LOCALE)
ロケール・ベースの比較を使用する場合に、有効なロケールに関連付けられた照合順序のアルゴリズムを使用してオペランドが比較されます。 末尾のスペースはオペランドから切り捨てられます。 例外として、すべてスペースで構成されるオペランドは、シングル・スペースに切り捨てられます。

長さの異なるオペランドを比較する場合、短い方のオペランドは、スペースを使用して延長されません。これは、そのような延長によって、ロケールの期待される結果が変化してしまうことがあるからです。

PROGRAM COLLATING SEQUENCE 節は、2 つのクラス国別オペランドの比較には影響しません。

関連概念
187 ページの『国別グループ』

関連タスク
191 ページの『国別グループの使用』

関連参照
274 ページの『NCOLLSEQ』
国別比較 (*COBOL for Linux on x86* 言語解説書)

クラス国別オペランドとクラス数値オペランドの比較

国別リテラルまたはクラス国別データ項目を、整数リテラルまたは整数として定義された数値データ項目 (すなわち、国別 10 進数項目またはゾーン 10 進数項目) と比較できます。 リテラルにできるのは多くても 1 つのオペランドです。

国別リテラルまたはクラス国別データ項目を、浮動小数点データ項目 (すなわち、表示浮動小数点または国別浮動小数点項目) と比較することもできます。

数値オペランドは、まだ国別表現でない場合には、 国別 (UTF-16) 表現に変換されます。 オペランドの国別文字値が比較されます。

関連参照

一般比較条件 (*COBOL for Linux on x86* 言語解説書)

国別数値オペランドと他の数値オペランドの比較

国別数値オペランド (国別 10 進数オペランドおよび国別浮動小数点オペランド) は、USAGE NATIONAL を持つクラス数値のデータ項目です。

USAGE とは無関係に、数値オペランドの代数値を比較できます。 そのため、国別 10 進数項目または国別浮動小数点項目を、バイナリー項目、内部 10 進数項目、ゾーン 10 進数項目、表示浮動小数点項目、または他の任意の数値項目と比較できます。

関連タスク
187 ページの『国別数値データ項目の定義』

関連参照

一般比較条件 (*COBOL for Linux on x86* 言語解説書)

国別と他の文字ストリング・オペランドとの比較

国別リテラルまたはクラス国別データ項目の文字値を、以下の他の文字ストリング・オペランド (USAGE DISPLAY の英字、英数字、英数字編集、DBCS、または数字編集) のいずれかの文字値と比較できます。

これらのオペランドは、基本国別データ項目へ移動されたかのように扱われます。 文字は国別 (UTF-16) 表現へ変換され、2 つの国別文字オペランドの比較が進行します。

関連タスク
186 ページの『国別文字形象定数の使用』
199 ページの『DBCS リテラルの比較』

関連参照
国別比較 (*COBOL for Linux on x86* 言語解説書)

国別データ・オペランドと英数字グループ・オペランドの比較

国別リテラル、国別グループ項目、または USAGE NATIONAL を持つ任意の基本データ項目を、英数字グループと比較できます。

どちらのオペランドも変換されません。国別オペランドは、国別オペランドと同じサイズ (バイト単位) の英数字グループ項目に移動されたかのように扱われ、2 つのグループが比較されます。 英数字比較は、英数字グループ・オペランドの従属項目の表現とは無関係に行われます。

例えば、Group-XN は、USAGE NATIONAL を持つ 2 つの従属項目からなる英数字グループです。

```
01  Group-XN.
    02 TransCode PIC NN   Value "AB"  Usage National.
    02 Quantity  PIC 999  Value 123   Usage National.
    ...
    If N"AB123" = Group-XN  Then Display "EQUAL"
    Else Display "NOT EQUAL".
```

上記の IF ステートメントが実行されると、国別リテラル N"AB123" の 10 バイトが、バイトごとに Group-XN の内容と比較されます。 項目は比較されて同じと見なされると、「EQUAL」が表示されます。

関連参照

グループ比較 (*COBOL for Linux on x86* 言語解説書)

UTF-16 (国別) データ・タイプを使用して UTF-8 データを処理

UTF-8 データを処理するには、最初に UTF-8 データを国別データ項目の UTF-16 に変換します。国別データを処理したあとで、データを出力のために再び UTF-8 に変換します。 この変換には、それぞれ、組み込み関数 NATIONAL-OF および DISPLAY-OF を使用します。 UTF-8 データにはコード・ページ 1208 を使用します。

USAGE UTF-8 データ項目を使用して UTF-8 データを処理するという推奨方法の代わりに、UTF-8 データを英数字データ項目に保管してから、国別データ項目で UTF-16 に変換することで処理することもできます。

ASCII または EBCDIC データを UTF-8 に変換するには、以下の手順に従ってください (有効なロケールのコード・ページが UTF-8 ではない場合に限ります。UTF-8 である場合、ネイティブ英数字データは UTF-8 でエンコード済みです)。

1. 関数 NATIONAL-OF を使用して、ASCII または EBCDIC ストリングを国別ストリング (UTF-16) に変換します。
2. 関数 DISPLAY-OF を使用して、国別ストリングを UTF-8 に変換します。

次の例は、ギリシャ語の EBCDIC データを UTF-8 に変換しています。

```
01 Greek-EBCDIC pic X(10) value "αβγδεζηθ".
01 UnicodeString pic N(10).
01 UTF-8-String pic X(20).
    Move function National-of(Greek-EBCDIC, 00875) to UnicodeString
    Move function Display-of(UnicodeString, 01208) to UTF-8-String
```

使用上の注意: 参照変更を使用して UTF-8 でエンコードされたデータを参照する場合には注意してください。 UTF-8 文字のエンコードでは、1 文字に使用されるバイト数が異なります。 マルチバイト文字を分割する可能性がある処理は避けてください。

関連タスク
99 ページの『データ項目のサブストリングの参照』
188 ページの『国別 (Unicode) 表現との間の変換』
401 ページの『UTF-8 でエンコードされた XML 文書の構文解析』

中国語 GB 18030 データの処理

GB 18030 は、中華人民共和国の政府機関によって指定された国別文字標準です。

COBOL for Linux は、GB 18030 をサポートしています。 有効化されたロケールで指定されたコード・ページが GB18030 (GB 18030 をサポートするコード・ページ) である場合、 GB18030 でエンコードされた、GB 18030 文字を含む USAGE DISPLAY データ項目は、プログラムで処理できます。 GB 18030 文字は、それぞれ 1 から 4 バイトです。 そのため、プログラム・ロジックでは、データのマルチバイトの性質を意識する必要があります。

以下の方法で、GB 18030 文字を処理できます。

- 国別データ項目を使用して、UTF-16、CCSID 01200 で表される GB 18030 文字を定義および処理します。
- データを UTF-16 へ変換し、その UTF-16 データを処理した後にデータを元のコード・ページ表現へ逆変換することにより、 任意のコード・ページ (GB18030 (CCSID 1392) を含む) のデータを処理します。

変換を必要とする中国語 GB 18030 を処理する必要がある場合、まず入力データを国別データ項目の UTF-16 に変換してください。 国別データ項目を処理した後、出力用としてそれを中国語 GB 18030 に逆変換してください。 この変換には、組み込み関数 NATIONAL-OF および DISPLAY-OF を 使用し、GB18030 または 1392 を各関数の 2 番目の引数として指定します。

次の例は、これらの変換を示しています。

```
 01 Chinese-ASCII pic X(16) value "奥林匹克运动会".
 01 Chinese-GB18030-String pic X(16).
 01 UnicodeString pic N(14).
 . . .
     Move function National-of(Chinese-ASCII, 1392) to UnicodeString
*  Process data in Unicode
     Move function Display-of(UnicodeString, 1392) to Chinese-GB18030-String
```

関連タスク
188 ページの『国別 (Unicode) 表現との間の変換』
198 ページの『DBCS サポート用のコーディング』

関連参照

193 ページの『文字データの保管』

DBCS サポート用のコーディング

IBM COBOL for Linux on x86 は、2 バイト文字セット (DBCS) を使用する言語を含む多数の各国語のいずれでもアプリケーションの使用をサポートします。

以下のリストは、DBCS のサポートを要約したものです。

- ユーザー定義語での DBCS 文字 (マルチバイト名)
- コメントでの DBCS 文字
- DBCS データ項目 (PICTURE N、G、または G と B で定義します)
- DBCS リテラル
- 照合シーケンス
- SOSI コンパイラー・オプション
- DBCS_CODEPAGE 環境変数

関連タスク
198 ページの『DBCS データの定義』
199 ページの『DBCS リテラルの使用』
200 ページの『有効な DBCS 文字に関するテスト』
200 ページの『DBCS データを含む英数字データ項目の処理』
201 ページの『第 11 章 ロケールの設定』
207 ページの『ロケール付きの照合シーケンスの制御』

関連参照
280 ページの『SOSI』

DBCS データの定義

DBCS データ項目を定義するには、PICTURE および USAGE 節を使用してください。DBCS データ項目では、PICTURE 記号の G、G と B、または N を使用できます。

DBCS データ項目は、USAGE DISPLAY-1 節を使用して指定できます。 PICTURE 記号の G を使用する場合、USAGE DISPLAY-1 を指定する必要があります。 PICTURE 記号の N を指定したが、USAGE 節を省略した場合、NSYMBOL コンパイラー・オプションの設定に応じて USAGE DISPLAY-1 または USAGE NATIONAL が暗黙指定されます。

DBCS 項目の定義で USAGE 節と一緒に VALUE 節を使用する場合、DBCS リテラルまたは形象定数 SPACE または SPACES を指定する必要があります。

データ項目に USAGE DISPLAY-1 が (明示的または暗黙的に) 存在する場合、選択されたロケールは、DBCS 文字を含んだコード・ページを示す必要があります。 ロケールのコード・ページに DBCS 文字が含まれない場合には、そのデータ項目はエラーとしてフラグを立てられます。

参照変更の処理の目的のため、DBCS データ項目のそれぞれの文字は、コード・ページ幅に相当するバイト数 (つまり、2) を占有するとみなされます。

関連タスク
201 ページの『第 11 章 ロケールの設定』

関連参照
204 ページの『サポートされるロケールおよびコード・ページ』
274 ページの『NSYMBOL』

DBCS リテラルの使用

DBCS リテラルを表すには、接頭部 N または G を使用できます。

すなわち、次のいずれかの方法で DBCS リテラルを指定できます。

- N'*DBCS* 文字' (コンパイラー・オプション NSYMBOL(DBCS) が有効である場合)
- G'*dbcs characters*'

APOST または QUOTE コンパイラー・オプションの設定にかかわりなく、引用符 (") またはアポストロフィ (') を DBCS リテラルの区切り文字として使用できます。DBCS リテラルに対して、同じ開始区切り文字と終了区切り文字をコーディングする必要があります。

SOSI コンパイラー・オプションが有効化されている場合、シフトアウト (SO) 制御文字 X'1E' は、開始区切り文字の直後に続けなければなりません。シフトイン (SI) 制御文字 X'1F' は終了区切り文字の直前に来るようにする必要があります。

DBCS リテラルのほかにも、英数字リテラルを使用して、サポートされるコード・ページの 1 つの任意の文字を指定できます。 ただし、SOSI コンパイラー・オプションが有効化されている場合には、 英数字リテラルの中にある DBCS 文字のストリングは、SO および SI 文字によって区切る必要があります。

マルチバイト文字を含んでいる英数字リテラルを継続させることはできません。 さらに DBCS リテラルの長さは、B 領域の単一ソース行で使用可能なスペースによって限定されます。 したがって、DBCS リテラルの最大長は 28 個の 2 バイト文字です。

マルチバイト文字を含んでいる英数字リテラルはバイトごとに、すなわち 1 バイト文字に適したセマンティクスによって、処理されます。ただし、(例えば、国別データ項目への割り当てや国別データ項目との 比較のように) 明示的または暗黙的に国別データ表現に変換された場合は、そのような仕方で処理されません。

関連タスク
199 ページの『DBCS リテラルの比較』
22 ページの『形象定数の使用』

関連参照
274 ページの『NSYMBOL』
280 ページの『SOSI』
DBCS リテラル (*COBOL for Linux on x86* 言語解説書)

DBCS リテラルの比較

DBCS リテラルの比較は、コンパイル時のロケールに基づきます。 このため、対象とする実行時のロケールがコンパイル時のロケールと同じでない限り、 2 つの DBCS リテラル間の暗黙的な関係条件を表すステートメント内で (VALUE G'*literal-1*' THRU G'*literal-2*' など) DBCS リテラルを使用することは避けてください。

関連タスク
194 ページの『国別 (UTF-16) データの比較』
201 ページの『第 11 章 ロケールの設定』

関連参照
258 ページの『COLLSEQ』
DBCS リテラル (*COBOL for Linux on x86* 言語解説書)
DBCS 比較 (*COBOL for Linux on x86* 言語解説書)

有効な DBCS 文字に関するテスト

漢字クラス・テストでは、有効な日本語図形文字に関するテストが行われます。 このテストには、カタカナ、ひらがな、ローマ字、および漢字の文字セットが含まれます。

漢字および DBCS クラス・テストは、IBM Z 定義と整合するように定義されます。 両方のクラス・テストは、2 バイト文字を z/OS に定義された 2 バイト文字に変換して、内部で実行されます。 変換された 2 バイト文字は、DBCS および日本語 GRAPHIC 文字がないかテストされます。

漢字クラス・テストは、最初のバイトの `X'41'` から `X'7E'` および 2 番目のバイトの `X'41'` から `X'FE'` の範囲、 さらにスペース文字 `X'4040'` について、変換された文字を検査することで行われます。

DBCS クラス・テストでは、コード・ページの有効な図形文字に関するテストが行われます。

DBCS クラス・テストは、それぞれの文字の最初と 2 番目のバイト双方 の `X'41'` から `X'FE'` の範囲、およびスペース文字 `X'4040'` について、変換された文字を検査することで行われます。

関連タスク
85 ページの『条件式のコーディング』

関連参照

クラス条件 (*COBOL for Linux on x86* 言語解説書)

DBCS データを含む英数字データ項目の処理

DBCS 文字を含んでいる英数字データ項目に対してバイト指向の操作 (例えば、`STRING`、`UNSTRING`、または参照変更) を行うと、結果は予測不能です。 そうではなく項目を国別データ項目に変換してから、処理する必要があります。

すなわち、以下のステップを実行してください。

1. `MOVE` ステートメントまたは `NATIONAL-OF` 組み込み関数を使用して、項目を国別データ項目の `UTF-16` に変換します。
2. 必要に応じて国別データ項目を処理します。
3. `DISPLAY-OF` 組み込み関数を使用して、結果を英数字データ項目に逆変換します。

関連タスク
93 ページの『データ項目の結合 (STRING)』
95 ページの『データ項目の分割 (UNSTRING)』
99 ページの『データ項目のサブストリングの参照』
188 ページの『国別 (Unicode) 表現との間の変換』

第 11 章 ロケールの設定

アプリケーションの実行時に有効になるロケールの国/地域別情報を反映させるようにアプリケーションを記述することができます。 国/地域別情報には、ソート順、文字種別、各国語、さらには日付と時刻、数値、通貨、住所、および電話番号の形式が含まれます。

COBOL for Linux では、適切なコード・ページや照合シーケンスを選択したり、 言語エレメントやコンパイラー・オプションを使用して、Unicode、1 バイト文字セット、 および 2 バイト文字セット (DBCS) を処理することができます。

関連概念
201 ページの『アクティブ・ロケール』

関連タスク
202 ページの『文字データのコード・ページの指定』
203 ページの『環境変数を使用したロケールの指定』
207 ページの『ロケール付きの照合シーケンスの制御』
210 ページの『アクティブ・ロケールおよびコード・ページ値へのアクセス』

アクティブ・ロケール

ロケール とは、国/地域別環境に関する情報をエンコードするデータの集合をいいます。 アクティブ・ロケールとは、プログラムをコンパイルまたは実行するときに有効なロケールです。 アプリケーションの国/地域別環境を設定するには、アクティブ・ロケールを指定します。

一度にアクティブにできるロケールは 1 つだけです。

アクティブ・ロケールは、プログラム全体を通して国/地域別依存のインターフェースの動作に影響します。

- 文字データに使用されるコード・ページ
- メッセージ
- 照合シーケンス
- 日時形式
- 文字の種別および大/小文字の変換

アクティブ・ロケールは、次の項目には影響しません。これらについては、85 COBOL 標準で特定の言語および動作が定義されています。

- 小数点およびグループ化された分離文字
- 通貨記号

アクティブ・ロケールは、プログラムをコンパイルおよび実行するためのコード・ページを決定します。

- コンパイルに使用されるコード・ページはコンパイル時のロケール設定に基づきます。
- アプリケーションの実行に使用されるコード・ページは実行時のロケール設定に基づきます。

ソース・プログラム内のリテラル値の評価は、コンパイル時にアクティブなロケールを使用して処理されます。 例えば、プログラムを実行するために国別リテラルをソース表現から UTF-16 に変換する処理では、コンパイル時のロケールを使用します。

COBOL for Linux は、該当する環境変数とシステム設定の組み合わせからアクティブ・ロケールの設定を決定します。 まず最初に、環境変数が使用されます。 該当するロケールのカテゴリーが環境変数によって定義されていない場合、 COBOL ではデフォルトとシステム設定を使用します。

関連概念
204 ページの『システム設定からのロケールの決定』

関連タスク
202 ページの『文字データのコード・ページの指定』

© Copyright IBM Corp. 2021

203 ページの『環境変数を使用したロケールの指定』
207 ページの『ロケール付きの照合シーケンスの制御』

関連参照
204 ページの『変換が使用可能なメッセージのタイプ』

文字データのコード・ページの指定

ソース・プログラムでは、COBOL の名前、リテラル、およびコメント内では、 サポートされるコード・ページで表現される文字を使用できます。 実行時に、USAGE DISPLAY、USAGE DISPLAY-1、または USAGE NATIONAL で記述されているデータ項目内では、サポートされるコード・ページで表現される文字を使用することができます。

特定の英数字データ項目に対して有効となるコード・ページは、以下の側面によって決まります。

- 使用される USAGE 文節
- NATIVE 句を USAGE 文節と共に使用したかどうか
- CHAR(NATIVE) または CHAR(EBCDIC) コンパイラー・オプションを使用したかどうか
- EBCDIC_CODEPAGE 環境変数の値
- DBCS_CODEPAGE 環境変数の値
- アクティブなロケール

USAGE NATIONAL データ項目の場合、コード・ページはデフォルトのリトル・エンディアン形式に設定されます。

USAGE DISPLAY データ項目の場合、COBOL for Linux は 以下のように ASCII、UTF-8、EUC、 および EBCDIC の各コード・ページから コード・ページを選択します。

- USAGE 文節の NATIVE 句で記述された データ項目、または CHAR(NATIVE) オプションを有効にして コンパイルされたデータ項目は、ASCII コード・ページ、EUC コード・ページ、 または UTF-8 コード・ページでエンコードされます。
- USAGE 文節の NATIVE 句を使用せずに記述された データ項目のうち、CHAR(EBCDIC) オプションを有効にして コンパイルされたデータ項目は、EBCDIC コード・ページでエンコードされます。

USAGE DISPLAY-1 データ項目の場合、COBOL for Linux は 以下のように ASCII コード・ページまたは EBCDIC コード・ページを選択します。

- USAGE 文節の NATIVE 句で記述された データ項目、または CHAR(NATIVE) オプションを有効にして コンパイルされたデータ項目は、ASCII DBCS コード・ページでエンコードされます。
- USAGE 文節の NATIVE 句を使用せずに記述された データ項目のうち、CHAR(EBCDIC) オプションを有効にして コンパイルされたデータ項目は、EBCDIC DBCS コード・ページでエンコードされます。

COBOL では、適切なコード・ページを次のように決定します。

ASCII、 UTF-8、EUC
実行時のアクティブ・ロケールから

ASCII DBCS
DBCS_CODEPAGE 環境変数が設定されている場合は、この変数から決定される。それ以外の場合は、現行のローカル設定からデフォルトの DBCS コード・ページが決定される。

EBCDIC
EBCDIC_CODEPAGE 環境変数が設定されている場合は、この変数から決定される。それ以外の場合は、現行のロケール設定からデフォルトの EBCDIC コード・ページが決定される。

関連タスク
203 ページの『環境変数を使用したロケールの指定』

関連参照
204 ページの『サポートされるロケールおよびコード・ページ』
220 ページの『ランタイム環境変数』

256 ページの『CHAR』
1 バイト文字の COBOL ワード
(*COBOL for Linux on x86* 言語解説書)
マルチバイト文字のユーザー定義語
(*COBOL for Linux on x86* 言語解説書)

環境変数を使用したロケールの指定

COBOL プログラムのロケール情報を提供するには、いずれかの環境変数を使用します。

すべてのロケール・カテゴリー (メッセージ、照合シーケンス、 日付と時刻形式、文字種別、および大/小文字の変換) に使用するコード・ページを指定するには、LC_ALL を使用します。

特定のロケール・カテゴリーの値を設定するには、以下の該当する環境変数を使用します。

- LC_MESSAGES は、肯定応答および否定応答の形式を指定するために使用します。 また、これを使用して、メッセージ (エラー・メッセージやリスト・ヘッダーなど) が米国英語か日本語の どちらになるかを決定できます。 日本語以外のロケールの場合は、米国英語が使用されます。
- LC_COLLATE は、関連条件内や SORT および MERGE ステートメント内などの、 より大比較またはより小比較に有効な照合シーケンスを指定するために使用します。
- LC_TIME は、コンパイラー・リストに示される日付と時刻の形式を指定するために使用します。 それ以外の日付および時刻の値はすべて COBOL 言語の構文に従います。
- LC_CTYPE は、文字の種別、大/小文字の変換、およびその他の文字属性を指定するために使用します。

上記のロケール環境変数のいずれかで指定されないロケール・カテゴリーは、LANG 環境変数の値から設定されます。

ロケールの環境変数を設定するには、次のような形式のコマンドを使用します (*.codepageID* はオプションです)。

```
export LC_xxxx=ll_CC.codepageID
```

ここで、 LC_*xxxx* は ロケール・カテゴリーの名前、 *ll* は小文字の 2 文字の言語コード、*CC* は 大文字の 2 文字の ISO 国別コード、および *codepageID* は ネイティブ DISPLAY と DISPLAY-1 データに使用されるコード・ページです。 COBOL for Linux では、POSIX で定められたロケール規約を使用しています。

例えば、ロケールを ISO 8859-1 でエンコードされるカナダ・フランス語に設定するには、COBOL アプリケーションをコンパイルして実行するコマンド・ウィンドウで、次のコマンドを発行します。

```
export LC_ALL=fr_CA.iso88591
```

ロケール名の有効な値 (*ll_CC*) と、 指定するコード・ページ (*codepageID*) はロケール名に有効でなければなりません。 有効な値は、後述のロケールおよびコード・ページの表に示されています。

関連概念
204 ページの『システム設定からのロケールの決定』

関連タスク
202 ページの『文字データのコード・ページの指定』

関連参照

204 ページの『サポートされるロケールおよびコード・ページ』
216 ページの『コンパイラー環境変数とランタイム環境変数』

システム設定からのロケールの決定

COBOL for Linux が該当するロケール・カテゴリーの値を環境変数から決定できない場合は、次のデフォルト設定が使用されます。

言語と国別コードは環境変数から決定されるが、コード・ページは決定されないという場合、COBOL for Linux では、言語と国別コードの組み合わせに関するデフォルト・システム・コード・ページが使用されます。

言語と国別コードの組み合わせに複数のコード・ページが適用される場合があります。 Linux システムでデフォルトのコード・ページを選択したくない場合は、コード・ページを明示的に指定する必要があります。

UTF-8: UTF-8 エンコードは、あらゆる言語と国別コードの組み合わせに対応しています。

関連タスク
202 ページの『文字データのコード・ページの指定』
203 ページの『環境変数を使用したロケールの指定』
210 ページの『アクティブ・ロケールおよびコード・ページ値へのアクセス』
215 ページの『環境変数の設定』

関連参照

204 ページの『サポートされるロケールおよびコード・ページ』
216 ページの『コンパイラー環境変数とランタイム環境変数』

変換が使用可能なメッセージのタイプ

以下のメッセージは各国語サポート、すなわちコンパイラー、ランタイム、およびデバッガー・ユーザー・インターフェース・メッセージ、 およびリスト・ヘッダー (ロケール・ベースの日付と時刻形式を含む) に対応しています。

アクティブ・ロケールで指定された該当するテキストおよび形式が、 これらのメッセージおよびリスト・ヘッダーに使用されます。

メッセージの言語とロケールに影響する LANG および NLSPATH 環境変数については、下の『関連参照』を参照してください。

関連概念
201 ページの『アクティブ・ロケール』

関連タスク
203 ページの『環境変数を使用したロケールの指定』

関連参照
216 ページの『コンパイラー環境変数とランタイム環境変数』

サポートされるロケールおよびコード・ページ

次の表に、ご使用のシステムで使用可能なロケールと、各ロケールでサポートされるコード・ページを示します。 COBOL for Linux は、システム上でコンパイル時と実行時に使用できるロケールをサポートしています。

システム上の使用可能なロケールを照会するには、以下のように入力します。

```
locale -a
```

表 *20.* **サポートされるロケールおよびコード・ページ**

ロケール名 [1]	言語 [2]	国または地域 [3]	ASCII ベースのコード・ページ [4]	EBCDIC コード・ページ [5]	言語グループ
any			utf-8	ロケールに適用可能な EBCDIC コード・ページは、ロケールのコード・ページ値にかかわらず、ロケール名 *language* および *COUNTRY* 部に基づきます。	
ar_AE	アラビア語	アラブ首長国連邦	iso88596	IBM-16804、IBM-420	アラビア語
be_BY	ベロルシア語	ベラルーシ	iso88595	IBM-1025、IBM-1154	Latin 5
bg_BG	ブルガリア語	ブルガリア	iso88595	IBM-1025、IBM-1154	Latin 5
ca_ES	カタロニア語	スペイン	iso88591	IBM-285、IBM-1145	Latin 1
cs_CZ	チェコ語	チェコ共和国	iso88592	IBM-870、IBM-1153	Latin 2
da_DK	デンマーク語	デンマーク	iso88591	IBM-277、IBM-1142	Latin 1
de_CH	ドイツ語	スイス	iso88591	IBM-500、IBM-1148	Latin 1
de_DE	ドイツ語	ドイツ	iso88591	IBM-273、IBM-1141	Latin 1
el_GR	ギリシャ語	ギリシャ	iso88597	IBM-4971、IBM-875	ギリシャ語
en_AU	英語	オーストラリア	iso88591	IBM-037、IBM-1140	Latin 1
en_GB	英語	英国	iso88591	IBM-037、IBM-1140	Latin 1
en_US	英語	米国	iso88591	IBM-037、IBM-1140	Latin 1
en_ZA	英語	南アフリカ	iso88591	IBM-037、IBM-1140	Latin 1
es_ES	スペイン語	スペイン	iso88591	IBM-284、IBM-1145	Latin 1
fi_FI	フィンランド語	フィンランド	iso88591	IBM-278、IBM-1143	Latin 1
fr_BE	フランス語	ベルギー	iso88591	IBM-297、IBM-1148	Latin 1
fr_CA	フランス語	カナダ	iso88591	IBM-037、IBM-1140	Latin 1
fr_CH	フランス語	スイス	iso88591	IBM-500、IBM-1148	Latin 1
fr_FR	フランス語	フランス	iso88591	IBM-297、IBM-1148	Latin 1
hr_HR	クロアチア語	クロアチア	iso88592	IBM-870、IBM-1153	Latin 2
hu_HU	ハンガリー語	ハンガリー	iso88592	IBM-870、IBM-1153	Latin 2
is_IS	アイスランド語	アイスランド	iso88591	IBM-871、IBM-1149	Latin 1
it_CH	イタリア語	スイス	iso88591	IBM-500、IBM-1148	Latin 1
it_IT	イタリア語	イタリア	iso88591	IBM-280、IBM-1144	Latin 1
iw_IL	ヘブライ語	イスラエル国	iso88598	IBM-12712、IBM-424	ヘブライ語
ja_JP	日本語	日本	IBMeucjp	IBM-930、IBM-939、IBM-1390、IBM-1399	表意文字言語

表 *20.* **サポートされるロケールおよびコード・ページ** *(続き)*

ロケール名 [1]	言語 [2]	国または地域 [3]	ASCII ベースのコード・ページ [4]	EBCDIC コード・ページ [5]	言語グループ
ko_KR	韓国語	大韓民国	euckr	IBM-933、IBM-1364	表意文字言語
lt_LT	リトアニア語	リトアニア	IBMiso885913	n/a	リトアニア語
lv_LV	ラトビア語	ラトビア	IBMiso885913	n/a	ラトビア語
mk_MK	マケドニア語	マケドニア	iso88595	IBM-1025、IBM-1154	Latin 5
nl_BE	オランダ語	ベルギー	iso8859-1	IBM-500、IBM-1148	Latin 1
nl_NL	オランダ語	オランダ	iso88591	IBM-037、IBM-1140	Latin 1
no_NO	ノルウェー語	ノルウェー	iso88591	IBM-277、IBM-1142	Latin 1
pl_PL	ポーランド語	ポーランド	iso88592	IBM-870、IBM-1153	Latin 2
pt_BR	ポルトガル語	ブラジル	iso88591	IBM-037、IBM-1140	Latin 1
pt_PT	ポルトガル語	ポルトガル	iso88591	IBM-037、IBM-1140	Latin 1
ro_RO	ルーマニア語	ルーマニア	iso88592	IBM-870、IBM-1153	Latin 2
ru_RU	ロシア語	ロシア連邦	iso88595	IBM-1025、IBM-1154	Latin 5
sk_SK	スロバキア語	スロバキア	iso88592	IBM-870、IBM-1153	Latin 2
sl_SI	スロベニア語	スロベニア	iso8859-2	IBM-870、IBM-1153	Latin 2
sq_AL	アルバニア語	アルバニア	iso88591	IBM-500、IBM-1148	Latin 1
sv_SE	スウェーデン語	スウェーデン	iso88591	IBM-278、IBM-1143	Latin 1
th_TH	タイ語	タイ	tis620	IBM-9030	タイ語
tr_TR	トルコ語	トルコ	iso88599	IBM-1026、IBM-1155	トルコ語
uk_UA	ウクライナ語	ウクライナ	iso88595	IBM-1123、IBM-1154	Latin 5
zh_CN	中国語	中国	gb18030	IBM-1388	表意文字言語
zh_TW	中国語 (繁体字)	台湾	IBMeuctw	IBM-1371、IBM-937	表意文字言語

表 20. **サポートされるロケールおよびコード・ページ** (続き)

ロケール名 [1]	言語 [2]	国または地域 [3]	ASCII ベースのコード・ページ [4]	EBCDIC コード・ページ [5]	言語グループ

1. サポートされる ISO 言語コードと ISO 国別コードの有効な組み合わせ (*language_COUNTRY*) を示します。 表に示されているロケール名の各文字が大文字か小文字かは重要で、ロケールに対して特定のコード・ページが選択 (または暗示) されているロケール名の大/小文字を反映しない場合があります。 選択したロケール名の各文字の適切な大/小文字については、「locale -a」コマンドの結果を参照してください。
2. 関連した言語を示します。
3. 関連した国または地域を示します。
4. 対応する *language_COUNTRY* 値を持つロケールのコード・ページ ID として有効なコード・ページを示します。 これらの表のエントリーは、確定したものではありません。 有効なロケールの現在のリストについては、お使いのシステムのマニュアルで実行している Linux の特定のバージョンと構成について確認してください。選択するロケールは有効でなければなりません。つまり、プログラムを開発する場所と実行する場所の両方にインストールされている必要があります。
5. 対応する *language_COUNTRY* 値を持つロケールのコード・ページ ID として有効なコード・ページを示します。 これらのコード・ページは、EBCDIC_CODEPAGE 環境変数の内容として有効です。 EBCDIC_CODEPAGE 環境変数が設定されていない場合は、この列内の右端に示すコード・ページ項目が、 対応するロケールの EBCDIC コード・ページとして選択されます。

関連タスク
202 ページの『文字データのコード・ページの指定』
203 ページの『環境変数を使用したロケールの指定』

ロケール付きの照合シーケンスの制御

比較、ソート、マージなどのさまざまな操作は、プログラムおよびデータ項目に有効な照合シーケンスを使用します。 照合シーケンスの制御方法は、データのクラスに対して有効なコード・ページに応じて、英字、英数字、DBCS、または国別のいずれかになります。

英字、英数字、または DBCS クラスである項目にロケール・ベースの 照合シーケンスが適用されるのは、`COLLSEQ(LOCALE)` コンパイラー・オプションが 有効な場合のみで、`COLLSEQ(BIN)` または `COLLSEQ(EBCDIC)` が 有効な場合には適用されません。 同様に、 国別クラス項目のロケール・ベースの照合シーケンスが適用されるのは、`NCOLLSEQ(LOCALE)` コンパイラー・オプションが 有効な場合のみで、`NCOLLSEQ(BIN)` が有効な場合は適用されません。

`COLLSEQ(LOCALE)` または `NCOLLSEQ(LOCALE)` コンパイラー・オプションが有効な場合、ロケール・ベースの照合順序によって影響を受ける構文 または意味体系規則を持つ言語エレメントには、コンパイル時のロケールが使用されます。 それらは以下のようなものです。

- 条件名 VALUE 文節内の THRU 句
- EVALUATE ステートメント内の *literal-3* THRU *literal-4* 句
- ALPHABET 文節内の *literal-1* THRU *literal-2* 句
- SYMBOLIC CHARACTERS 文節で指定された文字の順序位置
- CLASS 文節内の THRU 句

`COLLSEQ(LOCALE)` コンパイラー・オプションが有効な場合、SORT および MERGE ステートメントの英数字キーの照合シーケンスは常に実行時のロケールに基づきます。

関連タスク
6 ページの『照合シーケンスの指定』
159 ページの『ソートまたはマージ基準の設定』
202 ページの『文字データのコード・ページの指定』
203 ページの『環境変数を使用したロケールの指定』
208 ページの『ロケール付きの英数字照合シーケンスの制御』

209 ページの『ロケール付きの DBCS 照合シーケンスの制御』
209 ページの『ロケール付きの国別照合シーケンスの制御』
210 ページの『アクティブ・ロケールおよびコード・ページ値へのアクセス』

関連参照

204 ページの『サポートされるロケールおよびコード・ページ』
258 ページの『COLLSEQ』
274 ページの『NCOLLSEQ』

ロケール付きの英数字照合シーケンスの制御

プログラム照合シーケンスの場合、 1 バイト英数字の照合シーケンスは、 コンパイル時または実行時のロケールに基づきます。

ソース・プログラムで PROGRAM COLLATING SEQUENCE を指定すると、 照合シーケンスはコンパイル時に設定され、実行時のロケールに関係なく使用されます。 逆に、COLLSEQ コンパイラー・オプションを使用して照合シーケンスを設定すると、 実行時のロケールが優先されます。

有効なコード・ページが 1 バイト ASCII コード・ページの場合、SPECIAL-NAMES 段落で 以下の文節を指定できます。

- ALPHABET 文節
- SYMBOLIC CHARACTERS 文節
- CLASS 文節

有効なソース・コード・ページに DBCS 文字が含まれている場合にこれらの文節を指定すると、文節が診断され、コメントとして扱われます。 COBOL のユーザー定義の英字名およびシンボリック文字の規則では、 複数文字のシーケンスに依存する照合シーケンスではなく、 文字単位の照合シーケンスを前提としています。

PROGRAM COLLATING SEQUENCE 文節を OBJECT-COMPUTER 段落内で 指定した場合、*alphabet-name* に関連付けられている照合シーケンスを使用して、 英数字比較の真の値が決定されます。 また、PROGRAM COLLATING SEQUENCE 文節は、SORT または MERGE ステートメントで COLLATING SEQUENCE 句を指定していない限り、USAGE DISPLAY のソートおよびマージ・キーに適用されます。

COLLATING SEQUENCE 句または PROGRAM COLLATING SEQUENCE 文節を指定しない場合、有効な照合シーケンスはデフォルトで NATIVE になり、アクティブなロケール設定に基づきます。 この設定は、SORT および MERGE ステートメントと、プログラムの照合シーケンスに適用されます。

この照合シーケンスは、次の項目の処理に影響します。

- ALPHABET 文節 (for example, *literal-1* THRU *literal-2* など)
- SYMBOLIC CHARACTERS 仕様
- レベル 88 の項目、比較条件、SORT および MERGE ステートメントに対する VALUE 範囲の指定

関連タスク

6 ページの『照合シーケンスの指定』
207 ページの『ロケール付きの照合シーケンスの制御』
209 ページの『ロケール付きの DBCS 照合シーケンスの制御』
209 ページの『ロケール付きの国別照合シーケンスの制御』
159 ページの『ソートまたはマージ基準の設定』

関連参照

258 ページの『COLLSEQ』
データのクラスおよびカテゴリー (*COBOL for Linux on x86* 言語解説書)
英数字比較 (*COBOL for Linux on x86* 言語解説書)

ロケール付きの DBCS 照合シーケンスの制御

実行時のロケール・ベースの照合シーケンスは、リテラルの比較の場合を除き、常に DBCS データに適用されます。

クラス DBCS のデータ項目およびリテラルは、任意の関係演算子を持つ比較条件で使用することができます。 その他のオペランドはクラス DBCS かクラス national、または英数字グループでなければなりません。 DBCS 項目と編集済み DBCS 項目の間に区別はありません。

2 つの DBCS オペランドを比較する場合、COLLSEQ(LOCALE) コンパイラー・オプションが 有効であれば、照合シーケンスはアクティブ・ロケールによって決定されます。 有効でない場合、照合シーケンスは DBCS 文字のバイナリー値によって決定されます。 PROGRAM COLLATING SEQUENCE 文節は、 クラス DBCS のデータ項目またはリテラルを含む比較には影響しません。

DBCS 項目を国別項目と比較する場合、DBCS オペランドは、DBCS オペランドと 同じ長さの基本国別項目に移動されるかのように処理されます。 DBCS 文字は国別表現へ変換され、2 つの国別文字オペランドの比較が進行します。

DBCS 項目を英数字グループと比較する場合、変換または編集は行われません。 2 つの英数字オペランドについては、比較が実行されます。 比較は、データ表現には関係なくデータの各バイトを処理します。

関連タスク
6 ページの『照合シーケンスの指定』
199 ページの『DBCS リテラルの使用』
207 ページの『ロケール付きの照合シーケンスの制御』
208 ページの『ロケール付きの英数字照合シーケンスの制御』
209 ページの『ロケール付きの国別照合シーケンスの制御』

関連参照
258 ページの『COLLSEQ』
データのクラスおよびカテゴリー (*COBOL for Linux on x86* 言語解説書)
英数字比較 (*COBOL for Linux on x86* 言語解説書)
DBCS 比較 (*COBOL for Linux on x86* 言語解説書)
グループ比較 (*COBOL for Linux on x86* 言語解説書)

ロケール付きの国別照合シーケンスの制御

USAGE NATIONAL の国別リテラルまたはデータ項目は、任意の関係演算子を持つ比較条件で使用することができます。 PROGRAM COLLATING SEQUENCE 文節は国別オペランドを含む比較に影響を及ぼしません。

実行時にアクティブなロケールと関連付けられた照合順序のアルゴリズムに基づく比較を 有効にするには、NCOLLSEQ(LOCALE) コンパイラー・オプションを使用します。 NCOLLSEQ(BINARY) が有効な場合、照合シーケンスは国別文字のバイナリー値によって 決定されます。

SORT または MERGE ステートメントが クラス国別になるのは、NCOLLSEQ(BIN) オプションが 有効な場合のみです。

関連タスク
194 ページの『国別 (UTF-16) データの比較』
207 ページの『ロケール付きの照合シーケンスの制御』
209 ページの『ロケール付きの DBCS 照合シーケンスの制御』
159 ページの『ソートまたはマージ基準の設定』

関連参照
274 ページの『NCOLLSEQ』
データのクラスおよびカテゴリー (*COBOL for Linux on x86* 言語解説書)
国別比較 (*COBOL for Linux on x86* 言語解説書)

照合シーケンスに依存する組み込み関数

次の組み込み関数は、文字の順序位置によって異なります。

ASCII コード・ページの場合、これらの組み込み関数は有効な照合シーケンスに基づいてサポートされます。 EUC コード・ページまたは DBCS 文字を含むコード・ページの場合は、1 バイト文字の順序位置が、1 バイト文字の 16 進数表現に対応するものとします。 例えば、「A」の順序位置は 66 (X'41' + 1)、「*」の順序位置は 43 (X'2A' + 1) となります。

表 *21.* **照合シーケンスに依存する組み込み関数**

組み込み関数	戻り	コメント
CHAR	順序位置引数に対応する文字	
MAX	最大値を含む引数の内容	引数は、英字、英数字、国別、または数字です。[1]
MIN	最小値を含む引数の内容	引数は、英字、英数字、国別、または数字です。[1]
ORD	文字引数の順序位置	
ORD-MAX	最大値を含む引数の、引数リスト内での整数順序位置	引数は、英字、英数字、国別、または数字です。[1]
ORD-MIN	最小値を含む引数の、引数リスト内での整数順序位置	引数は、英字、英数字、国別、または数字です。[1]

1. 関数に数値の引数が含まれている場合は、コード・ページと照合シーケンスが適用されません。

これらの組み込み関数は、DBCS データ型ではサポートされません。

関連タスク
6 ページの『照合シーケンスの指定』
194 ページの『国別 (UTF-16) データの比較』
207 ページの『ロケール付きの照合シーケンスの制御』

アクティブ・ロケールおよびコード・ページ値へのアクセス

コンパイル時に有効なロケールを検証するには、コンパイラー・リストの最後の数行を検査します。

アプリケーションによっては、実行時にアクティブなロケールおよび EBCDIC コード・ページを検証し、コード・ページ ID を対応する CCSID に変換することが必要な場合もあります。 このような照会や変換は、呼び出し可能なライブラリー・ルーチンを使用して実行することができます。

実行時にアクティブなロケールおよび EBCDIC コード・ページにアクセスするには、 以下のようにライブラリー関数 _iwzGetLocaleCP を呼び出します。

```
CALL "_iwzGetLocaleCP" USING output1, output2
```

変数 *output1* は、 次のフォーマットでヌル終了ロケール値を表す、20 文字の英数字項目です。

- 2 文字の言語コード
- 下線 (_)
- 2 文字の国別コード
- ピリオド (.)
- ロケールのコード・ページ値

例えば、en_US.IBM-1252 は、言語コード en、国別コード US、コード・ページ IBM-1252 のロケール値を表しています。

変数 *output2* は、 有効なヌル終了 EBCDIC コード・ページ ID を表す、10 文字の 英数字項目 (IBM-1140 など) です。

コード・ページ ID を対応する CCSID に変換するには、次のようにライブラリー関数 _iwzGetCCSID を呼び出します。

```
CALL "_iwzGetCCSID" USING input, output RETURNING returncode
```

input は、 ヌル終了コード・ページ ID を表す英数字項目です。

output は、 4 バイトの符号付き 2 進数データ項目 (PIC S9(5) COMP-5 として 定義された項目など) です。 入力コード・ページ ID ストリングまたはエラー・コード -1 に対応する CCSID が戻されます。

returncode は、 次のように設定される、4 バイトの符号付き 2 進数データ項目です。

0
成功

1
コード・ページ ID は有効だが、関連する CCSID がない。*output* は -1 に設定されます。

-1
コード・ページ ID が有効なコード・ページではない。*output* は -1 に設定されます。

これらのサービスを呼び出すには、PGMNAME(MIXED) および NODYNAM コンパイラー・オプションを使用する必要があります。

211 ページの『例: コード・ページ ID の取得および変換』

関連タスク
201 ページの『第 11 章 ロケールの設定』

関連参照
264 ページの『DYNAM』
276 ページの『PGMNAME』
293 ページの『第 14 章 コンパイラー指示ステートメント』

例: コード・ページ ID の取得および変換

次の例は、呼び出し可能サービス _iwzGetLocaleCP および _iwzGetCCSID を使用して、 有効なロケールと EBCDIC コード・ページをそれぞれ取得し、 コード・ページ ID を対応する CCSID に変換する方法を示しています。

```
cbl pgmname(lm)
       Identification Division.
       Program-ID.  "Samp1".
       Data Division.
       Working-Storage Section.
       01 locale-in-effect.
          05 ll-cc             pic x(5).
          05 filler-period     pic x.
          05 ASCII-CP          Pic x(14).
       01 EBCDIC-CP            pic x(10).
       01 CCSID                pic s9(5) comp-5.
       01 RC                   pic s9(5) comp-5.
       01 n                    pic 99.

       Procedure Division.
       Get-locale-and-codepages section.
       Get-locale.
           Display "Start Samp1."
           Call "_iwzGetLocaleCP"
             using locale-in-effect, EBCDIC-CP
           Move 0 to n
           Inspect locale-in-effect
             tallying n for characters before initial x'00'
           Display "locale in effect: " locale-in-effect (1 : n)
           Move 0 to n
```

```
           Inspect EBCDIC-CP
              tallying n for characters before initial x'00'
           Display "EBCDIC code page in effect: "
             EBCDIC-CP (1 : n).

       Get-CCSID-for-EBCDIC-CP.
           Call "_iwzGetCCSID" using EBCDIC-CP, CCSID returning RC
           Evaluate RC
             When 0
               Display "CCSID for " EBCDIC-CP (1 : n) " is " CCSID
             When 1
               Display EBCDIC-CP (1 : n)
                       " does not have a CCSID value."
             When other
               Display EBCDIC-CP (1 : n) " is not a valid code page."
           End-Evaluate.

       Done.
           Goback.
```

ロケールを ja_JP.IBM-943 (`set LC_ALL=ja_JP.IBM-943`) に設定した場合、このサンプル・プログラムからの出力は次のようになります。

```
Start Samp1.
locale in effect: ja_JP.IBM-943
EBCDIC code page in effect: IBM-1399
CCSID for IBM-1399 is 0000001399
```

関連タスク

203 ページの『環境変数を使用したロケールの指定』

第 3 部 プログラムのコンパイル、リンク、実行、およびデバッグ

© Copyright IBM Corp. 2021

第 12 章 プログラムのコンパイル、リンク、実行

続くセクションでは、環境変数の設定、コンパイル、リンク、 実行、およびエラーの訂正の再配布の方法について説明します。

関連タスク
215 ページの『環境変数の設定』
224 ページの『プログラムのコンパイル』
229 ページの『ソース・プログラムのエラーの訂正』
234 ページの『プログラムのリンク』
237 ページの『リンク内のエラーの訂正』
238 ページの『プログラムの実行』

関連参照

232 ページの『cob2 オプション』

環境変数の設定

プログラムに必要な値を設定するには、環境変数を使用します。 環境変数の値を指定するには、 `export` コマンドまたは putenv() POSIX 関数を使用します。 環境変数を設定しなかった場合は、デフォルト値が適用されるか、またはその変数は定義されません。

環境変数 は、変化する可能性のあるユーザー環境またはプログラム環境の一部分を定義します。 例えば、プログラムが別のプログラムによって動的に呼び出されたときに、COBOL ランタイムでそのプログラムを検索できる場所を定義するには、COBPATH 環境変数を使用します。 環境変数は、コンパイラーおよびランタイム・ライブラリーの両方で使用されます。

IBM COBOL for Linux on x86 をインストールすると、インストール・プロセスによって、COBOL for Linux コンパイラーおよびランタイム・ライブラリーにアクセスするための環境変数が設定されます。 単純な COBOL プログラムをコンパイルおよび実行するために設定する必要があるのは LANG だけで、デフォルトの en_US メッセージ以外のメッセージを使用する場合に限り設定する必要があります。

`export` コマンドを使用すれば、2 つある以下の場所のどちらかで環境変数の値を変更できます。

- コマンド・シェルのプロンプト (例: XTERM ウィンドウ)。 この環境変数の定義は、該当のシェルまたはその下位シェル (該当のシェルから直接または間接的に呼び出されるシェル) から実行するプログラム (プロセスまたは子プロセス) に適用されます。
- ホーム・ディレクトリーにある .profile ファイル。 .profile ファイル内の環境変数を定義すると、それらの変数の値は Linux セッションを開始するときに必ず自動的に定義され、その値がすべてのシェル・プロセスに適用されます。

また、putenv() POSIX 関数を使用して COBOL プログラム内から環境変数を設定したり、getenv() POSIX 関数を使用して環境変数にアクセスすることもできます。

一部の環境変数 (COBPATH や NLSPATH など) は、ファイル検索を行うディレクトリーを定義します。 複数のディレクトリー・パスがリストされる場合は、各パスがコロンで区切られます。 環境変数で定義されたパスは、`export` コマンド内の最初のパスから最後のパスへの順番に評価されます。 環境変数のパス内で同じ名前のファイルが複数定義されている場合は、最初に見つかったファイルのコピーが使用されます。

例えば、次の `export` コマンドを発行した場合、2 つのディレクトリーを含むように COBPATH 環境変数 (この環境変数は、動的にアクセスされるプログラムを COBOL ランタイムが検索できる場所を定義します) が設定されます。 これらのディレクトリーのうち、最初に指定されたディレクトリーが先に検索されます。

```
export COBPATH=/users/me/bin:/mytools/bin
```

224 ページの『例: 環境変数の設定とアクセス』

© Copyright IBM Corp. 2021

関連タスク
228 ページの『コンパイルの調整』

関連参照
216 ページの『コンパイラー環境変数とランタイム環境変数』
218 ページの『コンパイラー環境変数』
220 ページの『ランタイム環境変数』

コンパイラー環境変数とランタイム環境変数

COBOL for Linux は以下の環境変数を使用します。これらは、コンパイラーおよびランタイムの両方で共通です。

DB2DBDFT
組み込み SQL ステートメントを含む、または Db2 ファイル・システムを使用するプログラムに使用するデータベースを指定します。

DBCS_CODEPAGE
DBCS リテラルおよび DBCS データ項目を含む DBCS データに適用可能な DBCS コード・ページを指定します。

DBCS コード・ページを設定するには、以下のコマンドを発行します。ここで、*codepage* は、International Components for Unicode (ICU) 変換ライブラリーでサポートされている DBCS コード・ページの名前 (IBM-943 や IBM-EUCjp など) です。

```
export DBCS_CODEPAGE=codepage
```

DBCS_CODEPAGE が設定されていない場合は、現行ロケールに関連付けられたデフォルト DBCS コード・ページが使用されます。

LANG
ロケールを指定します (環境変数を使用たてロケールの指定に関する関連タスクを参照)。 後述のように、LANG は NLSPATH 環境変数の値にも影響を与えます。

例えば、次のコマンドは言語ロケール名を米国英語に設定します。

```
export LANG=en_US
```

LC_ALL
ロケールを指定します。LC_ALL を使用するロケール設定は、LANG または その他の LC_*xx* 環境変数を使用する設定よりも優先されます (環境変数を使用したロケールの指定に関する関連タスクを参照)。

LC_COLLATE
ロケールの照合動作を指定します。 LC_ALL が指定されている場合は、この設定がオーバーライドされます。

LC_CTYPE
ロケールのコード・ページを指定します。 LC_ALL が指定されている場合は、この設定がオーバーライドされます。

LC_MESSAGES
ロケールのメッセージの言語を指定します。 LC_ALL が指定されている場合は、この設定がオーバーライドされます。

LC_TIME
日時情報の形式に使用するロケールを決定します。 LC_ALL が指定されている場合は、この設定がオーバーライドされます。

LD_LIBRARY_PATH
共有ライブラリーおよび EXIT コンパイラー・オプションで識別されるユーザー定義のコンパイラー出口プログラムの場所として使用されるディレクトリー・パスを指定します。

NLSPATH
メッセージ・カタログとヘルプ・ファイルの絶対パス名を指定し、*directory_name*/%L/%N という形式を使用します。%L には、LANG 環境変数で指定された値が代入されます。%N には、メッセージ・カタログ名が代入されます。

COBOL for Linux はコンパイラー・メッセージ・カタログを /opt/ibm/cobol/1.1.0/usr/share/locale/*xx* 内にインストールし、ランタイム・メッセージ・カタログを /opt/ibm/cobol/rte/usr/share/locale/*xx* にインストールします。*xx* は、COBOL for Linux でサポートされている言語です。デフォルトは en_US です。

NLSPATH を設定する際には、値を置き換えるのではなく NLSPATH に値を追加してください。 この環境変数は、他のプログラムが使用する可能性があります。 次に例を示します。

```
DIR=xxxx
NLSPATH=$DIR/%L/%N:$NLSPATH
export NLSPATH
```

xxxx は、COBOL がインストールされているディレクトリーです。 ディレクトリー *xxxx* には、COBOL メッセージ・カタログを含むディレクトリー *xxxx*/en_US (英語の言語設定の場合) が含まれている必要があります。

この製品には、次の言語によるメッセージが組み込まれています。

en_US
英語

ja_JP
日本語

メッセージの言語とロケール設定の言語は別に指定することができます。 例えば、環境変数 LANG を en_US に設定し、環境変数 LC_ALL を ja_JP.eucjp に設定することが可能です。 この例では、COBOL コンパイラーまたはランタイム・メッセージはすべて英語で処理され、プログラム内のネイティブ ASCII (DISPLAY または DISPLAY-1) データはコード・ページ ja_JP.eucjp (日本語 EUC コード・ページ) のエンコードとして処理されます。

コンパイラーは、NLSPATH および LANG 環境変数の値の組み合わせを使用して、 メッセージ・カタログにアクセスします。 NLSPATH が正しく設定されているが、LANG が上記のロケール値のいずれかに設定されていない場合は、警告メッセージが 生成され、コンパイラーは en_US メッセージ・カタログをデフォルトに設定します。 NLSPATH 値が無効の場合、終了エラー・メッセージが生成されます。

ランタイム・ライブラリーも NLSPATH を使用してメッセージ・カタログにアクセスします。 NLSPATH を正しく設定しないと、ランタイム・メッセージが短縮形で戻されます。コンパイラーおよびランタイムは両方とも自動的に NLSPATH を管理します。そのため、自分自身で NLSPATH を扱う必要はありません。

TMPDIR
コンパイラーとランタイムに使用される一時作業ファイルの場所を指定します。この値が設定されない場合、デフォルトは現行ディレクトリーになります。

次に例を示します。

```
export TMPDIR=/tmp
```

TZ
ロケールによって使用される時間帯情報を記述します。 TZ の形式は次のとおりです。

```
export TZ=SSS[+|-]nDDD[,sm,sw,sd,st,em,ew,ed,et,shift]
```

TZ がない場合は、デフォルトのロケール値として EST5EDT が使用されます。 標準時間帯しか指定しない場合は、*n* (GMT からの時間差) のデフォルト値が 5 ではなく 0 になります。

sm、*sw*、*sd*、*st*、*em*、*ew*、*ed*、*et*、*shift* のうち 1 つでも値を指定する場合は、これらすべての値を指定する必要があります。 これらの値のうち 1 つでも無効な場合は、ステートメント全体が無効と見なされ、時間帯情報は変更されません。

例えば、次のステートメントは、標準時間帯を CST、夏時間調整を CDT、CST と UTC の時間差を 6 時間に設定します。 夏時間調整の開始値と終了値は設定されません。

```
export TZ=CST6CDT
```

これ以外に考えられる値としては、Pacific United States を表す PST8PDT や、Mountain United States を表す MST7MDT があります。

関連タスク
203 ページの『環境変数を使用したロケールの指定』

関連参照
204 ページの『サポートされるロケールおよびコード・ページ』
218 ページの『TZ 環境パラメーター変数』

TZ 環境パラメーター変数

TZ 変数値の定義は次のとおりです。

表 *22.* ***TZ*** **環境パラメーター変数**

変数	説明	デフォルト値
SSS	標準時間帯 ID。 アルファベットで始まる 3 文字でなければなりませんが、スペースが含まれていても構いません。	EST
n	標準時間帯と、協定世界時 (UTC) (以前はグリニッジ標準時 (GMT)) との時間差。 正数はグリニッジ子午線より西の時間帯を示します。負数はグリニッジ子午線より東の時間帯を示します。	5
DDD	夏時間調整 (DST) 時間帯 ID。 アルファベットで始まる 3 文字でなければなりませんが、スペースが含まれていても構いません。	EDT
sm	DST の開始月 (1 から 12)	4
sw	DST の開始週 (-4 から 4)	1
sd	DST の開始日 (*sw* が 0 以外の場合は 0 から 6、*sw* が 0 の場合は 1 から 31)	0
st	DST の開始時刻 (秒単位)	3600
em	DST の終了月 (1 から 12)	10
ew	DST の終了週 (-4 から 4)	-1
ed	DST の終了日 (*ew* が 0 以外の場合は 0 から 6、*ew* が 0 の場合は 1 から 31)	0
et	DST の終了時刻 (秒単位)	7200
shift	時間変化量 (秒単位)	3600

コンパイラー環境変数

COBOL for Linux は、以下に示すように、コンパイラーのみの環境変数をいくつか使用します。

COBOL ワードは大/小文字を区別しないため、*library-name* や *text-name* などの COBOL ワードの文字はすべて、大文字として扱われます。 したがって、そのような名前に対応する環境変数名は大文字でなければなりません。 例えば、`COPY MyCopy` に対応する環境変数名は MYCOPY です。

COBCPYEXT
COPY *name* ステートメントにファイル接尾部が指定されていない場合は、コピーブックの検索に使用するためのファイル接尾部を指定します。 のファイル接尾部を 1 つ以上指定します。拡張子の前のピリオドはなくても構いません。 複数のファイル接尾部を指定する場合は、スペースまたはコンマで区切ります。

COBCPYEXT が定義されていない場合は、CPY、CBL、COB (小文字の cpy、cbl、cob も同等) の各接尾部が検索されます。

COBLSTDIR
コンパイラー・リスト・ファイルの書き込み先ディレクトリーを指定します。 有効な任意のパスを指定します。 絶対パスを指定するには、先行スラッシュを指定します。 それ以外の場合は、現行ディレクトリーからの相対パスになります。 末尾のスラッシュはオプションです。

COBLSTDIR が定義されていない場合は、コンパイラー・リストが現行ディレクトリーに書き込まれます。

COBOPT
コンパイラー・オプションを指定します。 複数のコンパイラー・オプションを指定するには、各オブジェクトをスペースまたはコンマで区切ります。 コマンド・シェルに対して意味を持つブランクまたは文字がリストに含まれている場合は、オプション・リストを引用符で囲みます。 以下に例を示します。

```
export COBOPT="TRUNC(OPT) TERMINAL"
```

個々のコンパイラー・オプションにデフォルト値が適用されます。

注 : コンパイラーは、特定のシェル・スクリプト文字を以下のように解釈します。

- 等号 (=) は左括弧 (に解釈されます。
- コロン (:) は右括弧) に解釈されます。
- 下線 (_) は単一引用符 (') に解釈されます。

円記号 (¥) エスケープ文字を追加すれば、このような解釈を防ぐことができるため、文字をストリングで渡すことができます。 円記号 (¥) が (エスケープ文字ではなく) 円記号そのものを表すようにしたい場合は、円記号 (¥) を重ねて (¥¥) 使用します。

library-name
ユーザー定義語として *library-name* を指定した場合は、名前が環境変数として使用され、この環境変数の値がコピーブックの位置を指定するパスに使用されます。

ライブラリー名を指定しない場合は、コンパイラーが次の順序でライブラリー・パスを検索します。

1. 現行ディレクトリー
2. -I*xxx* オプションで指定されたパス (設定されている場合)
3. SYSLIB 環境変数で指定されたパス

ファイルが検出されると、検索は終了します。

詳細については、コンパイラー指示ステートメントに関する関連参照 で COPY ステートメントのドキュメンテーションを参照してください。

SYSLIB
ライブラリー名による修飾なしのテキスト名で、COBOL の COPY ステートメントに使用するパスを指定します。 また、SQL INCLUDE ステートメントに使用するパスも指定します。

text-name
ユーザー定義語として *text-name* を指定した場合は、環境変数の値がファイル名として使用され、またコピーブックのパス名として使用される場合もあります。

複数のパス名を指定するには、各パス名をコロン (:) で区切ります。

詳細については、コンパイラー指示ステートメントに関する関連参照 で COPY ステートメントのドキュメンテーションを参照してください。

関連概念
377 ページの『Db2 コプロセッサー』

関連タスク
378 ページの『Db2 コプロセッサーを用いた SQL INCLUDE の使用』

関連参照
232 ページの『cob2 オプション』
248 ページの『コンパイラー・オプション』
293 ページの『第 14 章 コンパイラー指示ステートメント』

ランタイム環境変数

COBOL ランタイム・ライブラリーでは、次のランタイムのみの環境変数を使用します。

割り当て名
ASSIGN 節で COBOL ファイルの外部ファイル名に指定するユーザー定義語。例えば、次の ASSIGN 節の OUTPUTFILE が割り当て名です。

```
SELECT CARPOOL ASSIGN TO OUTPUTFILE
```

実行時に、環境変数を、COBOL ファイルと関連付けるシステム・ファイルの名前に設定します。 次に例を示します。

```
export OUTPUTFILE=january.car_results
```

上記のコマンドを発行した後に、COBOL ファイル CARPOOL に対して入出力ステートメントを行うと、現行ディレクトリー内のシステム・ファイル january.car_results に対して操作が行われます。

環境変数を設定しない場合、または空のストリングを設定した場合、COBOL は環境変数のリテラル名をシステム・ファイル名として使用します (上記の ASSIGN の例では、現行ディレクトリー内の OUTPUTFILE)。

ASSIGN 節では、標準言語ファイル・システム (STL) またはレコード順次区切りファイル・システム (RSD) など、デフォルト以外のファイル・システムに格納されたファイルを指定できます。 次に例を示します。

```
SELECT CARPOOL ASSIGN TO STL-OUTPUTFILE
```

この場合でも、環境変数 OUTPUTFILE (STL-OUTPUTFILE ではなく) を設定します。

CICS_TK_SFS_SERVER
完全修飾 CICS SFS サーバー名を指定します。 以下に例を示します。

```
export CICS_TK_SFS_SERVER=/.:/cics/sfs/sfsServer
```

COBPATH
COBOL ランタイムが共用ライブラリーなどの動的にアクセスされるプログラムを探すためのディレクトリー・パスを指定します。COBPATH が最初に探索され (設定していない場合、デフォルトでは現行ディレクトリー ("./") に設定される)、続けて LD_LIBRARY_PATH が探索されます。

例えば、次のようになっている場合を考えます。

```
export COBPATH=/pgmpath/pgmshlib
```

COBRTOPT
COBOL ランタイム・オプションを指定します。

ランタイム・オプションが複数ある場合は、コンマまたはコロンで区切ります。 サブオプションの区切り文字には、括弧または等号 (=) を使用します。 オプションは大/小文字の区別をしません。 例えば、次の 2 つのコマンドは同じです。

```
export COBRTOPT="CHECK(ON):UPSI(00000000)"
export COBRTOPT=check=on,upsi=00000000
```

特定のランタイム・オプションに複数の設定を指定すると、右端の設定が有効になります。

各ランタイム・オプションにはデフォルトが適用されます。 詳細については、 ランタイム・オプションに関する関連参照をご覧ください。

EBCDIC_CODEPAGE
CHAR(EBCDIC) または CHAR(S390) コンパイラー・オプションを使用してコンパイルされるプログラムが処理する EBCDIC データに適用できる EBCDIC コード・ページを指定します。

EBCDIC コード・ページを設定するには、次のコマンドを発行します。この場合、*codepage* には使用するコード・ページの名前が入ります。

```
export EBCDIC_CODEPAGE=codepage
```

EBCDIC_CODEPAGE が設定されていない場合は、 サポートされるロケールとコード・ページに関する関連参照に示されているように、 現在のロケールに基づいてデフォルトの EBCDIC コード・ページが選択されます。 CHAR(EBCDIC) コンパイラー・オプションが有効で、 有効なロケールに対して複数の EBCDIC コード・ページを適用できる場合は、 ロケールのデフォルト EBCDIC コード・ページが受け入れ可能でなければ、 EBCDIC_CODEPAGE 環境変数を設定する必要があります。

CICS_SFS_DATA_VOLUME
SFS ファイルが作成される SFS データ・ボリュームの名前を指定します。 以下に例を示します。

```
export CICS_SFS_DATA_VOLUME=sfs_SFS_SERV
```

このデータ・ボリュームは、アプリケーションがアクセスする SFS サーバーに対して定義されている必要があります。

この変数が設定されていない場合は、デフォルト名 sfs_SSFS_SERVER が使用されます。

CICS_SFS_INDEX_VOLUME
代替索引ファイルが作成される SFS データ・ボリュームの名前を指定します。 以下に例を示します。

```
export CICS_SFS_INDEX_VOLUME=sfs_SFS_SERV
```

このデータ・ボリュームは、アプリケーションがアクセスする SFS サーバーに対して定義されている必要があります。

この変数が設定されていない場合は、代替索引ファイルが、対応するベース索引ファイルと同じデータ・ボリューム上に作成されます。

CICS_VSAM_AUTO_FLUSH
CICS ファイルに対する入出力操作ごとのすべての変更が、制御がアプリケーションに戻される前にディスクにコミットされるかどうか (つまり、SFS の強制操作機能を使用可能にするかどうか) を指定します。 この環境変数に OFF を指定することによって、SFS ファイルを使用するアプリケーションのパフォーマンスを向上させることができます。 以下に例を示します。

```
export CICS_VSAM_AUTO_FLUSH=OFF
```

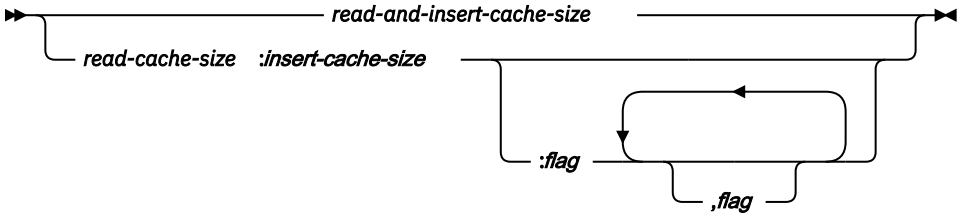
この環境変数が OFF に設定されている場合、SFS は遅延書き込み方針を使用します。つまり、SFS ファイルに対する変更は、ファイルがクローズされるまではディスクに書き込まれません。

SFS のクライアント側のキャッシュが有効な場合 (つまり、環境変数 CICS_VSAM_CACHE がゼロ以外の有効な値に設定されている場合)、CICS_VSAM_AUTO_FLUSH の設定は無視されます。 強制操作は使用不可です。

SFS のクライアント側のキャッシュが有効でない場合、CICS_VSAM_AUTO_FLUSH の値はデフォルトの ON に設定されます。

CICS_VSAM_CACHE
SFS ファイルに対するクライアント側のキャッシュが使用可能かどうかを指定します。 クライアント側のキャッシュを使用可能にすることによって、SFS ファイルを使用するアプリケーションのパフォーマンスを向上させることができます。

CICS_VSAM_CACHE の構文



サイズの単位はページ数です。 ゼロのサイズは、キャッシュが使用不可であることを示します。 指定できるフラグは次のとおりです。

ALLOW_DIRTY_READS
アクセスするファイルがロックされている必要があるという読み取りキャッシュの制限を取り除きます。

INSERTS_DESPITE_UNIQUE_INDICES
クラスター・ファイルのすべてのアクティブな索引と、入力順および相対ファイルのアクティブな代替索引が、重複を許可する場合のみ挿入をキャッシュするという、挿入キャッシュの制限を取り除きます。

例えば、次のコマンドは読み取りキャッシュのサイズを 16 ページに設定し、挿入キャッシュのサイズを 64 ページに設定します。 また、ダーティー読み取りを可能にし、ユニーク索引であっても挿入を可能にします。

```
export CICS_VSAM_CACHE=16:64:ALLOW_DIRTY_READS, \
    INSERTS_DESPITE_UNIQUE_INDICES
```

デフォルトでは、クライアント側のキャッシュは使用不可です。

CICS_SFS_CACHE_<*filename*>
特定の SFS ファイルに関するクライアント側のキャッシュが使用可能かどうかを指定します。 <*filename*> は SFS ファイル名に置き換えることができます。ファイル名に (.) 文字が含まれている場合は、(_) (下線) に置き換える必要があります。この設定は、特定のファイルに対して行われるファイル操作 (読み取りまたは書き込み) に基づくクライアント側のキャッシュを使用可能にするのに役立ちます。CICS_SFS_CACHE 設定と同じ構文仕様を使用して CICS_SFS_CACHE_<*filename*> を指定できます。

例えば、SFS 書き込み操作に常に使用される、VSAM.FILE1.TESTFILE という名前のファイルに関するファイル固有のクライアント側のキャッシュを使用可能にするには、WRITE 操作キャッシュのみを使用可能にするように環境を指定できます。

```
export CICS_SFS_CACHE_FILE1_TESTFILE=0:512
```

CICS_SFS_RDM_CACHE
ランダム読み取り操作に使用される SFS ファイルに関するクライアント側のキャッシュが使用不可かどうかを指定します。ランダム読み取り操作用に開かれるすべてのファイルを対象にクライアント側の読み取り操作キャッシュを使用不可にするには、環境値を 0 として指定します。

```
export CICS_SFS_RDM_CACHE=0
```

CICS_SFS_CACHE 設定はすべての SFS ファイルに適用できるグローバル設定なので、ランダム読み取り操作用に開かれるファイルを対象に読み取り操作キャッシュを設定することはお勧めできません。CICS_VSAM_RDM_CACHE 設定を使用して、SFS のクライアント側の読み取り操作キャッシュを使用不可にすることができます。

CICS_SFS_PREALLOC_<*filename*>

プログラムにより SFS サーバー上で初めてファイルが作成される際に、そのファイルに事前に割り振られるページ数を指定します。<*filename*> は SFS ファイル名に置き換えることができます。ファイル名に (.) 文字が含まれている場合は、(_) (下線) に置き換える必要があります。例えば、TESTFILE という名前の SFS ファイルを作成し、2000 ページを事前割り振りするには、以下の環境変数を設定します。

```
export CICS_SFS_PREALLOC_TESTFILE=2000
```

COBCORE
ランタイムが生成するコア・ファイルを置く位置を指定します。指定しなかった場合、デフォルトは現在の作業ディレクトリーです。名前がパーセント文字 (%) で終わる場合、ファイル名はプログラム名とタイム・スタンプから作成されます。

COBOUTDIR
変数にパスが含まれていない場合、COBOL DISPLAY ステートメント (SYSOUT、CONSOLE、SYSPUNCH) によって作成されるすべてのファイルがあるディレクトリー、およびコア・ファイル (COBCORE) が作成されるディレクトリーを指定します。

PATH
実行可能プログラムのディレクトリー・パスを指定します。

SYSIN、 SYSIPT、 SYSOUT、 SYSLIST、 SYSLST、 CONSOLE、 SYSPUNCH、 SYSPCH
これらの COBOL 環境名は、ACCEPT および DISPLAY ステートメントで使用される簡略名に対応する環境変数名として使用されます。

環境変数を設定しない場合、デフォルトでは、SYSIN および SYSIPT が論理入力装置 (キーボード) に割り当てられます。 SYSOUT、SYSLIST、 SYSLST、CONSOLE は、システムの論理出力装置 (画面) に割り当てられます。 SYSPUNCH および SYSPCH には、デフォルトでは値が割り当てられません。これらは明示的に定義しない限り無効です。これらの変数の値の末尾がパーセント文字 (%) の場合、ファイル名はプログラム名とタイム・スタンプから作成されます。

例えば、次のコマンドは CONSOLE を定義します。

```
export CONSOLE=/users/mypath/myfile
```

CONSOLE は、次のソース・コードとともに使用される場合があります。

```
SPECIAL-NAMES.
    CONSOLE IS terminal
    . . .
    DISPLAY 'Hello World' UPON terminal
```

関連タスク
113 ページの『ファイルの識別』
146 ページの『SFS ファイルの使用』
149 ページの『SFS パフォーマンスの向上』

関連参照
256 ページの『CHAR』
299 ページの『第 15 章 ランタイム・オプション』

例: 環境変数の設定とアクセス

次の例は、標準の POSIX 関数 getenv() および putenv() を呼び出すことによって、COBOL プログラムから環境変数に アクセスする方法と環境変数を設定する方法を示しています。

getenv() と putenv() は C 関数なので、BY VALUE によって引数を渡さなければなりません。 文字ストリングは、ヌル終了ストリングを指し示す BY VALUE ポインターとして渡さなければなりません。 これらの関数を呼び出すプログラムは、NODYNAM オプションと PGMNAME(LONGMIXED) オプションを指定してコンパイルします。

```
 CBL pgmname(longmixed),nodynam
 Identification division.
 Program-id. "envdemo".
 Data division.
 Working-storage section.
 01 P pointer.
 01 PATH pic x(5) value Z"PATH".
 01 var-ptr pointer.
 01 var-len pic 9(4) binary.
 01 putenv-arg pic x(14) value Z"MYVAR=ABCDEFG".
 01 rc pic 9(9) binary.
 Linkage section.
 01 var pic x(5000).
 Procedure division.
* Retrieve and display the PATH environment variable
     Set P to address of PATH
     Call "getenv" using by value P returning var-ptr
     If var-ptr = null then
         Display "PATH not set"
     Else
         Set address of var to var-ptr
         Move 0 to var-len
         Inspect var tallying var-len
           for characters before initial X"00"
         Display "PATH = " var(1:var-len)
     End-if
* Set environment variable MYVAR to ABCDEFG
     Set P to address of putenv-arg
     Call "putenv" using by value P returning rc
     If rc not = 0 then
         Display "putenv failed"
         Stop run
     End-if
     Goback.
```

プログラムのコンパイル

COBOL プログラムは、コマンド行から、またはシェル・スクリプトまたは Make ファイルを使用してコンパイルできます。

IBM 以外または自由形式の COBOL ソースの場合は、最初にソース変換ユーティリティー scu を使用して、コンパイルできるようにソースを変換する必要がある場合があります。scu 機能の要約を参照するには、コマンド scu -h を入力してください。 詳しくは、scu の man ページを参照するか、またはソース変換ユーティリティーに関する関連参照 を参照してください。

コンパイラー・オプションの指定: COBOL プログラムをコンパイルする際に使用するオプションは、次の方法で指定することができます。 例えば、次のようなことが可能です。

- コマンド行で COBOPT 環境変数を設定する。
- コンパイラー・オプションを cob2 コマンドへのオプションとして指定します。 このコマンドは、コマンド行、シェル・スクリプト、または Make ファイルで使用できます。
- PROCESS (CBL) または *CONTROL ステートメントを使用する。 PROCESS を使用して指定するオプションは他のオプション指定をオーバーライドします。

コンパイラー・オプションの設定と、設定方法の相対的な優先順位について詳しくは、矛盾するコンパイラー・オプションに関する関連参照を参照してください。

関連タスク

225 ページの『コマンド行からのコンパイル』
226 ページの『シェル・スクリプトを使用したコンパイル』
226 ページの『PROCESS (CBL) ステートメントでのコンパイラー・オプションの指定』
227 ページの『デフォルトのコンパイラー構成の変更』
425 ページの『第 21 章 プラットフォーム間でのアプリケーションの移植』

関連参照

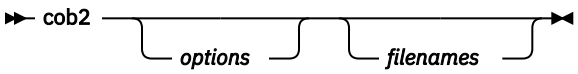
218 ページの『コンパイラー環境変数』
232 ページの『cob2 オプション』
248 ページの『コンパイラー・オプション』
250 ページの『矛盾するコンパイラー・オプション』

521 ページの『付録 A IBM Enterprise COBOL for z/OS との違いのまとめ』
参照形式 (*COBOL for Linux on x86* 言語解説書)
ソース変換ユーティリティー (scu) (*COBOL for Linux on x86* 言語解説書)

コマンド行からのコンパイル

コマンド行から COBOL プログラムをコンパイルするには、そのプログラムのタイプに 応じて、cob2 コマンドこのコマンドは、リンカーも呼び出します。

cob2 コマンドの構文



複数のファイルをコンパイルするには、コマンド行の任意の位置でファイル名を指定します。オプションおよびファイル名を区切るには、スペースを使用します。 例えば、次の 2 つのコマンドは同じです。

```
cob2 -g filea.cbl fileb.cbl -q"flag(w)"
cob2 filea.cbl -g -q"flag(w)" fileb.cbl
```

cob2 コマンドはそれぞれ、コマンド行に指定する順にコンパイラーとリンカー・オプションを受け入れます。 指定したオプションは、コマンド行のすべてのファイルに適用されます。

サフィックスが .cbl または .cob のソース・ファイルのみがコンパイラーに渡されます。 それ以外のすべてのファイルはリンカーに渡されます。

コンパイラー入出力のデフォルトの位置は、現行ディレクトリーです。

cob2 コマンドはスレッド・セーフです。cob2_r は COBOL for AIX との互換性を保つために準備されていますが、cob2 コマンドと同一のデフォルト・オプションを使用します。デフォルト・オプションについて詳しくは、227 ページの『デフォルトのコンパイラー構成の変更』 を参照してください。

注 : コンパイラーは、特定のシェル・スクリプト文字を以下のように解釈します。

- 等号 (=) は左括弧 (に解釈されます。
- コロン (:) は右括弧) に解釈されます。
- 下線 (_) は単一引用符 (') に解釈されます。

円記号 (¥) エスケープ文字を追加すれば、このような解釈を防ぐことができるため、文字をストリングで渡すことができます。 円記号 (¥) が (エスケープ文字ではなく) 円記号そのものを表すようにしたい場合は、円記号 (¥) を重ねて (¥¥) 使用します。

226 ページの『例: コンパイルでの cob2 の使用』

関連タスク
227 ページの『デフォルトのコンパイラー構成の変更』
234 ページの『プログラムのリンク』

関連参照
232 ページの『cob2 オプション』
248 ページの『コンパイラー・オプション』

例: コンパイルでの cob2 の使用

次の例に、さまざまな cob2 呼び出しで生成される出力を示します。

表 *23.* ***cob2*** **コマンドからの出力**

コンパイル対象	入力	生成されるファイル
alpha.cbl	`cob2 -c alpha.cbl`	alpha.o
alpha.cbl および beta.cbl	`cob2 -c alpha.cbl beta.cbl`	alpha.o, beta.o
alpha.cbl (LIST および ADATA オプションを使用)	`cob2 -qlist,adata alpha.cbl`	alpha.wlist、alpha.o[1]、alpha.lst、 alpha.adt、a.out

1. リンクが成功すると、これらのファイルは削除されます。

235 ページの『例: リンクでの cob2 の使用』

関連タスク
225 ページの『コマンド行からのコンパイル』

関連参照

251 ページの『ADATA』
270 ページの『LIST』

シェル・スクリプトを使用したコンパイル

シェル・スクリプトを使用すれば、the cob2 コマンドを自動化できます。

無効な構文がコマンドに渡されないようにするには、次の指針に従います。

- コンパイラー・サブオプションを区切るには、括弧ではなく等号とコロンを使用してください。
- コンパイラー・サブオプションを区切るには、単一引用符ではなく下線を使用してください。
- オプション・ストリングを引用符 (" ") で囲む場合を除き、ストリング内にはブランクを使用しないでください。

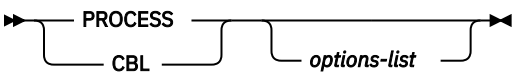
表 *24.* **シェル・スクリプト内のコンパイラー・オプション構文の例**

シェル・スクリプトでの使用	コマンド行での使用
`-qFLOAT=NATIVE:,CHAR=NATIVE:`	`-qFLOAT(NATIVE),CHAR(NATIVE)`
`-qEXIT=INEXIT=_String_,MYMODULE::`	`-qEXIT(INEXIT('String',MYMODULE))`

PROCESS (CBL) ステートメントでのコンパイラー・オプションの指定

COBOL プログラム内では、ほとんどのコンパイラー・オプションを PROCESS (CBL) ステートメント内にコーディングできます。 このステートメントは、IDENTIFICATION DIVISION ヘッダーの前、かつコメント行またはコンパイラー指示ステートメントの前にコーディングする必要があります。

PROCESS(CBL) ステートメントの構文



シーケンス・フィールドを使用しない場合、PROCESS ステートメントは 1 桁目以降から開始できます。 シーケンス・フィールドを使用する場合、シーケンス番号は 1 桁目で始まり、6 文字が含まれる必要があります。最初の文字は数字でなければなりません。 シーケンス・フィールドを付けて使用する場合には、PROCESS は 8 桁目以降で開始できます。

CBL を PROCESS の同義語として使用することができます。 CBL は、シーケンス・フィールドを使用しない場合、同様に 1 桁目以降で開始できます。 シーケンス・フィールドを付けて使用する場合には、CBL は 8 桁目以降で開始できます。

PROCESS および CBL ステートメントは、 プログラムで固定ソース形式を使用する場合は 72 桁目以前で、プログラムで拡張ソース形式 (Extended Source) を使用する場合は 252 桁目以前で終了させる必要があります。

PROCESS または CBL ステートメントと *options-list* の最初のオプションとは、1 つ以上のブランクで区切らなければなりません。 オプションはコンマまたはブランクで区切ります。 個々のオプションとそのサブオプションの間にスペースを入れてはなりません。

複数の PROCESS または CBL ステートメントをコーディングすることができます。 そうする場合は、間にステートメントを入れずに、ステートメントを続けて使用する必要があります。 複数の PROCESS または CBL ステートメントにわたってオプションを継続することはできません。

関連参照
283 ページの『SRCFORMAT』
参照形式 (*COBOL for Linux on x86* 言語解説書)
CBL (PROCESS) ステートメント (*COBOL for Linux on x86* 言語解説書)

デフォルトのコンパイラー構成の変更

cob2 コマンドで使用するデフォルト・オプションは、構成ファイル (デフォルトでは /opt/ibm/cobol/1.1.0/etc/cob2.cfg) から取得されます。cob2 で使用するオプションは、そのコマンドに -# オプションを指定することによって表示できます。

デフォルトの構成ファイルを使用している場合、コマンド cob2 -# abc.cbl によって次のような出力が表示されます。

```
exec: /opt/ibm/cobol/1.1.0/usr/bin/cob3 abc.cbl
exec: /usr/bin/gcc -m32 -shared -fPIC -rdynamic -fasynchronous-unwind-tables -Wl,--hash-
style=gnu
-Wl,--export-dynam -Wl,-Bsymbolic-functions -Wl,--build-id -Wl,--enable-new-dtags -Wl,-zrelro
-Wl,-znow -Wl,-zdefs -Wl,-z,noexecstack -Wl,--allow-shlib-undefined -pie -Wl,--as-needed abc.o
-Wl,--as-needed -L/opt/ibm/cobol/1.1.0/usr/lib/ -L/opt/ibm/cobol/rte/usr/lib/ -lcob2_32s
-lcob2_32r
-ldfp_32r -lm -lpthread -ldl -Wl,-rpath,/opt/ibm/cobol/rte/usr/lib/:
/opt/ibm/cobol/rte/:/opt/ibm/cobol/1.1.0/usr/lib/:/opt/ibm/cics/lib
```

cob2.cfg 構成ファイルを変更して、デフォルト・オプションを変更したり。

デフォルトの構成ファイルを変更する代わりに、目的に合うようにファイルのコピーを調整することができます。

注 : コンパイラーは、特定のシェル・スクリプト文字を以下のように解釈します。

- 等号 (=) は左括弧 (に解釈されます。
- コロン (:) は右括弧) に解釈されます。
- 下線 (_) は単一引用符 (') に解釈されます。

円記号 (¥) エスケープ文字を追加すれば、このような解釈を防ぐことができるため、文字をストリングで渡すことができます。 円記号 (¥) が (エスケープ文字ではなく) 円記号そのものを表すようにしたい場合は、円記号 (¥) を重ねて (¥¥) 使用します。

関連タスク
228 ページの『コンパイルの調整』

関連参照
232 ページの『cob2 オプション』
229 ページの『構成ファイル内のスタンザ』

コンパイルの調整

要件に合わせてコンパイルを調整するために、デフォルト構成ファイルの コピーを変更して、それをさまざまな方法で使用することができます。

構成ファイル /opt/ibm/cobol/1.1.0/etc/cob2.cfg には、cob2: で始まるセクション (スタンザ) があります。 これらのスタンザをリストするには、コマンド ls /opt/ibm/cobol/1.1.0/usr/bin/cob2*を使用します。 結果のリストは次の行を示しています。

```
/opt/ibm/cobol/1.1.0/usr/bin/cob2
/opt/ibm/cobol/1.1.0/usr/bin/cob2_r
/opt/ibm/cobol/1.1.0/usr/bin/cob2_cics
/opt/ibm/cobol/1.1.0/usr/bin/cob2_db2
/opt/ibm/cobol/1.1.0/usr/bin/cob2_oracle
```

コマンド cob2 および cob2_r は、同じモジュールを実行します。 ただし、cob2 は cob2 スタンザを使用し、cob2_r は cob2_r スタンザを使用します。

以下のステップを実行すれば、コンパイルを調整できます。

1. デフォルト構成ファイル /opt/ibm/cobol/1.1.0/etc/cob2.cfg のコピーを作成します。
2. 特定のコンパイル要件またはその他の COBOL コンパイル環境をサポートするよう、コピーを変更する。
3. (オプション) -# オプションを付けて cob2 コマンドを発行して、変更の影響を表示します。
4. /opt/ibm/cobol/1.1.0/etc/cob2.cfg の代わりにコピーを使用する。

コンパイラーを呼び出すたびに cob2.cfg のコピーを使用するには、etc.d ディレクトリーにあるその名前のシンボリック・リンクを変更して、コピーを指すようにします。 コンパイラーは、このシンボリック・リンクが指す構成ファイルを自動的に読み取ります。

特定のコンパイル用に変更した構成ファイルのコピーを、選択して使用するようにするには、cob2 コマンドに -F オプションを付けて発行します。例えば、myfile.cbl をコンパイルするために、構成ファイルとして /opt/ibm/cobol/1.1.0/etc/cob2.cfg ではなく /u/myhome/myconfig.cfg を使用するには、次のコマンドを発行します。

```
cob2 myfile.cbl -F/u/myhome/myconfig.cfg
```

mycob2 など、独自のスタンザを追加する場合、-F オプションを使用すれば、その独自のスタンザを指定できます。

```
cob2 myfile.cbl -F/u/myhome/myconfig.cfg:mycob2
```

または、mycob2 コマンドを定義することができます。

```
ln -s /usr/bin/cob2 /u/myhome/mycob2
mycob2 myfile.cbl -F/u/myhome/myconfig
```

ln コマンドで指定するディレクトリー (上記の /u/myhome など) はすべて、PATH 環境変数内になければなりません。

各スタンザ内の属性リストについては、スタンザに関する関連参照を参照してください。

関連タスク
215 ページの『環境変数の設定』

関連参照
232 ページの『cob2 オプション』
229 ページの『構成ファイル内のスタンザ』

構成ファイル内のスタンザ

次の表に示すように、構成ファイル内のスタンザには、複数ある属性のうちのいずれかを含めることができます。

表 *25.* **スタンザ属性**

属性	説明
compopts	コンマまたはスペースで区切られた、複数のコンパイラー・オプションのストリング。各オプションの前に -q フラグを付けるか、またはストリング全体を引用符で囲んで、その前に -q フラグを付けます。 コンマを含むオプション値がある場合は、そのオプションを引用符で囲む必要があります。
coprocessor	CICS または Db2 コプロセッサー・ライブラリーのパス。
runlib2, runlib2_64	CICS、Db2、または Oracle で作業している場合に、リンクする追加のランタイム・ライブラリー。
use	ローカル・スタンザとは別に、属性をとるスタンザ。 単一値の属性では、ローカル、つまりデフォルトのスタンザが提供されていない場合は、use スタンザ内の値が適用されます。 コンマで区切られたリストの場合、 use スタンザからの値がローカル・スタンザからの値に追加されます。

関連タスク
227 ページの『デフォルトのコンパイラー構成の変更』
228 ページの『コンパイルの調整』

関連参照

232 ページの『cob2 オプション』

ソース・プログラムのエラーの訂正

ソース・コード・エラーに関するメッセージは、そのエラーが発生した場所 (LINEID) を示します。 メッセージのテキストは、どんな問題であるかを知らせます。 この情報を使用して、ソース・プログラムを訂正することができます。

エラーを訂正するとしても、必ずしもすべての診断メッセージのソース・コードを修正する必要があるとは限りません。 警告レベルまたは通知レベルのメッセージは、プログラムの中に残っていても支障はなく、そのメッセージを除去するために、再コーディングやコンパイルを行う必要はありません。 ただし、重大レベルやエラー・レベルのエラーは、プログラム障害の可能性が大きいため、訂正しなければなりません。

低い方の 4 つのレベルの重大度とは対照的に、回復不能 (U レベル) エラーは、ソース・プログラムの間違いの結果として生じたものではない場合があります。 コンパイラー自体あるいはオペレーティング・システム中の欠陥から生じる可能性があります。 この場合は、コンパイラーが強制的に早期終了させられ、完全なオブジェクト・コードまたは完全なリストを作成しないため、問題を解決するしかありません。 多くの S レベルの構文エラーを持つプログラムに関するメッセージが発生した場合には、これらのエラーを訂正し、プログラムを再度コンパイルしてください。 また、コンパイル・ジョブに変更を加えることによって、ジョブ・セットアップの問題 (ファイル定義の欠落やコンパイラー処理用のストレージの不足など) を解決することが可能です。 コンパイル・ジョブ・セットアップが正しく、S レベルの構文エラーを訂正した場合は、IBM に連絡して他の U レベル・エラーの調査を要求してください。

ソース・プログラムのエラーを訂正した後で、プログラムを再コンパイルしてください。 この 2 回目のコンパイルが成功した場合は、リンク・エディット・ステップに進んでください。 コンパイラーによってまだ問題が検出される場合は、通知メッセージだけが戻されるようになるまで、上記の手順を繰り返さなければなりません。

関連タスク
231 ページの『コンパイラー・メッセージのリストの生成』
234 ページの『プログラムのリンク』

関連参照

231 ページの『コンパイラー検出エラーに関するメッセージおよびリスト』

コンパイラー診断メッセージの重大度コード

コンパイラーが検出できる条件は、5 つの重大度のレベルまたはカテゴリーに分けられます。

表 *26.* **コンパイラー診断メッセージの重大度コード**

メッセージのレベルまたはカテゴリー	戻りコード	目的
通知 (I)	0	通知にすぎません。アクションを取る必要はありません。プログラムは正しく実行されます。
警告 (W)	4	エラーの可能性があることを示します。プログラムはおそらく、書かれたとおりに正しく実行されます。
エラー (E)	8	明確にエラーである条件があることを意味します。 コンパイラーはエラーの訂正を試みましたが、プログラムの実行結果は予期したものではない可能性があります。 エラーを訂正しなければなりません。
重大 (S)	12	重大なエラーを示す条件があることを意味します。 コンパイラーはエラーを訂正できませんでした。 プログラムは正しく実行されず、また、実行を試みてはなりません。 オブジェクト・コードは作成されない可能性があります。
回復不能 (U)	16	コンパイルが終了するほどの重大なエラー条件があることを示します。

コンパイル終了時の最終の戻りコードは、通常、コンパイル中のメッセージで最も高い戻りコードになります。

コンパイラー診断メッセージを抑制したり、その重大度を変更したりすることができますが、 これは、最終のコンパイル戻りコードに影響する可能性があります。 詳細については、関連情報を参照してください。

関連タスク
594 ページの『コンパイラー・メッセージの重大度のカスタマイズ』

関連参照
593 ページの『MSGEXIT の処理』

コンパイラー・メッセージのリストの生成

ERRMSG というプログラム名を持つプログラムをコンパイルすることで、 コンパイラー診断メッセージとそのメッセージ番号、重大度、およびテキストの全リストを生成することができます。

次に示すように、PROGRAM-ID 段落のみをコーディングして、その他のプログラムを省略することができます。

```
Identification Division.
Program-ID. ErrMsg.
```

関連タスク
594 ページの『コンパイラー・メッセージの重大度のカスタマイズ』

関連参照
231 ページの『コンパイラー検出エラーに関するメッセージおよびリスト』
231 ページの『コンパイラー診断メッセージの形式』

コンパイラー検出エラーに関するメッセージおよびリスト

コンパイラーはソース・プログラムを処理するときに、COBOL 言語を調べてエラーがないかどうかを検査し、診断メッセージを発行します。 これらのメッセージは、コンパイラー・リスト内では (FLAG オプションに従って) 照合されます。

コンパイラー・リスト・ファイルはコンパイラー・ソース・ファイルと同じ名前ですが、サフィックス .lst が付きます。 例えば、myfile.cbl のリスト・ファイルは myfile.lst となります。 リスト・ファイルは、cob2 コマンドを発行したディレクトリーに書き込まれます。

リストに示されたメッセージには、問題の性質、重大度、およびその問題が検出されたコンパイラー・フェーズについての情報が記載されます。 可能な限り、メッセージはエラーを訂正するための具体的な指示を与えます。

コンパイラー・オプション、CBL および PROCESS ステートメント、および BASIS、COPY、または REPLACE ステートメントの処理中に検出されたエラーに関するメッセージは、リストの上部近くに表示されます。

(行番号で順序付けされた) コンパイル・エラーに関するメッセージは、プログラムごとにリストの終わり近くに表示されます。

コンパイル中に検出されたすべての問題の要約は、リストの下部に表示されます。

関連タスク
229 ページの『ソース・プログラムのエラーの訂正』
231 ページの『コンパイラー・メッセージのリストの生成』

関連参照
231 ページの『コンパイラー診断メッセージの形式』
230 ページの『コンパイラー診断メッセージの重大度コード』
267 ページの『FLAG』

コンパイラー診断メッセージの形式

コンパイラーが発行する各メッセージには、ソース行番号、メッセージ ID、およびメッセージ・テキストが含まれます。

各メッセージの形式は次のとおりです。

```
nnnnnn IGYppxxxx-l message-text
```

nnnnnn
コンパイラーが処理している最後の行のソース・ステートメントの番号。 ソース・ステートメント番号は、プログラムのソース印刷出力にリストされます。 コンパイル時に NUMBER オプションを指定す

ると、番号は元のソース・プログラム番号になります。 NONUMBER を指定すると、番号はコンパイラーによって生成された番号になります。

IGY
COBOL コンパイラーがメッセージを発行したことを識別する接頭部。

pp
メッセージの原因となった条件が、コンパイラーのどのフェーズまたはサブフェーズで検出されたかを識別する 2 文字。 アプリケーション・プログラマーはこの情報を無視することができます。 コンパイラー・エラーの疑いがあると診断した場合は、IBM にサポートを依頼してください。

xxxx
メッセージを識別する 4 桁の数字。

l
メッセージの重大度レベルを示す文字 (I、W、E、S、または U)。

message-text
メッセージ・テキスト。エラー・メッセージの場合はエラーの原因となった条件の簡単な説明。

ヒント: FLAG オプションを使用してメッセージを抑止した場合は、プログラム内にさらにエラーがある可能性があることを認識しておいてください。

関連参照
230 ページの『コンパイラー診断メッセージの重大度コード』
267 ページの『FLAG』

cob2 オプション

以下にリストされているオプションは、cob2 呼び出しコマンドに適用されます。

コンパイルに適用されるオプション

-c
プログラムをコンパイルしますが、リンクしません。

-comprc_ok=*n*
コンパイラーからの戻りコードに基づいて動作を制御します。 戻りコードが *n* 以下であれば、コマンドは継続してリンク・ステップに進むか、またはコンパイルの場合のみの場合には、ゼロの戻りコードで終了します。 コンパイラーによって生成された戻りコードが *n* より大きい場合は、コマンドは同じ戻りコードで終了します。

デフォルトは -comprc_ok=4 です。

-host
ホスト COBOL データ表現と言語セマンティクスに対して以下のコンパイラー・オプションを設定します。

- CHAR(EBCDIC)
- COLLSEQ(EBCDIC)
- NCOLLSEQ(BIN)
- FLOAT(S390)

-host オプションは、COBOL プログラムのコマンド行引数の形式を、ポインターの配列から、ストリングの長さを含むハーフワード接頭部がある EBCDIC 文字ストリングに変更します。 追加情報については、コマンド行引数に関する以下の関連タスクを参照してください。

-host オプションを使用するときに、C オブジェクト・ファイル内のメインルーチンをアプリケーションのメインの入り口点にしたい場合は、-cmain リンカー・オプションを以下で説明するように使用する必要があります。

-I*xxx*
library-name も SYSLIB も指定されていない場合に、コピーブックの検索に使用するディレクトリーへのパス *xxx* を追加します。 (このオプションは、小文字の *l* ではなく大文字の *I* です。)

各 -I オプションに使用できるパスは 1 つだけです。 複数のパスを追加するには、-I オプションを必要な数だけ使用してください。 I と *xxx* の間にスペースを入れないでください。

-q*xxx*
コンパイラーにオプションを渡します。*xxx* には任意のコンパイラー・オプションが入ります。 -q と *xxx* の間にスペースを入れないでください。

コンパイラー・オプションまたはサブオプションに括弧が含まれている場合や、一連のオプションが指定されている場合には、各オプションを引用符で囲みます。

複数のオプションを指定する場合は、各オプションをブランクまたはコンマで区切ります。 例えば、次の 2 つのオプション・ストリングは同じです。

```
-qoptiona,optionb
```

```
-q"optiona optionb"
```

シェル・スクリプトを使用して cob2 タスクを自動化する予定の 場合は、-q*xxx* オプションに対して特殊な構文が用意されています。 詳しくは、シェル・スクリプトでのコンパイルに関する関連タスクを参照してください。

リンクに適用されるオプション

-cmain
(-host の指定も行う場合のみ効果があります。) (メインルーチンを含む) C オブジェクト・ファイルを、実行可能ファイル内のメインの入り口点にします。C の場合、メインルーチンは関数名 main() で識別されます。

リンクする C オブジェクト・ファイルに、1 つ以上の COBOL オブジェクト・ファイルを持つメインルーチンが含まれている場合は、-cmain を使用して、C ルーチンをメインの入り口点として指定する必要があります。C メインルーチンを含む実行可能ファイル内のメインの入り口点として、COBOL プログラムを指定することはできません。これを行うと、予測不能な動作が発生し、診断は行われません。

-main:*xxx*
xxx を、リンカーに渡されるファイルのリスト内の最初のファイルにします。 このオプションの目的は、指定したファイルを実行可能ファイル内のメインプログラムにすることです。 *xxx* はオブジェクト・ファイルまたはアーカイブ・ライブラリーを一意的に識別する必要があり、接尾部はそれぞれ .o または .a でなければなりません。

-main が指定されていない場合は、コマンド内に指定した最初のオブジェクト、アーカイブ・ライブラリー、またはソース・ファイルが、リンカーに渡されるファイルのリスト内の最初のファイルです。

-main: *xxx* の構文が無効な場合、または *xxx* がコマンドによって処理されるオブジェクト・ファイルまたはソース・ファイルの名前ではない場合は、コマンドが終了します。

-o *xxx*
実行可能なモジュール *xxx* に名前を付けます。ここで、 *xxx* は任意の名前です。 -o オプションを使用しない場合、実行可能なモジュールの名前はデフォルトの a.out に指定されます。

コンパイルとリンクの両方に適用されるオプション

-F*xxx*
/opt/ibm/cobol/1.1.0/etc/cob2.cfg 構成ファイルに指定されているデフォルトではなく、構成ファイルまたはスタンザとして *xxx* を使用します。*xxx* の形式は以下のいずれかです。

- *configuration_file*:*stanza*
- *configuration_file*

- :*stanza*

-g
デバッガーで使用されるシンボリック情報を生成します。TEST コンパイラー・オプションを設定します。

-q32
32 ビット・オブジェクト・プログラムが生成されることを指定します。 ADDR(32) コンパイラー・オプションを設定します。 -m32 リンカー・オプションを設定します。このオプションは、32 ビットの実行可能モジュールを作成するようにリンカーに指示します。

-q64
このオプションは現在サポートされていません。コンパイラーはこのオプションを受け入れて無視します。

-v
コンパイルおよびリンクのステップを表示し、各ステップを実行します。

-#
コンパイル・ステップとリンク・ステップを表示しますが、それらを実行しません。

-?, ?
cob2 コマンドのヘルプを表示します。

関連タスク
225 ページの『コマンド行からのコンパイル』
226 ページの『シェル・スクリプトを使用したコンパイル』
235 ページの『リンカーへのオプションの引き渡し』
459 ページの『コマンド行引数の使用』

関連参照
218 ページの『コンパイラー環境変数』
252 ページの『ADDR』

プログラムのリンク

指定したオブジェクト・ファイルをリンクして、実行可能ファイルまたは共有オブジェクトを作成するには、リンカーを使用します。

リンカーは、次の方法で起動することができます。 以下のものを使用することができます。

- cob2 コマンド。

  このコマンドは、-c オプションを指定する場合を除き、リンカーを呼び出します。

  cob2 は COBOL マルチスレッド ライブラリーとリンクします。そのため、追加のスレッド・オプションを指定する必要はありません。

  C とのリンク: リンカーは、.o ファイルは受け入れますが、.c ファイルは受け入れません。 C と COBOL のファイルをまとめてリンクしたい場合は、gcc コマンドを使用して、最初に C ソース・ファイル用の .o ファイルを作成します。

- Make ファイル:

  Make ファイルを使用して、プログラムのビルドに必要な一連の操作 (コンパイルやリンクなど) を編成することができます。 Make ファイルでリンカー・ステートメントを使用して、必要な出力の種類を指定することができます。

リンカー・オプションは、上記のいずれかの方法を使用して指定することができます。

235 ページの『例: リンクでの cob2 の使用』

関連タスク
225 ページの『コマンド行からのコンパイル』
235 ページの『リンカーへのオプションの引き渡し』
463 ページの『第 24 章 共用ライブラリーの使用』

関連参照
232 ページの『cob2 オプション』
236 ページの『リンカーの入出力ファイル』
237 ページの『リンカーの検索規則』

リンカーへのオプションの引き渡し

呼び出しコマンド (cob2)または Makefile ステートメントのいずれにおいても、リンカー・オプションを指定できます。

呼び出しコマンドで指定するオプションのうち、このコマンドで認識されないものは、リンカーに渡されます。

呼び出しコマンド・オプションによっては、プログラムのリンクに影響するものがあります。 特定のコマンドのオプション・セットについて詳しくは、そのコマンドの資料を調べてください。

以下の表は、COBOL プログラムのために必要になることが多いと思われるオプションを示しています。 ここに示されているように、各オプションの前にはハイフン (-) を付けます。

表 *27.* **共通リンカー・オプション**

オプション	**目的**
-L*dir*	-l オプションで指定したライブラリーを検索するためのディレクトリーを指定します (デフォルト: /usr/lib)
-l*name*	指定した library-file を検索します。*name* は、ファイル lib*name*.a を選択します

235 ページの『例: リンクでの cob2 の使用』

関連参照
232 ページの『cob2 オプション』
236 ページの『リンカーの入出力ファイル』
237 ページの『リンカーの検索規則』

例: リンクでの cob2 の使用

COBOL 呼び出しコマンドを使用して、プログラムのコンパイルとリンクを両方とも実行することができます。 次の例では、cob2 の使用法を示します。

- 2 つのファイルをコンパイルした後でそれらのファイルをまとめてリンクするとき、-c オプションは使用しないでください。 例えば、alpha.cbl および beta.cbl をコンパイルしてリンクし、a.out を生成するには、次のように入力します。

```
cob2 alpha.cbl beta.cbl
```

  このコマンドは、alpha.o と beta.o を作成し、その後に alpha.o、beta.o、および COBOL ライブラリーをリンクします。 リンク・ステップが成功すると、a.out という名前の実行可能プログラムが作成され、alpha.o と beta.o が削除されます。
- コンパイルしたファイルをライブラリーとリンクするには、次のように入力します。

```
cob2 zog.cbl -lmylib
```

  このコマンドによりリンカーは、検索パス内の各ディレクトリーで最初にライブラリー libmylib.so を検索し、次にアーカイブ・ライブラリー・ファイル libmylib.a を検索します。これは、いずれかが検出されるまで続きます。

- オプションを使用してライブラリーの検索を制限するには、次のように入力します。

```
cob2 zog.cbl  -llib1  -llib2
```

この場合、最初のライブラリー指定を満たすために、リンカーによって各ディレクトリーでライブラリー `liblib1.so` が最初に検索され、次にアーカイブ・ライブラリー・ファイル `liblib1.a` が検索されます (前の例を参照)。 ただし、同時にリンカーは、それらの同じライブラリー内の `liblib2.a` を検索するだけです。

- コンパイルとリンクを別々のステップで実行するには、次のようなコマンドを入力します。

```
cob2 -c file1.cbl                # Produce one object file
cob2 -c file2.cbl file3.cbl      # Or multiple object files
cob2 file1.o file2.o file3.o     # Link with appropriate libraries
```

226 ページの『例: コンパイルでの cob2 の使用』

リンカーの入出力ファイル

リンカーはオブジェクト・ファイルどうしをリンクし、ライブラリー・ファイルが指定されている場合は実行可能出力ファイルを生成します。 実行可能出力は、実行可能プログラム・ファイルか共有オブジェクトのいずれかになります。

リンカー入力

- オプション
- オブジェクト・ファイル (*.o)
- アーカイブ・ライブラリー・ファイル (*.a)
- 動的ライブラリー・ファイル (*.so)

リンカー出力

- 実行可能ファイル (デフォルトでは a.out)
- 共有オブジェクト
- 戻りコード

ライブラリー・ファイル: ライブラリーは、接尾部が .a または .so のファイルです。 ライブラリーを指定するには、絶対または相対パス名を指定するか、 `-l` (小文字の L) オプションを `-l`*name* の形式で使用することができます。 後者の形式は、ファイル lib*name*.a を指定し、動的モードの場合はファイル lib *name*.so を指定して、複数のディレクトリー内で検索されるようにします。 これらの検索ディレクトリーには、 `-L` オプションを使用して指定するディレクトリーと、標準ライブラリー・ディレクトリ /usr/lib および /lib が含まれます。

環境変数 LD_LIBRARY_PATH は、コマンド行でいずれかを明示的に指定するライブラリー (例えば、`libc.a`) の検索や、`-l` オプション (例えば、`-lc`) を使用したライブラリーの検索には使用しません。 `-l` オプションを使用して指定したライブラリーを検索するディレクトリーを示すには、`-L` *dir* オプションを使用する必要があります。

Linux`ar` コマンドを使用して、単一のアーカイブ・ファイル内に 1 つ以上のファイルを結合してライブラリー・ファイルを作成することができます。

464 ページの『例: サンプルの共用ライブラリーの作成』

関連タスク
235 ページの『リンカーへのオプションの引き渡し』

関連参照
237 ページの『リンカーの検索規則』
237 ページの『リンカー・ファイル名のデフォルト』

リンカーの検索規則

リンカーは、オブジェクト・ファイル (.o) またはアーカイブ・ライブラリー・ファイル (.a) を検索するとき、その検索内容を満たすまで複数の場所を探します。

リンカーが検索する場所は次のとおりです。

1. ファイルに対して指定するディレクトリー

   ファイルとともにパスを指定した場合、リンカーはそのパスだけを検索し、そこでファイルが検出されなければリンクを停止します。
2. パスを指定しなかった場合は、現行ディレクトリー
3. 環境変数 LD_LIBRARY_PATH の値 (定義されている場合)

/usr/lib 内のデフォルトのライブラリー以外のライブラリーを使用する場合は、その他のライブラリーの場所を示す -L オプションを 1 つ以上指定することができます。 また、LD_LIBRARY_PATH 環境変数を設定することもできます。これにより、実行時にライブラリーの検索パスを指定することができます。

リンカーがファイルを検出できない場合は、エラー・メッセージを生成してリンクを停止します。

関連タスク
235 ページの『リンカーへのオプションの引き渡し』

関連参照
237 ページの『リンカー・ファイル名のデフォルト』

リンカー・ファイル名のデフォルト

ファイル名を入力しない場合、リンカーはデフォルト名を想定します。

表 *28.* **リンカーで想定されるデフォルトのファイル名**

ファイル	デフォルトのファイル名	デフォルトの接尾部
オブジェクト・ファイル	なし。 少なくとも 1 つのオブジェクト・ファイル名を入力する必要があります。	.o
出力ファイル	a.out	なし
ライブラリー・ファイル	オブジェクト・ファイルで定義されたデフォルトのライブラリー。 デフォルトのライブラリーを定義するには、コンパイラー・オプションを使用します。 追加のライブラリーを指定した場合は、デフォルトのライブラリーの前にそれらが検索されます。	.a または .so

リンク内のエラーの訂正

PGMNAME(UPPER) コンパイラー・オプションを使用すると、CALL ステートメントで参照されるサブプログラムの名前が大文字に変換されます。 リンカーは大/小文字を区別して名前を認識するため、この変換はリンカーに影響します。

例えば Call "RexxStart" は、コンパイラーによって Call "REXXSTART" に変換されます。 呼び出し先プログラムの実名が RexxStart の場合、リンカーはこのプログラムを検出できず、REXXSTART が 未解決の外部参照であることを示すエラー・メッセージを生成します。

このタイプのエラーは一般に、他のソフトウェア・プロダクトに備わっている API ルーチンを呼び出す場合に起こります。 API ルーチンの名前に大文字と小文字が混在している場合は、次の両方の処置を行ってください。

- PGMNAME(MIXED) コンパイラー・オプションを使用する。

- CALL ステートメントで、API ルーチンの名前が正しく (大/小文字が 正確に) 指定されていることを確認する。

プログラムの実行

COBOL プログラムを実行するには、環境変数を設定し、実行可能プログラムを実行するコマンドを発行して、実行時エラーがあればそのエラーを修正します。

1. 必要な環境変数がすべて設定されていることを確認します。

   以下に例を示します。

   - システム・ファイル名に値を割り当てる環境変数名をプログラムが使用する場合は、その環境変数を設定する必要があります。
   - プログラムでランタイム・オプションを使用する場合は、それらの値を COBRTOPT ランタイム環境変数に指定します。
2. プログラムを実行します。コマンド行で、実行可能モジュールの名前を入力するか、 そのモジュールを呼び出すコマンド・ファイルを実行します。 必要なプログラム実引数をコマンド行で含めます。

   例えば、コマンド `cob2 alpha.cbl beta.cbl-o gamma` が成功した場合は、`gamma` を入力してプログラムを実行することができます。
3. ランタイム・エラーを訂正します。

   デバッガーを使用して、エラーの発生時点でプログラム状態を調べることができます。

ヒント: ランタイム・メッセージが省略されているか不完全な場合は、環境変数 LANG または NLSPATH、あるいはその両方の設定が間違っている可能性があります。

関連タスク

215 ページの『環境変数の設定』
311 ページの『IBM Debug for Linux on x86 を使用したデバッグ』
459 ページの『コマンド行引数の使用』

関連参照
299 ページの『第 15 章 ランタイム・オプション』

603 ページの『付録 G ランタイム・メッセージ』

第 13 章 コマンド行でのコンパイラー・オプションの指定

コマンド行で指定されるほとんどのオプションは、オプションのデフォルト設定と、 構成ファイルに設定されたオプションをオーバーライドします。

コマンド行オプションには、次の 2 種類があります。

- フラグ・オプション
- **-q***option_keyword* (コンパイラー特定)

関連概念
239 ページの『フラグ・オプション』
247 ページの『-q オプション』

関連タスク
224 ページの『プログラムのコンパイル』

フラグ・オプション

COBOL for Linux は、Linux システムで使用される多くの共通の標準的フラグ・オプションをサポートします。フラグ・オプションでは、大/小文字が区別されます。 フラグ・オプションは、cob2 呼び出しコマンドに適用されます。

また、COBOL for Linux は、他のプログラミング・ツールおよび ユーティリティー (例えば、scu コマンド) に割り当てられたフラグもサポートします。

フラグ・オプションの中には、 フラグの一部を形成する引数を持つものがあります。 以下に例を示します。

```
cob2 stem.cbl -F/home/tools/test3/new.cfg:cob2
```

ここで、new.cfg はカスタム構成ファイルです。

1 つのストリングで引数を取らないフラグを指定することができます。例えば、次のように指定します。

```
cob2 -ocv file.cbl
```

```
cob2 -o -c -v file.cbl
```

引数を取るフラグ・オプションは単一ストリングの一部として指定することができますが、引数を取るフラグは 1 つしか使用できず、そのフラグは最後に指定されるオプションでなければなりません。例えば、他のフラグと一緒に -o フラグを (実行可能ファイルの名前を指定するために) 使用できるのは、-o オプションとその引数が最後に指定されている場合だけです。 例えば、次のように指定します。

```
cob2 -ocv test test.cbl
```

これは、以下の指定と同じ効果があります。

```
cob2 -o -c -vtest test.cbl
```

ほとんどのフラグ・オプションは 1 文字ですが、中には 2 文字のものもあります。該当する文字の組み合わせを使用する別のオプションがある場合は、単一ストリングで複数のオプションを指定しないように注意してください。

関連概念
239 ページの『第 13 章 コマンド行でのコンパイラー・オプションの指定』
247 ページの『-q オプション』

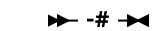
© Copyright IBM Corp. 2021

-# (ポンド記号)

目的

実際にコンパイラー・コンポーネントを呼び出さずに、コマンド行に指定されたコンパイルのステップをプレビューする。 このオプションが使用可能である場合、情報は標準出力に書き込まれ、呼び出されるプリプロセッサー、コンパイラー、およびリンカー内のプログラムの名前とそれぞれのプログラムに指定されるデフォルト・オプションが表示されます。プリプロセッサー、コンパイラー、およびリンカーは呼び出されません。この オプションは、コンパイルとリンクの両方に適用されます。

構文



デフォルト
コンパイラーはコンパイルの進行を表示しません。

USAGE
このコマンドを使用して、特定のコンパイルで呼び出されるコマンドとファイルを判別することができます。これにより、 ソース・コードのコンパイル、および既存のファイル (.lst ファイルなど) の上書きのオーバーヘッドが回避されます。このオプションは -v と同じ情報を表示しますが、コンパイラーは呼び出しません。-# オプションは、-v オプションをオーバーライドします。

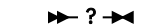
関連参照
247 ページの『-v』

-?, ?

目的

cob2 コマンドのヘルプを表示します。この オプションは、コンパイルとリンクの両方に適用されます。

構文



デフォルト
コンパイラーは、コマンド・ヘルプ情報を表示しません。

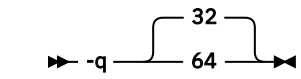
-q32、-q64

目的

-q32: 32 ビット・オブジェクト・プログラムが生成されることを指定します。ADDR(32) コンパイラー・オプションを設定します。 -m32 リンカー・オプションを設定します。このオプションは、32 ビットの実行可能モジュールを作成するようにリンカーに指示します。

-q64: このオプションは現在サポートされていません。コンパイラーはこのオプションを受け入れて無視します。

構文



デフォルト
-q32

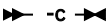
関連参照
252 ページの『ADDR』

-c

目的

完了済みのオブジェクトがリンカーへ送信されるのを防ぐ。このオプションが有効になっているとき、コンパイラーは出力オブジェクト・ファイル *file_name*.o を作成します。 このオプションはコンパイルにのみ適用されます。

構文



デフォルト

コンパイラーはデフォルトにより、リンカーを呼び出して、オブジェクト・ファイルを最終実行可能ファイルにリンクします。

例

- alpha.cbl という名前の 1 つのファイルをコンパイルするには、次のように入力します。

```
cob2 -c alpha.cbl
```

  コンパイルされたファイルには、alpha.o という名前が付けられます。
- alpha.cbl および beta.cbl という名前の 2 つのファイルをコンパイルするには、次のように入力します。

```
cob2 -c alpha.cbl beta.cbl
```

  コンパイルされたファイルには、alpha.o および beta.o という名前が付けられます。
- 2 つのファイルをリンクするには、-c オプションを指定せずにそれらのファイルをコンパイルします。例えば、alpha.cbl および beta.cbl をコンパイルしてリンクし、gamma を生成するには、次のように入力します。

```
cob2 alpha.cbl beta.cbl -o gamma
```

  このコマンドは、alpha.o と beta.o を作成し、その後に alpha.o、beta.o、および COBOL ライブラリーをリンクします。 リンク・ステップが成功すると、gamma という名前の実行可能プログラムが作成されます。
- LIST および NODATA オプションを使用して alpha.cbl をコンパイルするには、次のように入力します。

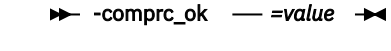
```
cob2 -qlist,noadata alpha.cbl
```

-comprc_ok

目的

コンパイラーからの戻りコードに基づいて動作を制御します。 戻りコードが *n* 以下であれば、コマンドは継続してリンク・ステップに進むか、またはコンパイルの場合のみの場合には、ゼロの戻りコードで終了します。 コンパイラーによって生成された戻りコードが *n* より大きい場合は、コマンドは同じ戻りコードで終了します。 このオプションはコンパイルにのみ適用されます。

構文



デフォルト

デフォルトは `-comprc_ok=4` です。

USAGE

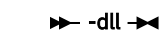
このオプションが有効になっているとき、コンパイラーは出力オブジェクト・ファイル *file_name*`.o` を作成します。

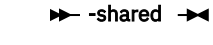
-dll | -shared

目的

リンケージ・エディターの出力をデフォルトの Position Independent Executable (PIE) から Dynamically Shared Object (DSO) に変更します。

構文





デフォルト

リンケージ・エディターの出力は PIE です。

USAGE

PIE は、他の実行可能プログラムと同じように呼び出すことができますが、COBOL 動的呼び出しまたは CICS トランザクションのターゲットとして使用することはできません。

DSO はその逆です。コマンド・ラインから呼び出すことはできませんが、COBOL 動的呼び出しまたは CICS トランザクションのターゲットとして使用できます。

-F

目的

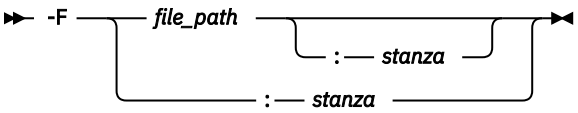
`/opt/ibm/cobol/1.1.0/etc/cob2.cfg` 構成ファイルに指定されているデフォルトではなく、構成ファイルまたはスタンザとして *xxx* を使用します。*xxx* の形式は以下のいずれかです。

- *configuration_file*:*stanza*
- *configuration_file*

- :*stanza*

この オプションは、コンパイルとリンクの両方に適用されます。

構文



デフォルト

デフォルトで、コンパイラーはインストール時に指定された構成ファイルを使用し、 そのファイルで定義されているスタンザを現在使用中の呼び出しコマンドに対して使用します。

パラメーター

File_path

使用される代替のコンパイラー構成ファイルの絶対パス名です。

stanza

コンパイルに使用する構成ファイル・スタンザの名前です。これは、使用中の呼び出しコマンドに関係なくコンパイラーに stanza の記入項目を使用するよう指示します。

関連参照

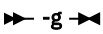
229 ページの『構成ファイル内のスタンザ』

-g

目的

デバッガーで使用されるシンボリック情報を生成します。TEST コンパイラー・オプションを設定します。この オプションは、コンパイルとリンクの両方に適用されます。

構文



デフォルト

デバッグ情報を生成しません。プログラム状態は保存されません。

例

myprogram.cbl をコンパイルしてデバッグ用に実行可能プログラム testing を生成するには、次のコマンドを使用します。

```
cob2 myprogram.cbl -o testing -g
```

関連タスク

311 ページの『IBM Debug for Linux on x86 を使用したデバッグ』

関連参照

285 ページの『TEST』

-host

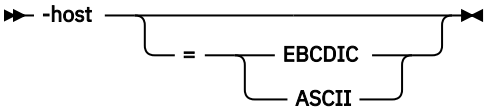
目的

ホスト COBOL データ表現と言語セマンティクスに対して以下のコンパイラー・オプションを設定します。

- BINARY(BE)
- CHAR(EBCDIC)
- COLLSEQ(EBCDIC)
- FLOAT(BE)
- NCOLLSEQ(BIN)
- UTF16(BE)

-host オプションは、COBOL プログラムのコマンド行引数の形式を、ポインターの配列から、ストリングの長さを含むハーフワード接頭部がある EBCDIC 文字ストリングに変更します。 追加情報については、コマンド行引数に関する以下の関連タスクを参照してください。

構文



デフォルト

コンパイラーは、ホスト COBOL データ表現と言語セマンティクスに対してコンパイラー・オプションを設定しません。

パラメーター

EBCDIC
-host と同じ。EBCDIC の z/OS スタイル・パラメーター・リストをプログラムに渡します。

ASCII
ASCII の z/OS スタイル・パラメーター・リストをプログラムに渡します。

関連参照
254 ページの『BINARY』
CHAR
COLLSEQ
FLOAT
NCOLLSEQ
288 ページの『UTF16』

-I

目的

library-name も SYSLIB も指定されていない場合に、コピーブックの検索に使用するディレクトリーへのパス *xxx* を追加します。 (このオプションは、小文字の *l* ではなく大文字の *I* です。)

各 -I オプションに使用できるパスは 1 つだけです。 複数のパスを追加するには、-I オプションを必要な数だけ使用してください。

構文

パラメーター

directory_path

コンパイラーがヘッダー・ファイルを検索するディレクトリーのパスです。

デフォルト

コピーブックを検索する場合の検索場所としてデフォルト・ディレクトリーはありません。

USAGE

構成ファイルとコマンド行の両方に -I directory オプションが指定されている場合は、 構成ファイルに指定されたパスが最初に検索されます。 -I directory オプションは、コマンド行に複数回指定することができます。 複数の -I オプションを指定した場合は、 ディレクトリーは、コマンド行に指定された順序で検索されます。 -I オプションは、絶対パス名を使用して組み込まれたファイルに対して影響を与えません。

例

myprogram.cbl をコンパイルして、 組み込まれたファイルを /usr/tmp and then /oldstuff/history で検索するには、次のように入力します。

```
cob2 myprogram.cbl -I/usr/tmp -I/oldstuff/history
```

-main

目的

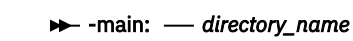
xxx を、リンカーに渡されるファイルのリスト内の最初のファイルにします。このオプションの目的は、指定したファイルを実行可能ファイル内のメインプログラムにすることです。 *xxx* はオブジェクト・ファイルまたはアーカイブ・ライブラリーを一意的に識別する必要があり、接尾部はそれぞれ .o または .a でなければなりません。

-main が指定されていない場合は、コマンド内に指定した最初のオブジェクト、アーカイブ・ライブラリー、またはソース・ファイルが、リンカーに渡されるファイルのリスト内の最初のファイルです。

-main: *xxx* の構文が無効な場合、または *xxx* がコマンドによって処理されるオブジェクト・ファイルまたはソース・ファイルの名前ではない場合は、コマンドが終了します。

このオプションはリンクにのみ適用されます。

構文



デフォルト

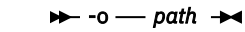
-main オプションは指定されません。

-o

目的

実行可能プログラムまたは共有ライブラリーに *xxx* という名前を付けます。ここで、*xxx* は任意の名前です。 -o オプションを使用しない場合、実行可能なモジュールの名前はデフォルトの a.out に指定されます。このオプションはリンクにのみ適用されます。

構文



パラメーター

path

ソース・ファイルからコンパイルするオプションを使用する場合、ファイルまたはディレクトリーの名前に *path* を使用できます。 *path* は、相対パスまたは絶対パスの名前にすることができます。 オブジェクト・ファイルからリンクするオプションを使用する場合、ファイル名に *path* を使用しなければなりません。*path* が既存のディレクトリーの名前である場合、 コンパイラーによって作成されたファイルは、そのディレクトリーに配置されます。 *path* が既存のディレクトリーではない場合、*path* は、コンパイラーによって作成されたファイルの名前です。 例については、以下を参照してください。

デフォルト

-c オプションを指定すると、出力オブジェクト・ファイル *file_name*.o が各入力ファイルに対して生成されます。リンカーは呼び出されず、オブジェクト・ファイルは現行ディレクトリーに入れられます。コンパイルの完了時にすべての処理が停止されます。 -o オプションを使用して、別のサフィックスを指定したり、サフィックスなしを指定したりしない限り、 コンパイラーはオブジェクト・ファイルに .o サフィックスを付けます (例: *file_name*.o)。

USAGE

-c オプションを -o と併用した場合に、 パスが既存のディレクトリーではないときは、一度にコンパイルできるソース・ファイルは 1 つのみです。 この場合、 コンパイラー呼び出し時に複数のソース・ファイル名がリストされる場合、 コンパイラーは、警告メッセージを出して -o を無視します。

例

myaccount という名前のディレクトリーが存在しないと想定し、結果として myaccount という実行可能ファイルが作成されるように myprogram.cbl をコンパイルするには、 以下のように入力します。

```
cob2 myprogram.cbl -o myaccount
```

test.cbl をオブジェクト・ファイルのみにコンパイルし、そのオブジェクト・ファイルに new.o と名付けるには、以下のように入力します。

```
cob2 test.cbl -c -o new.o
```

関連参照

-v

目的

コンパイルおよびリンクのステップを表示し、各ステップを実行します。この オプションは、コンパイルとリンクの両方に適用されます。 -v オプションが有効な場合、情報はコンマで区切られたリストに表示されます。この オプションは、コンパイルとリンクの両方に適用されます。

注 : -v オプションは、 -# オプションによってオーバーライドされます。

構文

デフォルト

コンパイラーはコンパイルの進行を表示しません。

例

myprogram.cbl をコンパイルして、コンパイルの進行を監視したり、コンパイルの進行、呼び出し中のプログラム、および指定されているオプションを記述するメッセージを表示できるようにするには、以下のように入力します。

```
cob2 myprogram.cbl -v
```

関連参照

240 ページの『-# (ポンド記号)』

-q オプション

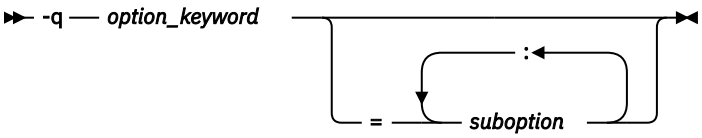
コンパイラーにオプションを渡します。*xxx* には任意のコンパイラー・オプションが入ります。 -q と *xxx* の間にスペースを入れないでください。 コンパイラー・オプションまたはサブオプションに括弧が含まれている場合や、一連のオプションが指定されている場合には、各オプションを引用符で囲みます。コンパイラー・オプションまたはサブオプションに括弧が含まれている場合や、一連のオプションが指定されている場合には、各オプションを引用符で囲みます。

複数のオプションを指定するには、各オプションをブランクまたはコンマで区切ります。 例えば、次の 2 つのオプション・ストリングは同じです。

```
-qoptiona,optionb
```

```
-q"optiona optionb"
```

シェル・スクリプトを使用して cob2 タスクを自動化する予定の 場合は、-q*xxx* オプションに対して特殊な構文が用意されています。 詳しくは、シェル・スクリプトでのコンパイルに関する関連タスクを参照してください。



関連概念

239 ページの『第 13 章 コマンド行でのコンパイラー・オプションの指定』

関連参照
248 ページの『コンパイラー・オプション』
250 ページの『矛盾するコンパイラー・オプション』

関連タスク
を使用したコンパイル

コンパイラー・オプション

コンパイルに対する指示および制御の方法には、コンパイラー・オプションを使用する方法と、コンパイラー指示ステートメント (コンパイラー指示) を使用する方法とがあります。

コンパイラー・オプションは、次の表にリストされているプログラムの局面に影響を与えます。 各オプションに結び付けられている情報は、そのオプションを指定するための構文を与えるもので、オプション、そのパラメーター、および他のパラメーターとの相互作用を説明しています。

表 *29.* **コンパイラー・オプション**

プログラムの局面	コンパイラー・オプション	デフォルト	オプションの省略形
ソース言語	253 ページの『ARITH』	ARITH(COMPAT)	AR(C|E)
	257 ページの『CICS』	NOCICS	なし
	260 ページの『CURRENCY』	NOCURRENCY	CURR|NOCURR
	274 ページの『NSYMBOL』	NSYMBOL(NATIONAL)	NS(NAT|DBCS)
	275 ページの『NUMBER』	NONUMBER	NUM|NONUM
	277 ページの『APOST/QUOTE』	QUOTE	Q|APOST
	279 ページの『SEQUENCE』	SEQUENCE	SEQ|NOSEQ
	280 ページの『SOSI』	NOSOSI	なし
	282 ページの『SQL』	SQL("")	なし
	283 ページの『SRCFORMAT』	SRCFORMAT(COMPAT)	SF(C|E)
日付処理	261 ページの『DATEPROC』	NODATEPROC、または DATEPROC(FLAG) (DATEPROC だけが指定された場合)	DP|NODP
	262 ページの『DATETIME』	DATETIME(1900 40)	なし
	291 ページの『YEARWINDOW』	YEARWINDOW(1900)	YW
マップおよびリスト	270 ページの『LINECOUNT』	LINECOUNT(60)	LC
	270 ページの『LIST』	NOLIST	なし
	271 ページの『LSTFILE』	LSTFILE(LOCALE)	LST
	271 ページの『MAP』	NOMAP	なし
	281 ページの『SOURCE』	SOURCE	S|NOS
	281 ページの『SPACE』	SPACE(1)	なし
	285 ページの『TERMINAL』	TERMINAL	TERM|NOTERM
	289 ページの『VBREF』	NOVBREF	なし
	290 ページの『XREF』	XREF(FULL)	X|NOX

表 29. **コンパイラー・オプション** (続き)

プログラムの局面	コンパイラー・オプション	デフォルト	オプションの省略形
オブジェクト・モジュールの生成	259 ページの『COMPILE』	NOCOMPILE(S)	C|NOC
	276 ページの『PGMNAME』	PGMNAME(UPPER)	PGMN(LU|LM)
	278 ページの『SEPOBJ』	SEPOBJ	なし
オブジェクト・コード制御	252 ページの『ADDR』	ADDR(32)	なし
	254 ページの『BINARY』	BINARY(NATIVE)	なし
	256 ページの『CHAR』	CHAR(NATIVE)	なし
	258 ページの『COLLSEQ』	COLLSEQ(BIN)	CS(L|E|BIN|B)
	262 ページの『DEFINE』	NODEFINE	DEF | NODEF
	264 ページの『DIAGTRUNC』	NODIAGTRUNC	DTR|NODTR
	269 ページの『FLOAT』	FLOAT(NATIVE)	なし
	274 ページの『NCOLLSEQ』	NCOLLSEQ(BINARY)	NCS(L|BIN|B)
	275 ページの『OPTIMIZE』	NOOPTIMIZE	OPT|NOOPT
	286 ページの『TRUNC』	TRUNC(STD)	なし
	292 ページの『ZWB』	ZWB	なし
CALL ステートメントの動作	264 ページの『DYNAM』	NODYNAM	DYN|NODYN
デバッグと診断	267 ページの『FLAG』	FLAG(I,I)	F|NOF
	268 ページの『FLAGSTD』	NOFLAGSTD	なし
	284 ページの『SSRANGE』	NOSSRANGE	SSR(MSG|ABD)|NOSSR
	285 ページの『TEST』	NOTEST	なし
その他	251 ページの『ADATA』	NOADATA	なし
	255 ページの『CALLINT』	CALLINT(SYSTEM,NODESC)	なし
	265 ページの『EXIT』	NOEXIT	NOEX|EX(INX|NOINX、LIBX|NOLIBX、PRTX|NOPRTX、ADX|NOADX、MSGX|NOMSGX)
	272 ページの『MAXMEM』	MAXMEM(2000)	なし
	273 ページの『MDECK』	NOMDECK	NOMD|MD|MD(C|NOC)
	279 ページの『SIZE』	SIZE(8388608)	SZ
	282 ページの『SPILL』	SPILL(512)	なし
	286 ページの『THREAD』	NOTHREAD	なし
	289 ページの『WSCLEAR』	NOWSCLEAR	なし

インストール先のデフォルト: 上記のオプションでリストされているデフォルトは、製品出荷時のデフォルトです。

オプション仕様:

- コンパイラー・オプションおよびサブオプションは大/小文字を区別しません。
- サブオプションとして引数が続くコンパイラー・オプションの場合は、option=*argument* を指定する代わりに、option(*argument*) というフォーマットを必ず使用する必要があります。

パフォーマンスに関する考慮事項: ADDR、ARITH、CHAR、DYNAM、FLOAT、OPTIMIZE、SSRANGE、TEST、TRUNC、および WSCLEAR の各コンパイラー・オプションは実行時のパフォーマンスに影響を及ぼす可能性があります。

関連タスク
224 ページの『プログラムのコンパイル』
497 ページの『第 27 章 プログラムのチューニング』

関連参照
293 ページの『第 14 章 コンパイラー指示ステートメント』
505 ページの『パフォーマンスに関連するコンパイラー・オプション』

85 COBOL Standard 準拠のオプション設定

85 COBOL Standard に準拠するには、コンパイラー・オプションおよびランタイム・オプションが必要です。

以下のコンパイラー・オプションが必要です。

- DYNAM
- NOCICS
- NOSOSI
- NOTHREAD
- PGMNAME(COMPAT) または PGMNAME(LONGUPPER)
- QUOTE
- TRUNC(STD)
- ZWB

FLAGSTD コンパイラー・オプションを使用して、IBM 拡張などの非準拠エレメントにフラグを立てることができます。

矛盾するコンパイラー・オプション

COBOL for Linux コンパイラーは、次の 2 つのいずれかの場合に、矛盾するコンパイラー・オプションに遭遇することがあります。つまり、同じオプションの肯定形式と否定形式の両方がオプションの優先順位の階層の中で同じレベルで指定されている場合、または相互に排他的なオプションが同じレベルで指定されている場合です。

コンパイラーは、次の優先順位 (高位から下位の順) に従ってオプションを認識します。

1. PROCESS (または CBL) ステートメントで指定されたオプション
2. cob2 コマンド呼び出しで指定されたオプション
3. COBOPT 環境変数のオプション・セット
4. 構成 (.cfg) ファイルの compopts 属性のオプション・セット
5. IBM デフォルト・オプション

矛盾するオプションを階層の同じレベルで指定した場合は、最後に指定したオプションの方が有効になります。

互いに排他的なコンパイラー・オプションを同じレベルで指定すると、コンパイラーは オプションの一方を対立しない値に強制し、エラー・メッセージを生成します。 例えば、PROCESS ステートメントで CICS と DYNAM の両方を指定すると、次の表に示すように、指定した順序に関係なく CICS が有効になり、DYNAM は無視されます。

表 30. 相互に排他的なコンパイラー・オプション		
指定される	**無視される**	**強制**
CICS	ADDR(64)	ADDR(32)
	DYNAM	NODYNAM
	THREAD	NOTHREAD

しかし、高レベルの優先順位で指定されたオプションは、低レベルの優先順位で指定されたオプションをオーバーライドします。 例えば、COBOPT 環境変数では CICS を コーディングし、PROCESS ステートメントでは DYNAM を コーディングすると、PROCESS ステートメントでコーディングされたオプションと、PROCESS ステートメントで コーディングされたオプションによって強制的にオンにされたオプションの優先順位の方が 高いため、DYNAM が有効になります。

関連タスク
224 ページの『プログラムのコンパイル』

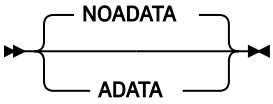
関連参照

218 ページの『コンパイラー環境変数』
229 ページの『構成ファイル内のスタンザ』
232 ページの『cob2 オプション』
293 ページの『第 14 章 コンパイラー指示ステートメント』

ADATA

ADATA は、コンパイラーに追加のコンパイル情報のレコードを収めた SYSADATA ファイルを作成させる場合に使用します。

ADATA オプションの構文



デフォルト: NOADATA

省略形: なし

SYSADATA 情報は他のツールによって使用され、使用のために ADATA ON が設定されます。

SYSADATA ファイルのサイズは、通常、関連するプログラムのサイズに比例します。

オプションの指定: PROCESS (または CBL) ステートメントで ADATA オプションを 指定することはできません。 指定できるのは、以下のいずれかの方法に限られます。

- cob2 コマンド またはそのバリアントのいずれかのオプションとして
- COBOPT 環境変数
- 構成 (.cfg) ファイルの compopts 属性

注 : ADATA オプションは現在サポートされていません。現時点ではデフォルトの NOADATA を使用してください。

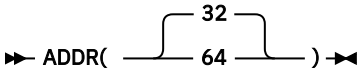
関連参照

218 ページの『コンパイラー環境変数』
229 ページの『構成ファイル内のスタンザ』
232 ページの『cob2 オプション』

ADDR

32 ビットのオブジェクト・プログラムを生成するのか、64 ビットのオブジェクト・プログラムを生成するのかを 指示するには、ADDR コンパイラー・オプションを使用します。

ADDR オプションの構文



デフォルト: ADDR(32)

省略形: なし

オプション指定:

注 : ADDR(64) オプションと -q64 オプションは現在サポートされていません。コンパイラーはこれらのオプションを受け入れて無視し、デフォルトの ADDR(32) にします。

他のコンパイラー・オプションを指定するいずれかの方法で ADDR オプションを指定できます。プログラムのコンパイルについては、関連タスクに説明があります。 ただし、PROCESS (または CBL) ステートメントで ADDR を指定する場合は、以下の点に注意が必要です。

- バッチ・コンパイルでは、最初のプログラムに対してのみ ADDR を指定できます。 バッチでは、後続のプログラムに対してオプションの値を変更することはできません。
- リンク・ステップで一致する 32 ビットまたは 64 ビット・オプションを使用する必要があります 。

cob2 コマンド の -q オプションを 使用してコンパイラー・オプションを指定すると、ADDR(32) を 32 に省略したり、ADDR(64) を 64 に省略したりできます。 次に例を示します。

```
cob2 -q64 prog64.cbl
```

ストレージ割り振り:

以下の COBOL データ型に対するストレージ割り振りは、ADDR コンパイラー・オプションの設定に依存します。

- USAGE POINTER (ADDRESS OF 特殊レジスターも同様。暗黙的にこの使用法を持つ)
- USAGE PROCEDURE-POINTER
- USAGE FUNCTION-POINTER
- USAGE INDEX

ADDR(32) が有効な場合、上記にリストされたいずれかの使用法を持つ項目が プログラムに含まれているとき、その項目ごとに 4 バイトが割り振られます。 ADDR(64) が有効な場合、項目ごとに 8 バイトが割り振られます。

上記のいずれかの USAGE を持つデータ項目に対して SYNCHRONIZED 文節が指定されている場合、ADDR(32) が 有効になっていると、その項目はフルワード境界に調整され、ADDR(64) が有効になっていると、その項目はダブルワード境界に調整されます。

ADDR オプションの設定は、いくつかのコンパイラー限界値に影響します。 詳細については、コンパイラー限界値に関する関連参照を参照してください。

LENGTH OF 特殊レジスター:

ADDR(32) が有効な場合、LENGTH OF 特殊レジスターには次の暗黙の定義があります。

```
PICTURE 9(9) USAGE IS BINARY
```

ADDR(64) が有効な場合、LENGTH OF 特殊レジスターには次の暗黙の定義があります。

```
PICTURE 9(18) USAGE IS BINARY
```

LENGTH 組み込み関数:

ADDR(32) が有効な場合、LENGTH 組み込み関数の戻り値は 9 桁の整数です。 ADDR(64) が有効な場合、戻り値は 18 桁の整数です。

プログラミングの要件および制約事項:

- アプリケーション内のプログラム・コンポーネントはすべて、同じ ADDR オプション設定を使用してコンパイルしなければなりません。 1 つのアプリケーションで 32 ビット・プログラムと 64 ビット・プログラムを混用することはできません。
- **言語間通信:** 複数言語アプリケーションでは、64 ビット COBOL プログラム は 64 ビット C/C++ プログラムと 結合でき、32 ビット COBOL プログラムは 32 ビット C/C++ プログラムと結合できます。
- **CICS:** CICS TXSeries または CICS TX 環境で実行される COBOL プログラムは 32 ビット・プログラムでなければなりません。

関連概念

関連タスク
109 ページの『データ項目の長さの検出』
224 ページの『プログラムのコンパイル』
382 ページの『CICS のもとで実行する COBOL プログラムのコーディング』
438 ページの『COBOL および C/C++ プログラム間の呼び出し』

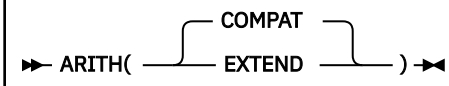
関連参照
232 ページの『cob2 オプション』
250 ページの『矛盾するコンパイラー・オプション』

コンパイラー限界値 (*COBOL for Linux on x86* 言語解説書)

ARITH

ARITH は、数値項目についてコーディングできる桁の最大数、および固定小数点の中間結果で使用される桁の数に影響します。

ARITH オプションの構文



デフォルト: ARITH(COMPAT)

省略形: AR(C|E)

ARITH(EXTEND) を指定すると、次のようになります。

- パック 10 進数、外部 10 進数、 および数字編集のデータ項目の PICTURE 節で指定できる桁位置の最大数が、18 から 31 へ引き上げられます。
- 固定小数点数値リテラルに指定できる桁数の最大数が、18 から 31 に上がります。 以下の場所を含め、数値リテラルが現在許可されているところであればどこでも、長精度の数値リテラルを使用することができます。
  - PROCEDURE DIVISION ステートメントのオペランド
  - VALUE 節 (長精度 PICTURE を含む数値データ項目に関する)

– 条件名の値 (長精度 PICTURE を含む数値データ項目に関する)

- NUMVAL、NUMVAL-C、および への引数の中で指定できる桁数の最大数が、18 から 31 に上がります。
- FACTORIAL 関数への整数引数の最大値は、29 です。
- 算術ステートメントの中間結果は、拡張モード を使用します。

ARITH(COMPAT) を指定すると、次のようになります。

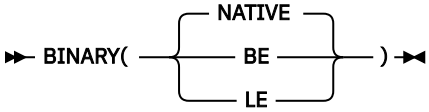
- パック 10 進数、外部 10 進数、 および数字編集のデータ項目の PICTURE 節の桁位置の最大数は 18 です。
- 固定小数点数値リテラルに指定できる桁数の最大数は、18 です。
- NUMVAL、NUMVAL-C、 への引数の中で指定できる桁数の最大数は 18 です。
- FACTORIAL 関数への整数引数の最大値は、28 です。
- 算術ステートメントの中間結果は、互換モード を使用します。

関連概念
531 ページの『付録 C 中間結果および算術精度』

BINARY

BINARY は、2 進数データ項目の表現形式を指定します。

BINARY オプションの構文



デフォルト: BINARY(NATIVE)

省略形: なし

プラットフォームのネイティブの 2 進数表現形式を使用するには、BINARY(NATIVE) を指定します。COBOL for Linux の場合、これはリトル・エンディアン 形式 (最小桁の数字が最下位アドレスに格納される) になります。

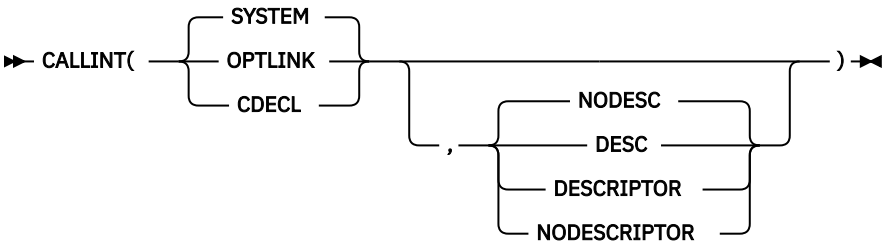
BINARY(BE) は、BINARY、COMP、COMP-4 の各データ項目が常に IBM Z、つまりビッグ・エンディアン形式 (最大桁の数字が最下位アドレスに格納される) で表現されることを示します。

BINARY(LE) は、BINARY、COMP、および COMP-4 の各データ項目が常にリトル・エンディアン 形式 (最小桁の数字が最下位アドレスに格納される) で表現されることを示します。

CALLINT

CALLINT は、CALL ステートメントによって行われる呼び出しに適用できる呼び出しインターフェース規約を 示したり、引数記述子を生成するのかどうかを示したりする場合に使用します。

CALLINT オプションの構文



デフォルト: CALLINT(SYSTEM,NODESC)

省略形: なし

特定の CALL ステートメントでこのオプションをオーバーライドするには、コンパイラー指示 >>CALLINT を使用します。

CALLINT には 2 つのサブオプション・セットがあります。

- 呼び出しインターフェース規約の選択

  SYSTEM
  SYSTEM サブオプションは、プラットフォームの標準システムのリンケージ規約が呼び出し規約となることを指定します。

  SYSTEM は、Linux でサポートされる唯一の呼び出しインターフェース規約です。

  OPTLINK
  OPTLINK サブオプションを コーディングする場合は、コンパイラーが I レベル診断メッセージを生成し、 ディレクティブ全体 (最初のキーワードのみではなく) が無視されます。

  CDECL
  CDECL サブオプションを コーディングする場合は、コンパイラーが I レベル診断メッセージを生成し、 ディレクティブ全体 (最初のキーワードのみではなく) が無視されます。

- 引数記述子が生成されるかどうかの指定

  DESC
  DESC サブオプションは、CALL ステートメントの各引数に、引数記述子が渡されることを指定します。 引数記述子の詳細については、下記の関連参照を参照してください。

  重要 : オブジェクト指向プログラムでは、DESC サブオプションを指定しないでください。

  DESCRIPTOR
  DESCRIPTOR サブオプションは、DESC サブオプションと同義です。

  NODESC
  NODESC サブオプションは、CALL ステートメントのどの引数に対しても、引数記述子を渡さないことを指定します。

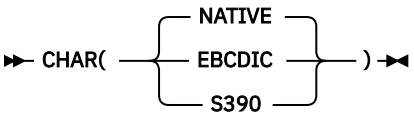
  NODESCRIPTOR
  NODESCRIPTOR サブオプションは、NODESC サブオプションと同義です。

関連参照
293 ページの『第 14 章 コンパイラー指示ステートメント』

CHAR

CHAR は、USAGE DISPLAY および USAGE DISPLAY-1 データ項目の表記および実行時の処置に影響を与えます。

CHAR オプションの構文



デフォルト: CHAR(NATIVE)

省略形: なし

プラットフォームのネイティブの文字表現 (ネイティブ形式) を使用するには、CHAR(NATIVE) を指定します。 COBOL for Linux の場合、ネイティブ形式は、実行時に有効なロケールによって示されるコード・ページによって定義されます。 コード・ページは、 単一バイトの ASCII コード・ページまたは ASCII ベースのマルチバイト・コード・ページ (UTF-8、EUC、または ASCII DBCS) に することができます。

CHAR(EBCDIC) と CHAR(S390) は同義です。これらは、DISPLAY および DISPLAY-1 データ項目が IBM Z の文字表現 (つまり EBCDIC) であることを示します。

ただし、USAGE 文節の NATIVE 句で 定義された DISPLAY および DISPLAY-1 データ項目 は、CHAR(EBCDIC) オプションの影響を受けません。 これらは常にプラットフォームのネイティブ形式で格納されます。

CHAR(EBCDIC) コンパイラー・オプションは、実行時の処理に次のような影響を及ぼします。

- **USAGE DISPLAY および USAGE DISPLAY-1 項目:** USAGE DISPLAY で記述されたデータ項目内の文字は、単一バイトの EBCDIC 形式として処理されます。 USAGE DISPLAY-1 で記述されたデータ項目内の文字は、 EBCDIC DBCS 形式として処理されます (後続の記述で、*EBCDIC* という用語は、 USAGE DISPLAY の単一バイトの EBCDIC 形式および USAGE DISPLAY-1 の EBCDIC DBCS 形式を指しています)。
  - ネイティブ形式でエンコードされたデータは、 端末からの ACCEPT で EBCDIC 形式に変換されます。
  - EBCDIC データは、端末への DISPLAY でネイティブ形式に変換されます。
  - 英数字リテラルおよび DBCS リテラルの内容は、 EBCDIC でエンコードされたデータ項目への割り当てのため、EBCDIC 形式に変換されます。 CHAR(EBCDIC) オプションが有効な場合の文字データの比較に関する 規則については、COLLSEQ オプションに関する下記の関連参照を参照してください。
  - 編集には EBCDIC 文字を使用します。
  - 埋め込みには EBCDIC スペースを使用します。 英数字操作 (割り当てや比較など) に使用されるグループ項目は、 グループ内の基本項目の定義に関わらず、単一バイトの EBCDIC スペースで埋め込まれます。
  - USAGE DISPLAY 項目への割り当て、またはこの項目との関係条件において、VALUE 文節内で使用される表意定数 SPACE または SPACES は、1 バイトの EBCDIC スペース (つまり X'40') として扱われます。
  - DISPLAY-1 項目への割り当て、またはこの項目との関係条件において、VALUE 文節内で使用される表意定数 SPACE または SPACES は、1 バイトの EBCDIC DBCS スペース (つまり X'4040') として扱われます。
  - Class テストは、EBCDIC の値範囲に基づいて実行されます。
- **USAGE DISPLAY 項目**
  - CALL *identifier*、CANCEL *identifier*、または形式 6 SET ステートメント内の program-name はネイティブ形式に変換されます (*identifier* で示されるデータ項目が EBCDIC でエンコードされている場合)。

- ASSIGN USING *data-name* の *data-name* で示されるデータ項目内の file-name は、 ネイティブ形式に変換されます (データ項目が EBCDIC でエンコードされている場合)。
- SORT-CONTROL 特殊レジスター内の file-name は、ソートまたはマージ関数に渡される前にネイティブ形式に変換されます (SORT-CONTROL には、暗黙的な定義 USAGE DISPLAY があります)。
- ゾーン 10 進数データ (USAGE DISPLAY を使用した数値 PICTURE 文節) および display 浮動小数点データは、EBCDIC 形式として処理されます。 例えば、PIC S9 value "1" の値は、X'31' ではなく X'F1' です。

- **グループ項目:** 英数字グループ項目 は、USAGE DISPLAY 項目と同様に扱われます。 (英数字グループ項目の USAGE 文節は、 グループ自体ではなくグループ内の基本項目に適用されます。)

16 進数リテラルが文字データに割り当てられるか、文字データと比較される場合は、16 進数リテラルが EBCDIC 文字を表すものと想定されます。 例えば、X'C1' は、値 'A' を持つ英数字項目と等しくなります。

表意定数 HIGH-VALUE または HIGH-VALUES、LOW-VALUE または LOW-VALUES、SPACE または SPACES、ZERO または ZEROS、および QUOTE または QUOTES は論理的に、EBCDIC で エンコードされたデータ項目の割り当てまたは比較を行うための EBCDIC 文字表現として扱われます。

英数字の USAGE DISPLAY 項目間の比較では、照合シーケンスとして、2 進数 (16 進数) 値に基づく 文字順序シーケンスが使用され、1 バイト文字の代替照合シーケンスが指定されている場合は、これによって変更されます。

関連タスク
202 ページの『文字データのコード・ページの指定』

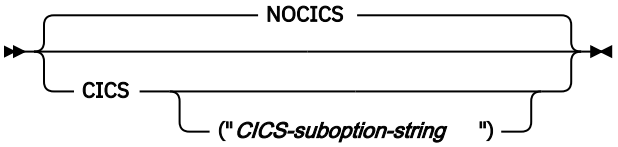
関連参照

258 ページの『COLLSEQ』
398 ページの『XML 文書のエンコード方式』
529 ページの『付録 B IBM Z ホスト・データ形式についての考慮事項』

CICS

CICS コンパイラー・オプションを指定すると、組み込みの CICS 変換プログラムが使用可能になり、CICS サブオプションを指定できるようになります。COBOL ソース・プログラムに EXEC CICS ステートメントが含まれており、プログラムが別個の CICS 変換プログラムで処理されていない場合は、CICS オプションを使用する必要があります。

CICS オプションの構文



デフォルト: NOCICS

省略形: なし

CICS オプションは、CICS プログラムをコンパイルする場合にのみ使用してください。 CICS オプションを指定してコンパイルされたプログラムは、非 CICS 環境では実行することができません。

NOCICS オプションを指定した場合、ソース・プログラム内で検出された CICS ステートメントはすべて診断され、破棄されます。

引用符またはアポストロフィのどちらかを使用して、CICS サブオプションのストリングを区切ります。

CBL または PROCESS ステートメントで、上記の構文を使用することができます。 cob2 または cob2_r コマンドで CICS オプションを使用した場合、ストリング区切り文字として使用できるのは、単一引用符 (') のみです (-q"CICS('*options*')"。

長い CICS サブオプション・ストリングを、複数のサブオプション・ストリングに分割して、複数の CBL または PROCESS ステートメントに置くことができます。CICS サブオプションは出現順に連結されます。 例えば、ソース・ファイル mypgm.cbl に以下のコードが含まれているとします。

```
cbl . . . CICS("string2") . . .
cbl . . . CICS("string3") . . .
```

コマンド cob2_r mypgm.cbl -q"CICS('string1')" を発行すると、コンパイラーは、次のサブオプション・ストリングを組み込み CICS 変換プログラムに渡します。

```
"string1 string2 string3"
```

ここに示すように、連結されたストリングはシングル・スペースで区切られます。 同じ CICS サブオプションの インスタンスが複数見つかった場合は、連結ストリングにおいて最後に指定されたものが有効となります。 コンパイラーでは、連結されたサブオプション・ストリングのサイズは 4KB に制限されます。

関連概念
387 ページの『組み込みの CICS 変換プログラム』

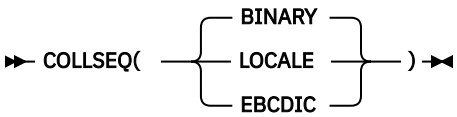
関連タスク
381 ページの『第 18 章 COBOL プログラムの開発 (CICS の場合)』

関連参照
250 ページの『矛盾するコンパイラー・オプション』

COLLSEQ

COLLSEQ は、英数字および DBCS オペランドを比較するための照合シーケンスを指定します。

COLLSEQ オプションの構文



デフォルト: COLLSEQ(BINARY)

省略形: CS(L|E|BIN|B)

COLLSEQ に次のサブオプションを指定できます。

- COLLSEQ(EBCDIC): ASCII 照合シーケンスではなく EBCDIC 照合シーケンスを使用します。
- COLLSEQ(LOCALE): (ロケールの照合に関する国/地域別情報と 整合した) ロケールに依存する照合を使用します。
- COLLSEQ(BIN): 16 進値の文字を使用します。ロケール設定は影響しません。 この設定を行うと、実行時のパフォーマンスが向上します。

STANDARD-1、STANDARD-2、または EBCDIC の英字名を持つソースに PROGRAM COLLATING SEQUENCE 文節を使用する場合は、英数字オペランドの比較では COLLSEQ オプションが 無視されます。 PROGRAM COLLATING SEQUENCE is NATIVE を指定した場合は、COLLSEQ オプションが適用されます。 それ以外の場合は、PROGRAM COLLATING SEQUENCE 文節で指定された英字名がリテラルで定義されているときに、COLLSEQ オプションによって指定された照合シーケンスが使用されます。このオプションは、この英

字名で指定されたユーザー定義のシーケンスによって変更されます。 (詳細については、ALPHABET 文節の関連参照をご覧ください。)

PROGRAM COLLATING SEQUENCE 文節は、DBCS 比較には影響しません。

前述のサブオプション NATIVE は推奨できません。 NATIVE サブオプションを指定する場合は、COLLSEQ(LOCALE) が前提となります。

次の表に、PROGRAM COLLATING SEQUENCE 文節が指定されていない場合に、比較に使用されるデータの型 (ASCII または EBCDIC) および 有効な COLLSEQ オプションに基づいて、適用できる変換および照合シーケンスをまとめます。 このオプションが指定されている場合は、コンパイラー・オプションの指定よりもソースの指定が優先されます。 CHAR オプションによって、データが ASCII になるか EBCDIC になるかが決まります。

表 *31.* **被比較数のデータ型および照合シーケンスが比較に与える影響**

被比較数	**COLLSEQ(BIN)**	**COLLSEQ(LOCALE)**	**COLLSEQ(EBCDIC)**
両方とも ASCII	変換は行われません。 比較は 2 進数値 (ASCII) に基づきます。	変換は行われません。 比較は現行のロケールに基づきます。	両方の被比較数が EBCDIC に変換されます。 比較は 2 進数値 (EBCDIC) に基づきます。
ASCII と EBCDIC の混在	EBCDIC の被比較数が ASCII に変換されます。 比較は 2 進数値 (ASCII) に基づきます。	EBCDIC の被比較数が ASCII に変換されます。 比較は現行のロケールに基づきます。	ASCII の被比較数が EBCDIC に変換されます。 比較は 2 進数値 (EBCDIC) に基づきます。
両方とも EBCDIC	変換は行われません。 比較は 2 進数値 (EBCDIC) に基づきます。	両方の被比較数が ASCII に変換されます。 比較は現行のロケールに基づきます。	変換は行われません。 比較は 2 進数値 (EBCDIC) に基づきます。

関連タスク
6 ページの『照合シーケンスの指定』
207 ページの『ロケール付きの照合シーケンスの制御』

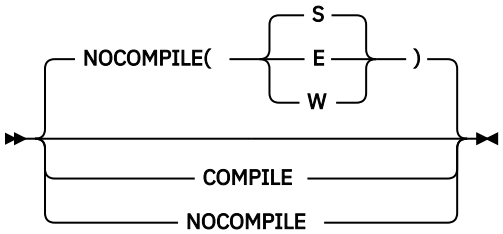
関連参照

256 ページの『CHAR』
ALPHABET 文節 (*COBOL for Linux on x86* 言語解説書)

COMPILE

COMPILE オプションは、重大エラーがあっても完全コンパイルを強制的に行う場合に限り、使用してください。 すべての診断およびオブジェクト・コードが生成されます。 コンパイルの結果として重大エラーが発生した場合は、生成されたオブジェクト・コードを実行しないでください。実行した場合の結果は保証されず、異常終了する場合があります。

COMPILE オプションの構文



デフォルト: NOCOMPILE(S)

省略形: C|NOC

NOCOMPILE にサブオプションを指定しないで使用すると、構文検査を要求します (診断だけが作成され、オブジェクト・コードは生成されません)。 サブオプションなしで NOCOMPILE を使用すると、オブジェクト・コードが生成されないため、 いくつかのコンパイラー・オプション (LIST、 OPTIMIZE、SSRANGE、および TEST) が無効になります。

NOCOMPILE にサブオプション W、E、または S を付けて使用すると、条件付き完全コンパイルを行います。コンパイラーが指定されたレベルのエラーを見つけると、完全コンパイル (診断およびオブジェクト・コード) は停止し、構文検査だけを継続します。

関連タスク
307 ページの『コーディング・エラーの検出』

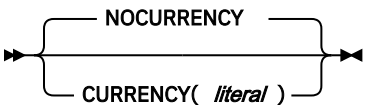
関連参照

231 ページの『コンパイラー検出エラーに関するメッセージおよびリスト』

CURRENCY

CURRENCY オプションを使用すれば、COBOL プログラムで使用する代替のデフォルト通貨記号を指定することができます。 (デフォルトの通貨記号はドル記号 ($) です。)

CURRENCY オプションの構文



デフォルト: NOCURRENCY

省略形: CURR|NOCURR

NOCURRENCY を指定すると、代替のデフォルト通貨記号が使用されません。

デフォルト通貨記号を変更するには、CURRENCY(*literal*) オプションを使用します。ここで、*literal* は、単一文字を表す有効な COBOL 英数字リテラル (または 16 進リテラル) です。 リテラルは、次のリストのものにすることはできません。

- 数値 0 から 9
- 英大文字 A B C D E G N P R S V X Z またはその英小文字
- スペース
- 特殊文字 * + - / , . ; ( ) " =
- 形象定数
- ヌル終了リテラル
- DBCS リテラル
- 国別リテラル

プログラムが 1 つの通貨タイプしか処理しない場合には、CURRENCY SIGN 節の代わりに CURRENCY オプションを使用して、プログラムの PICTURE 節で使用する通貨記号を指定できます。 プログラムで複数の通貨タイプを処理する場合は、CURRENCY SIGN 節と WITH PICTURE SYMBOL 句を併用して、異なる通貨記号タイプを指定しなければなりません。

CURRENCY オプションと CURRENCY SIGN 節の両方をプログラムで使用した場合は、CURRENCY オプションの方が無視されます。 CURRENCY SIGN 節で指定した通貨記号を、PICTURE 節で使用することができます。

NOCURRENCY オプションが有効なときに、CURRENCY SIGN 節を省略すると、通貨記号の PICTURE 記号としてドル記号 ($) が使用されます。

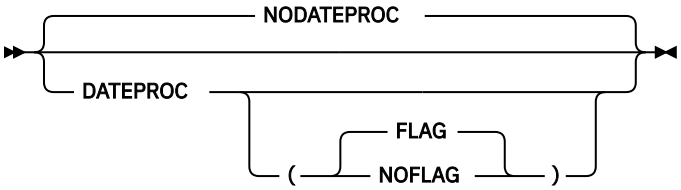
区切り文字: CURRENCY オプション・リテラルは、APOST|QUOTE コンパイラー・オプション設定に関係なく、引用符またはアポストロフィで区切ることができます。

関連タスク
55 ページの『通貨記号の使用』

DATEPROC

DATEPROC オプションは、COBOL コンパイラーの 2000 年言語拡張を使用可能にする場合に使用します。

DATEPROC オプションの構文



デフォルト: NODATEPROC、または DATEPROC(FLAG) (DATEPROC だけが指定された場合)

省略形: DP|NODP

DATEPROC(FLAG)
DATEPROC(FLAG) を指定すると、2000 年言語拡張が使用可能になり、言語エレメントが拡張機能を使用するか、または言語エレメントが拡張機能の影響を受けるたびに、コンパイラーは診断メッセージを作成します。 このメッセージは通常は通知レベルまたは警告レベルのメッセージであり、日付依存型処理に関連するステートメントを識別します。 日付構造のエラーまたは矛盾の可能性を識別する追加のメッセージが生成されることもあります。

診断メッセージの作成とソース・リスト内またはソース・リストの後にそれらのメッセージが示されるかどうかは、FLAG コンパイラー・オプションの設定に左右されます。

DATEPROC(NOFLAG)
DATEPROC(NOFLAG) を指定すると、2000 年言語拡張は有効になりますが、COBOL ソースにエラーまたは矛盾がない限り、コンパイラーは関連メッセージを作成しません。

NODATEPROC
NODATEPROC は、このコンパイル単位に関して拡張機能を使用可能にしないことを示します。 このオプションは、日付関連プログラム構造に、次のような影響を与えます。

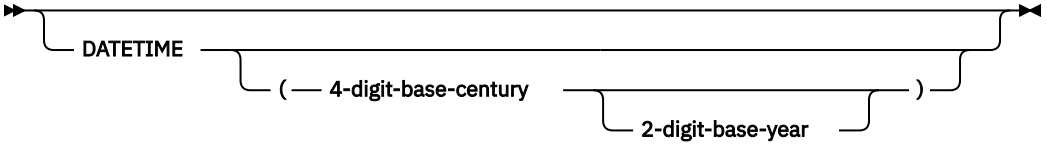
- DATE FORMAT 節は構文検査されますが、プログラムの実行には影響しなくなります。
- DATEVAL と UNDATE 組み込み関数は無効です。 すなわち、組み込み関数によって戻り値は引数の値と同じになります。
- YEARWINDOW 組み込み関数は値 0 を戻します。

関連参照
267 ページの『FLAG』
291 ページの『YEARWINDOW』

DATETIME

DATETIME オプションでは、ウィンドウ操作のアルゴリズムで使用される日付ウィンドウを指定します。

DATETIME オプションの構文



デフォルト: DATETIME(1900, 40)

省略形: なし

4-digit-base-century
これは最初の引数でなければなりません。ウィンドウ操作のアルゴリズムで使用する基本世紀を定義します。デフォルト値は 1900 です。

2-digit-base-year
これは 2 番目の引数でなければなりません。ウィンドウ操作のアルゴリズムで使用する基本年を定義します。デフォルト値は 40 です。

デフォルト・オプションの DATETIME(1900, 40) では、1940 から 2039 までの 100 年ウィンドウになります。DATETIME(1900 70) を指定すると、1970 から 2069 までの 100 年ウィンドウになります。

オプション指定:

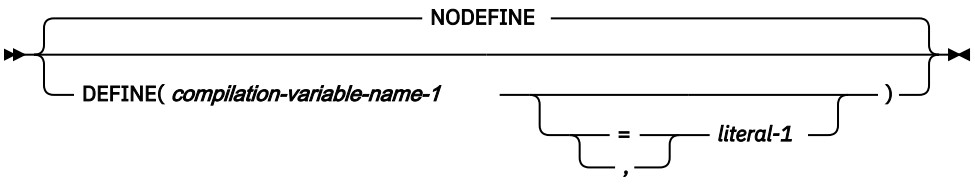
- cob2 コマンドのオプションでは、例えば DATETIME('1900 40') のように、引数は単一引用符で囲む必要があります。
- プロセス・ステートメントでは、例えば、DATETIME(1900 40) のように、引数は引用符なしで作成します。
- DATETIME オプションが未指定または引数が提供されない場合、基本世紀および基本年はデフォルト値になります。基本世紀は、後続の基本年の引数なしで指定でき、この場合、基本年はそのデフォルトの値になります。基本年を指定する場合は、基本世紀も指定しなければなりません。

DEFINE

DEFINE コンパイラー・オプションは、PARAMETER 句を持つ DEFINE ディレクティブを使用してプログラムで定義されたコンパイル変数にリテラル値を代入する場合に使用します。 DEFINE オプションでコンパイル変数に対して指定されるリテラル値は、対応するコンパイル変数のパラメーター値と呼ばれることがあります。 コンパイル変数は、任意の条件付きコンパイル・ディレクティブ (DEFINE、EVALUATE、IF など) 内で使用できます。 条件付きコンパイル・ディレクティブに含まれている条件付きコンパイル変数は、現在示しているリテラル値に対するシンボル参照として扱われます。

DEFINE コンパイラー・オプションを使用すれば、プログラム・ソースの外側からコンパイル変数に値を代入できます。 それが不要な場合は、プログラム・ソース内で DEFINE ディレクティブを使用してコンパイル変数を定義すれば十分です。

DEFINE オプションの構文



デフォルト: NODEFINE

省略形: DEF|NODEF

compilation-variable-name-1
プログラムにおいて条件付きコンパイル・ディレクティブで参照されるコンパイル変数の名前。 対応する DEFINE ディレクティブ (PARAMETER 句あり) がプログラム内の *compilation-variable-name-1* に対して存在しない場合は、そのコンパイル変数に対して指定された DEFINE コンパイラー・オプションのインスタンスはいずれも無視されます。 *compilation-variable-name-1* はデータ名のユーザー定義語の規則に従って構成されます。 ただし、DBCS 文字は名前には使用できません。 詳しくは、「*COBOL for Linux on x86* 言語解説書」 にある「ユーザー定義語」 を参照してください。

literal-1
プログラムにおいて条件付きコンパイル関連ディレクティブで *compilation-variable-name-1* がシンボリックに表すリテラル値。 *literal-1* は以下のいずれかの項目でなければなりません。

- 通常の英数字リテラル ('abcd') または 16 進数リテラル (x'F1F2F3') として指定可能な英数字リテラル。 国別リテラル、DBCS リテラル、およびヌル終了英数字リテラル (Z リテラル) はサポートされていません。
- 整数リテラル。
- ブール・リテラル (B'0' および B'1' のみサポート)

literal-1 が指定されない場合は、値 B'1' がコンパイル変数に代入されます。 例えば、次のように指定したとします。

```
>>define foo
```

foo には値 B'1' が割り当てられます。

注 : コンパイラーは、特定のシェル・スクリプト文字を以下のように解釈します。

- 等号 (=) は左括弧 (に解釈されます。
- コロン (:) は右括弧) に解釈されます。
- 下線 (_) は単一引用符 (') に解釈されます。

円記号 (¥) エスケープ文字を追加すれば、このような解釈を防ぐことができるため、文字をストリングで渡すことができます。 円記号 (¥) が (エスケープ文字ではなく) 円記号そのものを表すようにしたい場合は、円記号 (¥) を重ねて (¥¥) 使用します。

例えば、DEFINE オプションを使用してリテラル値 1 をコンパイル変数 VAR1 に割り当てるには、次のように DEFINE オプションを指定します。

```
DEFINE(VAR1=1)
```

コンパイラーの解釈からエスケープする等号、コロン、または下線が VAR1 に含まれている場合は、次のように DEFINE オプションを指定します。

```
DEFINE(VAR1\=1)
```

DEFINE オプションのインスタンスを複数指定して、異なる複数のコンパイル変数に対して値を定義できます。 単一の条件付きコンパイル変数が複数定義されている場合は、最後の変数定義が、対応する条件付きコンパイル変数の値として使用されます。 NODEFINE が、前の DEFINE オプション・インスタンスの後に指定されている場合は、すべての条件付きコンパイル変数に対する定義は取り消されます。

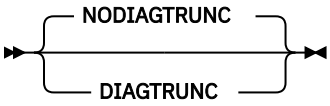
CBL ステートメントで指定される DEFINE オプションはバッチ・プログラムの最初のプログラムでのみ使用できます。 そのため、ファイルに複数の COBOL プログラムが含まれている場合は、最初のプログラムの前にのみ DEFINE オプションを指定した CBL ステートメントを使用できます。 最初のプログラムを対象に指定された DEFINE オプション (および cob2 コマンド・オプションとして指定された DEFINE オプション) は、ファイル内のすべてのプログラムに適用されます。

関連参照
条件付きコンパイル (*COBOL for Linux on x86* 言語解説書)
DEFINE (*COBOL for Linux on x86* 言語解説書)

DIAGTRUNC

DIAGTRUNC を使用すると、受け取り側が数値である MOVE ステートメントの場合に、受け取りデータ項目の整数桁数が送り出しデータ項目またはリテラルよりも少ないときには、コンパイラーは、重大度 4 (警告) の診断メッセージを出します。 複数の受け取り側があるステートメントでは、切り捨てられる可能性があるそれぞれの受け取り側ごとにメッセージが出されます。

DIAGTRUNC オプションの構文



デフォルト: NODIAGTRUNC

省略形: DTR|NODTR

診断メッセージは、次のようなステートメントに関連した暗黙の移動の場合にも出されます。

- INITIALIZE
- READ . . . INTO
- RELEASE . . . FROM
- RETURN . . . INTO
- REWRITE . . . FROM
- WRITE . . . FROM

送信フィールドが参照変更である場合を除いて、英数字データ名またはリテラルの送り出し側から数値の受け取り側への移動についても、診断メッセージが出されます。

TRUNC(BIN) オプションを指定した場合、COMP-5 の受け取り側についても、2 進数の受け取り側についても、診断メッセージはありません。

関連概念
39 ページの『数値データの形式』
100 ページの『参照修飾子』

関連参照

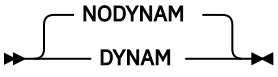
286 ページの『TRUNC』

DYNAM

DYNAM を使用すると、CALL *literal* ステートメントにより呼び出された、ネストされていない、別々にコンパイルされたプログラムを実行時に動的にロードしたり (CALL の場合)、削除したり (CANCEL の場合) することができます。

CALL *identifier* ステートメントの場合、ターゲット・プログラムは常に実行時にロードされ、このオプションの影響を受けません。

DYNAM オプションの構文



デフォルト: NODYNAM

省略形: DYN|NODYN

ON EXCEPTION 句の条件は、DYNAM オプションが有効化されている場合にのみ、CALL *literal* ステートメントで使用できます。

制約事項 : DYNAM コンパイラー・オプションは、EXEC CICS ステートメントまたは EXEC SQL ステートメントを含むプログラムに使用してはいけません。

NODYNAM を使用すれば、ターゲット・プログラム名は、リンカーで解決されます。

DYNAM オプションを使用した場合、次のステートメントの動作は、

```
CALL "myprogram" . . .
```

次のステートメントと同じ動作です。

```
MOVE "myprogram" to id-1
CALL id-1 ...
```

関連概念
436 ページの『CALL identifier および CALL literal』
CALLINTERFACE (*COBOL for Linux on x86* 言語解説書)

関連参照
250 ページの『矛盾するコンパイラー・オプション』

EXIT

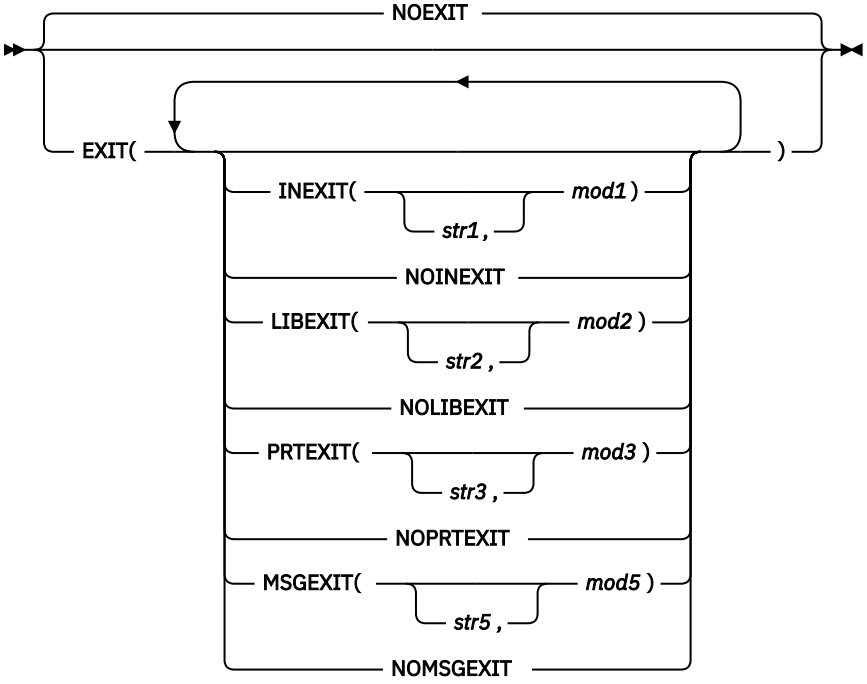
EXIT オプションを使用して、各種コンパイラー関数に代わるユーザー提供モジュールを指定します。

コンパイラー入力には、INEXIT サブオプションを使用して SYSIN に代わるモジュールを指定し (1 次コンパイラー入力)、LIBEXIT サブオプションを使用して SYSLIB に代わるモジュールを指定します (コピー・ライブラリー入力)。 コンパイラー出力には、PRTEXIT サブオプションを使用して、SYSPRINT に代わるモジュールを指定します (コンパイラー・リスト・ファイル)。

コンパイラー・メッセージをカスタマイズする (重大度を割り当てる診断メッセージへの FIPS (FLAGSTD) メッセージの変換など、 コンパイラー・メッセージの重大度を変更したり、それを抑制したりする) には、MSGEXIT サブオプションを使用します。 メッセージのカスタマイズに指定したモジュールは、コンパイラーが 診断メッセージまたは FIPS メッセージを出すたびに呼び出されます。

出口モジュールを作成するときには、モジュールが共用ライブラリーとしてリンクされていることを確認してから、COBOL コンパイラーで実行してください。 出口モジュールは、プラットフォームのシステム・リンケージ規約とともに呼び出されます。

EXIT オプションの構文



デフォルト: NOEXIT

省略形: NOEX|EX(INX|NOINX, LIBX|NOLIBX, PRTX|NOPRTX, ADX|NOADX, MSGX|NOMSGX)

オプションの指定: PROCESS (または CBL) ステートメントで EXIT オプションを 指定することはできません。 指定できるのは、以下のいずれかの方法に限られます。

- cob2 コマンドでのオプションとして
- COBOPT 環境変数

サブオプションは、コンマまたはスペースで区切って任意の順序で指定できます。 サブオプションの肯定形式と否定形式の両方を指定した場合は、最後に指定された形式が有効になります。 同じサブオプションを複数回指定すると、最後に指定したものが有効になります。

サブオプションをまったく指定せずに EXIT オプションを指定する (つまり、EXIT( ) を指定する) と、NOEXIT が有効になります。

INEXIT([*'str1'*,]*mod1*)
コンパイラーは、SYSIN ではなく、ユーザー提供のロード・モジュール (*mod1* はモジュール名) からソース・コードを読み取ります。

LIBEXIT([*'str2'*,]*mod2*)
コンパイラーは、*library-name* または SYSLIB ではなく、ユーザー提供のロード・モジュール (*mod2* はモジュール名) からコピーブックを入手します。 COPY ステートメントまたは BASIS ステートメントと一緒に使用するためです。

PRTEXIT([*'str3'*,]*mod3*)
コンパイラーは、プリンター宛先の出力を、SYSPRINT ではなく、ユーザー提供のロード・モジュール (*mod3* はモジュール名) に渡します。

MSGEXIT(['*str5*',]*mod5*)
コンパイラーは、メッセージ番号を渡し、 コンパイラー診断メッセージのデフォルトの重大度、または FIPS コンパイラー・メッセージのカテゴリー (数字コード) をユーザー提供ロード・モジュール に渡します (ここで、*mod5* はモジュール名です)。

名前 *mod1*、*mod2*、 *mod3*、*mod4*、および *mod5* は、同じものを参照することが可能です。

サブオプション *str1*、*str2*、 *str3*、*str4*、および *str5* は、ロード・モジュールに渡される文字ストリングです。 これらのストリングはオプションです。ストリングは最高 64 文字までの長さにすることができ、2 つのアポストロフィ (' ') で囲む必要があります。任意の文字をストリングに使用できますが、アポストロフィを組み込む場合は二重 (") にしなければなりません。小文字は大文字に変換されます。

文字ストリング形式: *str1*、*str2*、 *str3*、*str4*、または *str5* のいずれかが指定されると、 そのストリングは次のフォーマットで適切なユーザー出口モジュールに渡されます。 ここで、LL は、ストリングの長さを含む (ハーフワード境界に位置する) ハーフワードです。

LL	ストリング

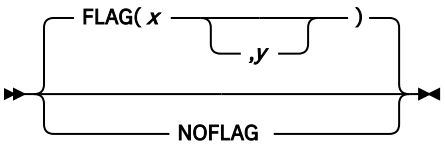
596 ページの『例: MSGEXIT ユーザー出口』

関連参照
268 ページの『FLAGSTD』
589 ページの『付録 F EXIT コンパイラー・オプション』

FLAG

重大度レベル *x* 以上のエラーのソース・リストの終わりに診断メッセージを作成するには、FLAG(*x*) を使用します。

FLAG オプションの構文



デフォルト: FLAG(I,I)

省略形: F|NOF

x および *y* は、I、W、E、S、U のいずれかになります。

FLAG(*x*,*y*) を 使用すると、重大度レベル *x* 以上のエラーに関して診断メッセージをソース・リストの終わりに 作成し、重大度レベル *y* 以上のエラーに関してはエラー・メッセージをソース・リストに 直接組み込むことができます。 *y* に指定する重大度は、*x* に指定する重大度より低い値にすることができません。 FLAG(*x*,*y*) を使用する場合は、SOURCE コンパイラー・オプションも指定する必要があります。

ソース・リスト内のエラー・メッセージは、メッセージ・コードを指す矢印の中にステートメント番号を埋め込むことによって、強調されます。 メッセージ・コードの後にメッセージ・テキストが続きます。 例えば、次のように指定します。

```
  000413     MOVE CORR WS-DATE TO HEADER-DATE
==000413==>    IGYPS2121-S     " WS-DATE " was not defined as a data-name.  . . .
```

FLAG(*x*,*y*) が有効である場合は、重大度 *y* 以上のほとんどのメッセージが、リスト内でそのメッセージの原因となった行の後に組み込まれます。IGYCB プレフィックスのあるメッセージがソース中に組み込まれることは決してありません。(例外のメッセージについては、以下に示す関連参照資料を参照してください。)

エラーのフラグ付けを抑止する場合は、NOFLAG を使用してください。 NOFLAG を使用しても、コンパイラー・オプションのエラー・メッセージは抑止されません。

組み込みメッセージ

- レベル U メッセージを組み込みに指定するのはお勧めできません。 レベル U のメッセージの組み込みの指定は受け入れられますが、ソース内には何のメッセージも作成されません。
- FLAG オプションは、コンパイラー・オプションの処理前に作成された診断メッセージには影響しません。
- コンパイラー・オプション、CBL ステートメントまたは PROCESS ステートメント、または BASIS、COPY、および REPLACE の各ステートメントの処理中に生成された診断メッセージがソース・リストに組み込まれることはありません。 このようなメッセージはすべて、コンパイラー出力の先頭に表示されます。
- IGYCB プレフィックスのある診断メッセージがソース・リストに組み込まれることはありません。このようなメッセージはすべて、FLAG オプションの設定に関係なく、コンパイラー出力の末尾に表示されます。
- *CONTROL または *CBL ステートメントの処理中に作成されたメッセージは、ソース・リストに組み込まれません。

関連参照
231 ページの『コンパイラー検出エラーに関するメッセージおよびリスト』

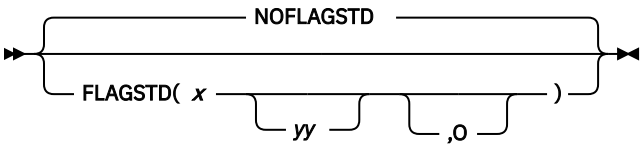
FLAGSTD

FLAGSTD は、準拠していると見なされる 85 COBOL 標準のレベルまたはサブセットを指定し、プログラムに組み込まれている 85 COBOL 標準エレメントに関して通知メッセージを取得する場合に使用します。

フラグ付け処理には、次の項目のどれかを指定することができます。

- 連邦情報処理標準 (FIPS) COBOL の選択されたサブセット
- オプション・モジュールのいずれか
- 廃止される言語エレメント
- サブセットとオプション・モジュールの任意の組み合わせ
- サブセットと古くなったエレメントの任意の組み合わせ
- IBM 拡張 (IBM 拡張にフラグが付けられるのは、FLAGSTD が指定され、かつ、「非規格準拠外」として識別された場合です。)

FLAGSTD オプションの構文



デフォルト: NOFLAGSTD

省略形: なし

x は、準拠していると見なされる 85 COBOL 標準のサブセットを指定します。

M
最小サブセットからのものではない言語エレメントに、「規格準拠外」というフラグを付けます。

I
最小サブセットまたは中間サブセットからのものではない言語エレメントに、「規格準拠外」というフラグを付けます。

H
高位サブセットが使用されており、言語エレメントにはサブセットによってフラグが付けられません。IBM 拡張であるエレメントは、「規格準拠外、IBM 拡張」とフラグが付けられます。

yy は、単一文字または 2 文字の組み合わせによって、サブセットに組み込むオプション・モジュールを指定します。

D
デバッグ・モジュール・レベル 1 のエレメントには、「規格準拠外」というフラグを付けません。

N
分割モジュール・レベル 1 のエレメントには、「規格準拠外」というフラグを付けません。

S
分割モジュール・レベル 2 のエレメントには、「規格準拠外」というフラグを付けません。

S を指定すると、N が含められます (N は S のサブセットです)。

O (英字) は、廃止された言語エレメントに「廃止」のフラグを付けるように指定します。

通知メッセージはソース・プログラム・リストに表示され、以下の情報を示しています。

- エレメントの「廃止」、「規格準拠外」、または「非規格準拠外」(廃止になり、しかも規格準拠外の言語エレメントには廃止のフラグだけを立てます)。
- そのエレメントが含まれている節、ステートメント、またはヘッダー。
- そのエレメントが含まれる節、ステートメント、またはヘッダーのソース・プログラム行および開始位置。
- そのエレメントが属するサブセットまたはオプション・モジュール。

FLAGSTD には、予約語の標準セットが必要です。

次の例では、関連メッセージ・コードおよびテキストとともに、フラグ付き節、ステートメント、またはヘッダーが使用された行番号と桁が示されています。 その後に、フラグ付き項目の総数と、その項目のタイプの要約があります。

```
   LINE.COL CODE       FIPS MESSAGE TEXT

            IGYDS8211  Comment lines before "IDENTIFICATION DIVISION":
                       nonconforming nonstandard, IBM extension to
                       ANS/ISO 1985.

      11.14 IGYDS8111  "GLOBAL clause":  nonconforming standard, ANS/ISO
                       1985 high subset.

      59.12 IGYPS8169  "USE FOR DEBUGGING statement":  obsolete element
                       in ANS/ISO 1985.


 FIPS MESSAGES TOTAL             STANDARD      NONSTANDARD      OBSOLETE

                 3                   1              1              1
```

EXIT コンパイラー・オプションの MSGEXIT サブオプションを使用することによって、FIPS 通知メッセージを診断メッセージに変換したり、 FIPS メッセージを抑制したりできます。 詳細については、MSGEXIT の処理に関する関連参照と、関連タスクを参照してください。

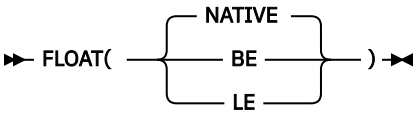
関連タスク
594 ページの『コンパイラー・メッセージの重大度のカスタマイズ』

関連参照
593 ページの『MSGEXIT の処理』

FLOAT

Float は、浮動小数点のデータ項目の表現形式を指定します。

FLOAT オプションの構文



デフォルト: FLOAT(NATIVE)

省略形: なし

プラットフォームのネイティブの浮動小数点表現形式を使用するには、FLOAT(NATIVE) を指定します。COBOL for Linux の場合、これはリトル・エンディアン 形式 (最小桁の数字が最下位アドレスに格納される) になります。

FLOAT(BE) は、COMP-1 および COMP-2 の各データ項目が常に IBM Z、つまりビッグ・エンディアン 形式 (最大桁の数字が最下位アドレスに格納される) で表現されることを示します。

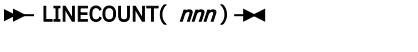
FLOAT(LE) は、COMP-1 および COMP-2 の各データ項目が常にリトル・エンディアン 形式 (最小桁の数字が最下位アドレスに格納される) で表現されることを示します。

関連参照
529 ページの『付録 B IBM Z ホスト・データ形式についての考慮事項』

LINECOUNT

LINECOUNT(*nnn*) は、コンパイル・リストの各ページに印刷する行数を指定する場合に使用します。ページ編集を抑止する場合には、LINECOUNT(0) を使用してください。

LINECOUNT オプションの構文



デフォルト: LINECOUNT(60)

省略形: LC

nnn は、10 から 255 の整数か、0 でなければなりません。

LINECOUNT(0) を指定すると、コンパイル・リストではページ替えが行われません。

コンパイラーは、タイトル用に *nnn* のうちの 3 行を使用します。 例えば、LINECOUNT(60) を指定すると、57 行のソース・コードが出力リストの各ページに印刷されます。

LIST

LIST コンパイラー・オプションは、ソース・コードのアセンブラー言語拡張のリストを作成する場合に使用します。

LIST オプションの構文

デフォルト: NOLIST

省略形: なし

PROCEDURE DIVISION でコーディングした *CONTROL (または *CBL) LIST、または NOLIST ステートメントはすべて無効です。コメントとして扱われます。

アセンブラー・リストは、名前がソース・プログラムと同じで、接尾部が .wlist のファイルに書き込まれます。

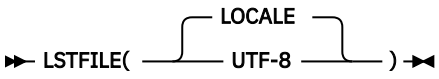
関連タスク
361 ページの『リストの入手』

関連参照
*CONTROL (*CBL) ステートメント (*COBOL for Linux on x86* 言語解説書)

LSTFILE

生成されたコンパイラー・リストを、有効なロケールで指定されたコード・ページでエンコードするには、LSTFILE(LOCALE) を指定します。 生成されたコンパイラー・リストを UTF-8 でエンコードするには、LSTFILE(UTF-8) を指定します。

LSTFILE オプションの構文



デフォルト: LSTFILE(LOCALE)

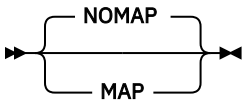
省略形: LST

関連参照
201 ページの『第 11 章 ロケールの設定』

MAP

DATA DIVISION で定義されている項目のリストを作成するには、MAP を使用します。

MAP オプションの構文



デフォルト: NOMAP

省略形: なし

出力には、以下の項目が含まれます。

- DATA DIVISION のマップ
- ネストされたプログラム構造マップ、およびプログラム属性
- プログラムの WORKING-STORAGE および LOCAL-STORAGE のサイズ

MAP 出力を制限したい場合は、DATA DIVISION で *CONTROL MAP または NOMAP ステートメントを使用してください。 *CONTROL NOMAP の後のソース・ステートメントは、*CONTROL MAP ステートメントによって出力が通常の MAP 形式に戻されない限り、リストには含められません。 次に例を示します。

```
*CONTROL NOMAP            *CBL NOMAP
    01  A                     01  A
    02  B                     02  B
*CONTROL MAP              *CBL MAP
```

MAP オプションが有効になっている場合は、ソース・コード・リストに MAP 報告書も組み込まれます。 圧縮 MAP 情報は、DATA DIVISION の WORKING-STORAGE SECTION、FILE SECTION、LOCAL-STORAGE SECTION、および LINKAGE SECTION のデータ名定義の右側に示されます。 XREF データと組み込み MAP 要約の両方が同じ行にある場合は、組み込み MAP 要約が先にリストされます。

365 ページの『例: MAP 出力』

関連概念
303 ページの『第 16 章 デバッグ』

関連タスク
361 ページの『リストの入手』

関連参照

*CONTROL (*CBL) ステートメント (*COBOL for Linux on x86* 言語解説書)

MAXMEM

MAXMEM を OPTIMIZE と一緒に使用して、コンパイラーが特定のメモリー集中の最適化のローカル・テーブルのために使用するメモリーの量を、 *size* KB に制限します。 特定の最適化に対して該当のメモリーが不十分な場合は、最適化の対象範囲が減らされます。

MAXMEM オプションの構文

デフォルト: MAXMEM(2048)

省略形: なし

値 -1 は、最適化ごとに、限界値のチェックなしで必要なだけのメモリーをとることを許可します。 コンパイルされるソース・ファイル、 ソース内のサブプログラムのサイズ、マシン構成、およびシステムでのワークロードによっては、この量が、使用可能なシステム・リソースを超える場合があります。

使用上の注意

- MAXMEM によって設定される限界値は、コンパイラー全体に対してではなく、特定の最適化に対するメモリー量です。 コンパイル処理の全体を通して必要なテーブルは、この限界値には影響しないか、組み込まれません。
- 限界値を大きく設定しても、コンパイラーが必要とするメモリーが少ないソース・ファイルのコンパイルにはマイナスの影響はありません。
- 最適化の対象範囲を制限しても、必ずしも結果のプログラムが低速になるわけではありません。単に、パフォーマンスを向上させるためのすべての方法を見つける前に、コンパイラーが終了することがあるだけです。
- 限界値を大きくしても、必ずしも結果のプログラムが高速になるわけではありません。単に、パフォーマンスを向上させるための方法が存在する場合に、コンパイラーがそれを見つけやすくなるだけです。

これは、コンパイルされるソース・ファイル、ソース内のサブプログラムのサイズ、マシン構成、およびシステムでのワークロードによっては、設定する限界値が高すぎるとページ・スペースが使い尽くされることがあります。 特に、MAXMEM(-1) を指定すると、コンパイラーが無制限の記憶量を使用しようとすることができ、最悪の場合マシンのリソースが使い尽くされることがあります。

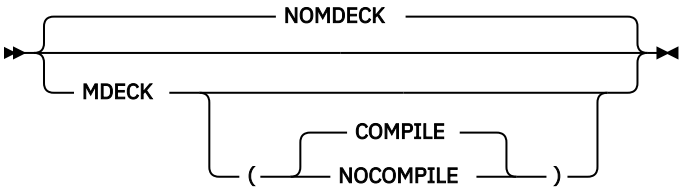
関連参照
275 ページの『OPTIMIZE』

MDECK

MDECK コンパイラー・オプションは、ライブラリー処理 (すなわち、COPY、BASIS、REPLACE、EXEC SQL INCLUDE ステートメントの結果) 後の更新済み入力ソースのコピーがファイルに書き込まれることを指定します。

MDECK 出力は、COBOL ソース・ファイルと同じ名前および接尾部 .dek を持つ、現行ディレクトリー内のファイルに書き込まれます。

MDECK オプションの構文



デフォルト: NOMDECK

省略形: NOMD | MD | MD(C | NOC)

オプション指定:

PROCESS (または CBL) ステートメントで MDECK オプションを指定することはできません。 指定できるのは、以下のいずれかの方法に限られます。

- cob2 コマンドでのオプションとして
- **COBOPT** 環境変数で
- 構成 (.cfg) ファイルの compopts 属性

サブオプション:

- MDECK(COMPILE) が有効である場合、ライブラリー処理および MDECK 出力ファイルの生成が完了した後、 正常にコンパイルが継続されますが、その際、コンパイルは COMPILE|NOCOMPILE オプションの 設定値に従って行われます。
- MDECK(NOCOMPILE) が有効である場合、構文検査が完了し、拡張ソース・プログラム・ファイルが書き込まれた後、コンパイルは終了します。 コンパイラーは、COMPILE オプションの 設定値に関係なく、コード生成をこれ以上行いません。

サブオプションなしの MDECKMDECK を指定した場合、MDECK(COMPILE)MDECK(COMPILE) が暗黙指定されます。

MDECK 出力ファイルの内容:

MDECK オプションを、EXEC CICS または EXEC SQL ステートメントを含むプログラムとともに使用する場合、これらの EXEC ステートメントは MDECK 出力にそのまま組み込まれます。ただし、SQL オプションを 使用してコンパイルを行うと、対応する EXEC SQL INCLUDE ステートメント が MDECK 出力において拡張されます。

CBL、PROCESS、*CONTROL、および *CBL カード・イメージは、MDECK 出力ファイルの適切な位置に渡されます。

バッチ・コンパイル (単一入力ファイル内に複数の COBOL ソース・プログラムが含まれている) の場合、完全な拡張ソースを含んでいる単一 MDECK 出力ファイルが作成されます。

SEQUENCE コンパイラー・オプション処理はすべて MDECK ファイル内に反映されます。

COPY ステートメントは、コメントとして MDECK ファイルに組み込まれます。

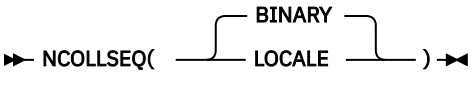
関連参照
229 ページの『構成ファイル内のスタンザ』

250 ページの『矛盾するコンパイラー・オプション』
293 ページの『第 14 章 コンパイラー指示ステートメント』

NCOLLSEQ

NCOLLSEQ は、クラス国別オペランドを比較するための照合シーケンスを指定します。

NCOLLSEQ オプションの構文



デフォルト: NCOLLSEQ(BINARY)

省略形: NCS(L|BIN|B)

16 進値の文字ペアを使用するには、NCOLLSEQ(BIN) を指定します。

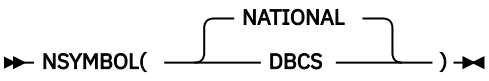
有効なロケール値と関連付けられた照合順序のアルゴリズムを 使用するには、NCOLLSEQ(LOCALE) を使用します。

関連タスク
194 ページの『2 つのクラス国別オペランドの比較』
207 ページの『ロケール付きの照合シーケンスの制御』

NSYMBOL

NSYMBOL オプションは、リテラルおよび PICTURE 節で使用される N 記号の解釈を制御し、国別処理や DBCS 処理が必要かどうかを指示します。

NSYMBOL オプションの構文



デフォルト: NSYMBOL(NATIONAL)

省略形: NS(NAT|DBCS)

NSYMBOL(NATIONAL) を指定した場合:

- USAGE 節のない、記号 N のみからなる PICTURE 節で定義されたデータ項目は、USAGE NATIONAL 節が指定されている場合のように扱われます。
- N". . ." または N'. . .' の形式のリテラルは、 国別リテラルとして扱われます。

NSYMBOL(DBCS) を指定した場合:

- USAGE 節のない、記号 N のみからなる PICTURE 節で定義されたデータ項目は、USAGE DISPLAY-1 節が指定されている場合のように扱われます。
- N". . ." または N'. . .' の形式のリテラルは、 DBCS リテラルとして扱われます。

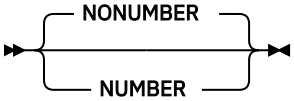
NSYMBOL(DBCS) オプションは、前リリースの IBM COBOL との互換性を提供します。
NSYMBOL(NATIONAL) オプションは、上記の言語エレメントの処理を、この点に関して 2002 COBOL 標準に準拠させます。

NSYMBOL(NATIONAL) は、Unicode データや を使用するアプリケーションの場合の推奨オプションです。

NUMBER

NUMBER コンパイラー・オプションは、ソース・コードの中に行番号があり、それらの番号がエラー・メッセージと SOURCE、MAP、LIST、および XREF のリストで必要な場合に使用してください。

NUMBER オプションの構文



デフォルト: NONUMBER

省略形: NUM|NONUM

NUMBER を要求すると、コンパイラーは、桁 1 から 6 に数字だけが含まれているかどうか、および番号が数字の照合シーケンスになっているかどうかを検査します。 (これに反して、SEQUENCE を使用すると、これらの桁の文字が EBCDIC 照合シーケンスになっているかどうかが検査されます。) 行番号が順序どおりになっていないことがわかると、コンパイラーは先行のステートメントの行番号より 1 だけ大きい値の行番号を割り当てます。 コンパイラーは、新規の値に 2 つのアスタリスクでフラグを立て、シーケンス・エラーを示すメッセージをリストに組み込みます。 シーケンス検査は、先行の行の新しく割り当てられた値に基づいて、次のステートメントから継続されます。

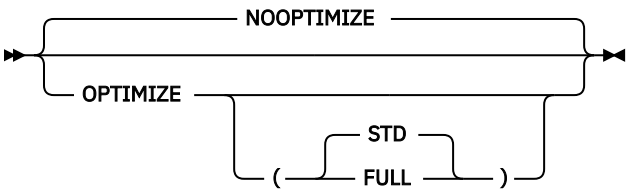
COPY ステートメントを使用する場合、NUMBER が有効なときは、ソース・プログラムの行番号とコピーブックの行番号が対応している必要があります。

ソース・コードの中に行番号がない場合や、コンパイラーにソース・コードの行番号を無視させる場合には、NONUMBER を使用してください。 NONUMBER が有効であると、コンパイラーは、ソース・ステートメントの行番号を生成し、それらの番号をリストで参照として使用します。

OPTIMIZE

OPTIMIZE は、オブジェクト・プログラムの実行時間を短縮するために使用します。 最適化によって、 オブジェクト・プログラムが使用するストレージの量を減らすこともできます。 実行される最適化には、 定数の伝搬、命令スケジューリング、および結果が使用されない計算の除去があります。

OPTIMIZE オプションの構文



デフォルト: NOOPTIMIZE

省略形: OPT|NOOPT

サブオプションなしで OPTIMIZE を指定すると、OPTIMIZE(STD) が有効になります。

FULL サブオプションは、OPT(STD) で実行される最適化に加えて、コンパイラーが DATA DIVISION から未参照のデータ項目を廃棄し、さらにこれらのデータ項目をそれぞれの VALUE 節の値に初期化するコードの生成を抑止するように要求します。 OPT(FULL) が有効であると、未参照のレベル 77 項目および基本レベル 01 項目がすべて破棄されます。 さらに、どの従属項目も参照されなければ、レベル 01 グループ項目も破棄されます。 削除された項目はリストの中で示されます。 MAP オプションが有効であれば、データ・マップ情報内の XXXXX の BL 番号は、そのデータ項目が破棄されたことを示します。

推奨: データベース・アプリケーションには、OPTIMIZE(FULL) を使用してください。 これによって、関連する COPY ステートメントに含まれる未使用の定数が除去されるため、大幅にパフォーマンスが向上する可能性があります。 ただし、データベース・アプリケーションが未使用のデータ項目に依存している場合には、以下の推奨を参照してください。

未使用データ項目: プログラムで未使用データ項目を意図的に利用している場合は、OPT(FULL) を使用しないでください。 従来は、次のような 2 つの方法が一般に使用されていました。

- 参照されたテーブルの後に未参照のテーブルを置き、2 番目のテーブルにアクセスするために最初のテーブルの範囲外添え字を使用する (以前の OS/VS COBOL プログラムで時折使用されていた手法)。 ご使用のプログラムがこの手法を使用するのかどうかを判別するには、SSRANGE コンパイラー・オプションと CHECK(ON) ランタイム・オプションを一緒に使用します。 この問題に対処するには、新しい COBOL の大きなテーブルのコーディング機能を使用して、テーブルを 1 つだけ使用します。
- 目印となるデータ項目を WORKING-STORAGE SECTION に置いて、プログラム・データの始めと終わりを識別できるようにする、あるいはそのデータを使用するライブラリー・ツール用のプログラムのコピーに印を付けてプログラムのバージョンを識別できるようにする方法。 この問題を解決するには、これらの項目を VALUE 節ではなく、PROCEDURE DIVISION ステートメントで初期化します。 この方法を使用すると、コンパイラーはこれらの項目が使用されているものと見なし、削除しません。

重大レベル以上のエラーが起こった場合、OPTIMIZE オプションはオフにされます。

関連概念
504 ページの『最適化』

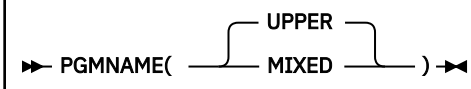
関連参照

250 ページの『矛盾するコンパイラー・オプション』
272 ページの『MAXMEM』

PGMNAME

PGMNAME オプションは、プログラム名および入り口点名の処理を制御します。

PGMNAME オプションの構文



デフォルト: PGMNAME(UPPER)

省略形: PGMN(LU|LM)

COBOL for OS/390® & VM との互換性を保つため、LONGMIXED および LONGUPPER もサポートされています。

LONGUPPER は、UPPER、LU、または U と省略することができ、LONGMIXED は、MIXED、LM、または M と省略することができます。

COMPAT: PGMNAME(COMPAT) を指定すると、PGMNAME(UPPER) が設定され、警告メッセージが戻されます。

PGMNAME オプションは、以下のコンテキストで使用される名前の処理を制御します。

- PROGRAM-ID 段落で定義されたプログラム名
- ENTRY ステートメントのプログラム入り口点名
- 以下におけるプログラム名参照:
  - ネストされたプログラムを参照する CALL ステートメント、静的にリンクされたプログラム、または共用ライブラリー

– 静的にリンクされたプログラム、または 共用ライブラリー を参照する SET プロシージャー・ポインターまたは関数ポインター・ステートメント
– ネストされたプログラムを参照する CANCEL ステートメント

PGMNAME(UPPER)

PGMNAME(UPPER) を使用する場合、PROGRAM-ID 段落で COBOL ユーザー定義語として指定されるプログラム名は、次のようなユーザー定義語に関する通常の COBOL 規則に従って いなければなりません。

- プログラム名の長さは最大 30 文字です。
- 名前で使用される文字はすべて英字、数字、ハイフン、または下線でなければなりません。
- 少なくとも 1 文字は英字にする必要があります。
- ハイフンを先頭文字や末尾文字として使用することはできません。
- 下線を先頭文字として使用することはできません。

定義または参照のいずれかで、プログラムをリテラルとして指定する場合は、次のようになります。

- プログラム名の長さは最高 160 文字まで。
- 名前で使用される文字はすべて英字、数字、ハイフン、または下線でなければなりません。
- 少なくとも 1 文字は英字にする必要があります。
- ハイフンを先頭文字や末尾文字として使用することはできません。
- 下線はどの位置でも使用できます。

外部プログラム名は、大文字に変換された英字で処理されます。

PGMNAME(MIXED)

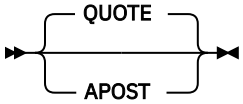
PGMNAME(MIXED) を使用する場合、プログラム名は、切り捨てられたり、変換されたり、 大文字への変換をされることなく、現状のまま処理されます。

PGMNAME(MIXED) を使用する場合、すべてのプログラム名定義は、プログラム名のリテラル形式を使用して、PROGRAM-ID 段落または ENTRY ステートメントで指定する必要があります。

APOST/QUOTE

APOST は、形象定数 [ALL] QUOTE または [ALL] QUOTES で 1 つ以上のアポストロフィ (') 文字を表したい場合に使用します。QUOTE は、形象定数 [ALL] QUOTE または [ALL] QUOTES で 1 つ以上の引用符 (") 文字を表したい場合に使用します。

APOST/QUOTE オプションの構文



デフォルト: QUOTE

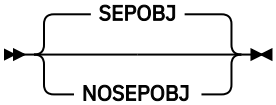
省略形: Q|APOST

区切り文字: APOST または QUOTE オプションが有効であるかどうかに関係なく、引用符 (") またはアポストロフィ (') のいずれかをリテラル区切り文字として使用できます。リテラルの開始の区切り文字として使用する区切り文字は、そのリテラルの終了の区切り文字としても使用しなければなりません。

SEPOBJ

SEPOBJ は、バッチ・コンパイル内で最外部にある各 COBOL プログラムを、 単一のオブジェクト・ファイルとしてではなく個別のオブジェクト・ファイルとして生成するかどうかを指定します。

SEPOBJ オプションの構文



デフォルト: SEPOBJ

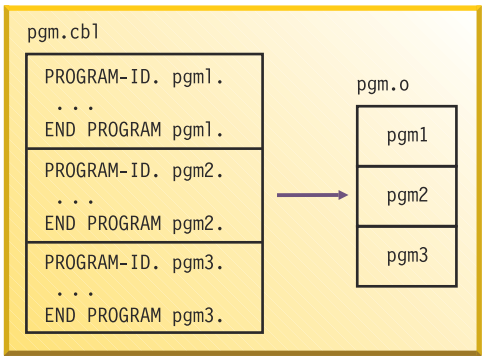
省略形: なし

バッチ・コンパイル

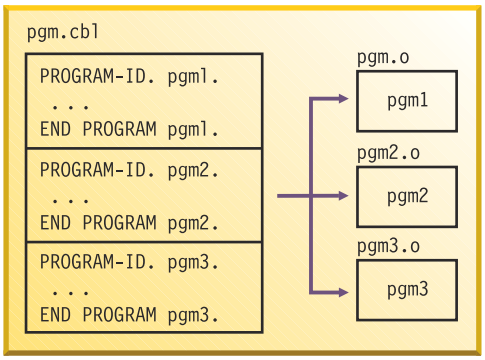
コンパイラーがバッチを 1 回呼び出すことで、複数の最外部プログラム (ネストされていないプログラム) が コンパイルされる場合は、コンパイラー・オプション SEPOBJ によって、 そのバッチ・コンパイルのオブジェクト・プログラム出力に対して生成されるファイルの数が決まります。

COBOL ソース・ファイル pgm.cbl に、pgm1、pgm2、pgm3 という名前の 3 つの COBOL 最外部プログラムが含まれているとします。 次の図に、オブジェクト・プログラム出力が 1 ファイル (NOSEPOBJ の場合) または 3 ファイルの場合 (SEPOBJ) のどちらで生成されるかを示します。

NOSEPOBJ を使用したバッチ・コンパイル



SEPOBJ を使用したバッチ・コンパイル



使用上の注意

- 上記の例の pgm2 または pgm3 が、別のプログラムからの CALL *identifier* によって呼び出される場合、85 COBOL 標準に準拠するには SEPOBJ オプションを使用する必要があります。

- NOSEPOBJ が有効な場合は、ソース・ファイルの名前に接尾部 .o を付けたものが、オブジェクト・ファイルの名前になります。 SEPOBJ が有効な場合は、PROGRAM-ID の名前に接尾部 .o を付けたものが、オブジェクト・ファイル の名前になります。
- PROGRAM-ID とオブジェクト・ファイルの名前が一致しない場合、CALL *identifier* で呼び出されるプログラムは、(PROGRAM-ID の名前ではなく) オブジェクト・ファイルの名前で参照されなければなりません。

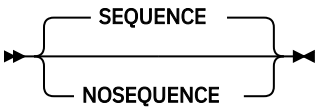
オブジェクト・ファイルには、当該プラットフォームおよびファイル・システムに対応した、有効なファイル名を付ける必要があります。

SEQUENCE

SEQUENCE を使用すると、コンパイラーは桁 1 から 6 を調べ、 ソース・ステートメントが ASCII 照合シーケンスに従って昇順に並んでいるかどうかを検査します。 昇順になっていないステートメントがあると、コンパイラーは診断メッセージを出します。

桁 1 から 6 がブランクの ソース・ステートメントはこのシーケンス検査には関与しないため、メッセージは出されません。

SEQUENCE オプションの構文



デフォルト: SEQUENCE

省略形: SEQ|NOSEQ

SEQUENCE オプションを有効にして COPY ステートメントを使用する場合、 ソース・プログラムのシーケンス・フィールドと、コピーブック・シーケンス・フィールドが対応している必要があります。

NUMBER と SEQUENCE を使用すると、シーケンス検査は、ASCII 照合シーケンスではなく、数字に従って行われます。

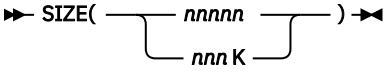
この検査と診断メッセージを抑止する場合は、NOSEQUENCE を使用してください。

関連タスク
308 ページの『行シーケンス問題の検出』

SIZE

SIZE は、コンパイラー・フロントエンドでコンパイルに使用できるようにする主記憶域の量を示す場合に使用します。 コンパイラー・フロントエンドとは、コードの生成や最適化の前に行われるコンパイルのフェーズのことです。

SIZE オプションの構文



デフォルト: 8388608 バイト (約 8 MB)

省略形: SZ

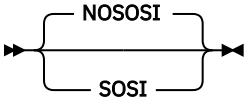
nnnnn には、少なくとも 800768 以上の 10 進数を指定します。

*nnn*K は、1 KB 単位で 10 進数を指定します。1 KB = 1024 バイトです。 受け入れられる最小値は 782K です。

SOSI

SOSI オプションは、コメント内、英数字、国別、および DBCS リテラル内、および DBCS ユーザー定義語内の値 X'1E' および X'1F' の扱いに影響を与えます。

SOSI オプションの構文



デフォルト: NOSOSI

省略形: なし

NOSOSI
NOSOSI を使用すると、値 X'1E' および X'1F' を持つ文字位置がデータ文字として扱われます。

NOSOSI は 85 COBOL 標準 に準拠しています。

SOSI
SOSI を使用すると、シフトアウト (SO) およびシフトイン (SI) 制御文字によって、COBOL ソース・プログラム内の ASCII DBCS 文字ストリングが区切られます。 SO 文字と SI 文字は、それぞれ X'1E' と X'1F' のエンコード値を持ちます。

SO 文字と SI 文字は、COBOL for Linux の COBOL ソース・コードには影響を与えません。ただし、リモート・ファイルが EBCDIC から ASCII に変換されるときのデータ処理を正しく行うために、これらの SO 文字と SI 文字がホスト DBCS の SO 文字と SI 文字のプレースホルダーとして機能する場合は例外です。

SOSI オプションが有効な場合は、COBOL for Linux の既存の COBOL 規則に加えて、次の規則が適用されます。

- すべての DBCS 文字ストリング (ユーザー定義語、DBCS リテラル、英数字リテラル、国別リテラル、およびコメント内) は、SO および SI 文字で区切る必要があります。
- ユーザー定義語には、DBCS 文字と SBCS 文字の両方を含めることはできません。
- DBCS ユーザー定義語の最大長は 14 DBCS 文字です。
- 2 バイトの小文字英字がユーザー定義語内で使用される場合は、それに対応する 2 バイトの大文字英字と同等になりません。
- DBCS ユーザー定義語には少なくとも 1 文字を含める必要がありますが、1 バイト表現に、その文字に対応する文字があってはなりません。
- A から Z、a から z、0 から 9、ハイフン (-)、および下線 (_) の 1 バイト文字の 2 バイト表現を、DBCS ユーザー定義語に含めることができます。 1 バイト表現でこれらの文字に適用される規則は、2 バイト表現でも適用されます。 例えば、ユーザー定義語では、ハイフンを先頭または末尾文字にすることはできず、下線を先頭文字にすることはできません。
- SOSI コンパイラー・オプションが有効な場合は、X'1E' または X'1F' 値を含む DBCS および国別リテラルに対して次の規則が適用されます。
  - X'1E' および X'1F' を使用した文字位置は、SO および SI 文字として扱われます。
  - X'1E' および X'1F' を使用した文字位置は、国別 16 進数表記の文字ストリングに組み込まれ、基本表記からは除去されました。
- SOSI コンパイラー・オプションが有効な場合は、X'1E' または X'1F' 値を含む英数字リテラルに対して次の規則が適用されます。
  - X'1E' および X'1F' を使用した文字位置は、SO および SI 文字として扱われます。
  - X'1E' および X'1F' を使用した文字位置は、16 進数表記の文字ストリングに組み込まれ、基本表記とヌル終了表記からは除去されました。

- 引用符で区切られた N-literal 内に DBCS の引用符を埋め込むには、DBCS の引用符 1 つを表現するために DBCS の引用符を連続して 2 個使用します。 N-literal が引用符で区切られている場合は、このリテラルに単一の DBCS 引用符を含めないでください。 この規則は、一重引用符にも適用されます。
- SHIFT-OUT および SHIFT-IN 特殊レジスターは、SOSI オプションが有効かどうかにかかわらず、X'0E' および X'0F' を使用して定義されます。

一般に、ホストの COBOL プログラムが SO および SI 文字のエンコード値に依存する場合は、Linux ワークステーションで同じ動作をしません。

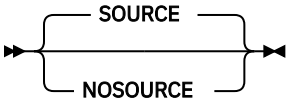
関連タスク
428 ページの『ASCII マルチバイト・ストリングと EBCDIC DBCS ストリングの違いの処理』

関連参照
文字ストリング (*COBOL for Linux on x86* 言語解説書)

SOURCE

SOURCE は、ソース・プログラムのリストを入手する場合に使用します。 このリストには、PROCESS または COPY ステートメントによって組み込まれたすべてのステートメントが入ります。

SOURCE オプションの構文



デフォルト: SOURCE

省略形: S|NOS

ソース・リストに組み込みメッセージが必要な場合は、SOURCE を必ず指定します。

コンパイラー出力リストにソース・コードを出したくない場合は、NOSOURCE を使用してください。

SOURCE 出力を制限したい場合は、PROCEDURE DIVISION で *CONTROL SOURCE または NOSOURCE ステートメントを使用してください。 Source *CONTROL NOSOURCE ステートメントの後のソース・ステートメントは、後続の *CONTROL SOURCE ステートメントによって出力が通常の SOURCE 形式に戻されない限り、リストには含められません。

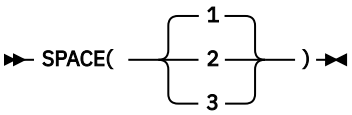
365 ページの『例: MAP 出力』

関連参照
*CONTROL (*CBL) ステートメント (*COBOL for Linux on x86* 言語解説書)

SPACE

SPACE は、ソース・コード・リストで 1 行送り、2 行送り、または 3 行送りを選択するために使用します。

SPACE オプションの構文



デフォルト: SPACE(1)

省略形: なし

SPACE が意味を持つのは、SOURCE コンパイラー・オプションが有効な場合だけです。

関連参照

281 ページの『SOURCE』

SPILL

このオプションは、レジスターの予備区域とは別に設定されるサイズ (KB) を指定します。 コンパイルするプログラムが非常に複雑であったり大きかったりする場合は、このオプションが必要になることがあります。

SPILL オプションの構文

デフォルト: SPILL(512)

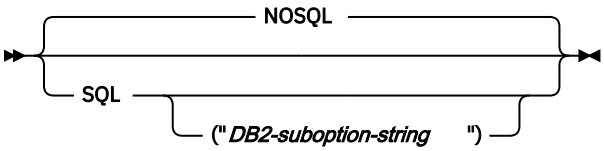
省略形: なし

予備サイズ *n* は 96 から 32704 までの任意の整数です。

SQL

SQL コンパイラー・オプションを使用すると、Db2 コプロセッサーを使用可能にし、Db2 サブオプションを指定できるようになります。COBOL ソース・プログラムに SQL ステートメントが含まれており、プログラムが Db2 プリコンパイラーで処理されていない場合には、SQL オプションを必ず指定しなければなりません。

SQL オプションの構文



デフォルト: NOSQL

省略形: なし

NOSQL が有効な場合は、ソース・プログラム内で検出された SQL ステートメントは 診断され、破棄されます。

Db2 サブオプションのストリングは、引用符または単一引用符を使用して区切ってください。

CBL または PROCESS ステートメントで、上記の構文を使用することができます。 cob2 コマンド で SQL オプションを 使用する場合、サブオプション・ストリング区切り文字として使用できるのは、 単一引用符 (') のみです: -q"SQL('*suboptions*')"。

注 : コンパイラーは、特定のシェル・スクリプト文字を以下のように解釈します。

- 等号 (=) は左括弧 (に解釈されます。
- コロン (:) は右括弧) に解釈されます。
- 下線 (_) は単一引用符 (') に解釈されます。

円記号 (¥) エスケープ文字を追加すれば、このような解釈を防ぐことができるため、文字をストリングで渡すことができます。 円記号 (¥) が (エスケープ文字ではなく) 円記号そのものを表すようにしたい場合は、円記号 (¥) を重ねて (¥¥) 使用します。

例えば、内蔵 Db2 コプロセッサーを操作して DEFERRED_PREPARE プリコンパイル・オプションを使用する場合は、 次のように SQL オプションを指定します。

```
SQL('... DEFERRED¥_PREPARE ...')
```

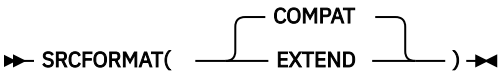
関連タスク
377 ページの『第 17 章 Db2 環境用のプログラミング』
379 ページの『SQL オプションを使用したコンパイル』
380 ページの『Db2 サブオプションの分離』

関連参照
250 ページの『矛盾するコンパイラー・オプション』

SRCFORMAT

SRCFORMAT を使用すると、COBOL ソースが 72 桁の固定ソース形式、または 252 桁の拡張ソース形式 (Extended Source) のどちらに準拠しているかを指示できます。

SRCFORMAT オプションの構文



デフォルト: SRCFORMAT(COMPAT)

省略形: SF(C|E)

SRCFORMAT(COMPAT) は、プライマリー・コンパイル入力 (primary compilation input)、および組み込まれている COPY テキストの各ソース行が 72 桁で終わることを指示します。ソース行が 72 バイトよりも短い場合は、スペース文字が最大 72 バイトまでソース行に論理的に追加されます。 ソース行が 72 バイトよりも長い場合は、最初の 72 バイトのみがプログラム・ソースとして使用されます。 (バイト 73 から 80 が存在する場合は、シリアル番号が含まれているものと見なされます。これらはコンパイラー・リストに出力されますが、それ以外は無視されます。)

SRCFORMAT(EXTEND) は、プライマリー・コンパイル入力 (primary compilation input)、および組み込まれている COPY テキストの各ソース行が 252 桁で終わることを指示します。 ソース行が 252 バイトよりも短い場合は、スペース文字が最大 252 バイトまでソース行に論理的に追加されます。 ソース行が 252 バイトよりも長い場合は、最初の 252 バイトのみがプログラム・ソースとして使用され、残りは無視されます。 (拡張ソース形式 (Extended Source) では、シリアル番号の指定はありません。)

どちらの形式でも、1 から 6 桁目はシーケンス番号として解釈されます。

オプションの指定: PROCESS (または CBL) ステートメントで SRCFORMAT オプションを 指定することはできません。 指定できるのは、以下のいずれかの方法に限られます。

- cob2 コマンドのオプションとして
- COBOPT 環境変数
- 構成 (.cfg) ファイルの compopts 属性

ソース変換ユーティリティー scu を使用すると、IBM 以外または自由形式の COBOL ソースを変換して、COBOL for Linux でコンパイルできるようになります。 scu 機能の要約を参照するには、コマンド scu -h を入力してください。 詳細については、scu の man ページを参照するか、または適切な関連参照を参照してください。

制約事項 : 拡張ソース形式 (Extended Source) は、スタンドアロン Db2 プリコンパイラーまたは分離型 CICS 変換プログラムと互換性がありません。

関連参照

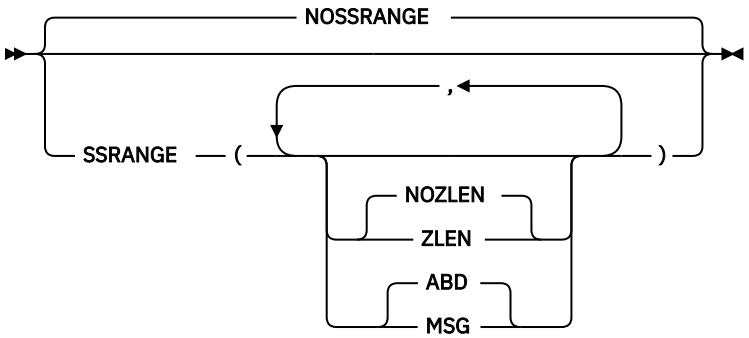
229 ページの『構成ファイル内のスタンザ』

232 ページの『cob2 オプション』
参照形式 (*COBOL for Linux on x86* 言語解説書)
ソース変換ユーティリティー (scu) (*COBOL for Linux on x86* 言語解説書)

SSRANGE

範囲外のストレージ参照を検査するコードを生成するには、SSRANGE を使用します。

SSRANGE オプションの構文



デフォルト: NOSSRANGE

SSRANGE のみが指定されている場合のサブオプションのデフォルトは NOZLEN,ABD です。

省略形: SSR|NOSSR

SSRANGE は、添え字 (ALL 添え字を含む) または指標が、関連するテーブルの領域外の区域を参照しようとしているかどうかを検査するコードを生成します。 各添え字または指標の妥当性は、個別に検査されることはありません。 代わりに、テーブルの範囲外を参照しないように、有効アドレスが検査されます。

サブオプションなしで指定された SSRANGE は、SSRANGE(NOZLEN,ABD) を指定したものとして受け入れられます。

注 : SSRANGE オプションが有効になっている場合は、範囲検査がコンパイラーによって生成され、実行時には必ずその検査が実施されます。 ランタイム・オプション CHECK(OFF) を指定しても、コンパイルされた範囲検査を実行時に無効にすることはできません。

定義された最大長の範囲内で参照を行うようにするために、可変長項目も検査されます。

次の点を確認するために、参照変更式が検査されます。

- 開始位置が 1 以上である。
- 開始位置が、サブジェクト・データ項目の現在の長さより大きくない。
- 開始位置と長さの値 (指定されている場合) がサブジェクト・データ項目の終わりを超えた区域を参照していない。
- 長さの値 (指定されている場合) が 1 以上である。

ZLEN サブオプションおよび NOZLEN サブオプションは、コンパイラーによる参照変更長の検査方法を制御します。

- ZLEN が有効になっている場合、コンパイラーは、参照変更長がゼロ以上になるようにコードを生成します。 ゼロ長の参照変更が指定されると、実行時に SSRANGE エラーが出力されません。
- NOZLEN が有効になっている場合、コンパイラーは、参照変更長が 1 以上になるようにコードを生成します。ゼロ長の参照変更が指定されると、実行時に SSRANGE エラーが発生します。 これは、旧バージョンの COBOL での SSRANGE の動作方法と互換性があります。

サブオプション MSG および ABD は、範囲検査が失敗したときの COBOL プログラムの実行時動作を制御します。

- MSG が有効になっているときに範囲検査が失敗すると、ランタイム警告メッセージが発行されます。 つまり、プログラムの実行は継続され、他の範囲外状態も特定される可能性があります。
- ABD が有効になっているときに範囲検査が失敗すると、最初の範囲外状態でランタイム・エラー・メッセージが発行され、プログラムが異常終了します。 その次の潜在的な範囲外状態を見つけるには、最初の範囲外状態を修正し、プログラムを再コンパイルして再実行します。 他のすべての潜在的な範囲外状態を特定するには、このプロセスを何度も繰り返さなければならない可能性があります。

無制限グループまたはその従属項目の場合、検査は参照変更式に対してのみ行われます。 無制限グループに従属するテーブルに対する添字付きまたは索引付きの参照は、検査されません。

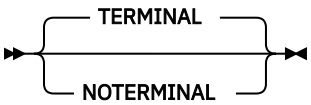
関連概念
100 ページの『参照修飾子』

関連タスク
308 ページの『有効範囲の検査』

TERMINAL

TERMINAL を使用すると、進行メッセージと 診断メッセージをディスプレイ装置に送ることができます。

TERMINAL オプションの構文



デフォルト: TERMINAL

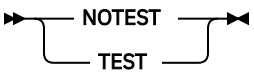
省略形: TERM|NOTERM

この追加の出力が不要な場合は、NOTERMINAL を使用してください。

TEST

TEST は、デバッガーがシンボリック・ソース・レベルのデバッグを実行できるようにするシンボルおよびステートメントの情報が含まれる オブジェクト・コードを生成する場合に使用します。

TEST オプションの構文



デフォルト: NOTEST

省略形: なし

デバッグ情報が入ったオブジェクト・コードを生成しない場合は、NOTEST を使用します。 NOTEST を使用してコンパイルされたプログラムは、デバッガーで実行されますが、デバッグ・サポートは限られています。

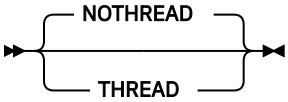
WITH DEBUGGING MODE 節を使用した場合には、TEST オプションはオフになります。 TEST は、オプション・リストに表示されますが、診断メッセージが出されて、競合のため TEST が無効である旨が通知されます。

関連タスク
311 ページの『IBM Debug for Linux on x86 を使用したデバッグ』

THREAD

THREAD オプションは受け入れられて無視されます。COBOL プログラムで、複数のスレッドがある実行単位で実行可能にする必要があることを指定する必要はなくなりました。

THREAD オプションの構文



デフォルト: NOTHREAD

省略形: なし

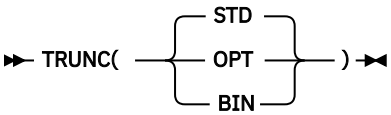
関連タスク

225 ページの『コマンド行からのコンパイル』

TRUNC

TRUNC は、バイナリー・データが移動および算術演算時に切り捨てられる方法に影響を与えます。

TRUNC オプションの構文



デフォルト: TRUNC(STD)

省略形: なし

TRUNC は、COMP-5 データ項目には効力を持ちません。COMP-5 項目は、TRUNC サブオプションの指定に関係なく、TRUNC(BIN) が有効である場合と同様に処理されます。

TRUNC(STD)
TRUNC(STD) は、MOVE ステートメントおよび算術式の中の USAGE BINARY 受信フィールドにのみ適用されます。 TRUNC(STD) が有効であると、算術式の最終結果または MOVE ステートメント中の送信フィールドは、BINARY 受信フィールド の PICTURE 節の桁数に切り捨てられます。

TRUNC(OPT)
TRUNC(OPT) はパフォーマンス・オプションです。 TRUNC(OPT) が有効であると、コンパイラーは、データが MOVE ステートメントおよび算術式にある USAGE BINARY 受信フィールドの PICTURE の指定に従うものと想定します。 結果は最適な方法で処理され、PICTURE 節の中の桁数か、またはストレージ内の 2 進数フィールドのサイズ (ハーフワード、フルワード、またはダブルワード) に切り捨てられます。

ヒント: TRUNC(OPT) オプションを使用するのは、2 進数区域に移動されるデータが、2 進数項目に対する PICTURE 節で定義された値よりも高い精度の値にならないことが確実である場合に限定してください。 そうしないと、結果は予測できません。 この切り捨ては、想定される最も効果的な方法で実行されます。そのため、結果は、生成される特定のコード・シーケンスに左右されます。 特定のステートメントに対して生成されたコード・シーケンスを見なければ、切り捨ての予想は不可能です。

TRUNC(BIN)
TRUNC(BIN) オプションは、USAGE BINARY データを処理するすべての COBOL 言語に適用されます。TRUNC(BIN) が有効な場合、すべての 2 進数項目 (USAGE COMP、COMP-4、または BINARY) は、固有

ハードウェア 2 進数項目として、すなわち、それぞれが個々に USAGE COMP-5 と宣言されたものとして処理されます。

- BINARY 受信フィールドは、ハーフワード、フルワード、またはダブルワード境界でのみ切り捨てられます。
- BINARY 送信フィールドは、受け取り側が数値であれば、ハーフワード、フルワード、またはダブルワードとして処理されます。受け取り側が数値でない場合、TRUNC(BIN) は効力を持ちません。
- フィールドの全 2 進内容が重要です。
- DISPLAY は切り捨てを行わずに、2 進フィールドの内容全体を変換します。

推奨事項: 他のプロダクトによって設定される 2 進値を使用するプログラムの場合、推奨オプションは TRUNC(BIN) です。 他のプロダクト (Db2 および C/C++、 など) は、COBOL の 2 進数データ項目に、データ項目の PICTURE 節に従わない値を入れることがあります。データが BINARY データ項目用の PICTURE 節に矛盾しない場合は、CICS プログラムで TRUNC(OPT) を使用することができます。

USAGE COMP-5 には、個々のデータ項目に TRUNC(BIN) の性質を適用する効果があります。 したがって、すべての 2 進数データ項目に対して TRUNC(BIN) を 使用することによるパフォーマンス上のオーバーヘッドは、非 COBOL プログラムまたは 他のプロダクトやサブシステムに渡されるデータ項目など、一部の 2 進数データ項目に のみ COMP-5 を指定することによって回避できます。 COMP-5 の使用は、どの TRUNC サブオプションが有効であっても影響を受けません。

VALUE 節における大きなリテラル: コンパイラー・ オプション TRUNC(BIN) を使用する場合、2 進数データ項目 (COMP、COMP-4、 または BINARY) 用の VALUE 節に指定された数字リテラルは、PICTURE 節の 9 の 数により暗黙指定される値に制限されることはなく、通常、固有 2 進表現 (2、4、 または 8 バイト) の容量までの大きさの値を持つことができます。

注 : TRUNC(BIN) と NUMCHECK(BIN) が両方とも有効で、エラー・メッセージか異常終了が生成される際に、パフォーマンスを改善するために後で TRUNC(STD|OPT) に切り替えようとしている場合は、データを訂正する必要があります。切り替えない場合は、NUMCHECK(BIN) をオフにして、アプリケーションの実行時間を減らし、エラー・メッセージや異常終了が生成されないようにすることができます。

TRUNC の例 1

```
01  BIN-VAR     PIC S99 USAGE BINARY.
. . .
    MOVE 123451 to BIN-VAR
```

次の表に、MOVE ステートメント後のデータ項目の値を示します。

データ項目	10 進数	16 進数	表示
送り出し側	123451	3B|E2|01|00	123451
受け取り側 TRUNC(STD)	51	33|00	51
受け取り側 TRUNC(OPT)	-7621	3B|E2	2J
受け取り側 TRUNC(BIN)	-7621	3B|E2	762J

ハーフワードのストレージが BIN-VAR に割り振られます。 プログラムが TRUNC(STD) オプションでコンパイルされた場合は、この MOVE ステートメントの結果は 51 で、フィールドは、PICTURE 節に適合するように切り捨てられます。

プログラムが TRUNC(BIN) オプションでコンパイルされた場合、MOVE ステートメントの結果は -7621 です。 このような異常に見える結果になるのは、非ゼロの高位桁が切り捨てられたためです。 ここでは、生成されたコード・シーケンスは、下位のハーフワード量 X'E23B' を受け取り側に移動させるだけです。 切

り捨てられた新しい値はオーバーフローして 2 進数ハーフワードの符号ビットになるため、値が負の数になります。

123451 は BIN-VAR の PICTURE 節より高い精度を持つため、この MOVE ステートメントを TRUNC(OPT) オプションでコンパイルしてはなりません。 TRUNC(OPT) を使用した場合も、結果は -7621 になります。これは、10 進数の切り捨てを行わないことによって、最高のパフォーマンスが得られたためです。

TRUNC の例 2

```
01  BIN-VAR     PIC 9(6)  USAGE BINARY
. . .
    MOVE 1234567891 to BIN-VAR
```

次の表に、MOVE ステートメント後のデータ項目の値を示します。

データ項目	10 進数	16 進数	表示
送り出し側	1234567891	D3|02|96|49	1234567891
受け取り側 TRUNC(STD)	567891	53|AA|08|00	567891
受け取り側 TRUNC(OPT)	567891	53|AA|08|00	567891
受け取り側 TRUNC(BIN)	1234567891	D3|02|96|49	1234567891

TRUNC(STD) を指定すると、送り出しデータは BINARY 受け取り側の PICTURE 節に適合するように、6 桁の整数に切り捨てられます。

TRUNC(OPT) を指定すると、コンパイラーは送り出しデータの精度が BINARY 受け取り側の PICTURE 節の精度よりも大きくないと想定します。 この場合、最も効率のよいコード・シーケンスは、TRUNC(STD) が指定されているものとして切り捨てを行うことです。

TRUNC(BIN) を指定すると、BIN-VAR に割り振られた 2 進数フルワードにすべての送り出しデータが収まるため、切り捨ては行われません。

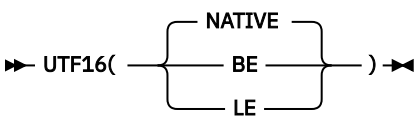
関連概念
39 ページの『数値データの形式』

関連参照
VALUE 節 (*COBOL for Linux on x86* 言語解説書)

UTF16

UTF16 は、UTF-16 データ項目の表現形式を指定します。

UTF16 オプションの構文



デフォルト: UTF16(NATIVE)

省略形: なし

プラットフォームのネイティブの UTF-16 表現形式を使用するには、UTF16(NATIVE) を指定します。 COBOL for Linux の場合、これはリトル・エンディアン 形式 (最小桁の数字が最下位アドレスに格納される) になります。

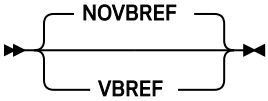
UTF16(BE) は、UTF-16 のデータ項目が常に IBM Z、つまりビッグ・エンディアン 形式 (最大桁の数字が最下位アドレスに格納される) で表現されることを示します。

UTF16(LE) は、UTF-16 のデータ項目が常にリトル・エンディアン 形式 (最小桁の数字が最下位アドレスに格納される) で表現されることを示します。

VBREF

VBREF は、ソース・プログラムの中で使用されるすべてのステートメントと、これらのステートメントが使用されている行番号の間の相互参照を入手するために使用します。また、VBREF はプログラムの中でそれぞれのステートメントが使用された回数の合計も出します。

VBREF オプションの構文



デフォルト: NOVBREF

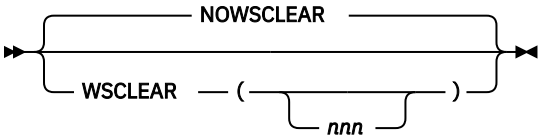
省略形: なし

コンパイルの効率を高める場合は、NOVBREF を使用してください。

WSCLEAR

WSCLEAR を使用すると、初期化時に、プログラムの WORKING-STORAGE の EXTERNAL 以外のデータ項目をクリアして 2 進ゼロにすることができます。 ストレージは、VALUE 文節が適用される前にクリアされます。

WSCLEAR オプションの構文



デフォルト: NOWSCLEAR

省略形: なし

ストレージのクリア処理をバイパスするには、NOWSCLEAR を使用します。

nnn は 0 から 255 までの任意の整数です。 サブオプションを指定しない WSCLEAR は、WSCLEAR(*0*) と同じです。

WSCLEAR オプションは、コマンド・ラインまたは COBOL ステートメントのいずれかで指定できます。 ただし、COBOL ステートメントで指定された WSCLEAR オプションは、コマンド・ラインで指定されたオプションよりも優先されます。

- コマンド・ラインで WSCLEAR を *nnn* サブオプションとともに指定するには、フォーマット -qwsclear*(nnn)* を使用します。 この場合、コマンド・プロセッサーは 0 から 255 の範囲内にある文字のみをスキャンし、範囲外の他の文字は無視します。 エラー・メッセージは出されません。

  例えば、-qwsclear*(99999)* を指定する場合、コマンド・プロセッサーは WSCLEAR*(99)* のみを取ります。

- COBOL ステートメントで WSCLEAR を *nnn* サブオプションとともに指定するには、フォーマット WSCLEAR*(nnn)* を使用します。

WSCLEAR(*nnn*) を指定する場合、*nnn* によって表されるバイト値は、WORKING-STORAGE データの各バイトを初期化して特定の値にするために使用されます。 これは VALUE 属性が指定されていないデータ項目にのみ適用されます。

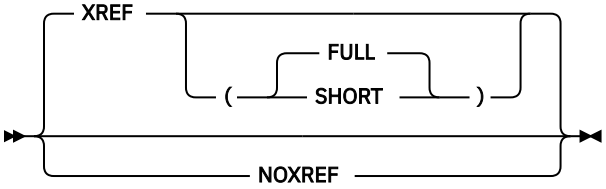
パフォーマンスの 考慮事項: WSCLEAR を使用するときに、オブジェクト・プログラムのサイズやパフォーマンスが懸念される 場合は、OPTIMIZE(FULL) も 使用します。 これにより、DATA DIVISION から未参照のデータ項目をすべて除去するよう コンパイラーに命令が出されるため、初期化の時間が短縮されます。

関連参照
275 ページの『OPTIMIZE』

XREF

XREF は、ソート済みの相互参照リストを作成するために使用します。

XREF オプションの構文



デフォルト: XREF(FULL)

省略形: X|NOX

XREF、XREF(FULL)、または XREF(SHORT) を選択できます。 サブオプションを付けないで XREF を指定すると、XREF(FULL) が有効になります。

リストのセクションには、プログラム内で参照されるすべてのプログラム名、データ名、およびプロシージャー名、およびそれらの名前が定義されている行番号が表示されます。 外部プログラム名が識別されます。

369 ページの『例: XREF 出力: データ名相互参照』
370 ページの『例: XREF 出力: プログラム名相互参照』

また、関連コピーブックを取得したファイルでプログラム内の COPY または BASIS ステートメントを相互参照するセクションも含まれています。

370 ページの『例: XREF 出力: COPY/BASIS 相互参照』

ロケール設定によって指定された照合シーケンスの順序で、名前はリストされます。 名前が 1 バイト文字であっても、マルチバイト文字 (DBCS など) を含んでいても、この順序が使用されます。

XREF と SOURCE を使用した場合は、データ名とプロシージャー名の相互参照情報が元のソースと同じ行に印刷されます。 行番号参照またはその他の情報は、リスト・ページの右側に表示されます。 組み込み関数を参照するソース行の右側には、IFN という文字と、その関数の引数が定義されている場所の行番号が印字されます。 組み込み参照に含められた情報によって、ID が未定義であるか (UND)、 複数回定義されているか (DUP)、項目が暗黙定義であるか (IMP) (特殊レジスターや形象定数など)、 プログラム名が外部プログラム名であるか (EXT) がわかります。

XREF と NOSOURCE を使用すると、ソート済みの相互参照リストだけが得られます。

XREF(SHORT) は、相互参照リスト内の明示的に参照されたデータ項目だけを印刷します。 XREF(SHORT) は、マルチバイトデータ名とプロシージャー名、および単一バイトの名前に適用されます。

NOXREF を使用すると、このリストは抑止されます。

使用上の注意

- MOVE CORRESPONDING ステートメントで使用されるグループ名は、XREF リストに入れられます。 それらのグループの基本名もリストされています。
- データ名の XREF リストでは、文字 M が前に付いている行番号は、そのデータ項目がその行のステートメントによって明示的に変更されたことを示しています。
- XREF リストは追加のストレージを使用します。

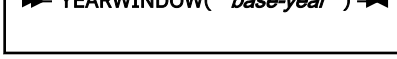
関連概念
303 ページの『第 16 章 デバッグ』

関連タスク
361 ページの『リストの入手』

YEARWINDOW

YEARWINDOW を使用すると、COBOL コンパイラーによるウィンドウ表示日付フィールド処理に適用する 100 年ウィンドウ (世紀ウィンドウ) の最初の年を指定することができます。

YEARWINDOW オプションの構文



デフォルト: YEARWINDOW(1900)

省略形: YW

base-year は、世紀ウィンドウの最初の年を表します。 次のいずれかの値で指定します。

- 1900 から 1999 の間の符号なし 10 進整数

  符号なし整数は固定ウィンドウの開始年号を指定します。 例えば、YEARWINDOW(1930) は 1930 から 2029 年の世紀ウィンドウを指定します。

- -1 から -99 の負の整数

  負の整数は、スライディング・ウィンドウを示します。 ウィンドウの最初の年は、現在の年に負の整数を加えて計算されます。 例えば、YEARWINDOW(-80) は、世紀ウィンドウの最初の年がプログラム実行年より 80 年前であることを指定します。

使用上の注意

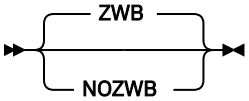
- YEARWINDOW オプションは、DATEPROC オプションも有効でない限り、効力を持ちません。
- 実行時には、次の 2 つの条件が真でなければなりません。
  - 世紀ウィンドウの開始年号が 1900 年代の年号である。
  - 現在の年号がコンパイル単位の世紀ウィンドウ内にある。

  例えば、現在の年号が 2010 年で、DATEPROC オプションが有効な場合に、YEARWINDOW(1900) というオプションを指定すると、プログラムは終了し、エラー・メッセージが出されます。

ZWB

ZWB を使用してコンパイルすると、コンパイラーは、実行時に符号付きゾーン 10 進数 (DISPLAY) フィールドを英数字基本フィールドと比較する前に、 そのフィールドから符号を除去します。

ZWB オプションの構文



デフォルト: ZWB

省略形: なし

ゾーン 10 進数項目がスケール項目である場合 (すなわち、記号 P をその PICTURE ストリング内に含んでいる場合)、比較でその 10 進数項目を使用しても ZWB の影響を受けません。 そのような項目では常に、英数字フィールドとの比較が行われる前に符号が除去されます。

ZWB はプログラムの実行方法に影響します。 同一の COBOL プログラムでも、このオプションの設定に応じて異なる結果が生成されることがあります。

NOZWB は、入力数字フィールドで SPACES をテストする場合に使用します。

第 14 章 コンパイラー指示ステートメント

プログラムのコンパイルを指示するには、いくつかのコンパイラー指示ステートメントが役に立ちます。

以下のコンパイラー指示ステートメントがあります。

BASIS ステートメント
この拡張ソース・プログラム・ライブラリー・ステートメントは、コンパイルのソースとして、完全な COBOL プログラムを提供します。形式化および処理の規則については、COPY ステートメントの *text-name* の説明を参照してください。

***CONTROL（*CBL）ステートメント**
このコンパイラー指示ステートメントは、出力の作成を抑制するかまたは可能にするかを選択します。キーワードの *CONTROL と *CBL は同義語です。

CALLINTERFACE 指示
このコンパイラー指示は呼び出しのインターフェース規約を指定し、引数記述子を生成するかどうかを示します。 >>CALLINTERFACE を使用して指定された規約は、別の >>CALLINTERFACE 指定が行われるまで有効です。 >>CALLINT は、>>CALLINTERFACE の略語です。

>>CALLINTERFACE を使用できるのは、PROCEDURE DIVISION の中だけです。

>>CALLINTERFACE ディレクティブの構文と使用法は、CALLINT コンパイラー・オプションと類似しています。 例外は次のとおりです。

- この指示構文には括弧が含まれません。
- この指示は、後述のように、選択した呼び出しに適用できます。
- この指示構文には、キーワード DESCRIPTOR とその変形が含まれます。

サブオプションなしで >>CALLINT を指定すると、使用される呼び出し規約が CALLINT コンパイラー・オプションで区切られます。

DESCRIPTOR のみ: >>CALLINT ディレクティブは、以下の形式の場合以外はコメントとして扱われます。

- >>CALLINT SYSTEM DESCRIPTOR、または同等に >>CALLINT DESCRIPTOR
- >>CALLINT SYSTEM NODESCRIPTOR、または同等に >>CALLINT NODESCRIPTOR

これらのディレクティブは、DESCRIPTOR のオン/オフを切り替えます。SYSTEM は無視されます。

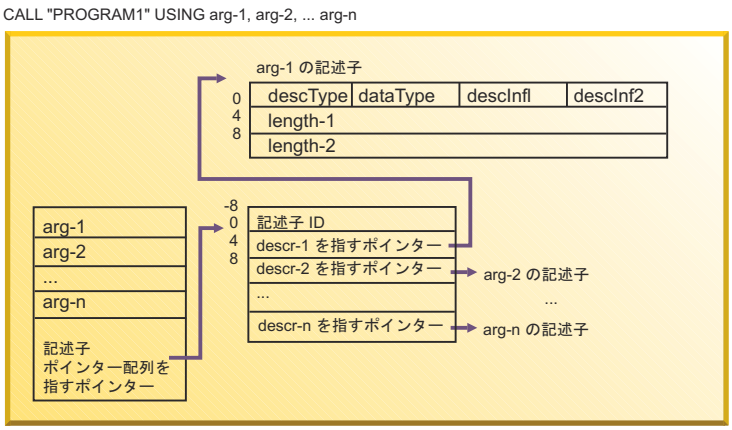
>>CALLINT 指示を指定する場所は、COBOL プロシージャー・ステートメントが指定可能な場所であればどこでも構いません。 例えば、次の構文は有効です。

```
MOVE 3 TO
>>CALLINTERFACE SYSTEM
RETURN-CODE.
```

>>CALLINT の影響は、現行のプログラムに限定されます。 ネストされたプログラム、または同じバッチ内でコンパイルされたプログラムは、>>CALLINT コンパイラー指示で指定された規約ではなく、CALLINT コンパイラー・オプションで指定された呼び出し規約を継承します。

>>CALLINT SYSTEM DESCRIPTOR を使用して呼び出されるルーチンを記述する場合、引数受け渡しの仕組みは次のようになります。

© Copyright IBM Corp. 2021



pointer to descr-n
特定の引数に対する記述子を指します。引数に対する記述子が存在しない場合は 0 になります。

descriptor-ID
このバージョンの記述子を識別するには、COBDESC0 に設定します。これにより、記述子の入力形式が将来変更される場合にも対応できるようになります。

descType
PICTURE X(*n*) を使用した USAGE DISPLAY、あるいは PICTURE G(*n*) または N(*n*) を使用した USAGE DISPLAY-1 の基本データ項目の場合は、X'02' (descElmt) に設定します。 それ以外 (数値フィールド、構造体、テーブル) の場合はすべて、X'00' に設定します。

dataType
次のように設定します。

- descType = X'00' の場合: dataType = X'00'
- descType = X'02' で、USAGE が DISPLAY の場合: dataType = X'02' (typeChar)
- descType = X'02' で、USAGE が DISPLAY-1 の場合: dataType = X'09' (typeGChar)

descInf1
常に X'00' に設定します。

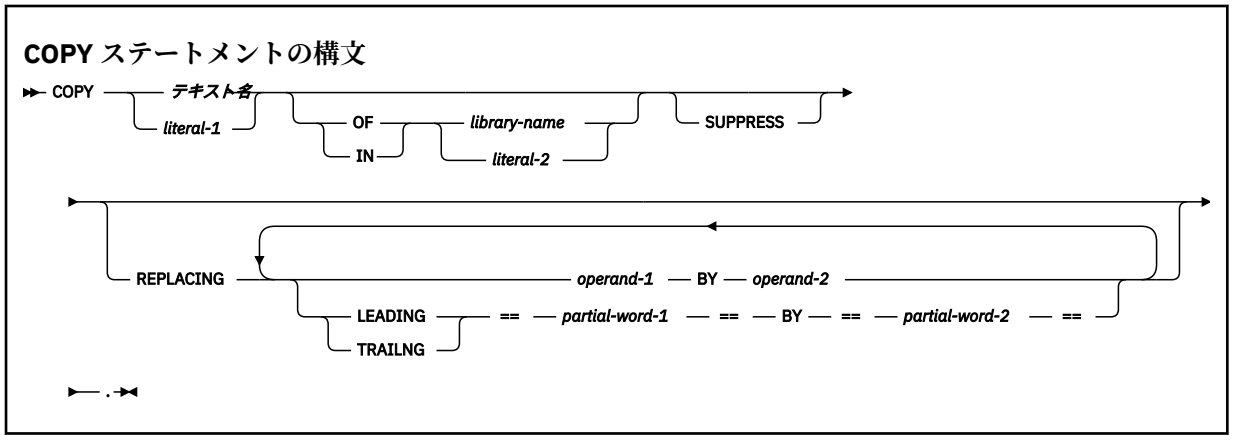
descInf2
次のように設定します。

- descType = X'00' の場合: descInf2 = X'00'
- descType = X'02'の場合:
  - CHAR(EBCDIC) オプションが有効で、USAGE 文節内の NATIVE オプションで引数が定義されていない場合: descInf2 = X'40'
  - それ以外の場合: descInf2 = X'00'

length-1
引数記述子には、固定長引数の引数長か、可変長項目の現行の長さが入ります。

length-2
引数が可変長項目の場合は、その引数の最大長を示します。 固定長引数の場合、length-2 と length-1 は等しくなります。

COPY ステートメント



このコンパイラー指示ステートメントは、事前に作成されたテキストを COBOL プログラムに入れます。

text-name も *library-name* もプログラム内で固有である必要はありません。これらは、プログラム内の他のユーザー定義語と同じにすることができます。

事前に作成されたテキストを含む *text-name* (コピーブックの名前) を指定する必要があります。例えば、COPY *my-text* のようになります。 *text-name* に修飾子として *library-name* を付けることができます。例えば、COPY *my-text* of *inventory-lib* のようになります。 *text-name* が修飾されない場合は、SYSLIB の *library-name* が想定されます。

library-name

library-name をリテラルとして指定した場合、リテラルの内容は実際のパスとして扱われます。 ユーザー定義語として *library-name* を指定した場合は、名前が環境変数として使用され、この環境変数の値がコピーブックの位置を指定するパスに使用されます。 複数のパス名を指定するには、各パス名をコロン (:) で区切ります。

library-name を指定しない場合、このパスは『*text-name*』に説明するとおりに使用されます。

text-name

text-name をリテラルとして指定した場合、リテラルの内容は実際のパスとして扱われます。 *text-name* をユーザー定義語として指定した場合、処理は *text-name* に対応する環境変数が設定されているかどうかによって異なります。 環境変数が設定されている場合、環境変数の値はコピーブックのファイル名 (およびパス名) として使用されます。

次の 3 つの条件がすべて満たされる場合は、 *text-name* が絶対パスとして扱われます。

- *library-name* は使用されません。
- *text-name* がリテラル、または環境変数である。
- 最初の文字が '/' である。

例えば、次のパスは絶対パスとして扱われます。

```
COPY "/mycpylib/mytext.cpy"
```

text-name に対応する環境変数が設定されていない場合、コピーブックは次の名前で検索されます。

1. 接尾部が .cpy の *text-name*
2. 接尾部が .cbl の *text-name*
3. 接尾部が .cob の *text-name*
4. 接尾部のない *text-name*

例えば、COPY MyCopy は次の順序で検索を行います。

1. MYCOPY.cpy (前述のとおり、指定されたすべてのパス内)

2. MYCOPY.cbl (前述のとおり、指定されたすべてのパス内)
3. MYCOPY.cob (前述のとおり、指定されたすべてのパス内)
4. MYCOPY (前述のとおり、指定されたすべてのパス内)

COBOL は、その名前がリテラルに含まれている場合 (「MyCopy」) 以外は、デフォルトでは *library-name* と *text-name* を大文字にします。 この例では、MyCopy と MYCOPY は同じではありません。 ファイル名が大/小文字混合 (MyCopy.cbl など) の場合は、*text-name* を COPY ステートメントでリテラルとして定義します。

-I オプション

それ以外の場合 (*library-name* も *text-name* もパスを示さない場合)、検索パスは -I オプションに依存します。

COPY A と COPY A OF MYLIB を等しくするには、-I$MYLIB を指定します。

上記の規則に基づくと、COPY "/X/Y" はルート・ディレクトリー内で検索され、COPY "X/Y" は現行ディレクトリー内で検索されます。

COPY A OF SYSLIB は COPY A と 同等です。 -I オプションは、*library-name* 修飾が明示的に指定された COPY ステートメントや、 ライブラリー名が SYSLIB のステートメントには影響しません。

library-name と *text-name* の両方を指定すると、コンパイラーは、*library-name* の末尾が / でない場合に、2 つの値の間にパス区切り文字 (/) を挿入します。 例えば、COPY MYCOPY OF MYLIB を次のように設定します。

```
export MYCOPY=MYPDS(MYMEMBER)
export MYLIB=MYFILE
```

この場合、結果として MYFILE/MYPDS(MYMEMBER) が作成されます。

text-name をユーザー定義語として指定した場合は、ローカル・ファイルにアクセスすることができ、メインフレームのソースを変更せずに z/OS 上の PDS メンバーにアクセスすることもできます。 次に例を示します。

```
COPY mycopybook
```

この例では、環境変数 *mycopybook* が h/mypds(mycopy) に設定されている場合は、次のようになります。

- h は特定のホストに割り当てられます。
- mypds は z/OS PDS 名です。
- mycopy は PDS メンバー名です。

NFS (ネットワーク・ファイル・システム) を 使用して Linux から z/OS ファイルにアクセスできます。 NFS により、Linux パス名を使用して z/OS ファイルにアクセスできます。ただし、NFS では、z/OS 命名規約に従うためにパス区切り文字が「.」に変換されます。 名前の形式が適切になるように、環境変数に値を割り当てるときはこのことに注意してください。 例えば、次のように設定します。

```
export MYCOPY=(MYMEMBER)
export MYLIB=M/MYFILE/MYPDS
```

結果として次のパスが作成されるため、これは機能しません。

```
M/MYFILE/MYPDS/(MYMEMBER)
```

これは、パス区切り文字の変換後に次のようになります。

```
M.MYFILE.MYPDS.(MYMEMBER)
```

DELETE ステートメント
この拡張ソース・ライブラリー・ステートメントは、BASIS ソース・プログラムから COBOL ステートメントを除去します。

EJECT ステートメント
このコンパイラー指示ステートメントは、次のソース・ステートメントが次のページの最上部に印刷されるように指定します。

ENTER ステートメント
ステートメントは、コメントとして扱われます。

EVALUATE ディレクティブ
EVALUATE ディレクティブは、コンパイル・グループに組み込むソース行を選択する分岐方式を提供します。

IF ディレクティブ
IF ディレクティブは、片方向または両方向の条件付きコンパイルを提供します。

INSERT ステートメント
このライブラリー・ステートメントは、COBOL ステートメントを BASIS ソース・プログラムに追加します。

PROCESS（CBL）ステートメント
このコンパイラー指示ステートメントは、最も外側の IDENTIFICATION DIVISION ヘッダーの前に置かれるもので、プログラムのコンパイル時に使用されるコンパイラー・オプションを指定します。

REPLACE ステートメント
このステートメントは、ソース・プログラム・テキストを置き換えるために使用されます。

SKIP1/2/3 ステートメント
このステートメントは、ある行がソース・リストでスキップされることを示します。

TITLE ステートメント
このステートメントは、ソース・リストの各ページの最上部に表題 (ヘッダー) が印刷されるように指定します。

USE ステートメント
USE ステートメントは、以下のエレメントを指定するための宣言 を提供します。

- エラー処理プロシージャー: EXCEPTION/ERROR
- ユーザー・ラベル処理プロシージャー: LABEL
- デバッグ行およびセクション: DEBUGGING

関連タスク
4 ページの『ソース・リストのヘッダーの変更』
225 ページの『コマンド行からのコンパイル』
226 ページの『PROCESS (CBL) ステートメントでのコンパイラー・オプションの指定』

関連参照
232 ページの『cob2 オプション』
CALLINTERFACE (*COBOL for Linux on x86* 言語解説書)
PROCESS (CBL) ステートメント (*COBOL for Linux on x86* 言語解説書)
*CONTROL (*CBL) ステートメント (*COBOL for Linux on x86* 言語解説書)
COPY ステートメント (*COBOL for Linux on x86* 言語解説書)
DEFINE ディレクティブ (*COBOL for Linux on x86* 言語解説書)
EVALUATE ディレクティブ (*COBOL for Linux on x86* 言語解説書)
IF ディレクティブ (*COBOL for Linux on x86* 言語解説書)

第 15 章 ランタイム・オプション

次の表に、サポートされているランタイム・オプションの一覧を示します。

表 32. **ランタイム・オプション**			
オプション	**説明**	**デフォルト**	**省略形**
299 ページの『CHECK』	エラー検査のフラグを立てます。	CHECK(ON)	CH
300 ページの『DEBUG』	USE FOR DEBUGGING 宣言で指定された COBOL デバッグ・セクションがアクティブかどうかを指定します。	NODEBUG	なし
300 ページの『ERRCOUNT』	重大度 1 (W レベル) の条件が何回発生すると実行単位が異常終了するかを指定します。	ERRCOUNT(20)	なし
300 ページの『FILESYS』	ASSIGN 文節または環境変数のいずれかを通した、明示的なファイル・システム選択が行われないファイルに使用されるファイル・システムを指定します。	FILESYS(VSA)	FS(DB2|QSAM|RSD|SdU|SFS |STL|VSA|VSAM)
302 ページの『TRAP』	COBOL が例外を代行受信するかどうかを指定します。	TRAP(ON)	なし
302 ページの『UPSI』	COBOL ルーチンを使用するアプリケーションに対して、8 つの UPSI スイッチのオン/オフを設定します。	UPSI(00000000)	なし

ランタイム・オプションは、COBRTOPT ランタイム環境変数を設定することにより指定します。

関連タスク

215 ページの『環境変数の設定』
238 ページの『プログラムの実行』

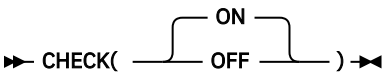
関連参照

220 ページの『ランタイム環境変数』

CHECK

CHECK により、エラー検査にフラグが設定されます。 COBOL では、索引、添え字、参照変更範囲によって、エラー検査が行われます。

CHECK オプションの構文



デフォルト: CHECK(ON)

省略形: CH

ON
ランタイム検査を実行することを指定します。

© Copyright IBM Corp. 2021

OFF
ランタイム検査を実行しないことを指定します。

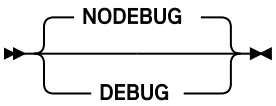
使用上の注意: コンパイル時に NOSSRANGE が 有効だった場合は、CHECK(ON) が無効になります。

パフォーマンスの考慮事項: SSRANGE を使用して COBOL プログラムを コンパイルした後、アプリケーションのテストまたはデバッグを行わない場合は、CHECK(OFF) を 指定するとパフォーマンスが向上します。

DEBUG

DEBUG は、USE FOR DEBUGGING 宣言で指定された COBOL デバッグ・セクションがアクティブかどうかを指定します。

DEBUG オプションの構文



デフォルト: NODEBUG

DEBUG
デバッグ・セクションをアクティブにします。

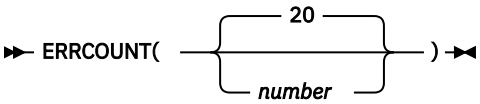
NODEBUG
デバッグ・セクションを抑制します。

パフォーマンスについての考慮事項: パフォーマンスを高めるには、このオプションをデバッグ時だけに使用してください。

ERRCOUNT

ERRCOUNT は、警告メッセージが何回発生すると実行単位が異常終了するかを指示します。

ERRCOUNT オプションの構文



デフォルト: ERRCOUNT(20)

number は、正の場合、実行単位の実行中に発生し得る警告メッセージ数です。 警告メッセージ数が *number* を超えると、実行単位が異常終了します。

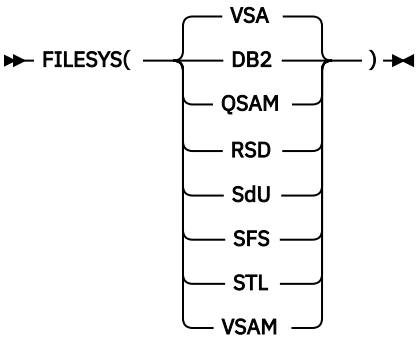
number が 0 の場合は、発生し得る警告メッセージ数に制限がないことを指示します。 *number* を負にすることはできません。

重大度が警告より高い条件によるメッセージが発生すると、ERRCOUNT オプションの値に関係なく、実行単位が終了します。

FILESYS

FILESYS は、ASSIGN 文節または環境変数によってファイル・システムが明示的に指定されなかったファイルに使用されるファイル・システムを指定します。 このオプションは、順次ファイル、相対ファイル、および索引付きファイルに適用されます。 このオプションは、行順次ファイルには適用されません。行順次ファイルの場合、ファイル・システムは LSQ (行順次) として指定するか、またはデフォルトでそれになるように設定する必要があります。

FILESYS オプションの構文



デフォルト: FILESYS(VSA)

省略形: FS(DB2|QSA|RSD|SdU|SFS|STL|VSA)

DB2
ファイル・システムは Db2 リレーショナル・データベースです。

QSAM
ファイル・システムはメインフレーム QSAM ファイルと互換性があります。

RSD
ファイル・システムはレコード順次区切りです。

SdU
ファイル・システムは SMARTdata ユーティリティーです。

SFS
ファイル・システムは CICS Structured File Server です。

STL
ファイル・システムは STL (標準言語) ファイル・システムです。

VSA または VSAM
VSA または VSAM (仮想記憶アクセス方式) は SFS または STL ファイル・システムのいずれかを暗黙に示します。

システム・ファイル名が値 /.:/cics/sfs で始まる場合、このサブオプションは SFS ファイル・システムを暗黙に示します。 その他の場合は、STL ファイル・システムを暗黙に示します。

関連概念
117 ページの『ファイル・システム』
122 ページの『行順次ファイル編成』

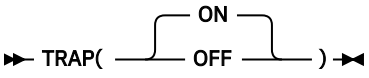
関連タスク
113 ページの『ファイルの識別』

関連参照
116 ページの『ファイル・システム決定の優先順位』
220 ページの『ランタイム環境変数』
ASSIGN 節 (*COBOL for Linux on x86* 言語解説書)

TRAP

TRAP は、COBOL が例外を代行受信するかどうかを指示します。

TRAP オプションの構文



デフォルト: TRAP(ON)

TRAP(OFF) が有効であり、なおかつ例外条件を処理する独自のトラップ・ハンドラーを使用しない場合は、これらの条件が発生すると、オペレーティング・システムによるデフォルトのアクションが実行されます。 例えば、プログラムが無許可の場所にデータを格納しようとすると、デフォルトのシステム・アクションとして、メッセージが出されて処理が終了します。

ON
COBOL による例外の代行受信をアクティブにします。

OFF
COBOL による例外の代行受信を非アクティブにします。

使用上の注意

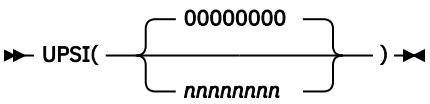
- TRAP(OFF) を使用するのは、プログラムの例外を COBOL が処理する前に分析する必要がある場合だけにしてください。
- 非 CICS 環境で TRAP(OFF) を指定すると、例外ハンドラーが設定されません。
- (例外診断の目的で) TRAP(OFF) を使用して実行すると、COBOL が TRAP(ON) を必要とするため、多数の副次作用が発生する可能性があります。 TRAP(OFF) を使用して実行すると、ソフトウェアで発生した条件、プログラム・チェック、または異常終了が検出されない場合でも、副次作用が発生する可能性があります。 TRAP(OFF) が有効な場合にプログラム・チェックまたは異常終了が検出されると、次の副次作用が発生する可能性があります。
  - COBOL によって取得されたリソースが解放されない。
  - COBOL によってオープンされたファイルがクローズしない。このため、レコードが失われる可能性があります。
  - メッセージまたはダンプ出力が生成されない。

  このような条件が発生すると、実行単位が異常終了します。

UPSI

UPSI は、COBOL ルーチンを使用するアプリケーションに対して、8 つの UPSI スイッチのオン/オフを設定します。

UPSI オプションの構文



デフォルト: UPSI(00000000)

それぞれの *n* は、UPSI スイッチ (0 から 7) のうちの 1 つを表します。 左端の *n* が 1 番目のスイッチを表しています。 それぞれの *n* は 0 (オフ) または 1 (オン) のいずれかになります。

第 16 章 デバッグ

アプリケーションの動作における問題の原因を判別するには、ソース言語デバッグと対話式デバッグの 2 つの異なる方法を使用することができます。

ソース言語デバッグの場合、COBOL は、デバッグを容易にするいくつかの言語エレメント、コンパイラー・オプション、およびリスト出力を提供します。

対話式デバッグの場合は、IBM Debug for Linux on x86 を使用できます。

関連タスク
303 ページの『ソース言語によるデバッグ』
307 ページの『コンパイラー・オプションを使用したデバッグ』
311 ページの『IBM Debug for Linux on x86 を使用したデバッグ』
361 ページの『リストの入手』
372 ページの『オフセット情報を含むメッセージによるデバッグ』

373 ページの『アセンブラー・ルーチンのデバッグ』

ソース言語によるデバッグ

さまざまな COBOL 言語機能を使用して、プログラムの障害の原因を正確に示すことができます。

障害のあるプログラムが既に実動中の大規模なアプリケーションの一部である場合 (ソース更新を除外する) は、プログラムの障害部分をシミュレートするような小さいテスト・ケースを作成してください。 テスト・ケースでは、以下の問題の検出に役立つようなデバッグ機能をコーディングしてください。

- プログラム・ロジックのエラー
- 入出力エラー
- データ型のミスマッチ
- 初期化されていないデータ
- プロシージャーの問題

関連タスク
303 ページの『プログラム・ロジックのトレース』
304 ページの『入出力エラーの検出および処理』
304 ページの『データの妥当性検査』
305 ページの『初期化されていないデータの移動、初期化、または設定』
305 ページの『プロシージャーに関する情報の生成』

関連参照
ソース言語のデバッグ (*COBOL for Linux on x86* 言語解説書)

プログラム・ロジックのトレース

プログラムのロジックは、DISPLAY ステートメントを追加することによってトレースしてください。

例えば、問題が EVALUATE ステートメントまたは 1 組のネストされた IF ステートメントにあると判断した場合は、それぞれのパスで DISPLAY ステートメントを使用して、ロジック・フローを調べます。 問題の原因が数値の計算方法にあると判断した場合は、DISPLAY ステートメントを使用して、いくつかの中間結果の値を検査することができます。

プログラムの中で明示範囲終了符号を使用してステートメントを終了させるようにすると、ロジックはより明確であり、したがってトレースしやすくなります。

© Copyright IBM Corp. 2021

例えば、特定のルーチンが開始して終了したかどうかを判別する場合は、プログラムに次のようなコードを挿入してみてください。

```
DISPLAY "ENTER CHECK PROCEDURE"
    .
    .  (checking procedure routine)
    .
DISPLAY "FINISHED CHECK PROCEDURE"
```

ルーチンが正しく作動していることを確認したら、次のいずれかの方法で DISPLAY ステートメントを使用不可にします。

- 各 DISPLAY ステートメントの行の 7 桁目にアスタリスクを置き、コメント行に変換する。
- 各 DISPLAY ステートメントの 7 桁目に D を置き、コメント行に変換する。 これらのステートメントを再活動化したい場合は、ENVIRONMENT DIVISION に WITH DEBUGGING MODE 節を含めると、7 桁目の D は無視され、DISPLAY ステートメントが実施されます。

プログラムを実動に移す前に、使用したすべてのデバッグ・エイドを削除するか使用不可にしてから、プログラムを再コンパイルします。 プログラムはより効果的に実行され、使用するストレージは小さくなります。

関連概念
16 ページの『範囲終了符号』

関連参照

DISPLAY ステートメント (*COBOL for Linux on x86* 言語解説書)

入出力エラーの検出および処理

ファイル状況キーは、プログラムのエラーが、ストレージ・メディアで起こっている入出力エラーによるものかどうかを 判別するのに役立ちます。

ファイル状況キーをデバッグ・エイドとして使用するためには、各入出力ステートメントの後で、状況キーの値がゼロ以外かどうか検査します。 値がゼロでない (エラー・メッセージで報告される) 場合には、プログラム内の入出力プロシージャーのコーディングを調べます。 状況キーの値に基づいてエラーを訂正するためのプロシージャーを組み込むこともできます。

問題がプログラムの入出力プロシージャーにあると判断した場合は、USE EXCEPTION/ERROR 宣言を組み込んで、問題のデバッグに役立てることができます。 その後、ファイルのオープンに失敗すると、適切な EXCEPTION/ERROR 宣言が実行されます。 適切な宣言とは、ファイルに固有なもの、あるいはオープン属性 (INPUT、OUTPUT、I-O、または EXTEND) 用に提供されたものです。

各 USE AFTER STANDARD ERROR ステートメントを、PROCEDURE DIVISION の中の DECLARATIVES キーワードに続くセクションにコーディングします。

関連タスク
170 ページの『ERROR 宣言のコーディング』
170 ページの『ファイル状況キーの使用』

関連参照

ファイル状況キー (*COBOL for Linux on x86* 言語解説書)

データの妥当性検査

プログラムが非数値データに対して算術を実行しようとしているか、または入力レコードの誤ったデータ型を受け取ろうとしている可能性がある場合、クラス・テスト (クラス条件) を使用してデータ型を妥当性検査してください。

クラス・テストを使用すると、データ項目の内容が、ALPHABETIC、ALPHABETIC-LOWER、ALPHABETIC-UPPER、DBCS、KANJI、または NUMERIC のいずれであるかを検査できます。 データ項目が暗黙的または

明示的に USAGE NATIONAL として記述されている場合、クラス・テストは、指定された文字クラスに関連した文字の国別文字表現を検査します。

関連タスク
85 ページの『条件式のコーディング』
200 ページの『有効な DBCS 文字に関するテスト』

関連参照

クラス条件 (*COBOL for Linux on x86* 言語解説書)

初期化されていないデータの移動、初期化、または設定

問題の原因がテーブルまたはデータ項目のフィールドに残された残余データにあると考えられる ときは、INITIALIZE または SET ステートメントを使用して、 テーブルまたはデータ項目を初期化してください。

問題が起きたり起きなかったりし、しかも同一のデータで起きるとは限らない場合、スイッチが初期化されていないが多くの場合正しい値 (0 または 1) に偶然に設定されることが原因であると考えられます。SET ステートメントを使用してスイッチを初期化するようにすれば、初期化されていないスイッチが問題の原因であると判断できるか、考えられる原因からそのスイッチを除外することができます。

関連参照
INITIALIZE ステートメント (*COBOL for Linux on x86* 言語解説書)
SET ステートメント (*COBOL for Linux on x86* 言語解説書)

プロシージャーに関する情報の生成

プログラムまたはテスト・ケースおよびその実行方法に関する情報を生成するには、USE FOR DEBUGGING 宣言をコーディングします。 この宣言を使用すると、ステートメントをプログラムに組み込んで、プログラムの実行時にいつそれらのステートメントを実行しなければならないかを指示することができます。

例えば、プロシージャーが何度実行されるかを決定するには、デバッグ・プロシージャーを USE FOR DEBUGGING 宣言に組み込み、カウンターを使用して、制御がそのプロシージャーに渡される回数の記録をとることができます。 カウンター技法を使用して、次のような項目を検査できます。

- PERFORM ステートメントが実行される回数。特定のルーチンが使用されているかどうか、および制御構造が正しいかどうか。
- ループが実行される回数、ループが実行されているかどうか、およびループの回数が正確かどうか。

プログラムに、デバッグ行かデバッグ・ステートメント、またはその両方を入れることができます。

デバッグ行 は、桁 7 の D で識別されているステートメントです。プログラムのデバッグ行をアクティブにするには、ENVIRONMENT DIVISION の SOURCE-COMPUTER 行に WITH DEBUGGING MODE 節をコーディングする必要があります。この節が含まれていないと、デバッグ行はコメントとしてしか扱われません。

デバッグ・ステートメント とは、PROCEDURE DIVISION の DECLARATIVES セクションにコーディングされたステートメントです。 それぞれの USE FOR DEBUGGING 宣言は別個のセクションにコーディングしなければなりません。デバッグ・ステートメントは、次のようにコーディングしてください。

- DECLARATIVES セクション内のみ。
- ヘッダー USE FOR DEBUGGING の後に置く。
- 最外部のプログラムだけに置く (ネストされたプログラムでは無効です)。 デバッグ・ステートメントが、ネストされたプログラムに含まれているプロシージャーによって起動されることはありません。

プログラムでデバッグ・ステートメントを使用するには、WITH DEBUGGING MODE 節を指定し、さらに、DEBUG ランタイム・オプションを使用する必要があります。

オプションに関する制約事項:

- USE FOR DEBUGGING 宣言は、WITH DEBUGGING MODE 節が指定されていると、TEST コンパイラー・オプションと相互に排他的な関係にあります。 USE FOR DEBUGGING 宣言と WITH DEBUGGING MODE 節が存在していると、TEST オプションは取り消されます。

306 ページの『例: USE FOR DEBUGGING』

関連参照
SOURCE-COMPUTER 段落 (*COBOL for Linux on x86* 言語解説書)
デバッグ行 (*COBOL for Linux on x86* 言語解説書)
デバッグ・セクション (*COBOL for Linux on x86* 言語解説書)
DEBUGGING 宣言 (*COBOL for Linux on x86* 言語解説書)

例: USE FOR DEBUGGING

この例は、以下のプログラム・セグメントは、DISPLAY ステートメントと USE FOR DEBUGGING 宣言を使用してプログラムをテストする際に必要となるステートメントの種類を示しています。

DISPLAY ステートメントは、情報を端末または出力ファイルに書き込みます。 USE FOR DEBUGGING 宣言は、ルーチンが実行される回数を示すカウンターと一緒に使用されます。

```
Environment Division.
. . .
Data Division.
. . .
Working-Storage Section.
. . . (other entries your program needs)
01  Trace-Msg    PIC X(30) Value "  Trace for Procedure-Name : ".
01  Total        PIC 9(9)  Value 1.
. . .
Procedure Division.
Declaratives.
Debug-Declaratives Section.
    Use For Debugging On Some-Routine.
Debug-Declaratives-Paragraph.
    Display Trace-Msg, Debug-Name, Total.
End Declaratives.

Main-Program Section.
    . . . (source program statements)
    Perform Some-Routine.
    . . . (source program statements)
    Stop Run.
Some-Routine.
    . . . (whatever statements you need in this paragraph)
    Add 1 To Total.
Some-Routine-End.
```

プロシージャー Some-Routine が実行されるたびに、DECLARATIVES SECTION の DISPLAY ステートメントがこのメッセージを出します。

```
  Trace For Procedure-Name : Some-Routine 22
```

メッセージの終わりにある番号 22 は、データ項目 Total の累算された値で、Some-Routine が実行された回数を示しています。 デバッグ宣言内のステートメントは、名前を指定されたプロシージャーが実行される前に実行されます。

DISPLAY ステートメントを使用して、プログラムの実行をトレースし、プログラム中のフローを示すこともできます。 これを行うには、DISPLAY ステートメントから Total を除去し、DECLARATIVES SECTION の USE FOR DEBUGGING を次のように変更します。

```
USE FOR DEBUGGING ON ALL PROCEDURES.
```

これで、最外部プログラムのそれぞれの非デバッグ・プロシージャーが実行される前にメッセージが表示されるようになります。

コンパイラー・オプションを使用したデバッグ

ある種のコンパイラー・オプションは、プログラム内のエラーの検出、プログラム内の各種エレメントの検出、リストの取得、およびデバッグのためのプログラムの準備に役立ちます。

コンパイラー・オプション (括弧内に示されているもの) を使用して、以下のエラーを検出することができます。

- 重複データ名のような構文エラー (NOCOMPILE)
- セクションの欠落 (SEQUENCE)
- 無効な添え字値 (SSRANGE)

コンパイラー・オプションを使用して、プログラム内にある以下のエレメントを検出することができます。

- エラー・メッセージおよび関連するエラーの発生場所 (FLAG)
- プログラム・エンティティー定義および参照。 (XREF)
- DATA DIVISION 内のデータ項目 (MAP)
- ステートメント参照 (VBREF)

ソースのコピー (SOURCE) または生成されたコードのリスト (LIST) を取得できます。

TEST コンパイラー・オプションを使用して、デバッグできるようプログラムを準備します。

関連タスク
307 ページの『コーディング・エラーの検出』
308 ページの『行シーケンス問題の検出』
308 ページの『有効範囲の検査』
308 ページの『診断するエラーのレベルの選択』
310 ページの『プログラム・エンティティー定義および参照の検出』
311 ページの『データ項目のリスト』
361 ページの『リストの入手』

関連参照

248 ページの『コンパイラー・オプション』

コーディング・エラーの検出

条件付きでコンパイルしたり、構文検査のみを行ったりする場合は、NOCOMPILE オプションを使用してください。 SOURCE オプションと一緒に使用すると、NOCOMPILE は、コーディングの間違い (欠落している定義、正しく定義されていないデータ項目、重複するデータ名など) を見つけるのに役立つリストを作成します。

構文のみの検査: プログラムの構文検査のみを行い、オブジェクト・コードを生成しないようにするには、サブオプションなしの NOCOMPILE を使用してください。 一緒に SOURCE オプションを指定すると、コンパイラーはリストを作成します。

NOCOMPILE を指定すると、いくつかのコンパイラー・オプションが抑制されます。 詳細については、COMPILE オプションに関する下記の関連参照を参照してください。

条件付きコンパイル: 条件付きでコンパイルするには、NOCOMPILE(*x*) (ここで、*x* はエラーの重大度レベルの 1 つです) を使用してください。 エラーすべてが *x* より低い重大度である場合に、プログラムはコンパイルされます。使用できる重大度レベルは、S (重大)、E (エラー)、および W (警告) であり、この順序で低くなります。

レベル *x* またはそれ以上のエラーが発生した場合、コンパイルは停止し、プログラムの構文検査のみが行われます。

関連参照
259 ページの『COMPILE』

行シーケンス問題の検出

順序どおりになっていないステートメントを見つけるには、SEQUENCE コンパイラー・オプションを 使用してください。 シーケンス中断は、ソース・プログラムのセクションが移動または削除されたことを表します。

SEQUENCE を使用すると、コンパイラーはソース・ステートメント番号を検査し、昇順になっているかどうかを調べます。 順序どおりになっていないステートメント番号の横には、2 つのアスタリスクが入れられます。 これらのステートメントの合計数はソース・リストに続く診断の最初の行として印刷されます。

関連参照
279 ページの『SEQUENCE』

有効範囲の検査

SSRANGE コンパイラー・オプションを使用して、アドレスが適切な範囲内にあるかどうかを調べます。

SSRANGE を使用すると、以下のアドレスが検査されます。

- 添え字付きまたは指標付きデータ参照: 指定されたテーブルの最大境界内に、必要なエレメントの有効アドレスがあるかどうか。
- 可変長データ参照 (OCCURS DEPENDING ON 節を含むデータ項目への参照): 実際の長さが正であるかどうか、 そしてグループ・データ項目に対して定義された最大長より短いかどうか。
- 参照変更データ参照: オフセットと長さが正であるかどうか。 オフセットと長さの合計がデータ項目の最大長より短いかどうか。

SSRANGE オプションが有効である場合、以下の両方の条件が真であれば、 実行時に検査が行われます。

- 指標付き、添え字付き、可変長、または参照変更データ項目が含まれている COBOL ステートメントが実行される。
- CHECK ランタイム・オプションが ON である。

参照されるデータを含むデータ項目の範囲外に有効アドレスがある場合は、エラー・メッセージが生成され、プログラムは停止します。 メッセージでは、参照されたテーブルまたは ID、およびエラーが発生した行番号が識別されます。 エラーの原因となった参照のタイプによっては、追加情報が提供されます。

与えられたデータ参照内のすべての添え字、指標、または参照修飾子がリテラルであり、データ項目の外側を参照する結果になる場合は、SSRANGE コンパイラー・オプションの設定値に関係なく、コンパイル時にエラーが診断されます。

パフォーマンスの考慮: SSRANGE が指定されると、パフォーマンスは少し低下することがあります。これは、それぞれの添え字付きまたは指標付き項目を検査するために必要なオーバーヘッドが余分にかかるためです。

関連参照
284 ページの『SSRANGE』
505 ページの『パフォーマンスに関連するコンパイラー・オプション』

診断するエラーのレベルの選択

FLAG コンパイラー・オプションを使用すると、コンパイル時に診断するエラーのレベルを 指定すること、およびエラー・メッセージをリストに組み込みかどうかを指示することができます。 すべてのエラーが通知されるようにするには、FLAG(I) または FLAG(I,I) を使用してください。

最初のパラメーターには、発行される構文エラー・メッセージのうち最も重大度レベルの低いものを指定してください。 オプションとして、2 番目のパラメーターには、ソース・リストに組み込む構文メッセージのうち最も重大度レベルの低いものを指定します。 この重大度レベルは、最初のパラメーターのレベル

と同じかそれ以上でなければなりません。 両方のパラメーターを指定する場合は、一緒に SOURCE コンパイラー・オプションも指定する必要があります。

表 *33.* **コンパイラー・メッセージの重大度レベル**	
重大度レベル	**結果のメッセージ**
U (回復不能)	U (メッセージのみ)
S (重大)	すべての S および U メッセージ
E (エラー)	すべての E、S、および U メッセージ
W (警告)	すべての W、E、S、および U メッセージ
I (通知)	すべてのメッセージ

2 番目のパラメーターを指定すると、コンパイラーがエラーを検出するために利用できる情報が十分ある時点で、 構文エラー・メッセージ (U レベルのメッセージを除く) がソース・リストに組み込まれます。 ライブラリー・コンパイラー・フェーズで出されるメッセージ以外のすべての組み込みメッセージは、それらが参照するステートメントの直後に続きます。 エラーがあるステートメントの番号もメッセージに示されます。 組み込みメッセージは、ソース・リストの終わりに出される残りの診断メッセージで繰り返されます。

NOSOURCE コンパイラー・オプションを指定した場合は、構文エラー・メッセージはリストの終わりにだけ入れられます。 回復不能エラーに関するメッセージはソース・リストに組み込まれません。この重大度のエラーはコンパイルを終了させるからです。

309 ページの『例: 組み込みメッセージ』

関連タスク
231 ページの『コンパイラー・メッセージのリストの生成』

関連参照

230 ページの『コンパイラー診断メッセージの重大度コード』
231 ページの『コンパイラー検出エラーに関するメッセージおよびリスト』
267 ページの『FLAG』

例: 組み込みメッセージ

次の例は、FLAG オプションに 2 番目のパラメーターを指定することによって生成される 組み込みメッセージを示しています。 要約の中のメッセージのいくつかは複数の COBOL ステートメントに適用されます。

```
   LineID  PL SL  ----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7-|--+----8  Map
and Cross Reference
 ...                                                                                    |
   000977               /                                                               |
   000978               ****************************************************************  |
   000979               ***          I N I T I A L I Z E   P A R A G R A P H           **  |
   000980               ***  Open files. Accept date, time and format header lines.    **  |
   000981         IA4690***   Load location-table.                                      **  |
   000982               ****************************************************************  |
   000983                  100-initialize-paragraph.                                       |
   000984                    move spaces to ws-transaction-record                          | IMP
339
   000985                    move spaces to ws-commuter-record                             | IMP
315
   000986                    move zeroes to commuter-zipcode                               | IMP
326
   000987                    move zeroes to commuter-home-phone                            | IMP
327
   000988                    move zeroes to commuter-work-phone                            | IMP
328
   000989                    move zeroes to commuter-update-date                           | IMP
332
   000990                    open input update-transaction-file                            | 203
 ==000990==> IGYPS2052-S An error was found in the definition of file "LOCATION-FILE".  The      |
```

```
                        reference to this file was discarded.                                                  |
   000991                       location-file                                                                  | 192
   000992                       i-o commuter-file                                                              | 180
   000993                       output print-file                                                              | 216
   000994                   if loccode-file-status not = "00" or                                               | 248
   000995                      update-file-status not = "00" or                                                | 247
   000996                      updprint-file-status not = "00"                                                 | 249
   000997      1               display "Open Error ..."                                                       |
   000998      1               display "  Location File Status = " loccode-file-status                         | 248
   000999      1               display "  Update   File Status = " update-file-status                         | 247
   001000      1               display "  Print    File Status = " updprint-file-status                       | 249
   001001      1             perform 900-abnormal-termination                                                 | 1433
   001002                   end-if                                                                           |
   001003         IA4760    if commuter-file-status not = "00" and not = "97"                                | 240
   001004      1              display "100-OPEN"                                                            |
   001005      1              move 100 to comp-code                                                         | 230
   001006      1              perform 500-stl-error                                                         | 1387
   001007      1              display "Commuter File Status (OPEN) = "                                      |
   001008      1                       commuter-file-status                                                 | 240
   001009      1              perform 900-abnormal-termination                                              | 1433
   001010         IA4790    end-if                                                                           |
   001011                   accept ws-date from date                                                       | UND
 ==001011==> IGYPS2121-S "WS-DATE" was not defined as a data-name.  The statement was discarded.  |
   001012         IA4810    move corr ws-date to header-date                                               | UND
463
 ==001012==> IGYPS2121-S "WS-DATE" was not defined as a data-name.  The statement was discarded.  |
   001013                   accept ws-time from time                                                       | UND
 ==001013==> IGYPS2121-S "WS-TIME" was not defined as a data-name.  The statement was discarded.  |
   001014         IA4830    move corr ws-time to header-time                                               | UND
457
 ==001014==> IGYPS2121-S "WS-TIME" was not defined as a data-name.  The statement was discarded.  |
   001015         IA4840    read location-file                                                             | 192
 ...
 LineID  Message code  Message text
      192  IGYDS1050-E   File "LOCATION-FILE" contained no data record descriptions.
                         The file definition was discarded.
      899  IGYPS2052-S   An error was found in the definition of file "LOCATION-FILE".
                         The reference to this file was discarded.
                         Same message on line:    990
     1011  IGYPS2121-S   "WS-DATE" was not defined as a data-name.  The statement was discarded.
                         Same message on line:   1012
     1013  IGYPS2121-S   "WS-TIME" was not defined as a data-name.  The statement was discarded.
                         Same message on line:   1014
     1015  IGYPS2053-S   An error was found in the definition of file "LOCATION-FILE".
                         This input/output statement was discarded.
                         Same message on line:   1027
     1026  IGYPS2121-S   "LOC-CODE" was not defined as a data-name.  The statement was discarded.
     1209  IGYPS2121-S   "COMMUTER-SHIFT" was not defined as a data-name.  The statement was discarded.
                         Same message on line:   1230
     1210  IGYPS2121-S   "COMMUTER-HOME-CODE" was not defined as a data-name.  The statement was
discarded.
                         Same message on line:   1231
     1212  IGYPS2121-S   "COMMUTER-NAME" was not defined as a data-name.  The statement was discarded.
                         Same message on line:   1233
     1213  IGYPS2121-S   "COMMUTER-INITIALS" was not defined as a data-name.  The statement was
discarded.
                         Same message on line:   1234
     1223  IGYPS2121-S   "WS-NUMERIC-DATE" was not defined as a data-name.  The statement was discarded.
 Messages    Total    Informational    Warning    Error    Severe    Terminating
 Printed:      19                                   1        18
 * Statistics for COBOL program FLAGOUT:
 *    Source records = 1755
 *    Data Division statements = 279
 *    Procedure Division statements = 479
 Locale = en_US.ISO8859-1                  (1)
 End of compilation 1,  program FLAGOUT,  highest severity: Severe.
 Return code 12
```

(1)
コンパイラーが使用したロケール

プログラム・エンティティー定義および参照の検出

XREF(FULL) コンパイラー・オプションを使用すると、データ名、 プロシージャー名、またはプログラム名が定義および参照されている場所を見つけることができます。 また、コピーブックを取得したファイルへの、COPY または BASIS ステートメントの相互参照を作成するためにも使用します。

ソート済み相互参照には、そのデータ名、プロシージャー名、またはプログラム名が定義されている行番号およびそのデータ名、プロシージャー名、またはプログラム名へのすべての参照の行番号が入れられます。

明示的に参照されているデータ項目だけを含める場合は、XREF(SHORT) オプションを使用します。

XREF (FULL または SHORT) と SOURCE オプションの両方を使用すると、変更された相互参照がソース・リストの右側に印刷されます。 この組み込み相互参照は、データ名またはプロシージャー名が定義されている行番号を示します。

詳細については、XREF コンパイラー・オプションに関する関連参照を参照してください。

369 ページの『例: XREF 出力: データ名相互参照』
370 ページの『例: XREF 出力: プログラム名相互参照』
370 ページの『例: XREF 出力: COPY/BASIS 相互参照』
371 ページの『例: XREF 出力: 組み込み相互参照』

関連タスク
361 ページの『リストの入手』

関連参照
290 ページの『XREF』

データ項目のリスト

DATA DIVISION 項目および暗黙的に宣言されたすべての項目のリストを作成するには、MAP コンパイラー・オプションを使用します。

MAP オプションを指定すると、圧縮 MAP 情報を含む組み込み MAP 要約が、COBOL ソース・データ定義の右側に生成されます。 XREF データと組み込み MAP 要約の両方が同じ行にあるときは、組み込み要約の方が先に印刷されます。

MAP リストおよび組み込み MAP 要約の各部分は、ソース全体を通じ、*CONTROL MAP|NOMAP (または *CBL MAP|NOMAP) ステートメントを使用して、選択または抑制することができます。 次に例を示します。

```
*CONTROL NOMAP
    01  A
    02  B
*CONTROL MAP
```

365 ページの『例: MAP 出力』

関連タスク
361 ページの『リストの入手』

関連参照

271 ページの『MAP』

IBM Debug for Linux on x86 を使用したデバッグ

IBM Debug for Linux on x86 を使用します。これは、COBOL プログラムをデバッグするための、COBOL for Linux に付属しているインタラクティブなソース・レベルのデバッガーです。

IBM Debug for Linux on x86 の概要

IBM Debug for Linux on x86 は、ソース・レベルの対話式デバッガーです。Linux on x86 で実行されているデバッガー・エンジンにリモートで接続している Windows および Linux ベースのワークステーション上で動作します。 IBM Debug for Linux on x86 を使用して、COBOL で作成されたプログラムをデバッグすることができます。

このデバッガーは、アプリケーション・ソース・ファイルと、そのソース・ファイルに含まれるエレメントを表示します。指定した行または条件での、単一ステップ、ステップスルー、ステップオーバー、または実行停止を行うことができます。実行を制御しながら、変数、レジスター、メモリー、コール・スタックなどの要素をモニターできます。

IBM Debug for Linux on x86 は、インターネット・プロトコル、バージョン 6 (IPv6) に対応しています。

インストール

IBM Debug for Linux on x86 のコンポーネントをインストールする方法について説明します。

デバッガー・コンポーネント

IBM Debug for Linux on x86 は、次の 2 つのコンポーネントから構成されるクライアント/サーバー・モデルを使用します。

- デバッグ・エンジン (`irmtdbgc`)。 これは Linux on x86 マシンにインストールされたサーバー・コンポーネントです。
- デバッグ・クライアントである Remote Debug Eclipse User Interface (p2 リポジトリー)。 これは、既存の Eclipse インスタンスを拡張する Eclipse 機能のセットとして提供されており、 Linux on x86 または Windows 10 ワークステーションにインストールすることができます。

Linux on x86 デバッグ・エンジンのインストール

IBM Debug for Linux on x86 1.0 のデバッグ・エンジンは、IBM COBOL for Linux on x86 の 1.1.0.1 フィックスパック以降で利用可能です。 これは、製品に付属しているデフォルト・インストール・ユーティリティーを使用すると、デフォルトでインストールされます。 コンパイラーとデバッグ・エンジンのインストール方法について詳しくは、インストール・ガイドを参照してください。

Linux on x86 または Windows 10 デバッグ・クライアントのインストール

デバッグ・クライアントである Remote Debug Eclipse User Interface (p2 リポジトリー) は、既存の Eclipse インスタンスを拡張する Eclipse 機能のセットとして提供されています。

p2 リポジトリーをダウンロードして機能をインストールする方法について詳しくは、IBM Debug for Linux on x86 Remote Debug Eclipse User Interface installation を参照してください。

アクセシビリティー機能

アクセシビリティー機能は、運動障害または視覚障害など身体に障害を持つユーザーがソフトウェア・プロダクトを快適に使用できるように サポートします。

IBM Debug for Linux on x86 には、次のアクセシビリティー機能が備わっています。

- 編集可能なオブジェクトをカーソルの代わりにビジュアル・フォーカスしたインディケーター、および強調表示されたボタン、メニュー項目および他の選択項目。
- ボタンおよび他の選択に応じたツールチップ・ヘルプ。
- ウィザードおよびダイアログ・ボックスによる完全な支援テクノロジーのイネーブルメント。
- 閉じるまで存在するメッセージ、ダイアログ・ボックス、およびウィザード。
- 吹き出しによるイメージの説明およびマークされたテーブル・ヘッダーを含むドキュメンテーション。
- ユーザー・インターフェース・キーボード・ナビゲーション。

  注 : Eclipse IDE では、 Ctrl+F10 を押すと、エディターのマーカー・バーのコンテキスト・メニューが表示されます。

キーボード使用によるユーザー・インターフェースのナビゲート

ユーザー・インターフェースは、キーボードを使用してナビゲート可能です。タブ・キーを繰り返して使用することによって、 特定のスコープ (例えば、あるダイアログ、またはビューとその関連アイコン) 内でコントロールからコントロールへの移動を行うことができます。ユーザー・インターフェースの主要なコ

ントロールまでナビゲートするか、またはタブ・キーを使用するビュー (エディター など) の外に移動するには、Ctrl+Tab を使用します。

メニュー

次のように、キーボードを使用してメニューに アクセスできます。

- メインメニュー・バーにあるメニューにアクセスするには、F10 を使用します。
- 現行ビューのコンテキスト・メニューを開くには、Shift+F10 を使用します。

  注: このショートカットは、 使用しているウィンドウ・マネージャーに依存します。ほとんどの場合は Shift+F10 です。
- 現行ビューのプルダウン・メニュー (ある場合) を開くには、Ctrl+F10 を使用します。 エディターの場合、Ctrl+F10 を使用すると、エディター領域の左にマーカー・バーのメニューが開きます。
- 特定の項目のメインメニューをアクティブにするには、Alt キーとニーモニックを 同時に使用します (例えば、「ウィンドウ」メニューを開くには、Alt+W を使用します)。
- Microsoft Windows のみ: Alt を押すと、メニュー・バーにフォーカスが移ります。

コントロール

ダイアログ・ボックス、設定ページ、およびプロパティー・ページ内の 大部分のコントロールのラベル (例えば、ボタン、チェック・ボックス、ラジオ・ボタン) には、 ニーモニックが割り当てられています。あるラベルに関連付けられたコントロールにアクセスするには、 そのラベル中の下線付きの文字と一緒に Alt キーを使用します。

ナビゲーション・コンテキスト

設定ダイアログ およびプロパティー・ダイアログに対して、ナビゲーション・コンテキストが保管されます。設定ダイアログおよびプロパティー・ダイアログで選択されたページは、ダイアログの起動と起動の間に保管されますが、ユーザー・インターフェースの起動と起動の間には保管されません。

エディター、ビュー、およびパースペクティブの循環

エディター、 ビュー、およびパースペクティブ間での切り替えのため、ユーザー・インターフェースには、Ctrl キーとファンクション・キーによって起動される循環機能が備わっています。これらの循環機能はすべて、 最後に選択されていたものを撤回し、2 つの項目間を迅速に移動すること を可能にします。循環機能には、次のものがあります。

- Ctrl+F6 - エディターに移動する
- Ctrl+F7 - ビューに移動する
- Ctrl+F8 - パースペクティブに移動する

これらに加えて、Ctrl+E を使用して、エディターのドロップダウンをアクティブにでき、 Ctrl+PageUp および Ctrl+PageDown を使用して、オープンしている複数エディター間での切り替えを行うことができます。

キー・アシスト

ユーザー・インターフェースの多くのアクションには、 キーボードでの操作が結び付けられています。メインメニューから 「**ヘルプ**」**>**「**キー・アシスト**」 を 選択すると、使用可能なキーボード操作の一覧にアクセスできます。

ヘルプ・システム

以下のキー組み合わせを使用して、 キーボードでヘルプ・システムをナビゲートできます。

- フレーム (ページ) の内側でタブを押すと、次のリンク、ボタン、またはトピック・ノード に移動します。

- ツリー・ノードを展開するには、右矢印を押します。ツリー・ノードを 縮小するには、左矢印を押します。
- 次のトピック・ノードに移動するには、下矢印またはタブを押します。
- 前のトピック・ノードに移動するには、上矢印または Shift+Tab を押します。
- 選択されたトピックを表示するには、Enter を押します。
- 一番上までスクロールするには、Home を押します。一番下までスクロールするには、End を押します。
- 戻るには、Alt と左矢印を押します。進むには、Alt と右矢印を押します。
- 次のフレームまたはツールバーに移動するには、Ctrl+Tab (Mozilla または Mozilla ベースの ブラウザーを使用している場合は Ctrl+F6) を押します。
- 前のフレームに移動するには、Shift+Ctrl+Tab を押します。(Mozilla または Mozilla ベースのブラウザーを使用している場合は、Shift+Ctrl+F6 を押します)。
- トピック内容を表示しているフレームに移動するには、Alt+K を押します (Windows 上で 組み込みヘルプ・ブラウザーを使用しているか、Internet Explorer を使用している場合)。
- 「目次」タブに移動するには、Alt+C を押します。
- 「検索結果」タブに移動するには、Alt+R を押します。
- タブ間を移動するには、右矢印/左矢印を押します。
- 別のビューに切り替えるには、タブを選択してから Enter を押します。
- 別のビューに切り替え、そのビューに移動するには、タブを選択してから上矢印を押します。
- 検索入力フィールドに移動するには、Alt+S を押します。
- 現行ページまたはアクティブなフレームを印刷するには、Ctrl+P を押します。
- 現行ページまたはアクティブなフレーム内の文字列を検索するには、Ctrl+F を押します (Windows 上で組み込みヘルプ・ブラウザーを使用しているか、Internet Explorer を使用している場合)。

ヘルプ・システムのポップアップ・ダイアログ にある大部分のコントロールのラベルには、ニーモニックが割り当てられています。あるラベルに関連付けられたコントロールにアクセスするには、 下線付きの文字と一緒に Alt キーを使用します。

デバッグの準備

デバッガー・ユーザー・インターフェース・デーモン (デバッグ・デーモン) が コンパイル型言語デバッガー・エンジン (デバッグ・エンジン) を listen していないと、デバッグ・セッションを開始することはできません。また、 アプリケーションが適切なデバッグ・オプションを指定してコンパイルされている必要もあります。

デバッグ・デーモンがデバッグ・エンジンの listen を行うようにする 設定については、関連トピックを参照してください。

ソース・コード・レベルでプログラムをデバッグするためには、オブジェクト・ファイル内にシンボル情報とデバッグ・フックを生成するようコンパイラーに指示する特定のコンパイラー・オプションを指定して、プログラムをコンパイルする必要があります。最適化 (NOOPTIMIZE) を指定せず、 -g または TEST オプションを指定してコンパイルしてください。

-g オプション
OPTIMIZE オプション
OPTIMIZE は、オブジェクト・プログラムの実行時間を短縮するために使用します。 最適化によって、 オブジェクト・プログラムが使用するストレージの量を減らすこともできます。 実行される最適化には、 定数の伝搬、命令スケジューリング、および結果が使用されない計算の除去があります。

TEST オプション

TEST は、デバッガーがシンボリック・ソース・レベルのデバッグを実行できるようにするシンボルおよびステートメントの情報が含まれる オブジェクト・コードを生成する場合に使用します。

デバッグ・エンジンの *listen*

デバッグ・デーモンは、ユーザー・インターフェースの一部であり、 エンジン接続を listen します。デバッグする対象のプログラム言語またはデバッグ・セッションを起動した方法に基づいて、 デバッグ・デーモンが自動的にデバッグ・エンジンを listen するか、または、 そうするように設定する必要があります。

下記で説明するように、一部のリモート・デバッガーは SSL セキュア接続の使用をサポートしています。

- 316 ページの『デバッグ・デーモンの SSL サポート』

デバッグ・デーモンがデバッグ・エンジンを listen しているかどうかを 判別するには、次の 3 つの方法があります。

- 「デバッグ」ビューでデーモン・アイコンの状態を監視します。デーモンが listen している 場合、アイコンは  の ように表示されます。デーモンが listen していない場合、アイコンは  のように表示されます。
- デーモン・アイコンの右にある下矢印をクリックします。デーモンが listen している場合の先頭のメニュー項目は、「**デバッグ UI デーモンが次のポートで listen しています: <port number>**」です。デーモンが listen していない場合の先頭の メニュー項目は、「**ポートでの listen の開始: <port number>**」です。
- デーモン・アイコンの上にカーソルを置いて吹き出しを表示します。デーモンが listen している場合の吹き出しのツールチップは、「**デバッグ UI デーモンが次のポートで listen しています: <port number>。このボタンを選択すると、listen が停止します**」です。デーモンが listen していない場合の吹き出しのツールチップは、「**デバッグ UI デーモンは listen していません。このボタンを選択すると、ポート <port number> での listen を開始します**」です。

セキュリティー上の理由があるか、または、デーモンのポート番号をマルチユーザー・マシン上の別のユーザーが 必要としている場合、デバッグ・デーモンを停止したいことがあります。ただし、 コンパイル型言語のデバッグ・セッションを開始するには、デーモンを listen する必要があります。

デバッグ・デーモンがデバッグ・エンジンを listen していない場合、 それを開始するには、以下のいずれかのタスクを実行します。

- デーモン・アイコン () をクリックします。デーモンが開始したことを示すアイコンに変わります。
- デーモン・アイコンの右にある下矢印をクリックし、メニューから「**ポートでの listen の開始: <port number>**」を選択します。

デバッグ・デーモンが listen している場合、それを停止するには、 以下のいずれかのタスクを実行します。

- デーモン・アイコン () をクリックします。デーモンが停止したことを示すアイコンに変わります。
- デーモン・アイコンの右にある下矢印をクリックし、メニューから 「**listen の停止**」を選択します。

デバッグ・デーモンがデバッグ・エンジンの listen に 使用するデフォルトのポートは、8001 です。「デバッグ」ビューまたは「デバッグ・デーモン設定」ページで、 デーモンのポート番号を変更できます。また、デバッグ・デーモンが listen する ポートの範囲を指定することもできます。

「デバッグ」ビューからポート番号を変更するには、 以下のステップを実行します。

1. デーモン・アイコンの右にある下矢印をクリックし、メニューから 「**ポートの変更**」を選択します。
2. 「設定」ダイアログ・ボックスが開きます。「**デーモン・ポート**」フィールドに、使用するポート番号またはポート番号の範囲 (このトピックで後述します) を入力します。
3. 「**OK**」をクリックして、ポート番号を変更します。ポート番号をそのデフォルト値に戻すには、「**デフォルトを復元**」ボタンをクリックします。

「デバッグ・デーモン設定」ページからポート番号を変更するには、 デバッグ設定に関する関連トピックを参照してください。

ポート番号の範囲を指定するには、 値をコンマおよびハイフンで区切ります。例えば、`8001,8003,8900-8903` と指定すると、 デバッグ・デーモンは、8001、8003、8900、8901、8902、および 8903 の番号の範囲にある最初に 使用可能なポートを使用します。デーモン接続が確立された後、 デーモン・アイコンの上にカーソルを置くと、吹き出しツールチップに、どのポートが使用 されたのかが示

されます。あるいは、デーモン・アイコンの右にある下矢印を クリックして、メニューに含まれているポート番号を確認することもできます。

注 :

- まだシステムで使用されていなければ (既に使用されている場合はクライアントでメッセージが表示されます)、デフォルト・ポートを使用することをお勧めします。
- 以前に設定されたデーモン・ポートが、ワークベンチでデバッグ・セッション用に 現在使用されている場合、デーモン・ポートの変更は、ポートを通して作成された、前の接続には 影響しません。以降のエンジン接続には、新しいポート番号が 使用されます。
- 新しいポート番号が既に別のアプリケーションによって使用されている場合、 その新規ポートでデーモンが listen しようとすると、エラー・メッセージを伴うプロンプトが 出されます。この場合、別のアプリケーションが使用していないデーモン・ポート番号 を選択してください。

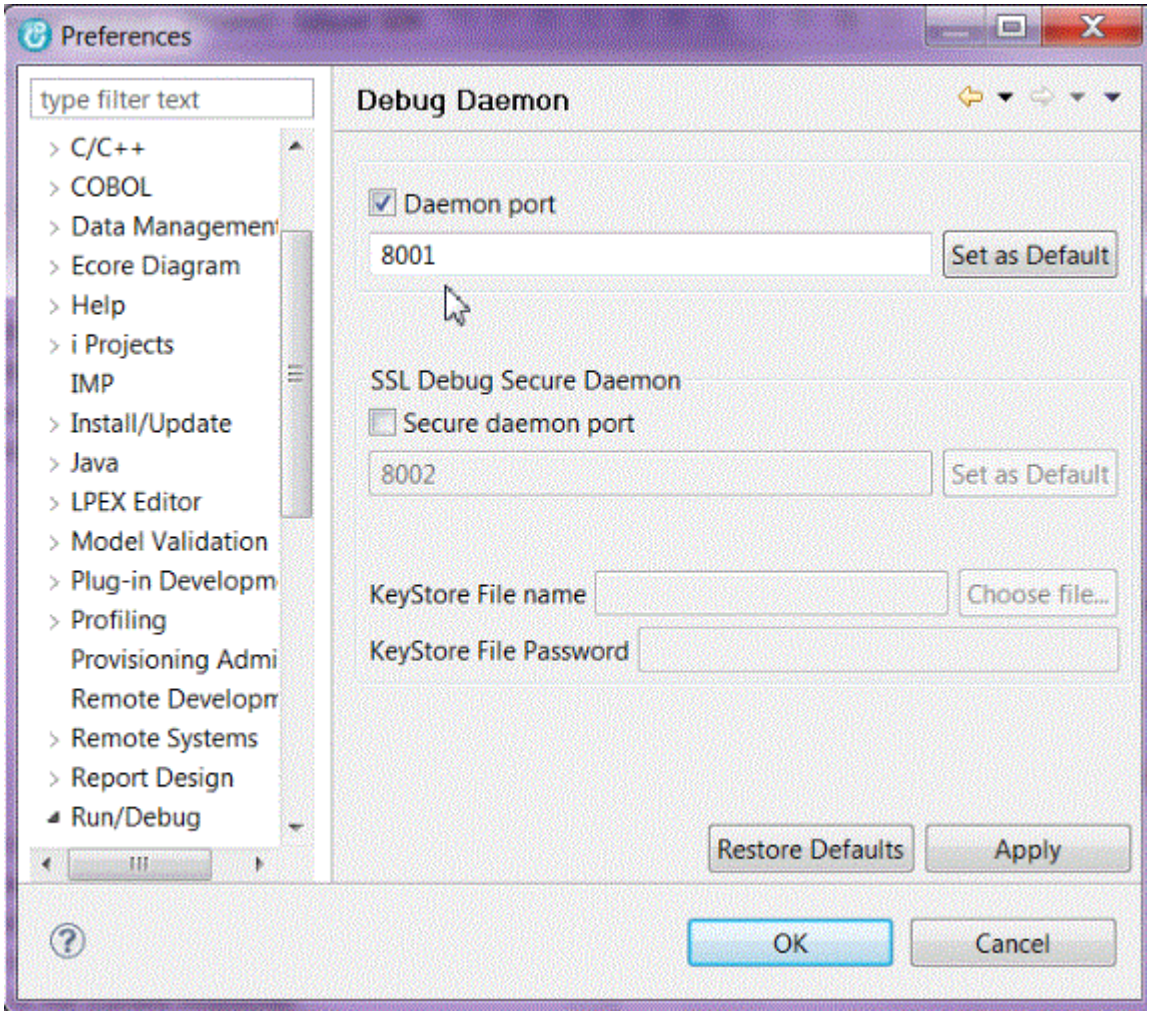
デバッグ・デーモンの *SSL* サポート
従来のデバッグ UI デーモンに加え、SSL セキュア・デバッグ・デーモンを使用できます。 ただし、SSL は、SSL をサポートするリモート・デバッガーと共に使用する場合にのみ機能します。

注 : デバッグ接続を保護するために SSH トンネリングを使用している場合は、**デーモン・ポート**への接続が必要です。

SSL セキュア・デバッグ・デーモンを有効にするためには、以下の 2 つの方法があります。

- デーモン・アイコン  の右にある 下矢印をクリックし、「**ポートの変更...**」を選択します。
- 「**ウィンドウ**」**>**「**設定**」をクリックして設定ページを開きます。 メニューで「**実行/デバッグ (Run/Debug)**」を展開し、「**デバッグ・デーモン**」を選択します。

次のダイアログが表示されます。



SSL セキュア・デーモンを有効にするには、「SSL デバッグ・セキュア・デーモン」チェック・ボックスを選択し、ポートを指定します。 鍵ストア・ファイルおよびパスワードも定義する必要があります。

ポート番号の範囲を指定するには、 値をコンマおよびハイフンで区切ります。例えば、 8001,8003,8900-8903 と指定すると、 デバッグ・デーモンは、8001、8003、8900、8901、8902、および 8903 の番号の範囲にある最初に 使用可能なポートを使用します。

デバッガー・ユーザー・インターフェースからのクライアント・マシン *IP* アドレス取得
デバッグ・セッションを起動できるためには、デバッガー・クライアント (ユーザー・インターフェース) を 実行するマシンの IP アドレスが必要です。

デバッガー・ユーザー・インターフェースを実行するクライアント・マシンの IP アドレスを 取得するには、以下のステップを実行します。

1. 「デバッグ」ビューで、デーモン・アイコンの右にある下矢印をクリックし、メニューから 「**ワークステーション IP の取得**」を選択します。
2. 「ワークステーション IP の取得」メッセージが開き、クライアント・マシンの 現行 IP アドレスが示されます。

このダイアログ・ボックスから IP アドレスを選択し、それをコピーして貼り付ける ことができます。

注 : ワークステーションに複数の LAN アダプターが設置されている場合や、 ワークステーションとサーバーの間にルーターまたは仮想プライベート・ネットワーク (VPN) が介在している場合は、 このダイアログに複数の IP アドレスがリストされることがあります。 サーバーが使用できるアドレスを見つけるには、それぞれの IP アドレスを試してみなければならない場合があります。

デバッグ関連のコンパイラー・オプション

COBOL for Linux on x86 のプログラムのデバッグに関連するコンパイラー・オプションには、以下のものがあります。

コンパイラー・オプション	定義
-g	ソース・コードのデバッグ情報を生成するようコンパイラーに指示 します。作成したコードをデバッグしたい場合は、このオプションを 指定する必要があります。
TEST	-g と同等です。

デバッグ設定値の設定

使用するデーモン・ポート番号、デバッガー・エディター設定、およびデバッグ・エンジンからの応答を待機する時間などの、さまざまなデバッグ関連設定値を設定できます。

Eclipse IDE メニュー・バーから「**ウィンドウ**」 > 「**設定**」を選択すると、「設定」ダイアログ・ボックスが開きます。このダイアログ・ボックスで、「**実行/デバッグ**」ノードを選択かつ展開して、さまざまなデバッグの設定を設定できます。 これには、コンパイル型言語アプリケーションのデバッグ時に設定できる、(「**アニメーション表示ステップイン**」、「**デバッグ・デーモン**」、 および「**コンパイル型デバッグ**」ノードにある) 次の設定が含まれます。

アニメーション表示ステップインの設定

「設定」ダイアログ・ボックスで、「**実行/デバッグ**」 > 「**コンパイル済みデバッグ**」 > 「**アニメーション表示ステップイン**」を選択すると、「アニメーション表示ステップイン」設定ページが開きます。 このページでは、アニメーション表示ステップイン・アクションの現行のステップイン・ペース (または現行のステップイン遅延) および最大ペース (または最大ステップイン遅延) を設定できます。さらに、「デバッグ」ビューでアニメーション表示ステップイン・アクションの「**スピードアップ**」または「**スローダウン**」を選択したときに、ペースを引き上げるまたは引き下げる時間量を設定できます。

この設定ページの各フィールドのデフォルト値は以下のとおりです。

- 「**現在のペース (ミリ秒)**」フィールド: 2 秒、つまり 2000 ミリ秒
- 「**スピードアップ/スローダウンする割合 (ミリ秒)**」フィールド: 200 ミリ秒
- 「**最大ペース (ミリ秒)**」フィールド: 5 秒、つまり 5000 ミリ秒

デバッグ・デーモンの設定

「設定」ダイアログ・ボックスで、「**実行/デバッグ**」 > 「**デバッグ・デーモン**」 を選択すると、 「デバッグ・デーモン」ページが開きます。 このページで、デーモンがデバッグ・エンジン接続を listen する、ポート、ポートの範囲、またはポートの組み合わせを設定できます。 ポートの範囲と組み合わせは、コンマで区切られたリスト、ハイフンによる範囲設定、またはこの 2 つの組み合わせで指定できます。デフォルトでは、ポートは 8001 に設定されています。

注 : 問題が起きたか、またはデフォルト・ポートが既に使用されているマルチユーザー・マシンで実行されている場合を除き、デフォルト・ポートはそのままにしておくことをお勧めします。

このデーモン・ポートを「デバッグ・デーモン設定」ページで変更する場合、 設定ページの「**デフォルトを復元**」ボタンをクリックすれば、 簡単にデフォルト値の設定に戻せます。

デーモンが、デバッグ・エンジンを listen するようにユーザー・インターフェースで設定されている場合に、この設定ページでデーモン・ポート番号を変更すると、デバッガーは新しいポート番号でデーモンを開始します。

デバッガー・エディターの設定

「設定」ダイアログ・ボックスで、「**実行/デバッグ**」 > 「**コンパイル済みデバッグ**」 > 「**デバッグ・エディター (Debug Editor)**」を選択すると、「デバッガー・エディター」ページが開きます。 このページでは、吹き出しおよびタイプ評価を許可するようエディターを設定できます。「**ホバー評価を許可する**」チェック・

ボックスを選択すると、 デバッガー・エディター内の式に移動したときに、その値をポップアップ表示させることができます。「**タイプを吹き出しで表示**」 チェック・ボックスが選択されている場合、式のタイプがポップアップに表示されます。

「**デバッグ中は常に使用 (Always use while debugging)**」チェック・ボックスによって、デバッグ中にソースを開くエディターが決まります。 これにより、ステップの際の表示内容も決まります。 このチェック・ボックスのデフォルト設定は、 このデバッガーをインストールした際の製品によって異なります。 このチェック・ボックスが選択解除になると、次のようになります。

- ワークベンチ設定のソース・ファイル・タイプに関連付けられたデフォルト・エディターで、ソースが開きます。
- ソースまたはリストがホスト・デバッグ・エンジンによってのみ検出可能な場合、 ソースまたはリストはデバッガー・エディターで開きます。

このセクションでは、次の設定も行うことができます。

- ソース・ファイル全体をロードするようエディターを設定します。デフォルトでは、 この設定はオフです。「**ファイル内容全体の読み込み**」チェック・ボックスが選択されている場合、 ソース・ファイル全体がロードされますが、パフォーマンスが悪化する可能性 があります。この 設定をオンにする必要がある場合としては、特定の拡張 LPEX エディター・アクションを使用する場合、 例えば、ファイル内で増分検索を行ったり、大括弧の突き合わせ機能を使用する場合などが考えられます。
- モニター対象の式がエディターでダブルクリックされたときに、「モニター」 ビューにその式を追加できるように、デバッガーを設定します。
- すべてのデバッグ・セッションのデバッガー・エディターで、実行中の現在行を中央そろえとする場合、「**実行行のビューの中央そろえ**」チェック・ボックスを 選択します。
- 実行行の色を選択します。

コンパイル型デバッグの設定

「設定」ダイアログ・ボックスで、「**実行/デバッグ**」 > 「**コンパイル型デバッグ**」を 選択すると、「コンパイル型デバッグ」ページが開きます。 このページでは、これらの設定を設定できます。

プログラム・プロファイル

プログラム・プロファイルを削除するように選択することができます。 プログラム・プロファイルは、デバッガーによって、デバッグしたそれぞれのプログラムごとに保管されます。 プログラム・プロファイルには、 例えばブレークポイントおよびモニター設定のような情報が組み込まれています。現在保管されているプログラム・プロファイルをすべて削除するには、 このボタンを選択します。

現在のデバッグ・セッションでデバッグ中のプログラムにのみ 例外ブレークポイント設定を適用する場合は、 「**プログラムによる例外ブレークポイント設定を保管**」チェック・ボックスを選択します。このチェック・ボックスが選択されていない場合、 例外ブレークポイント設定は、現在のデバッグ・エンジンによってデバッグされる すべてのプログラムに適用されます。

エンジン応答時間

デバッグ・エンジンからの応答をデバッガーが待つ時間の長さを指定する場合は、 「**待ち時間 (秒)**」ラジオ・ボタンを選択し、 フィールドに待ち時間の長さを秒単位で入力します。 デフォルトでは、デバッガーはエンジン応答を 15 秒待ちます。 「**待ち時間**」ラジオ・ボタンが選択されているときに、 エンジンが指定された待ち時間内に応答しない場合は、ダイアログ・ボックスが表示され、 エンジンの応答を待機し続けるよう指示するプロンプトが出されます。 待機し続けないことを選択すると、バッグ・セッションは終了します。

デバッグ・エンジンからの応答をデバッガーが連続して待機するようにする場合は、 「**連続**」ラジオ・ボタンを選択します。 このラジオ・ボタンを選択すると、エンジンが応答しない場合、手動でデバッグ・セッションを終了する必要があります。

「**エンジン接続のトレース**」設定は、診断目的に使用されます。 この設定を選択すると、IBM でのみ読み取り可能な大容量ファイルをディスクに書き込めます。この設定は、IBM サービス技術員の指示があった場合にのみ選択してください。

コンパイル型言語のデバッガー・エンジン

デバッガーのクライアント/サーバー設計により、ネットワーク内の他のシステム上でリモートに実行しているプログラムをデバッグし、 ワークステーションのローカル・リソースを使用してデバッグ・セッションを表示および制御できます。

デバッガー・バックエンド (デバッグ・エンジン とも呼ばれます) は、 デバッグ対象のプログラムと同じシステム上で稼働します。このシステムは、 ネットワーク経由でアクセス可能ないずれかの Linux on x86 システムです。

注 : この製品付属のデバッグ・エンジンは、バージョン 1.0 エンジンと識別されます。

デバッガー・エンジンの開始

ユーザー・インターフェース・クライアントからデバッグを行う場合、 ユーザー・インターフェース・デーモン・モード を使用してデバッガー・エンジンを開始します。このモードでは、ユーザー・インターフェースが最初に開始され、ユーザー・インターフェースはエンジンが接続されるのを待ちます。

irmtdbgc コマンドは、リモート・システム上の デバッグ・エンジンを開始します。irmtdbgc コマンドの構文は、irmtdbgc [debugger parms] debuggee_name [debuggee parms] です。ここで、[debugger parms] には、以下のパラメーターを任意の順序で指定できます。

パラメーター	説明
-qhost= <host:port>	<host> は、デバッガー・ユーザー・インターフェースが 実行するマシンのホスト名を指定します。これは、ホスト名または IP アドレスのいずれかです。 指定されない場合、環境変数 DER_DBG_ADDR の値が使用されます。 どちらも指定されていない場合、値 localhost が使用されます。 <port> はオプション です (デフォルトでポート 8001 が想定されます)。
-i	指定されると、デバッガーがデバッグ対象をロード後すぐに停止し、アプリケーションのメイン・エントリー・ポイントまでの実行をしないことを指定します。
-a xxxx	xxxx はプロセス ID であるか、またはアプリケーションの名前が固有である場合は ps コマンドで示されるプロセスの名前です。
-qconsole=<remote, local, or GUI>	これは、デバッグされるプログラムが表示 されるコンソールを制御します。 -qconsole=remote を指定した場合は、出力がローカル・セッションおよびユーザー・インターフェースに送信されます。 -qconsole=local が指定されている場合、irmtdbgc コマンドを入力したコンソール・ウィンドウにコンソールが表示されます。 -qconsole=GUI が 指定されている場合、別のウィンドウにコンソールが表示されます。 この パラメーターのデフォルト値は remote です。

パラメーター	説明
-s	デバッグ対象がすぐに実行することを指定します。デバッグ 対象は、プロファイルからブレークポイントに到達したときか、シグナルが発生した場合に停止します。
--	これは、次のパラメーターがデバッグ対象の名前であることを示します。これが 必要なのは、デバッグ対象の名前が文字 '-' で始まる場合だけです。

デバッガーは、 デバッグするプログラムを *PATH* 環境変数を 使用して検索します。

デバッガー・エンジンの環境変数

デバッグ・エンジン環境変数は Linux 環境で設定されます。

次の環境変数が、エンジンの動作を制御します。

環境変数	説明
DER_DBG_PATH	デバッガーがソース・ファイルを見つけるために 使用する一連のパスを設定します。これらのパスは、デバッグ情報にソース・ファイルの完全修飾名が含まれて いない場合に使用されます。
DER_DBG_ADDR	ユーザー・インターフェース・デーモン・モードで使用されるデフォルト・ホストを指定します。 これは、ホスト名または IP アドレスのどちらです。デフォルトは localhost です。 これは、コマンド行パラメーター -qhost で上書きされます。 アドレスを 指定する場合は、ユーザー・インターフェース・デーモン・モードで使用されるデフォルト・ポートを 組み込むこともできます。ポート番号を組み込むには、DER_DBG_ADDR=<ホスト名またはアドレス>:<ポート> と 指定します。デフォルトでは、使用されるポート番号は 8001 です。 この環境変数で指定されたポートは、コマンド行パラメーター -quiport で上書きされます。
DER_DBG_TRACE	この環境変数は、エンジン・トレース・ファイルのロケーションを指定するのに使用します。
DER_DBG_PICLDUMP	この環境変数は、EPDC トレース・ファイルの ロケーションを指定するのに使用します。

ファイアウォールに関する考慮事項

エンジンとユーザー・インターフェースとの間にファイアウォールがある場合、 エンジンとユーザー・インターフェースとの間で通信が実行されるよう、適切なファイアウォール規則を設定する必要があります。

ファイアウォールでクライアントに転送するように構成されたポート上のファイアウォールの WAN IP アドレスに接続するよう、エンジンに指示する必要があります。例えば、次のように指定します。

```
Client - ip 10.1.1.7, daemon port 8101
Firewall - wan ip 10.10.10.3, configured to forward to port 8101 to 10.1.1.7
Server - start the engine to connect to the client:
irmtdbgc -qhost=10.10.10.3:8101 a.out
```

注 :

- ファイアウォールの多くは、ポート 8001 をブロックします。場合によっては、上の例のように別のポートを使用する必要があります。
- また、セキュア・シェル (ssh) トンネル経由でワークステーションに接続するように、デバッグ・エンジンに指示することもできます。 技術情報 1438892、Debugging through a Secure Shell tunnel を参照してください。 ワークステーション上のクライアントのデバッグ構成によってデバッグ・セッションを開始する場合は、 デバッグ構成の「**詳細設定**」タブに、接続をトンネルするオプションが存在する場合があります。

アプリケーションのデバッグ

デバッグ・セッションを起動した後、デバッグ・ビューでさまざまなデバッグ・タスクを使用することができます。

デバッグに使用可能なビューには、以下のものがあります。

- 「デバッグ」ビュー: プログラムのデバッグを管理します。
- デバッガー・エディター: アプリケーションのソースを表示します。
- 「ブレークポイント」ビュー: ブレークポイントを設定して、操作します。
- 「変数」ビュー: アプリケーション内の変数をリストして、編集します。 詳しくは、334 ページの『変数のインスペクション』を参照してください。
- 「レジスター」ビュー: プログラム内のレジスターを表示します。
- 「モニター」ビュー: モニター対象として選択した変数、式、およびレジスターを扱います。
- 「モジュール」ビュー: プログラムの実行中にロードされるモジュールのリストが表示されます。 アプリケーション内の個々のコンパイル・ユニットやソース・ファイルまでナビゲートし、関数エントリー・ポイントを確認して、それらにブレークポイントを設定することができます。
- 「コンソール」ビュー: プログラムの画面出力を表示します。
- 「メモリー」ビュー: アプリケーションにより使用されるメモリーを表示およびマップします。

コンパイル型言語デバッガー

コンパイル型言語デバッガーは、ソース・レベルの対話式デバッガーです。これは、 デバッグ・エンジンへのネットワーク接続を介して接続されたクライアントで機能します。コンパイル型言語デバッガーを使用すると、COBOL プログラムをデバッグできます。

デバッガーにより、アプリケーション・ソース・ファイルとこのソース・ファイル内の関数が表示されます。指定した行または条件での、単一ステップ、ステップスルー、ステップオーバー、または実行停止を行うことができます。実行を制御しながら、変数、レジスター、メモリー、コール・スタックなどの要素をモニターできます。

コンパイル型言語デバッガーは、インターネット・プロトコル、バージョン 6 (IPv6) に対応しています。

デバッガー・エディター

デバッグ・セッションを起動すると、デバッガーがデバッガー・エディターを使用してソースを表示します。このエディターは、いくつかのデバッグ・アクションを提供します。

デバッグ・セッションを起動すると、 ソースが参照モードのエディターで開き、それは変更できません。ソースをデバッガー・エディターで変更できるのは、それがデバッグ・セッションの外のエディターで開かれているときに限られます。

ソースが見つからないときには、エディターはソースなしで開きます。ソースを見つける方法については、325 ページの『ソースの探索』を参照してください。

関連タスク

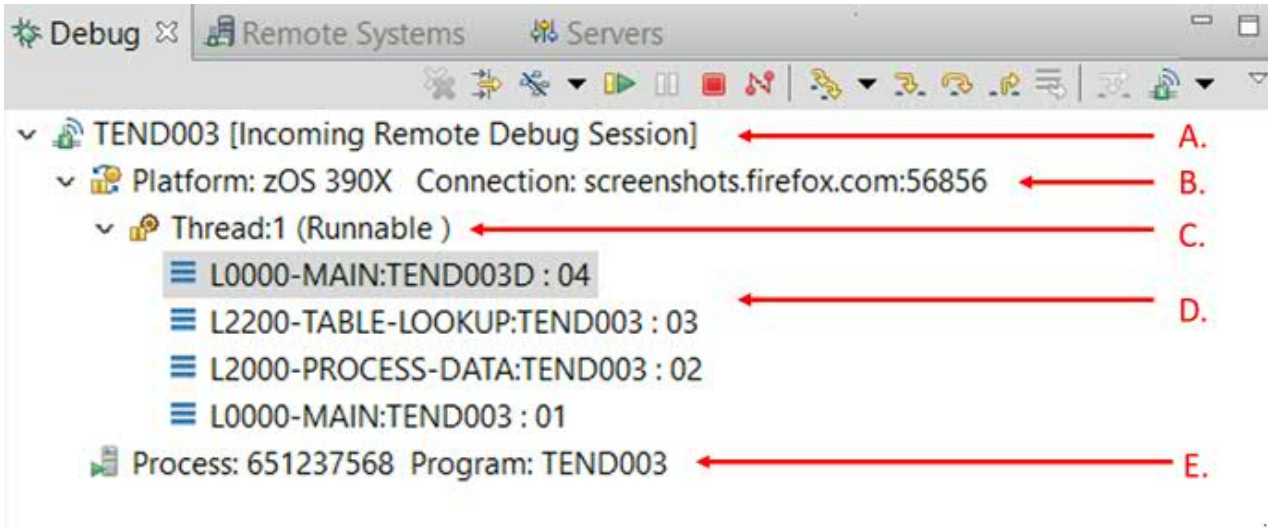
326 ページの『異なるデバッグ・ビュー間の切り替え』

デバッガー・エディターの「**ビューの切り替え**」メニューを使用すると、デバッグ・セッション中に異なるデバッグ・ビュー間で切り替えることができます。「**デフォルト・ビューの設定**」アクションを使用すると、特定のデバッグ・ビューをデフォルト・ビューとして選択できます。

「デバッグ」ビューの使用

「デバッグ」ビューを使用すると、プログラムのデバッグを管理することができます。 このビューには、デバッグするターゲットごとの中断スレッドのスタックが表示されます。 デバッグする各プログラムまたはアプリケーションのデバッグ・ターゲット (スレッドおよびスタック・フレームと関連付けられています) が、「デバッグ」ビューに表示されます。

「デバッグ」ビューでは、プログラムの各スレッドは、ツリー内のノードとして表示されます。「デバッグ」ビューにおける典型的なデバッグ・ターゲットは、次の図のように表されます。

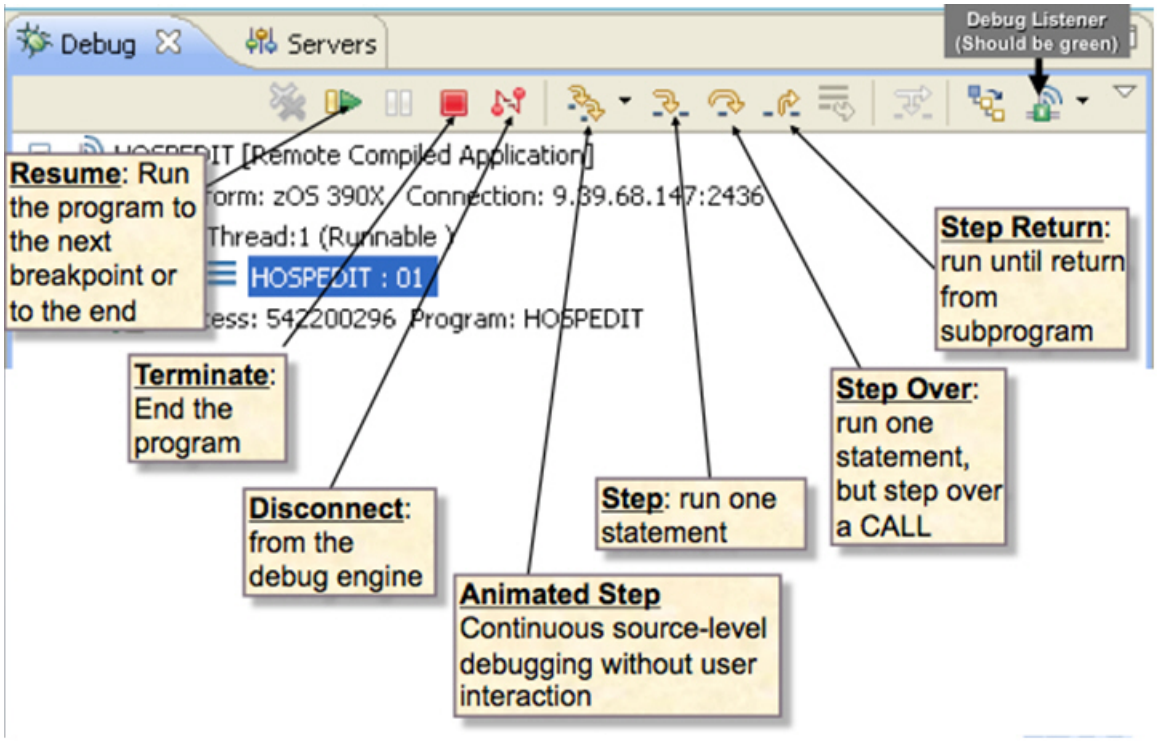


「デバッグ」ビューでは、プログラムのデバッグ・セッションを開始 するために使用される起動が、最上位のノード・レベルに 表示されます (図のポインター A.)。起動のすぐ下に、デバッグ・エンジンを 表すノードが表示されます (図のポインター B.)。その次に、プログラム内の各 スレッドが表示されます (図のポインター C.)。プログラム実行が停止すると、デフォルトでは、 停止するスレッドのノードが自動的に展開され、そのスタック・フレームが 表示されます (図のポインター D.)。他のスレッドを手動で展開すると、それらのスレッドは 次回にプログラムが中断したときに自動的に展開されます。最後に、デバッグされるプロセスおよび プログラムを表すノードが表示されます (図のポインター E.)。

注 : 段落フレームはサポートされません。

プログラム実行が中断すると、選択したスタック・フレームのソースがエディターで表示されて、プログラムでこれから実行するソース行が強調表示されます。プログラム内に 多数のスレッドがある場合、停止を引き起こしたスレッドのスタックが、デバッグ・フレームの最後から スクロールオフすることがあります。

この後のセクションでは、「デバッグ」ビューのツールバー・アイコンを使って実行できる操作について説明します。 また、下の図に示されているように、デバッガー・デーモンを設定するために「デバッグ」ビューを使用することもできます。 この詳細情報については、デバッグ・エンジンの listen に関する関連トピックを参照してください。



プログラムの実行、終了、および切り離し

「デバッグ」ビューで、以下の基本的なデバッグ・アクションを実行できます。

- アプリケーションを実行するには「**再開**」 をクリックするか、F8 を押します。
- デバッグ・セッションを終了するには、「終了」( ) をクリックするか、Shift+F8 を押します。あるいは、終了するデバッグ・ターゲット (または、そのスレッドまたはスタックの 1 つ) を右クリックし、終了アクションのいずれか 1 つを選択します。
- プログラムから切り離し、プログラムをそのまま実行するには、「切断」( ) をクリックします。デバッグ対象のプログラムが開始された方法によっては、このアクションが使用不可になる場合もあります。

プログラムのステップスルー

スレッドが中断されている場合は、ステップ関連のコントロールを使用して、プログラムの実行を 1 行ずつ ステップスルーできます。ステップ・オペレーション中にブレークポイントまたはイベントが 検出されると、実行はブレークポイントまたはイベントで中断し、ステップ・オペレーションは終了します。 **ステップ・コマンド**を使用すると、プログラムを一度に 1 命令または 1 ロケーションごとにステップスルーできます。

以下のステップ・コマンドを使用できます。

- **ステップオーバー** ( ) (F6): ステップオーバーを発行すると、 プログラムは次のソース行に進みます。
- **ステップイントゥ** ( )(F5): ステップイントゥを発行すると、プログラムは次のステートメントに進みます。 現在行が別の関数の呼び出しを含んでいる場合、 デバッガーは、その関数で停止します。

  このコマンドの動作には、 「**ステップ・フィルターの使用**」アクション ( )(Shift+F5) が影響します。フィルターが オフ (ボタンが選択されていない) の場合、デバッガーは、呼び出された関数で、その関数にデバッグ情報が含まれていなくても停止し、逆アセンブルが表示されます。フィルターがオン (ボタンが選択されている) の場合、デバッガーは、 ソースが表示可能である場合のみ、呼び出された関数で停止します。ソースが表示可能でない場合、 デバッガーは、「**ステップオーバー**」が発行された場合と同じ動作をします。*DER_DBG_ STEP_DEBUG* デバッグ・エンジン環境変数によって、「**ステップ・フィルターの使用**」アクションの 動作が変わります。

注 : COBOL では、ステップイン・アクションは通常、ステップ・フィルター・アクションが常にオンになっている場合と同じように動作します。これらの言語で作成されたプログラムをデバッグすると、デバッガーはソース・コードで停止しようとします。

- **ステップ・リターン** () (F7): ステップ・リターンを発行すると、 プログラムは、呼び出し側プログラム内の、現在の関数への呼び出しの直後のポイントまで 実行します。 通常、呼び出し命令に続く場所で停止するようにします。 呼び出し側プログラムにデバッグ情報がある場合、これは、ソース行の中央です。
- **アニメーション表示ステップイン** (): このアクションを発行すると、デバッガーは、ステップイン・アクションを繰り返し発行します。 「**アニメーション表示ステップイン**」アクションを再度選択することで、各ステップの間の遅延を制御できます。

関連タスク

327 ページの『ブレークポイントの使用』
ブレークポイントは、指定したポイントでプログラムを停止するようデバッガーに指示するために、実行可能プログラム内に設定される一時マーカーです。 ブレークポイントが検出されると、その行が実行される前に ブレークポイントで実行が中断されます。このポイントで、スレッドのスタックを調べ、 変数、レジスター、およびメモリーの内容を検査できます。 その後、行をステップオーバー (実行) して、それが引数に与える影響を確認することができます。

ソースの探索

アプリケーションをデバッグするとき、デバッグ・エンジンはアプリケーションのソースを見つけようとします。 デバッグ・エンジンがソースを検出できる場合は、デバッガー・エディターでソースが開きます。デバッグ・エンジンがソースを検出できない場合、ソースの「逆アセンブル」ビューがデバッガー・エディターで開きます。

デバッグ・エンジンが容易にソース・ファイルを検出できるようにするには、以下の方法を使用できます。

- 「デバッグ」ビューまたはデバッガー・エディターで、ソース・ロケーションを追加することができます。例えば「**ソース・ルックアップの編集**」によって開く「ソース・ルックアップ・パスの編集」ダイアログでは、追加するソース・ロケーションのタイプを選択できます。 あるいは、 「デバッグ」ビューでスタック・フレームまたはスレッドを右クリックし、 「**ソース・ルックアップの編集**」アクションを選択して、ソース・ロケーション・リストを変更することもできます。

ソース・ロケーション・リストの変更

デバッグ・セッションの開始後は、以下のステップを完了して、ソース・ロケーション・リストの変更または追加を行うことができます。

1. デバッグ・ターゲット (あるいは、そのスレッドまたはスタック・フレームの 1 つ) を右クリックして、「**ソース・ルックアップの編集**」を選択します。

   「**ソース・ルックアップ・パスの編集**」ウィンドウが開きます。
2. 以下のいずれかの手順を実行します。
   - ソース・ロケーションを追加するには、「**追加**」をクリックします。 「**ソースの追加**」ダイアログが開きます。 以下のいずれかのオプションを選択します。
     - 「**ファイル・システム・ディレクトリー**」を選択すると、ローカル・ファイル・システム・ディレクトリーがソース・ロケーション・リストに追加されます。 サブディレクトリーもまた検索するには、「**サブフォルダーの検索**」を選択します。
     - 「**デバッグ・エンジン**」を選択すると、デバッグ・エンジンがソース・ロケーション・リストに追加されます。
     - 「**デバッグ・エンジン・パス**」を選択すると、 デバッガー・エンジンで指定されたパスがソース・ロケーション・リストに追加されます。 複数のパスを指定する場合は、エンジンのプラットフォームに応じて、適切な分離文字を使用して区切ります。 例えば、z/OS エンジンまたは Linux エンジンには、コロン (:) を使用します。「**デバッグ・エンジン・パス**」の設定に対する 変更は、それ以降のデバッグ・セッションで有効です。
   - エントリーを除去するには、ソース・ロケーションを選択して 「**除去**」をクリックします。

- エントリーの順序を変更するには、ソース・ロケーションを選択して「**上へ**」または「**下へ**」をクリックします。

3. ソース・ロケーション・リストにあるソース・ファイル名のすべてのインスタンスを検索するには、「**パス上の重複ソース・ファイルを検索**」を選択します。 デバッガーがそのファイル名の複数インスタンスを見つけた場合は、正しいソース・ファイルを選択するよう求められます。
4. 変更を保存するには、「**OK**」をクリックします。

エディターでのソース・ファイルの変更

以下の条件に 1 つでも当てはまる場合、デバッガーは現在のスタック・フレームに不適切なソースを見つける可能性があります。

- ソースが移動されている。
- プログラムがビルドされたシステムではないシステムでデバッグしている。

この場合は、エディターでテキスト・ファイルを開いて変更します。

1. エディターの中を右クリックして「**テキスト・ファイルの変更**」を選択します。
2. 開くファイルのパスと名前を入力または参照します。

   注 : ローカル・ワークステーション上のファイルを指定する場合は、完全修飾パスとファイル名を入力してください。
3. 指定したソース・ファイルをエディターにロードしてウィンドウを閉じるには、「**OK**」をクリックします。

エディターでソース・ファイルを検出する

ソースが見つからないときには、ソースなしでエディターが開きます。

ソースを見つけるには、以下のいずれかの手順を行ってください。

- 別のエディター・ソース・ファイル名を指定するには、「**テキスト・ファイルの変更**」をクリックします。開くファイルのパスと名前を参照または入力します。

  注 : ワークステーション上のファイルを指定するには、完全修飾パスとファイル名を入力してください。エディター・ソース・ファイルを変更する機能は、デバッグを行う言語、環境、およびプラットフォームによって異なります。
- ソース・ルックアップ・パスを編集するには、「**ソース・ロケーションの追加**」を選択します。 「**ソース・ルックアップ・パスの編集**」ウィンドウが開きます。 ソース・ロケーションの追加については、325 ページの『ソース・ロケーション・リストの変更』を参照してください。

ソースの「逆アセンブル」ビューを開くには、「**逆アセンブル・ビューの表示**」をクリックします。

異なるデバッグ・ビュー間の切り替え

デバッガー・エディターの「**ビューの切り替え**」メニューを使用すると、デバッグ・セッション中に異なるデバッグ・ビュー間で切り替えることができます。 「**デフォルト・ビューの設定**」アクションを使用すると、特定のデバッグ・ビューをデフォルト・ビューとして選択できます。

「**ビューの切り替え**」メニューから、 「展開されたソース」ビュー、「混合」ビュー、および「逆アセンブル」ビューの 3 つのビューを選択できます。 「展開したソース」ビューでは、COBOL ソースの COPY ステートメントが、参照先コピーブックの実際の内容に置換されます。 「混合」ビューには、展開したソースと逆アセンブル指示が表示されます。 「逆アセンブル」ビューには、逆アセンブル指示が表示されます。

別のデバッグ・ビューへの切り替え

「ビューの切り替え」アクションを使用すると、別のデバッグ・ビューに切り替えることができます。 デバッグ・ビューの設定は、現行デバッグ・セッションの現行ファイルにのみ適用されます。

1. デバッガー・エディター内で右クリックします。
2. 「**ビューの切り替え**」メニューを展開します。
3. 「表示」アクションのいずれかを選択して、別のデバッグ・ビューに切り替えます。

「表示」アクションには、「展開されたソースの表示」、「混合の表示」、および「逆アセンブル・ビューの表示」があります。

デフォルト・ビューとなるデバッグ・ビューの選択

「**デフォルト・ビューの設定**」アクションによって、選択したデバッグ・ビューがデフォルト・ビューとして設定されます。 このアクションによって、現行デバッグ・セッションの現行デバッグ・ファイルが、選択されたビューに切り替わります。 その後のデバッグ・セッションでは、選択されたビューがデフォルト・ビューとして使用されます。 特定の言語またはアプリケーションにおいてデフォルト・ビューが使用できない場合には、次のビューが選択されます。

1. デバッガー・エディター内で右クリックします。
2. 「**ビューの切り替え**」 > 「**デフォルト・ビューの設定**」を選択します。
3. いずれかのデバッグ・ビューをデフォルト・ビューとして選択します。
   デバッグ・ビューには、「展開したソース」、「混合」、および「逆アセンブル」が含まれます。

ブレークポイントの使用

ブレークポイントは、指定したポイントでプログラムを停止するようデバッガーに指示するために、実行可能プログラム内に設定される一時マーカーです。 ブレークポイントが検出されると、その行が実行される前に ブレークポイントで実行が中断されます。このポイントで、スレッドのスタックを調べ、 変数、レジスター、およびメモリーの内容を検査できます。 その後、行をステップオーバー (実行) して、それが引数に与える影響を確認することができます。

デバッガーは、以下の種類のブレークポイントをサポートします。

- **行ブレークポイント** : これが設定されている行が実行されようとしている時点で、トリガーされます。
- **エントリー・ブレークポイント** : 適用されるエントリー・ポイントに入ると、トリガーされます。
- **アドレス・ブレークポイント** : 特定のアドレスにある逆アセンブル命令が実行される前に、トリガーされます。
- **ロード・ブレークポイント** : DLL またはオブジェクト・モジュールがロードされると、トリガーされます。
- **条件付きブレークポイント**は、該当するブレークポイントの動作を制御するパラメーターによってトリガーされます。すべてのブレークポイント・タイプが条件をサポートしているわけではありません。
- **イベント・ブレークポイント**は、アプリケーションからスローされた例外をデバッガーが認識した時点でトリガーされます。
- **監視ブレークポイント** は、特定のアドレスにあるデータが実行で変更されると、トリガーされます。
- **オカレンス・ブレークポイント**は、イベントが発生するか、または特定の例外がスローされると、トリガーされます。 ブレークポイントがトリガーされたときに、アクションを実行できます (オプション)。

イベント・ブレークポイントを設定するには、「ブレークポイント」ビューで「**コンパイル型言語のイベント・ブレークポイントの管理**」プッシュボタンをクリックした後、「イベント・ブレークポイントの管理」ダイアログで、デバッガーにキャッチさせるイベントのタイプを選択します。これらのブレークポイントには、すべての標準シグナル、C++ 例外などの関心の高い多数のイベント、およびライブラリー関数 (`exit()` など) が含まれます。 POSIX シグナルについては、個々のシグナルのすべての発生ごとに通知を受けるようにする (取扱シグナル) か、ハンドラーが用意されていない場合のこれらシグナルの発生についてのみ通知を受けるようにする (非取扱シグナル) かを選択できます。

行ブレークポイントは、デバッグの際にエディターで、実行可能な行の左にあるルーラー域をダブルクリックするか、ポップアップ・メニュー・アクション を右クリックすることで設定できます。 または、「ブレークポイント」ビュー でウィザードを使用して設定できます。スレッド固有の行ブレークポイントが必要であれば、アクティブなデバッグ・セッションがある間に、「ブレークポイント」ビューで設定しなければなりません。 エントリー・ブレークポイントを設定するには、「モジュール」ビューでエントリー・ポイントを右クリックし、 ポップアップ・メニューから「**エントリー・ブレークポイントの設定**」を選択できます。 あるいは、「ブレークポイント」ビューでウィザードを使って設定することもできます。 ま

た、「デバッグ」ビューでデバッグ・ターゲット (あるいは、そのスレッドまたはスタック・フレームの 1 つ) を右クリックし、 ポップアップ・メニューから「**オプション**」**>**「**すべての関数エントリーで停止**」を選択すると、 すべてのエントリー・ポイントで停止させることができます (このオプションは「ブレークポイント」ビューのポップアップ・メニューからも選択可能です)。 他のすべてのブレークポイント・タイプは、「ブレークポイント」ビューでウィザードによって設定できます。 ブレークポイントを設定するウィザードにアクセスするには、 「ブレークポイント」ビューで右クリックして、ポップアップ・メニューから「**ブレークポイントの追加**」を選択します。 これによりメニューが展開されて、設定するブレークポイント・タイプを選択できます。ブレークポイントを設定するのに ウィザードを使用する場合、オプションのブレークポイント・パラメーターを指定し、条件付きブレークポイントを設定できます (関連トピックを参照してください)。

すべてのデバッグ・セッションに関するすべてのブレークポイントのリストが「ブレークポイント」ビューに表示されます。 以下のいずれかの方法に従い、表示されるブレークポイントの数を減らすことができます。

- 現在のデバッグ・セッションに関連していないブレークポイントをフィルターで除くには、「ブレークポイント」ビューの「**選択したターゲットでサポートされるブレークポイントを表示**」プッシュボタンをクリックします。
- 「ブレークポイント」ビューを「デバッグ」ビューとリンクさせるには、「**デバッグ・ビューにリンク**」トグルをクリックします。このトグルを選択した場合、デバッグ・セッションがブレークポイントで中断すると、「ブレークポイント」ビューでそのブレークポイントが自動的に選択状態になります。

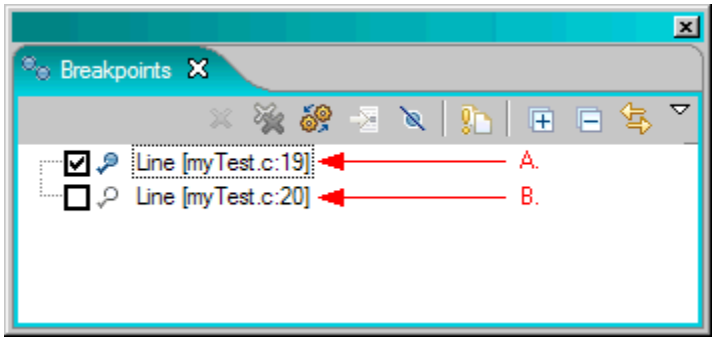
「ブレークポイント」ビューで見やすくするために、ブレークポイントをグループ化 することもできます。ブレークポイントのグループ化は、ブレークポイントのグループ (標準ブレークポイント・リスト)、 ブレークポイント・タイプによるグループ (例えば、行ブレークポイントとエントリー・ブレークポイントを別にグループ化する)、および、 ブレークポイント作業セット別のグループ (ユーザー自身で定義するグループ) が可能です。ブレークポイントをグループ化するには、 「ブレークポイント」ビューの下矢印アイコンを選択し、「ブレークポイント」ビューに表示したいグループを 選択します。メニューで「**詳細設定**」をクリックすると、 ダイアログ・ボックスが開き、グループをネスト構造で作成できます。 作業セットを作成するには、「ブレークポイント」ビューの下矢印アイコンのメニューから「**作業セット**」を 選択します。

リストの中のブレークポイント項目には、ブレークポイントのプロパティーの要約が大括弧で囲まれて表示されます。 ポップアップ・メニュー・オプションを使用すると、ブレークポイントを追加、ブレークポイントを除去、およびブレークポイントを使用可能または使用不可に設定できます。 ポップアップ・メニューには、ブレークポイントのプロパティーを編集 するオプションもあります。「ブレークポイント」ビューのボタンを使用すると、ブレークポイントを除去できます。

ブレークポイントの編集を選択すると、ブレークポイントを作成したウィザードが開きます (ブレークポイントの作成にウィザードを使用しなかった場合は、ブレークポイント・タイプのウィザードが開きます)。このウィザードの間に、「**次へ**」**>** または **<** 「**戻る**」をクリックすると、ウィザードでのブレークポイント設定を表示または編集できます。作業が終わったら、 「**完了**」をクリックしてブレークポイントを変更するか、 「**キャンセル**」をクリックして変更を行わずにウィザードを終了 します。

ブレークポイントは、「ブレークポイント」ビューまたはエディターのポップアップ・メニュー、 および「ブレークポイント」ビューのチェック・ボックスによって使用可能/使用不可にできます。 ブレークポイントを使用可能にすると、ブレークポイントにヒットするたびにすべてのスレッドが中断されます。ブレークポイントを使用不可にすると、スレッドは中断されません。 ブレークポイントを使用可能/使用不可にする方法については、 関連トピックを参照してください。

「ブレークポイント」ビューには、設定ブレークポイントの左方に 2 つの インディケーターがあります ()。 一番左は、ブレークポイントが使用可能かどうかを示すチェック・ボックスです。 使用可能になっている場合、このチェック・ボックスにチェック・マークが付きます。 (下図のポインター A. を参照。) 使用不可になっている場合、このチェック・ボックスにはチェック・マークが付きません。 (下図のポインター B. を参照。)



チェック・マーク・オーバーレイのあるインディケーターは、デバッグ・エンジンによって正常にインストールされたブレークポイントを示しています。 ブレークポイントが使用可能の場合、このインディケーターは塗りつぶされ、ブレークポイントが使用不可の場合、このインディケーターは塗りつぶされません。エディターでは、行ブレークポイントについて、チェック・マーク・オーバーレイのインディケーターが、正常にデバッグ・エンジンによってインストールされた ブレークポイントを示します (ブレークポイントが使用可能に されている場合はこのインディケーターが塗りつぶされ、ブレークポイントが使用不可にされている場合は インディケーターは塗りつぶされません)。

実行を中断するにはブレークポイントがインストールされている必要があります。 現在のデバッグ・セッションでは無効なブレークポイントを追加することができます。 ブレークポイントを認識するデバッグ・エンジンが含まれるデバッグ・セッションの一部となる時点までは、このブレークポイントはインストールされません。

エディターでは、行ブレークポイント、および エントリー・ブレークポイントのインディケーターが、エディターの左方のマーカー・バーに表示されます。「ブレークポイント」ビューでは、行、 エントリー、アドレス、監視、およびロードの各ブレークポイントのインディケーターが表示されます。

「ブレークポイント」ビューでは、下記のいずれか１つを実行することによって、ソース・エディターがブレークポイントのロケーションに対して開かれます。

- ブレークポイントをダブルクリックします。
- ブレークポイントを選択して、 「**ブレークポイント用のファイルへジャンプ**」プッシュボタンをクリックします。
- ブレークポイントを右クリックし、ポップアップ・メニューから 「**ファイルへジャンプ**」を選択します。

ブレークポイントの設定

以下の情報は、行ブレークポイント、エントリー・ブレークポイント、例外ブレークポイント、その他のブレークポイント・タイプを設定する方法を説明しています。

行のブレークポイント

行のブレークポイントを設定するには、以下のいずれかの手順を実行します。

- デバッグ時に、停止する場所となるステートメントを右クリックして「**ブレークポイントの追加**」>「**行**」を選択します。
- デバッガー・エディターで、行の左側の余白領域をダブルクリックします。 ブレークポイントを設定または除去するには、この方法を使用できます。
- 「ブレークポイント」ビューで、空白領域を右クリックして「**ブレークポイントの追加**」を選択します。

エントリー・ブレークポイントの設定

エントリー・ブレークポイントを設定するには、以下のいずれかの手順を実行します。

- 「モジュール」ビューでエントリー・ポイントを右クリックして、「**エントリー・ブレークポイントの設定**」を選択します。
- 「デバッグ」ビューで、デバッグ・ターゲットまたは、そのスレッドあるいはスタック・フレームの１つを右クリックし、「**オプション**」 > 「**すべての関数エントリーで停止**」を選択すると、すべてのエントリー・ポイントで停止します。

- アウトライン・ビューで、エントリー・ポイントの名前を右クリックして「**エントリー・ブレークポイントの切り替え**」を選択します。 アウトライン・ビューからエントリー・ブレークポイントを追加するときには、条件式などの情報を追加することはできません。
- 「ブレークポイント」ビューで、「**すべての関数エントリーで停止**」を選択します。
- 「ブレークポイント」ビューで、「**ブレークポイントの追加**」 > 「**エントリー...**」を選択します。

イベント・ブレークポイントの設定

イベント・ブレークポイントを設定するには、以下の手順を実行します。

1. 「ブレークポイント」ビューで、「**コンパイル型言語のイベント・ブレークポイントの管理**」をクリックします。
2. 「イベント・ブレークポイントの管理」ダイアログで、デバッガーにキャッチさせるイベント・タイプを選択します。

その他のブレークポイント・タイプの設定

ソース・エントリー・ブレークポイントを設定するには、以下のいずれかの手順を実行します。

1. 「ブレークポイント」ビューを右クリックして、メニューから「**ブレークポイントの追加**」を選択します。全メニューに拡張されて、サポートされるブレークポイント・タイプが表示されます。
2. 設定するブレークポイント・タイプを選択します。 このウィザードを使ってブレークポイントを設定するときには、オプションのブレークポイント・パラメーターの指定、および条件付きブレークポイントの設定を行うことができます。

ブレークポイントのエクスポートとインポート

ブレークポイントのエクスポート
ブレークポイントをエクスポートするには、以下の手順を実行します。

1. ブレークポイント・ビューの中を右クリックして、「**ブレークポイントのエクスポート**」を選択します。
2. 「ブレークポイントのエクスポート」ウィンドウで、エクスポートするブレークポイントを選択します。
3. 「**エクスポート先ファイル**」フィールドで、パスとファイル名を入力します。
4. 同じ名前の既存のファイルをデバッガーが上書きするときに警告を出さないようにするには、「**警告を出さずに既存のファイルを上書き**」を選択します。
5. 「**終了**」をクリックすると、ファイルに保存されます。

   このファイルに保存されたブレークポイントをインポートすることができます。また、同じプログラムをデバッグしている他のユーザーにこのファイルを送信すると、それらのユーザーも同じブレークポイントをインポートできます。

ブレークポイントのインポート
ブレークポイントをインポートするには、以下の手順を実行します。

1. エクスポート元と同じプログラムの中にブレークポイントをインポートしようとしていることを確認します。 異なるプログラムの中にブレークポイントをインポートしようと試みた場合、デバッガーがブレークポイントをインストールできない可能性があります。
2. ブレークポイント・ビューの中を右クリックして、「**ブレークポイントのインポート**」を選択します。

   「ブレークポイントのインポート」ウィンドウが開きます。
3. 「ブレークポイントのインポート」ウィンドウの「**インポート元ファイル**」フィールドで、パスとファイル名を指定し、ファイル拡張子として `bkpt` を指定します。 また、「**参照**」を使ってファイルまでナビゲートすることもできます。
4. 既存のブレークポイントをファイル内のブレークポイントで置換するようデバッガーに指示するには、「**既存のブレークポイントの更新**」チェック・ボックスを選択します。
5. これらのブレークポイント用に新しいワーキング・セットを作成するようデバッガーに指示するには、「**ブレークポイント・ワーキング・セットの作成**」チェック・ボックスを選択します。
6. 「**完了**」をクリックします。

ブレークポイントを使用可能/使用不可にする

ブレークポイントを削除するのではなく、 使用不可にするとプログラムの実行が停止されません。ブレークポイントを使用可能にすると、ブレークポイントにヒットするたびにすべてのスレッドが中断されます。ブレークポイントを使用不可にすると、スレッドは中断されません。 アプリケーションの実行中に、ブレークポイントを追加、削除、または使用可能/使用不可にすることができます。

ブレークポイントは、使用不可にしても、「ブレークポイント」ビューに残ります。 使用不可にしたブレークポイントでプログラムを停止させたい場合は、 そのブレークポイントを選択して使用可能にします。ブレークポイントを削除しないで無効にする利点は、 ブレークポイントを再設定するために、そのロケーションをソース内で探索する必要がない点です。さらに、ブレークポイントを使用不可にすると、そのブレークポイント内の追加の設定がすべて保存されます。

設定ブレークポイントの左には 2 つのインディケーターがあります。 一番左は、ブレークポイントが使用可能かどうかを示すチェック・ボックスです。 使用可能にされたブレークポイントは、チェック・ボックス内のチェック・マークで示され、 使用不可にされたブレークポイントは、チェック・ボックス内にチェック・マークがないことで示されます。 ブレークポイントが使用不可にされている場合、エディターまたは「ブレークポイント」ビューの そのブレークポイントのポップアップ・メニューから「**使用可能にする**」を選択 できます (エディターの場合は、メニュー項目は「**ブレークポイントを 使用可能にする**」です)。ブレークポイントが使用可能である場合、 そのポップアップ・メニューから「**使用不可にする**」を選択できます。

ブレークポイント・ビューでブレークポイントを使用可能/使用不可にする

「ブレークポイント」ビューから 1 つのブレークポイントを使用可能または使用不可にするには、以下のようにします。

1. 「ブレークポイント」ビューをクリックして、このビューを前景にします。
2. 使用可能または使用不可にしたいブレークポイントが見つかるまで、ブレークポイントのリストをスクロールします。 使用可能または使用不可にするブレークポイントが複数の場合は、それらを、キーボードの Shift キーまたは Ctrl キーを使用して選択します。
3. 以下のいずれかを実行します。
   - ブレークポイントを使用可能または使用不可にするには、ブレークポイントの一番左のチェック・ボックスを使用します。 ブレークポイントを使用可能にするには、チェック・ボックスを選択状態にします。 ブレークポイントを使用不可にするには、チェック・ボックスをクリアします。
   - 使用可能または使用不可にするブレークポイントを右マウス・ボタン・クリックして、「**使用可能にする**」または「**使用不可にする**」を選択します。

エディターでブレークポイントを使用可能/使用不可にする

エディターから 1 つのブレークポイントを使用可能または使用不可にするには、以下のようにします。

1. エディター内でブレークポイントを見つけます。
2. 以下のいずれかのタスクを実行します。
   - エディター・ルーラー・バー内のブレークポイント・インディケーターを右クリックし、「**ブレークポイントを使用可能にする**」または「**ブレークポイントを使用不可にする**」を選択します。
   - エディター内でブレークポイントを右クリックし、ポップアップ・メニューから「**ブレークポイントを使用可能にする**」 または「**ブレークポイントを使用不可にする**」を選択します。

   エディター・ブレークポイント・インディケーターは、そのブレークポイントが使用不可になっている場合は透明なドットに、有効になっていた場合は塗りつぶされたドットに変わります。

すべてのブレークポイントを使用不可にする

すべてのブレークポイントを使用不可にするには、次のようにします。

1. 「**すべてのブレークポイントをスキップ**」トグル・ボタンをクリックします。

   これにより、すべてのブレークポイントが一時的に使用不可になります。
2. 「**ブレークポイントを使用不可にする**」アクションによって明示的に使用不可にしたものを除く、すべてのブレークポイントを再び使用可能にするには、「**すべてのブレークポイントをスキップ**」トグル・ボタンを再びクリックします。

条件付きブレークポイント

ブレークポイントの動作を制御するため、オプションのブレークポイント・パラメーターが 使用されます。

注 : すべてのブレークポイントが条件をサポートしているわけではありません。

ブレークポイントを設定する際、任意のブレークポイント・ウィザードの「**オプション・パラメーター**」ページで以下のパラメーターを設定することにより、そのブレークポイントを条件付きにすることができます。

オプションのブレークポイント・パラメーター	**説明**	**サポートされるブレークポイントのタイプ**
スレッド	ブレークポイントをスレッド固有にできます。ブレークポイントをすべてのスレッドに適用するか (デフォルト動作)、それとも 1 つの特定のスレッドだけに適用するか (n=one) を指定できます。 すべてのスレッドを指定するには、「**すべて**」を選択してください。個別のスレッドを指定するには、そのスレッドを選択します。	このパラメーターは、すべてのタイプのブレークポイントでサポートされます。

オプションのブレークポイント・パラメーター	説明	サポートされるブレークポイントのタイプ
頻度	いつブレークポイントで停止するのか、いつブレークポイントをスキップするのかを示します。 デバッガーは、各ブレークポイントが検出された回数を カウントします。このセクション内のフィールドは、何回目のブレークポイント検出で デバッガーが初めて停止するのか、デバッガーがどのような頻度で停止するのか、何回目のブレークポイント検出でデバッガーがもう停止しなくなるのかをデバッガーに指示します。 以下のパラメーターが、ブレークポイント頻度を 設定するために使用されます。 • **From（開始）:** 何回目のブレークポイント検出でデバッガーを初めて 停止させるのかを入力します。例えば、最初の 5 回のブレークポイント検出をデバッガーにスキップさせる場合、6 を入力します。 • **To（終了）:** デバッガーを停止させる最後のブレークポイントの出現回数を 入力します。例えば、ブレークポイントを 20 回検出した後に、デバッガーにそれ以降のブレークポイントを無視させるようにするには、20 を入力します。すべての検出で停止するには、「無限大」を入力します。 • **Every（頻度）:** このブレークポイントでデバッガーを 停止させる頻度を入力します。例えば、4 回の検出につき 1 回停止させるには、4 を入力します。	このパラメーターは、すべてのタイプのブレークポイントでサポートされます。

オプションのブレークポイント・パラメーター	説明	サポートされるブレークポイントのタイプ
式	このフィールドに式を入力することができます。このフィールドに指定した条件が真の時にのみ (ゼロ以外の値は真と見なされる)、プログラムの実行が ブレークポイントで停止します。 例えば、C++ プログラムをデバッグする場合、次の式を入力できます。 `(i==1) || (j==k) && (k!=5)` 条件式 として使用できるのは、ブレークポイントの場所の言語での有効な式であって、評価結果が数値になり、副次作用がなく、 関数を呼び出したりしない任意の式です。C および C++ の場合、すべての代入演算子、および増分演算子と減分演算子 (++ と --) は許可されません。 **重要 :** 条件を満足していないブレークポイントでアプリケーションが停止しているようには見えなくても、 デバッガーは、条件を評価している間はアプリケーションを一時的に中断します。ほとんどの 目的に対して、この短い一時停止は重要ではありません。ただしマルチスレッド・アプリケーションでは、一時停止が原因で、スレッドのディスパッチ順序がオペレーティング・システムによって変更される可能性があります。	行、エントリー、アドレス。

変数のインスペクション

変数を検査する方法は 2 つあり、デバッグ・エディター・ウィンドウ内で変数の名前の上にマウスを移動するか、「変数」ビューを使用することができます。マウスを移動した場合、デバッガーは小さなウィンドウのツリー構造の中に変数の値を表示し、変数名からマウスを離すとその表示が消えます。 デバッガーがこのウィンドウを表示している間、ツリー構造を展開または省略表示して詳しい情報を確認できます。 変数の値を吹き出しウィンドウから編集することもできます。 デバッガーの「変数」ビューを使用すると、プログラム内の変数に簡単にアクセスして、変数を確認および編集できます。

あるスレッドが中断すると、スレッドの一番上のスタック・フレームが自動的に選択されます。 スタック・フレームが選択されると、 そのスタック・フレームの可視変数が「変数」ビューに表示されます。 複雑な変数を展開して、その変数を構成するエレメントを表示することができます。

- 吹き出しウィンドウで変数値を変更するには、以下の手順を実行します。

  a) 吹き出しウィンドウ内のツリー構造から、変数またはフィールドを選択します。

選択した要素の値が、ツリー構造の下にある詳細ペインに表示されます。

b) 詳細ペイン内をクリックして、変数値を編集します。

詳細ペインで値を編集するとき、「**切り取り**」、「**コピー**」、「**貼り付け**」、または「**すべてを選択**」アクションを使用できます。

c) 変数値を編集したら、詳細ペインの下にある「**値の代入**」ボタンをクリックして、新しい値を変数に割り当てます。

- 「変数」ビューで変数値を変更するには、「**値**」列にある変数の値をクリックしてからインラインで値を変更するか、または、以下のステップを実行します。

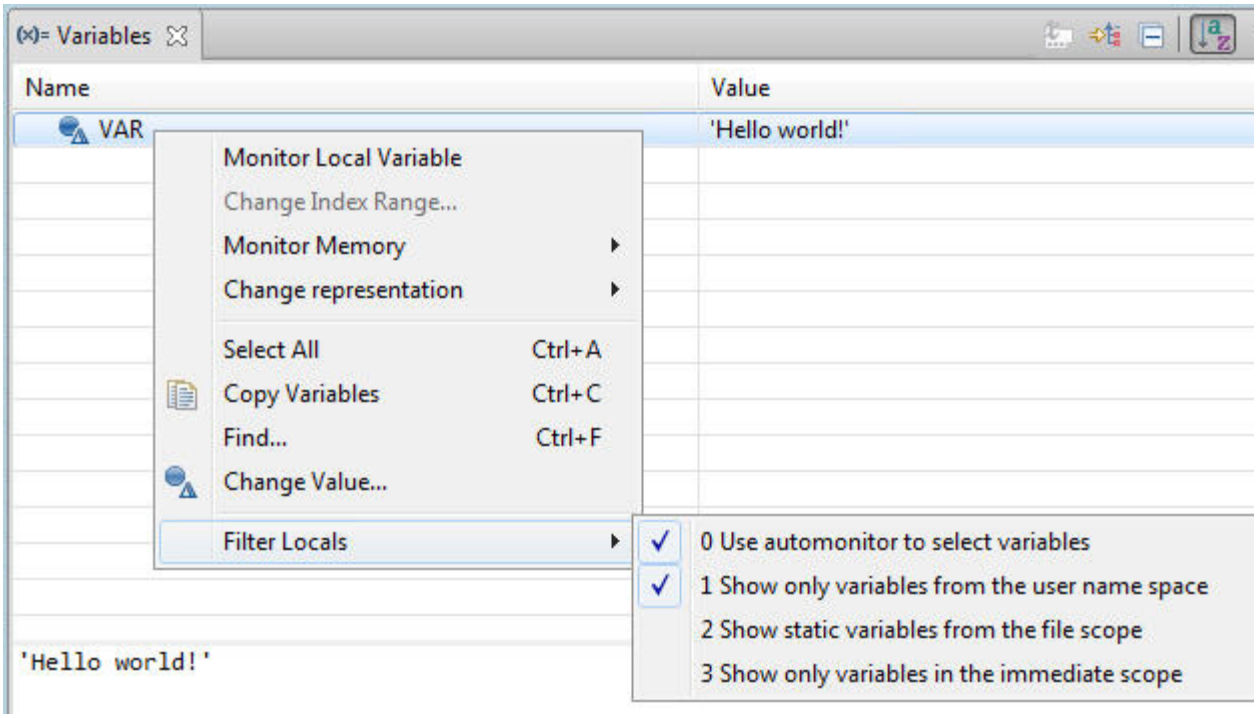
  a) 編集する変数を右クリックし、ポップアップ・メニューから 「**値の変更**」を選択します。

  b) 表示されたダイアログで、変数値を変更します。

変数値が変更されたことを示すため、 そのインディケーターの横にデルタ・シンボルが表示されます。変更の影響を受けた他のすべての変数にも、そのインディケーターの横にデルタ・シンボルが表示されます。

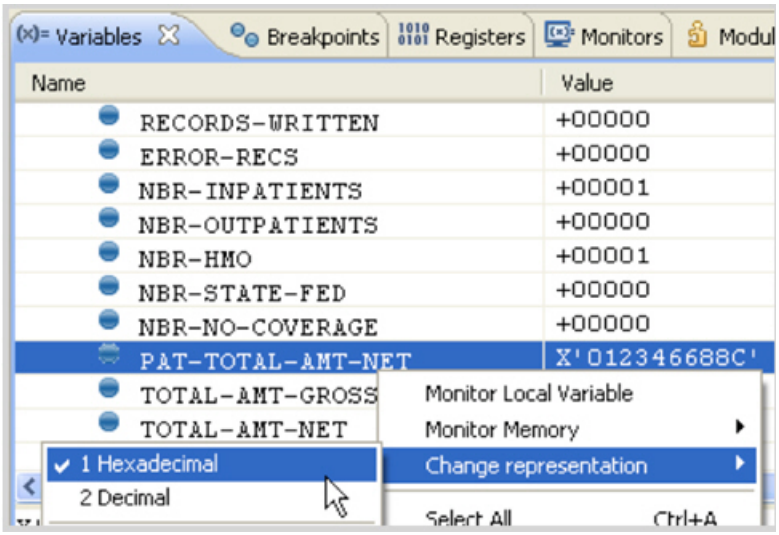
「変数」ビューは、選択されたスタック・フレームのすべての変数を 表示します。ビューは現在のスコープ内の変数を動的に表示します。プログラムがステップスルーまたは再開されると、変数が表示されたり、表示されなくなったりします。「変数」ビューの代わりに、「モニター」ビューで変数をモニターすることもできます。 「モニター」ビューでは、変数の値が取得可能な場合、それが常にデバッガーによって表示されます。 一度に 1 つ以上の変数を表示および検査するには、変数 (複数可) を右クリックして、ポップアップ・メニューから 「**ローカル変数のモニター**」を選択し、「モニター」ビューで変数の操作を行います。

デバッグする言語に基づいて、 一定の変数のみが表示されるように「変数」ビューをフィルター操作できます。これを行うには、「変数」ビューで右クリックして、 次の画像に示されているように「**ローカルのフィルター**」サブメニューから項目を選択します。



注 :「**ローカルのフィルター**」サブメニューで使用できるフィルター・オプションと内容は、デバッグしている言語によって異なります。

変数ビューから、値を 10 進数または 16 進数 形式で表記するよう設定できます。表記を選択するには、変数を強調表示し、右クリックして「**表記の変更**」を選択した後、適切な形式を次のように選択します。



データ例外をデバッグするときには、16 進表示で値を表示すると役立つ場合があります。

「モニター」ビューへの、変数、式、またはレジスターの追加

「モニター」ビューには、モニターする対象に選択した、変数、式、およびレジスターが 表示されます。変数または式は、ダイアログ・ボックスで入力するか、デバッガー・エディターで選択することができます。「モニター」ビューを使用して、 デバッグ・セッション中に常に確認したい、グローバル変数 (または、変数、式、およびレジスター) をモニター できます。「モニター」ビューからは、変数、式、またはレジスターの内容を変更したり、値の表記を変更することもできます。注: 一部のプログラミング言語の式サポートは、ご使用のサーバーにインストールされているその言語のコンパイラーまたはランタイムのバージョン、あるいはその両方に応じて異なります。

式に関する新しいプログラム・モニターを「モニター」ビューから追加するには、次のようにします。

1. エディター内で、式を評価したいコンテキストを表しているソース行を 選択します。または、「デバッグ」ビューで、モニター対象の式が含まれるスレッドを選択します。
2. 「モニター」ビューの「**式のモニター**」ボタン () をクリックします。
3. 「式のモニター」ダイアログ・ボックスで、変数、式、またはレジスターをフィールドに 入力します。
4. 「**OK**」をクリックします。

追加アクション

以下の追加情報は、エディター、変数ビュー、レジスター・ビューからプログラム・モニターを追加する方法について、およびモニター・ビューで変数、式、またはレジスターの内容を変更する方法について説明しています。

変数または式に対する新規プログラム・モニターをエディターから追加するには、次のようにします。

1. エディターで、モニターしようとする式を強調表示し、右クリックします。
2. ポップアップ・メニューから「**式のモニター**」を選択します。

変数または式に対する新規プログラム・モニターを「変数」ビューから追加するには、次のようにします。

1. 「変数」ビューで、モニターしようとする変数を右クリックします。複数のモニターを追加するには、キーボードの Ctrl または Shift キーを使って複数の変数を選択します。
2. ポップアップ・メニューから「**ローカル変数のモニター**」を選択します。

各行をステップスルーするときに、各行の変数を「モニター」ビューに自動的に追加するには、次のようにします。

1. モニターを開始する最初の行で、プログラムを停止します。
2. 「デバッグ・コンソール」ビューで SET AUTOMONITOR ON コマンドを入力します。

レジスターに対する新規プログラム・モニターを「レジスター」ビューから追加するには、次のようにします。

1. 「レジスター」ビューで、モニターしようとするレジスターを右クリックします。
2. ポップアップ・メニューから「**レジスターのモニター**」を選択します。

変数、式、またはレジスターの内容を「モニター」ビューで変更するには、次のようにします。

1. 値を変更する式を選択します。
2. 式が構造体または配列である場合は、展開して、個々の要素を表示します。
3. 変更する式までスクロールダウンし、次のいずれかを行います。
   - 式をダブルクリックします。
   - 式を右クリックし、ポップアップ・メニューから「**値の変更**」を選択します。

   注 : 変数をダブルクリックしてもその値フィールドを編集できない場合、 その変数は変更不能なタイプです。
4. 式の新規の値を入力して、**Enter** を押します。 副次作用のない有効な任意の式を、新しい値にできます。式の値が変更されたことを示すため、 そのインディケーターの横にデルタ・シンボルが表示されます。変更の影響を受ける他のすべての式にも、そのインディケーターの横にデルタ・シンボルが表示されます。

上記の制約事項のいずれかが 満たされていない場合、デバッガーはリカバリーを試みます。ただし、デバッグされるアプリケーションの状態が取り返しのつかないほど 変更されることがないという保証はありません。

最適化された COBOL プログラムで変数をモニターする場合、その変数の値を変更するステートメントを実行するたびに、次のエラー・メッセージが表示される可能性があります。エラーが発生しました: EQA2421E 最適化のために代入値がプログラムで使用されない可能性があるため、この代入は実行されませんでした。(`Error occurred: EQA2421E The assignment was not performed because the assigned value might not be used by the program, due to optimization.`) デバッガーはステートメントを実行しません。 このステートメントを実行するには、以下のステップを実行します。

1. 前述のいずれかの方法で、変数を「モニター」ビューに追加します。 デバッガーは、変数の名前と現行値を「モニター」ビューに表示します。
2. その変数の値を変更するステートメントに達するまで、プログラムをステップスルーします。
3. 「デバッグ・コンソール」ビューに **SET WARNING OFF** コマンドを入力します。 「デバッグ・コンソール」ビューに、**SET WARNING OFF** コマンドを受け取ったというメッセージが表示されます。
4. ステートメントをステップスルーします。モニターしている変数の新しい値が、「モニター」ウィンドウに表示されます。

モニター内容の表記の設定

「**モニター**」ビューおよび「**変数**」ビューでの 変数と式の表記を変更することができ、変数または式の現行の表記をデフォルトに 設定できます。

1. 表記を変更しようとする変数または式を右クリックします。
2. ポップアップ・メニューから「**表記の変更**」を選択します。
3. 設定したい表記を選択します。現行の表記の横に チェック・マークが付きます。
4. 現行の表記をデフォルトに設定するには、次のようにします。
   a. 変数または式に希望通りの表記を上記ステップに従って設定済みであることを確認します。
   b. 変数または式を右クリックし、ポップアップ・メニューから「**表記の変更**」**>**「**デフォルト表記の設定**」を 選択します。

   変数または式に対して、このデフォルト表記がアプリケーションの後続のデバッグ・セッションで使用されます。

変数と式の間接参照

評価結果がアドレスになる変数または式は、その変数または式をモニターしている場合、「**変数**」ビューまたは「**モニター**」ビューで 間接参照することができます。間接参照を行うには、以下のステップを実行します。

1. 「変数」ビューまたは「モニター」ビューで、間接参照できる変数または式 (例えば、ポインター) を右クリックします。
2. ポップアップ・メニューから「**間接参照**」を選択します。

レジスターの内容のビュー

レジスターの内容は、「レジスター」ビュー、「モニター」ビュー、または「メモリー」ビューから表示できます。これらのビューで、 プログラムの現行スレッドのレジスターの内容を確認し、変更することができます。「レジスター」ビューでは、レジスターはカテゴリー化されているので、必要なのは表示したいレジスターのカテゴリーを展開するだけです。

「レジスター」ビューでレジスターの内容を表示する

1. 「デバッグ」ビューで、レジスターを表示する対象となるスレッド (またはスレッドのスタック・フレーム) を選択します。
2. 「レジスター」ビューで、表示するレジスター・グループを展開します。
3. 必要に応じて、スクロール・バーまたは PageUp および PageDown キーを使用して、そのレジスターが表示されるまで「レジスター」ビューをスクロールします。

   注 : 「レジスター」ビューを使用している場合、値をコピーするには、コピー操作の前に編集モードにしておく必要があります。 「レジスター」ビューで値を編集するには、それをダブルクリックするか、右クリックして、メニューから「**値の変更**」を選択します。どちらのアクションでも「値の設定」ダイアログ・ボックスが開き、そこからレジスターの値をコピーできます。

   ヒント : パフォーマンスの向上のため、 使用または編集していないレジスター・グループは、縮小表示にしてください。

   「レジスター」ビューでレジスター値を変更するには、以下の手順を実行します。

   a. 編集するレジスターを右クリックし、ポップアップ・メニューから 「**値の変更**」を選択します。
   b. 表示されたダイアログで、変数値を変更します。
   c. 「**OK**」をクリックします。 レジスター値が変更されたことを示すため、 そのインディケーターの横にデルタ・シンボルが表示されます。

   重要 : ダイアログを完了するには、キーボードの Enter キーではなく、「**OK**」をクリックする必要があります。 キーボードの Enter キーを押すと、レジスター値に改行が挿入されます。

「モニター」ビューに既に追加したレジスターの内容を表示する

1. 必要に応じて、スクロール・バーまたは PageUp および PageDown キーを使用して、そのレジスターが表示されるまで「モニター」ビューをスクロールします。
2. レジスターの表記を変更する場合は、レジスター名を右クリックし、ポップアップ・メニューから「**表記の変更**」を選択します。 次に、その結果表示されたポップアップ選択肢から、必要な表記を選択します。

「メモリー」ビューで、メモリー・モニターに既に追加したレジスターの内容を表示する

必要に応じて、スクロール・バーまたは PageUp および PageDown キーを使用して、 レジスターのアドレスが表示されるまでメモリー・モニターをスクロールします。

「モジュール」ビューの使用

プログラムの実行中にロードされるモジュールのリストを表示するには、モジュール・ビューを使用します。

- デバッグ情報を持つモジュールを表示。

「モジュール」ビューでは、リスト内のアイテムを展開してコンパイル・ユニット、ファイル、および関数を表示できます。 モジュールを表示しているときに「**デバッグ情報を持つモジュールを表示**」ボタンをクリックすると、デバッグ情報のないモジュールがフィルターによって除かれ、デバッグ情報を持つモジュールだけが残ります。 デフォルトでは、この設定はオンです。

- ロードされたモジュールに関連付けられているソース・ファイルをリストします。

  モジュールをインスペクションしているときに、ソース・ファイル・ノードをダブルクリックすると、エディターでソース・ファイルが開きます。 「**ファイル・フィルター・ダイアログの表示**」ボタンをクリックすると、ダイアログ・ボックスが開き、ロードされたモジュールに関連付けられているすべてのソース・ファイルがリストされます。 テキスト・フィールドにファイル名を入力することで、選択リストを絞り込むことができます。「**OK**」をクリックすると、ソース・ファイルがエディターで開きます。
- 次のようにして、コンパイル・ユニット、ファイル、および関数のプロパティーを「プロパティー」ビューで表示します。

  a) 「プロパティー」ビューを開くには、「**ウィンドウ**」 > 「**ビューの表示**」 > 「**プロパティー**」を選択します。

  b) 「モジュール」ビューで、プロパティーを表示するモジュールに進みます。 必要に応じて、モジュール・ノードを展開し、スクロール・バー、上下キー、 または PageUp および PageDown キーを使用して、そのモジュールが表示されるまで 「モジュール」ビューをスクロールします。

  c) そのプロパティーを「プロパティー」ビューに表示するモジュールを選択します。
- モジュール・ビューからエントリー・ブレークポイントを設定します。

  エントリー・ポイントを右クリックして、メニューから「**エントリー・ブレークポイントの設定**」を選択します。

メモリーのモニター

メモリー・ビューを使用すると、プログラムによって使われるメモリーまたはメモリー域の内容を変更できます。

「変数」ビュー、「モニター」ビュー、「レジスター」ビュー、またはエディターから新しいメモリー・モニターを追加する

1. 「変数」ビュー、「モニター」ビュー、または「レジスター」ビューで、 メモリーをモニターしようとする対象の、変数、式、またはレジスターを右クリックします。あるいは、 エディターで、メモリーをモニターしようとする対象の式を強調表示し、 右クリックします。

   注: 式がポインターである場合、その式の値が、 メモリーをアドレッシングするのに使用されます。式が `lvalue` (メモリー内の アドレスを伴う) である場合、その式のアドレスが、メモリーをアドレッシングするのに使用されます。それ以外の場合、 式の値がアドレスとして使用されます。例えば、 `int i = 0x44;` という宣言があると想定します。式が `i` である場合、 メモリー・モニターはアドレス `i` に置かれ、式が `i+1` である場合、 メモリー・モニターは、式 `i+1` の値で取得されるロケーション、つまり `0x45` に置かれます。
2. ポップアップ・メニューから「**メモリーのモニター**」>「***<rendering>***」を選択します。ここで、*<rendering>* は、「メモリー」ビューの「**レンダリング**」部分に表示したいレンダリングです。

「メモリー」ビューで式に関する新しいメモリー・モニターを追加する

1. 「**メモリー・モニターの追加**」 をクリックします。
2. 「メモリーのモニター」ダイアログ・ボックスで、 フィールドに式を入力します (この式の評価結果がアドレスにならなければなりません)。
3. 「**OK**」をクリックします。

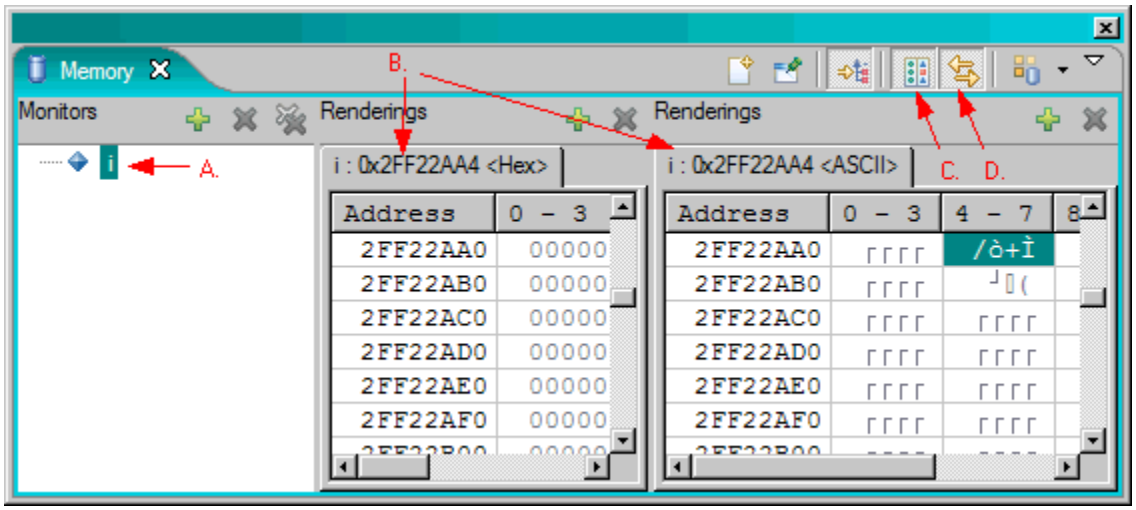
   「メモリー」ビューの「**モニター**」 (左方) 部分に、 モニターのために入力した式が表示されます。メモリー・モニターが複数ある場合、このセクションに、モニターしている式のリストが表示されます。

   「メモリー」ビューの「**レンダリング**」(右方) 部分に、 16 進法と ASCII のレンダリングが取り込まれます。

「メモリー」ビューでのメモリーのインスペクション

「メモリー」ビューで、特定のアドレスにあるメモリーの内容を 確認できます。アドレスは、変数またはレジスターを指す式から 取得できます。メモリーをモニターするとき、モニターが、16 進、ASCII、EBCDIC、UTF-8、 符号付き整数、符号なし整数などのデータ・フォーマットでレンダリングされるように設定できます。

「メモリー」ビューは、次の図のように表されます。



- ビューの左側には、「**モニター**」ペインが 配置されています (図のポインター A.)。ビューのこの部分には、 モニター対象として追加した、式、変数、およびレジスターのリストが表示されます。 この文書では、モニター とは、「**モニター**」ペインに リストされるメモリー・モニターのことです。
- ビューの右側 (このビューが水平方向に設定されている場合) には、「**レンダリング**」ペインが あり、選択されているモニターのレンダリングが表示されます (図のポインター B.)。このペインを使用して、 モニターされるメモリーが表示されるデータ・フォーマット (複数可) を設定できます。
- 「**分割ペインを切り替え**」コントロール (図のポインター C.) を使用して、「**レンダリング**」ペインを分割できます。デフォルトでは、 「メモリー」ビューに表示される「レンダリング」ペインは 1 つのみです。「**分割ペインを切り替え**」を クリックすると、2 つ目の「レンダリング」が開き、分割ペインとして表示されます。
- 「**メモリー・レンダリング・ペインをリンク**」コントロール (図の ポインター D.) を使用して、分割した「**レンダリング**」ペインを、 1 つのレンダリングでナビゲートするとき、または、1 つのレンダリングのフォーマットを変更するときに、 同期させることができます。

「**モニター**」ペインからアドレスまたは式をモニターすると、 「**レンダリング**」ペインには、使用しているオペレーティング・システムのデフォルトの テキスト・ベース・レンダリングが取り込まれます。「**レンダリング**」ペインからアドレスまたは式をモニターすると、 このペインには、レンダリングのリストが取り込まれ、そのうちの 1 つを表示用に選択できます。

注 : メモリーを UTF-8 形式で表示する場合は、 「**従来型**」または「**16 進法と文字**」レンダリングを使用します。 UTF-8 エンコードで文字を表示するには、レンダリングを右クリックして、 メニューから「**テキスト**」 > 「**UTF-8**」を選択します。

「メモリー」ビューで、 複数の変数、式、およびレジスターをモニターすることができます。 また、「**レンダリング**」ペインに複数のレンダリングを追加することができます。 「**モニター**」ペインには、 追加した各変数、式、またはレジスターがリストされます。「**レンダリング**」ペインには、 「**メモリー**」ビューで現在選択されているモニターに対する 1 つ以上のメモリー・レンダリングのみが表示されます (複数のレンダリングは、タブまたは分割ペインで分離されます)。

「メモリー」ビューで 水平方向 (横並び) または垂直方向 (上下) に表示されるように、これら 2 つの ペインを設定できます。ビューのレイアウトを設定するには、 「メモリー」ビューの下矢印アイコンをクリックし、メニューから「**レイアウト**」を 選択します。そうすると、サブメニューが開き、表示したい方向を選択できます。

メモリー・モニターを使用してメモリーの内容を表示する

「**メモリー**」ビューからメモリーの内容を表示するには、次のようにします。

1. 「**モニター**」ペインで、表示しようとするメモリー・ロケーションが含まれている メモリー・モニターを選択します。メモリーは「**レンダリング**」ペインに表示され、 以降の操作はすべてこのペインで実行します。複数のレンダリングを追加した場合は、表示したいレンダリングが含まれているタブを選択してください。
2. 必要であれば、 「**分割ペインを切り替え**」ボタン () を選択して、「**レンダリング**」ペインを分割します。 デフォルトでは、 「メモリー」ビューに表示される「レンダリング」ペインは 1 つのみです。「**分割ペインを切り替え**」を クリックすると、2 つ目の「レンダリング」が開き、分割ペインとして表示されます。「**16 進法と文字**」のレンダリングを選択した場合は、 このボタンを選択すると両方のレンダリングを見ることができます。
3. 必要な場合、レンダリングのスクロール・バーを使用すると、 現在のレンダリングで表示されているメモリー・モニターの基底アドレスの上下のメモリー・ロケーション を表示できます。あるいは、レンダリングの中で右クリックし、「**アドレスへジャンプ**」ポップアップ・メニュー項目を選択するか、Ctrl+G を 押します。そうすると、レンダリングの一番下にセクションが開き、そこで以下のアクションを 実行できます。

   a) 「**アドレスへジャンプ**」プルダウン・メニュー項目を 選択し、ジャンプ先のアドレスを入力します。レンダリングの表示が、入力されたアドレスが表示されるように変わり、 そのアドレスが選択された状態になります。

   b) 「**オフセットへジャンプ**」プルダウン・メニュー項目を 選択し、オフセットを入力します。レンダリングの表示が、入力されたオフセットを式 (基底アドレス) に 加算したアドレスが表示されるように変わり、 そのアドレスが選択された状態になります。負の値になった場合は、レンダリングは基底アドレスから戻って 表示されます。

   c) 「**ジャンプ・メモリー・ユニット**」プルダウン・メニュー項目を 選択します。この機能は、現在選択されているアドレスに、指定した数のメモリー・ユニットを加算 します。結果のアドレスが、選択された状態になります。 負の値になった場合は、レンダリングは現在アドレスから戻って 表示されます。

   これらのすべての入力で、16 進数での入力を行うことができます。そのためには、 「**16 進数で入力**」チェック・ボックスを選択します (このチェック・ボックスが選択されていない場合、 入力は 10 進数になります)。フィールドへの入力が終わったら、Enter を押すか、「**OK**」を クリックして、レンダリング内で指定のロケーションにジャンプします。このセクションを閉じるには、 「**キャンセル**」をクリックするか、Ctrl+G を押します。

   注 : 入力は、接頭部 `0x` が付いている場合も 16 進数として扱われます。
4. 特定のセルに入っているアドレスにジャンプするには、そのセルの中で右クリックし、 ポップアップ・メニューから「**間接参照ポインター**」を 選択します。
5. 必要であれば、列のヘッダー・セルの左端または右端をクリックしてからドラッグすることによって、列幅を 変更できます。または、レンダリングの内側で右クリックし、ポップアップ・メニューから「**フィットするようサイズ変更**」を選択すると、 列に入っているテキストがすべて表示されるように、すべての列のサイズが変わります。あるいは、 レンダリングの内側で右クリックし、ポップアップ・メニューから「**フォーマット**」を 選択します。そうすると、「フォーマット」ダイアログ・ボックスが開きます。このダイアログ・ボックスで、行あたりのユニット数、および列あたりのユニット数を設定できます。これらの設定を 行うときには、ダイアログ・ボックス内の「**プレビュー**」ウィンドウに、 設定しようとしているレンダリング・レイアウトが表示されます。これらの設定を デフォルト・レイアウトとして保管するには、「**デフォルトとしての保管**」をクリックします。
6. メモリー・レンダリングをオフセット・モード に切り替えるには、 レンダリングの内側で右クリックし、ポップアップ・メニューから「**表示モードの変更**」>「**オフセット・モード**」を 選択します。メモリー・レンダリングをアドレス・モード に切り替えるには、 レンダリングの内側で右クリックし、ポップアップ・メニューから「**表示モードの変更**」>「**アドレス・モード**」を 選択します。オフセット・モードに切り替えると、モニターされている式のアドレスが、 レンダリング内で最初のセルになり、「**アドレス**」列にはオフセットが 表示されるようになります。
7. 次のようにして、「メモリー」ビューの要素を非表示にし、表示を見やすくすることもできます。

- 「**メモリー・モニター・ペインを切り替え**」を選択解除することによって、 「**モニター**」ペインを非表示にできます。
- 「**アドレス**」列を非表示にするには、 レンダリングの内側で右クリックし、「**アドレス列の非表示**」を選択します。 非表示になっているアドレス列を復元するには、レンダリングの内側で右クリックし、ポップアップ・メニューから「**アドレス列の表示**」を選択します。

メモリー・レンダリング内にカーソルがあるが、モニターするよう最初に設定したアドレスから離れてしまった場合、 「**基底アドレスにリセット**」ポップアップ・メニュー項目を選択すると、 メモリー・モニターの基底アドレスにカーソルを戻すことができます。あるいは、メモリー・モニターの すべてのレンダリングをリセットすることができます。そうするには、モニターを右クリックし、「**リセット**」を選択します (または、 複数のモニターを選択して、このアクションを選択できます)。モニターを リセットすると、デフォルトで、可視のレンダリングが基底アドレスにリセットされます。 現在の「メモリー」ビュー内のすべてのレンダリングを基底アドレスにリセットするには、 「メモリー」ビュー設定を変更してください。

メモリー・ロケーションの内容の変更

デバッグ中に、メモリー・ロケーションの内容を変更することができます。

「モニター」ビューでメモリー・モニター内のメモリー・ロケーションの内容を変更するには、次のようにします。

1. 「**モニター**」ペインで、編集しようとするメモリー・ロケーションが含まれている メモリー・モニターを選択します。メモリーは「**レンダリング**」ペインに表示され、 以降の操作はすべてこのペインで実行します。複数のレンダリングを追加してある場合は、編集するレンダリングが入っているタブを選択します。
2. 変更しようとするメモリー・ロケーションまでスクロールダウンします。あるいは、モニター内で右クリックし、 「**アドレスへジャンプ**」ポップアップ・メニュー項目を選択します。 そうすると、レンダリングの一番下に「**アドレスへジャンプ**」セクションが 開き、そこでジャンプ先のアドレスを入力できます。
3. 変更しようとする値が入っている行を選択し、 その値をダブルクリックします。

   ヒント : レンダリングにフォーカスがある場合は、 変更しようとする値をダブルクリックしなくても、その値を編集できます。単に、変更内容の 入力を始めることができ、そうするとエディターがアクティブになります。
4. そのメモリー・ロケーションの有効な値を入力します。
5. **Enter** を押して、変更を実行依頼します。 デバッガーが、有効な値をチェックします。

「メモリー」ビュー設定

テーブル・レンダリング、コード・ページ、および埋め込みストリングの設定値を メモリー・レンダリングに対して設定できます。さらに、メモリー・レンダリングをリセットするための優先動作を変更することができます。

「メモリー」ビューの設定ダイアログ・ボックスは、「メモリー」ビューの 下矢印アイコンのメニューから開きます。「メモリー」ビューの「設定」ダイアログ・ボックスを開くには、 「メモリー」ビューの下矢印アイコンをクリックし、メニューから「**設定**」を 選択します。「メモリー」ビューの「テーブル・レンダリング設定」ダイアログ・ボックスを開くには、 「メモリー」ビューの下矢印アイコンをクリックし、メニューから「**テーブル・レンダリング設定**」を 選択します。

設定に対して行った変更をデフォルト設定に復元するには、 「**デフォルトを復元**」をクリックします。

設定*:* メモリー・モニターのリセット

基底アドレスから離れてしまった場合、レンダリングを基底アドレスに リセットすることができます。レンダリングを基底アドレスにリセットするとき、可視のレンダリングのみをリセットするよう設定するか、すべてのレンダリングをリセットするよう設定できます。 すべてのレンダリングをリセットすることを選択すると、リセット操作の パフォーマンスに悪い影響を与える可能性があります。この設定を指定するには、「設定」ダイアログ・ボックスを開き、「**メモリー・モニターのリセット**」ノードを選択します。「メモリー・モニターのリセット」ページで、適当なラジオ・ボタンを選択します。

設定: 埋め込みストリング

埋め込みストリングとは、メモリーをリトリーブできないときに、メモリー内容に 表示される文字列です。埋め込みストリングを指定するには、「設定」ダイアログ・ボックスを開き、「**埋め込みストリング**」ノードを選択します。「埋め込みストリング」ページで、メモリー内容が判別できないときに表示する文字列を 指定します。

設定: コード・ページの選択

ASCII および EBCDIC テキスト・ベースのレンダリング (および、このデバッガーを一緒にインストールした製品で使用可能であれば、 マップされたメモリー) を「**レンダリング**」ペインでモニターしている場合、レンダリングの 表示に使用されるコード・ページを指定できます。

メモリーのレンダリングに対するコード・ページを ASCII/EBCDIC に 設定するには、「設定」ダイアログ・ボックスを開き、「**コード・ページの選択**」ノードを選択します。「コード・ページの選択」ページで、 文字セットのコード・ページをどう変更するのかを指定します (ASCII レンダリング、EBCDIC レンダリング、または両方について)。

テーブル・レンダリング設定

テーブルに表示されるメモリー・レンダリングの設定を指定するには、 「メモリー」ビューの下矢印アイコンをクリックし、「**テーブル・レンダリング設定**」を 選択します。「設定」ダイアログ・ボックスが開きます。このダイアログ・ボックスで、 バッファーの終わりまでスクロールするとデバッガーが自動的にメモリーの次のページをロードする ようにするかどうかを指定できます。この設定が選択されていない場合、ページ当たりの行数に指定した行数だけが、「**レンダリング**」ペインに ロードされます。このページ・サイズ設定で定義されたバッファーの外にスクロール することはできません。その代わりに、次のページまたは前のページからメモリーを 表示するには、右クリックし、ポップアップ・メニューから「**前ページ**」および 「**次ページ**」アクションを使用しなければなりません。

複数の「メモリー」ビューの操作

追加の「メモリー」ビューをワークベンチに追加することができます。これを行うには、「**新規メモリー・ビュー**」( ) をクリックします。 複数の「メモリー」ビューが 開いている場合、それらのレンダリングを互いにリンクすることはできません。ただし、 1 つの「メモリー」ビューの内容を固定して、一方のビューに追加されたメモリー・レンダリングが他方のビューに 影響しないようにできます。1 つのメモリー・モニターを固定するには、「メモリー」ビュー内の「**メモリー・モニターをピン留め**」ボタン ( ) が オンに切り替えられているようにします。そうすると、別の「メモリー」ビューに移動してメモリー・モニターを追加した場合に、そのメモリー・モニターが両方の「メモリー」ビューに表示されますが、固定されたモニターに現在表示されているメモリー・レンダリングは変化しなくなります。

新しい「メモリー」ビューを 追加すると、その「**レンダリング**」ペインには、 メモリー・レンダリング選択リストが取り込まれます。このリストから、 メモリー・レンダリングで使用するデータ・フォーマットを選択して、 「**レンダリングの追加**」をクリックします。

「メモリー」ビューからのメモリー・モニターの除去

「**メモリー**」ビューからメモリー・モニターを除去するには、次のようにします。

1. 除去するメモリー・モニターを選択します (「**モニター**」ペインのリストから選択します)。
2. 「**メモリー・モニターの除去**」ボタン ( ) をクリックします。

複数のメモリー・モニターを除去するには、キーボードの Ctrl と Shift キーを使用してそれらを 選択し、「**メモリー・モニターの除去**」をクリックします。 すべてのメモリー・モニターを除去するには、「**すべて除去**」をクリックします。

注 : 1 つのメモリー・モニターに 対して複数のレンダリングが追加されている場合、そのモニターの除去を選択すると、すべてのレンダリングが除去されます。

メモリーのマッピング

「メモリー」ビューでは、自ら定義したレイアウトまたはサンプル・レイアウトに 従ってマップされたメモリーの内容を表示できます。

実行している製品に基づいて、サンプル・レイアウトまたは文書タイプ定義 (DTD) ファイル、あるいはその両方が、 以下の場所にあります。

- <product installation directory>¥plugins¥com.ibm.debug.memorymap.<platform>.samples¥samples (<product installation directory> は、この製品をインストールしたディレクトリーです)。
- <product installation directory>¥maps。

事前定義されたメモリー・レイアウトは XML ファイルに保管されます (テキスト ・エディターまたは XML エディターを使って作成された、レイアウトごとに 1 つの XML ファイル)。 XML ファイル・フォーマットでは、事前定義した基本タイプ・エレメントの構造を記述できることに加えて、ネストされたレイアウト (レイアウト・エレメントが別のメモリー・レイアウト・ファイルを指すことができる) も記述できます。 レイアウト・ファイルは、レイアウトされるメモリー・ブロックの長さも指定します。

モニターするメモリー・ブロックのサイズは、 選択したレイアウトのサイズによって決まります。 指定したメモリー・ブロックが保護されているか、それにアクセスできない場合には、ユーザー設定で埋め込みストリングとして設定されているストリング (デフォルトは複数の「?」) として値が表示されます。

当初は、ネストされたレイアウトを表すレイアウト・エレメントおよび サブエレメントは取り込まれません (まだ、サブエレメントは生成されていません)。 レイアウト・エレメントは、 最初に展開するときに XML レイアウト・ファイルに応じて取り込まれます。 レイアウト・エレメントを取り込むということは、XML に指定されているレイアウト・エレメントに応じて メモリー・ブロックをフラグメントに分割することを意味します。 レイアウト・サブエレメントに関して表示される値は、XML ファイルに指定されているデフォルトの 基本タイプに応じてフォーマットされます。

マップされたメモリーの操作

「メモリー」ビューを使用して、メモリー・マップによって、式、変数、およびレジスター のメモリーをモニターすることができます。

「メモリー」ビュー に追加したモニターでマップされたメモリーを表示するには、次のようにします。

1. 「メモリー」ビューの「**レンダリング**」ペインで列を設定します。 列を表示または非表示にするには、ペイン内を右クリックして、ポップアップ・メニューから「**列の選択**」を選択します。表示されるダイアログ・ボックスで、表示する列を選択してから、「**OK**」をクリックします。 列をドラッグ・アンド・ドロップすることによって、ビュー内で移動することもできます。
2. 必要であれば、レンダリングのスクロール・バーを使用して、フィールドを表示します。 あるいは、モニターを右クリックして、「**フィールドの検索**」ポップアップ・メニュー項目 を選択します。 フィールドの検索について詳しくは、関連トピックを参照してください。
3. 必要であれば、タイプの表示または非表示を行うレンダリングを設定します。これを行うには、メモリー・マップ・モニターを右クリックして、ポップアップ・メニューから「**型の表示**」または「**型の非表示**」を選択します。
4. 必要であれば、表示するフィールドのメモリー内容の表記を変更します。 これを行うには、フィールドまたはその値を右クリックして、ポップアップ・メニューから「**表記**」**>**「**<表記フォーマット>**」 を選択します。
5. 「**オフセット**」列での値に適切な表示タイプを選択するには、「**レンダリング**」ペインを右クリックして、「**オフセット表示の選択**」 **>** **オフセット表示タイプ**を選択します。
6. 表示しやすくするために、メモリー・フィールドをグループ化して、 このグループにフィルターを設定することができます。マッピング・レイアウト・フィールドの グループ化についての詳細は、関連トピックを参照してください。

「メモリー」ビューでは、複数の変数、式、およびレジスターをモニターできます。単一のメモリー・モニターについて複数のマップ・レンダリングを追加できます。複数の「メモリー」ビューを ワークベンチに追加することもできます。「メモリー」ビューの「**モニター**」ペインに、追加したそれぞれの変数、式、またはレジスターがリストされます。「**レンダリング**」ペインには、 「**メモリー**」ビューで現在選択されているモニターのメモリー・レンダリング のみが表示されます (複数のレンダリングは、タブまたは分割ペインで分離されます)。

メモリー・マップの設定を行う

メモリー・マップの設定では、メモリー・マップのロケーションを設定することができます。 また、「グループ管理」ダイアログで、「すべてのグループを除去」を 選択したときに、デバッガーからメッセージを出すかどうか 設定することもできます。フィールドを検出する前にビルドされるようにマップを設定することもできます。

デバッガーと一緒にインストールした製品には、サンプルのメモリー・マップ・ディレクトリー <product installation directory>¥plugins¥com.ibm.debug.memorymap.<platform>.samples¥samples が含まれていることがあります (<product installation directory> はこの製品をインストールしたディレクトリー)。このディレクトリーが製品に含まれている場合、 デフォルトで、デバッガーは、メモリー・マップをこのディレクトリーの中で探します。そうでない場合、 デフォルトのメモリー・マップのディレクトリーは、メモリー・マップ設定に指定されています。 メモリー・マップのディレクトリーには、「メモリー」ビューで必要な layout.dtd ファイルが含まれていなければ なりません。メモリー・マップのロケーションは変更できますが、 変更する場合は、layout.dtd ファイルを新規メモリー・マップのロケーションにコピーしておく必要があります (このロケーションにマップをエクスポートすると、エクスポート・プロシージャーによって layout.dtd ファイルが自動的に生成されます)。このファイルは、常にメモリー・マップのロケーションになければなりません。

注 : layout.dtd ファイルは、このデバッガーと同時にインストールした製品のダウンロード・サイトからでも入手可能な場合があります。このデバッガーと一緒にインストールした製品で使用できる layout.dtd が存在しない場合は、 346 ページの『マッピング・レイアウトの定義』 の説明に従って layout.dtd ファイルを作成できます。

デバッガーに、作成したメモリー・マップを検索させるために、メモリー・マップをデフォルト・ディレクトリーに追加することも、以下のように、メモリー・マップのロケーションを別のディレクトリーを指すように変更することもできます (この別のディレクトリーには、layout.dtd ファイルのコピーが必ず含まれるようにしてください)。

1. 「メモリー」ビューで、下矢印アイコンをクリックして、メニューから「**メモリー・マップ設定** 」を選択します。
2. 「**メモリー・マップ設定**」ダイアログ・ボックスで、 「**メモリー・マップのロケーション**」フィールドに設定するメモリー・マップのロケーションを入力するか、参照します。

   注 :

   - このデバッガーと共に実行する製品にリモート・システム・エクスプローラーが付属している場合、メモリー・マップのロケーション設定は、このダイアログ・ボックスの 「**メモリー・マップのロケーション**」セクションで行われます。このセクションでは、 リモート・サーバー上のロケーションを入力または参照できます。これを行うには、メモリー・マップのロケーションに関連付けられた「**プロファイル**」と「**接続**」を選択します (プロファイルを指定していないか、ワークスペースに何も存在しないか、あるいはその両方の場合、「**ディレクトリー**」フィールドに入力したファイル名はローカル・ファイルとして扱われ、どのプロファイルにも関連付けられません)。 次に、「**ディレクトリー**」フィールドにメモリー・マップのロケーションのフォルダーを指定します。 メモリーのマップを行うとき、指定されたロケーションにあるマップのリスト が表示されます。このロケーションがリモートである場合、 使用可能なマップのリストを取得するため、リモート・サーバーへの接続が試みられます。 「**マップ**」オプションが選択されている場合、これによって、 リモート・システムとローカル・システム両方にあるマップをブラウズすることができます。選択された マップ・ファイルがリモート・システム上にある場合、必要なリモート・ファイルが ローカル・システムにキャッシュされます。
   - デフォルトのメモリー・マップの場所を変更した場合は、「**メモリー・マップ設定**」ダイアログ・ボックスの「**デフォルトを復元**」プッシュボタンをクリックすると、製品のデフォルト値の設定に簡単に戻すことができます。
3. 取得するメモリー・ブロックのサイズを制御するには、「**最小メモリー・ブロック取得サイズ (バイト)**」フィールドと「**最大メモリー・ブロック取得サイズ (バイト)**」フィールドに入力します。取得したメモリーのブロックは、最小メモリー・ブロック取得サイズと同じ大きさのセグメントに分割されます。その後、取得要求は、最大メモリー・ブロック取得サイズまで統合されます。

   注 :

- 指定された最大メモリー・ブロック取得サイズが、デバッガー・エンジンでサポートされる最大サイズを超えると、デバッガー・エンジンによってサポートされる最大サイズが使用されます。
- メモリーのマッピング中にパフォーマンス上の問題が発生した場合は、最小ブロック・サイズを増やすと役立つことがあります。連続する大きいマップの場合は、最小ブロック・サイズの値を大きくするとパフォーマンスが改善されます。

4. グループをすべて除去するときにプロンプトが出されるようにする 場合は、「**グループをすべて除去する際にプロンプトを出す**」チェック・ボックスを選択します。
5. マップの再ビルド前にグループ化情報と記述情報を保存または廃棄するプロンプトを出すかどうかを選択します。このチェック・ボックスを選択しない場合は、最後の保存/廃棄アクションが記憶されます (例えば、マップの最後の再ビルドでのアクションが保存だった場合、情報は保存されます)。
6. レンダリングでグループと記述を変更したときに、XML マップ・ファイルを保存するかどうかを指定します。このチェック・ボックスを選択すると、変更時にレンダリングが再ビルドされます。関連する XML ファイルを使用する「メモリー」ビュー内のすべてのレンダリングが再ビルドされます。
7. 「フィールドの検索」ダイアログ・ボックスを開く前にマップをビルドするには、 「「**フィールドの検索」ダイアログのオープン時にマップを自動的にビルド**」チェック・ボックスを 選択します。このチェック・ボックスが選択されていない場合、既にビルド済み (または展開済み) の要素のみが「フィールドの検索」ダイアログ・ボックスに表示されます。デフォルトでは、このチェック・ボックスは選択されます。
8. マップのエクスポートによって他のメモリーのレンダリングが影響を受けるときに警告メッセージを受信するための、希望する設定を入力します。

メモリーをマップする際に、表示された使用可能なマップのリストが、 メモリー・マップのロケーションにあるマップです。 同様に、「**マップ**」アクションを使用してメモリーを マップするときは、このロケーションのマップを見つけるようプロンプトが出されます。 しかし、このアクションを使用すると、ローカル・システムの別の場所を参照して、 メモリーのマップを探すこともできます。 ローカル・システムの別の場所を参照し、 そのロケーションからマップを選択すると、 そのロケーションがデフォルトのメモリー・マップのロケーションになります。

注 : このデバッガーと共に実行する製品にリモート・システム・エクスプローラーが付属している場合、リモート・システムまたはローカル・システム上のマップを参照できます。リモート・システムまたは ローカル・システム上の別のロケーションからマップを選択する場合、そのロケーションがデフォルトのメモリー・マップのロケーションに なります。

式、変数、またはレジスターに対するメモリーのマップピング

式または変数にメモリーをマップするには、 「メモリー」ビューに式または変数を追加する指示に従い、メモリー・レンダリングを選択するときに「**マップ**」 オプションを選択します。これと同様に、レジスターにメモリーをマップするには、 「メモリー」ビューにレジスターを追加する指示に従い、メモリー・レンダリングを選択するときに 「**マップ**」オプションを選択します。

式、変数、およびレジスターの「メモリー」ビューへの追加については、 「関連トピック」を参照してください。

エラーがあるマップを使用したメモリーのレンダリングを選択すると、「メモリー」ビューの「**レンダリング**」ペインに、エラーを解決するためのオプションを示したエラー・メッセージが表示されます。 例えば、エラー・ページには、ファイルを開く (ファイルの編集と 保存も可能) ためのオプションと、ファイルの再ビルドのためのオプションが含まれている可能性があります。メモリーを マップすると、そのマップは展開されたノードのエレメントのみをビルドします。エラーのある ノードが展開されるまで、エラーは発生しません。これらのエラーを修正するには、マップを 開いてエラーを修正してから、そのマップを再ビルドしてください。 メモリー・レイアウトの編集については、「関連トピック」を参照してください。

マッピング・レイアウトの定義

以下では、マップのレイアウト定義について説明し、例を示します。

レイアウト XML ファイルの作成

以下に示すように、XML ファイル・フォーマットは、layout.dtd 文書タイプ定義 (DTD) ファイルの中に定義されます。

```
<?xml version="1.0"?>
  <!ELEMENT LAYOUT (FIELD)+>
```

```
  <!ATTLIST LAYOUT Header CDATA #REQUIRED length CDATA #REQUIRED>
  <!ELEMENT GROUP EMPTY>
  <!ATTLIST GROUP Name CDATA #REQUIRED>
  <!ELEMENT FIELD (FIELD)*>
  <!ATTLIST FIELD
    Header CDATA #REQUIRED
    Type (16_BIT_INT|16_BIT_UINT|16_BIT_HINT|32_BIT_INT|32_BIT_UINT|32_BIT_HINT|32_BIT_FLOAT|
64_BIT_INT|64_BIT_FLOAT|CHARACTER|HEX|ASCII|EBCDIC|STRUCTURE|PADDING|BIT|BITMASK|MAP) #REQUIRED
    length CDATA #REQUIRED
    layout CDATA #IMPLIED
    filename CDATA #IMPLIED
      Groups CDATA #IMPLIED>
```

これは、XML レイアウト・ファイルが最初にヘッダー (タイトル) およびレイアウトの全長を指定し、 それに続いて、サブエレメント (FIELD) のリストを指定することを意味します。 サブエレメントは、ヘッダー (名前)、長さ、および、そのサブエレメントのデフォルト表記を決めるのに使用される基本タイプによって記述されています。

以下のような特殊なサブエレメント・タイプもあります。

- PADDING は、特定的にレイアウトする必要のないバイトのブロックを定義するために使用されます。
- STRUCTURE は、ネストされた構造体を導入します。 このサブエレメントは値を持ちません。
- BITMASK は、ビット・マスクされたサブエレメントを定義するために使用されます。 そのサブエレメントは、ビット、グループ、または BIT タイプによって定義されるビットを表します。
- UNION は、複数の方法でメモリーの同じ部分を定義します。

以下の例は、 次のような C 言語構造体のレイアウトを定義します。

```
typedef struct {
  unsigned short ushort_val;
  short short_val;
  unsigned long ulong_val;
  long long_val;
  char string_val[12];
  char char_val;
} _test;
```

_test 構造体のツリー・ビューを記述していて、このフォーマットに準拠している XML ファイルは次のとおりです。

```
<?xml version="1.0"?>
  <LAYOUT Header="A Layout" description="Tree view" length="25">
  <FIELD Header="ushort_val" Type="16_BIT_UINT" length="2"></FIELD>
  <FIELD Header="short_val" Type="16_BIT_INT" length="2"></FIELD>
  <FIELD Header="ulong_val" Type="32_BIT_UINT" length="4"></FIELD>
  <FIELD Header="long_val" Type="32_BIT_INT" length="4"></FIELD>
  <FIELD Header="string_val" Type="ASCII" length="12"></FIELD>
  <FIELD Header="char_val" Type="ASCII" length="1"></FIELD>
</LAYOUT>
```

offset 属性と offset_mode 属性を使用すると、マップの開始を基準とする (offset_mode=absolute) か、現行アドレスを基準として (offset_mode=relative) フィールドの正確なロケーションを指定できます。 次の例の b という要素では、offset に 10 が指定され、offset_mode は relative として定義されています。これらの属性がない場合は、この要素のオフセットは 80 になります。ただし、オフセットは、現在位置を基準とする 10 として定義されているため、オフセットは 90 です。要素 a の長さには接頭部 0x が付いているため、長さは 16 進数で定義されることに注意してください。一般に、length 属性および offset 属性は接頭部 0x の付いた 16 進数で指定できます。 要素 c のオフセットは 4 で、モードは absolute です。これは、この要素のオフセットが、レイアウトの開始から 4 バイト離れていることを意味します。フィールド c によってマップされる 10 バイトは、フィールド a によってもカバーされます。

```
<Header="offset_Test" length="190">
  <FIELD Header="a" Type="HEX" length="0x64"></FIELD>
  <FIELD Header="b" description="offset = 90" Type="HEX" length="80" offset="10"
offset_mode="relative"></FIELD>
  <FIELD Header="c" Type="HEX" length="10" offset="4" offset_mode="absolute"></FIELD>
</LAYOUT>
```

パディング・フィールドの定義

バイト位置合わせ構造を扱ったり、メモリー・マップでの定義が不要なマップ内のデータ領域をスキップしたりするために、埋め込みフィールドを使用できます。 例えば、上で定義された _test 構造の場合、long_val フィールドを無視し、string_val タイプをレイアウトに表示するマップを作成できます。 XML ファイルは次のようになります。

```
<?xml version="1.0"?>
  <LAYOUT Header="A Layout" description="Tree view" length="0x19">
  <FIELD Header="ushort_val" Type="16_BIT_UINT" length="2"></FIELD>
  <FIELD Header="short_val" Type="16_BIT_INT" length="2"></FIELD>
  <FIELD Header="ulong_val" Type="32_BIT_UINT" length="4"></FIELD>
  <FIELD Header="" Type="PADDING" length="4"></FIELD>
  <FIELD Header="string_val" Type="ASCII" length="12"></FIELD>
  <FIELD Header="char_val" Type="ASCII" length="1"></FIELD>
</LAYOUT>
```

string_val の offset に 4 を指定し、offset_mode に relative を指定することにより、offset 属性をここで使用して long_val フィールドをスキップすることもできます。

```
  <FIELD Header="string_val" Type="ASCII" length="12" offset="4" offset_mode="relative"></FIELD>
```

これは、string_val フィールドのアドレスが実際には ulong_val フィールドの最終バイトを基準にして 4 バイトであり、そのため string_val フィールドで使用されるバイトをスキップすることを意味します。

構造体の定義

以下の XML サンプルは、ネストされた構造体をマップするための STRUCTURE フィールドの使用法を示しています。 構造体トップ・エレメントは、関連付けられた値をもちません。 それは、そのサブエレメントを示すために展開できます。 STRUCTURE フィールドの長さは XML レイアウトの合計サイズに加えられますが、インクルードされたフィールド・サイズについては、表示のためだけを意図しています。 例えば、以下の構造体は合計レイアウト・サイズの中の 344 バイトのみを意味します。

```
<FIELD Header="MACHINE CHECK LOG OUT AREA" Type="STRUCTURE" length="344">
  <FIELD Header="reserved" Type="HEX" length="16"></FIELD>
  <FIELD Header="FLCSID" Type="HEX" length="4"></FIELD>
  <FIELD Header="FLCIOFP" Type="HEX" length="4"></FIELD>
  <FIELD Header="reserved" Type="HEX" length="20"></FIELD>
  <FIELD Header="FLCESAR" Type="HEX" length="4"></FIELD>
  <FIELD Header="FLCCTSA" Type="HEX" length="8"></FIELD>
  <FIELD Header="FLCCCSA" Type="HEX" length="8"></FIELD>
  <FIELD Header="FLCMCIC" Type="HEX" length="8"></FIELD>
  <FIELD Header="reserved" Type="HEX" length="8"></FIELD>
  <FIELD Header="FLCFSA" Type="HEX" length="4"></FIELD>
  <FIELD Header="reserved" Type="HEX" length="4"></FIELD>
  <FIELD Header="FLCFLA" Type="HEX" length="16"></FIELD>
  <FIELD Header="FLCRV110" Type="HEX" length="16"></FIELD>
  <FIELD Header="FLCARSAV" Type="STRUCTURE" length="64">
    <FIELD Header="AR0" Type="HEX" length="4"></FIELD>
    <FIELD Header="AR1" Type="HEX" length="4"></FIELD>
    <FIELD Header="AR2" Type="HEX" length="4"></FIELD>
    <FIELD Header="AR3" Type="HEX" length="4"></FIELD>
    <FIELD Header="AR4" Type="HEX" length="4"></FIELD>
    <FIELD Header="AR5" Type="HEX" length="4"></FIELD>
    <FIELD Header="AR6" Type="HEX" length="4"></FIELD>
    <FIELD Header="AR7" Type="HEX" length="4"></FIELD>
    <FIELD Header="AR8" Type="HEX" length="4"></FIELD>
    <FIELD Header="AR9" Type="HEX" length="4"></FIELD>
    <FIELD Header="AR10" Type="HEX" length="4"></FIELD>
    <FIELD Header="AR11" Type="HEX" length="4"></FIELD>
    <FIELD Header="AR12" Type="HEX" length="4"></FIELD>
    <FIELD Header="AR13" Type="HEX" length="4"></FIELD>
    <FIELD Header="AR14" Type="HEX" length="4"></FIELD>
    <FIELD Header="AR15" Type="HEX" length="4"></FIELD>
  </FIELD>
  <FIELD Header="FLCFPSAV" Type="HEX" length="32"></FIELD>
  <FIELD Header="" Type="PADDING" length="64"></FIELD>
  <FIELD Header="" Type="PADDING" length="64"></FIELD>
</FIELD>
```

構造体は、レイアウトの内部にも外部にも 定義できます。外部構造体は、 構造体フィールドに filename="<file name>" を指定することによって、 ネストされたレイアウトのように作成できます。ここで、<file name> が指すファイルは、 構造体の実際の定義を含みます。

例えば、MACHINE CHECK LOG OUT AREA 構造体をマッピング・レイアウトに外部的に指定するには、<FIELD Header="MACHINE CHECK LOG OUT AREA" Type="STRUCTURE" length="344" filename="machine.xml"></FIELD> のように指定します。

ビット・マスク・フィールドの定義

以下の XML の一例は、BITMASK フィールドを記述しているサンプルです。 BITMASK の長さはバイト数で指定されます。 それには、長さがビット数で指定される BIT フィールドのセットが含まれます。 BIT フィールドに対して表示されるオフセットは、BITMASK フィールド内のビット・オフセットです。 ビット・マスク・フィールドの長さは XML レイアウトの合計サイズに加えられるが、個別の BIT フィールド・サイズについては、表示のためだけを意図しています。

```
<FIELD Header="BITMASK" Type="BITMASK" length="1">
  <FIELD Header="BIT 1" Type="BIT" length="1"></FIELD>
  <FIELD Header="BIT 2" Type="BIT" length="1"></FIELD>
  <FIELD Header="BIT 3" Type="BIT" length="1"></FIELD>
  <FIELD Header="BIT 4" Type="BIT" length="1"></FIELD>
  <FIELD Header="BIT 5" Type="BIT" length="1"></FIELD>
  <FIELD Header="BIT 6" Type="BIT" length="1"></FIELD>
  <FIELD Header="BIT 7" Type="BIT" length="1"></FIELD>
  <FIELD Header="BIT 8" Type="BIT" length="1"></FIELD>
</FIELD>
```

ユニオンの定義

次の例では、以下の C 言語共用体のレイアウトを定義します。

```
union my_union {
  int my_intVal;
  double my_doubleVal;
};
```

以下のサンプル XML では、共用体の記述方法を説明します。XML では、共用体の長さは最大フィールドのサイズであることに注意してください。

```
<LAYOUT Header="UNIONS" length="8">
  <FIELD Header="my_union" Type="UNION" length="8">
    <FIELD Header="my_intVal" Type="HEX" length="4" description="value within the union"></
FIELD>
    <FIELD Header="my_doubleVal" Type="HEX" length="8"></FIELD>
  </FIELD>
</LAYOUT>
```

ネストされたレイアウトの定義

MAP フィールド・タイプとオプションのレイアウト・フィールドを一緒に使用すれば、以下の DSA レイアウト例のように、ネストされたレイアウトを記述できます。

```
<?xml version="1.0"?>
<!DOCTYPE LAYOUT SYSTEM "Layout.dtd">
<LAYOUT Header="DSA" length="72">
  <FIELD Header="FLAGS" Type="HEX" length="2"></FIELD>
  <FIELD Header="junk" Type="HEX" length="2"></FIELD>
  <FIELD Header="Back Chain" Type="MAP" length="4" layout="dsa.xml"></FIELD>
  <FIELD Header="Forward Chain" Type="MAP" length="4" layout="dsa.xml"></FIELD>
  <FIELD Header="R14" Type="HEX" length="4"></FIELD>
  <FIELD Header="R15" Type="HEX" length="4"></FIELD>
  <FIELD Header="R0" Type="HEX" length="4"></FIELD>
  <FIELD Header="R1" Type="HEX" length="4"></FIELD>
  <FIELD Header="R2" Type="HEX" length="4"></FIELD>
  <FIELD Header="R3" Type="HEX" length="4"></FIELD>
  <FIELD Header="R4" Type="HEX" length="4"></FIELD>
  <FIELD Header="R5" Type="HEX" length="4"></FIELD>
  <FIELD Header="R6" Type="HEX" length="4"></FIELD>
  <FIELD Header="R7" Type="HEX" length="4"></FIELD>
  <FIELD Header="R8" Type="HEX" length="4"></FIELD>
  <FIELD Header="R9" Type="HEX" length="4"></FIELD>
  <FIELD Header="R10" Type="HEX" length="4"></FIELD>
```

```
  <FIELD Header="R11" Type="HEX" length="4"></FIELD>
  <FIELD Header="R12" Type="HEX" length="4"></FIELD>
</LAYOUT>
```

この整形式 XML レイアウトは、DSA.XML という名前のファイルに保管されます。 お気付きのように、フィールド 3 および 4 には異なる DSA 構造体を指すポインターが入っているので、2 つのネストされたレイアウト定義を追加することになります。

注 : そのレイアウトの実際のメモリー・マッピングが実行されるのは、再帰的なレイアウト展開を防ぐために、初めてレイアウト・エレメントを展開するときだけです。

グループの定義

グループ構文を使用すると、マッピング・レイアウトのフィールドを グループに編成することができるため、操作がしやすくなります。グループを定義するには、 レイアウト・ファイルの先頭に <GROUP Name="groupName"></GROUP> を 置く必要があります。次に、 <FIELD Header="RESERVED" Type="HEX" length="12" Groups="groupName"></FIELD> と指定して、事前定義されたグループにフィールドが属していることを示します。

1 つのフィールドが複数のグループに 属すことができます。複数のグループを定義するには、 Groups 属性で、コンマで区切ったリストに指定してください。Groups 属性内の各グループは、<GROUP> タグを使用してレイアウトで定義されている必要があります。

ALL グループ名は 特別なグループです。これをフィールドに指定すると、すべてのグループに属する ことになり、このフィールドはすべてのグループから見えるようになります。次の コードのサンプルに、グループが入っています。

```
<?xml version="1.0"?>
<!DOCTYPE LAYOUT SYSTEM "Layout.dtd">
<LAYOUT Header="GROUP_EXAMPLE" length="32">
  <GROUP Name="GroupA"></GROUP>
  <GROUP Name="GroupB"></GROUP>
  <FIELD Header="FIELD_A" Type="HEX" length="8" Groups="GroupA"></FIELD>
  <FIELD Header="FIELD_B" Type="HEX" length="8" Groups="GroupB"></FIELD>
  <FIELD Header="FIELD_AB" Type="HEX" length="8" Groups="GroupA,GroupB"></FIELD>
  <FIELD Header="FIELD_ALL" Type="HEX" length="8" Groups="ALL"></FIELD>
</LAYOUT>
```

ORG グループの定義

ORG_GROUP タグを使用して、メモリーの以前に定義した部分のレイアウトを定義できます。これは、アセンブラーでの ORG 命令の動作と似ています。FIELD 属性を使用して、新規レイアウトの開始位置を指定できます。単純なケースでは、FIELD 属性の値に、マップで以前に定義したフィールドの名前を指定できます。*NONE または * を値として使用することもできます。これは、メモリー内の現在位置のレイアウトを意味します。Header 属性は、単に新規レイアウトの名前です。

```
<LAYOUT Header="SW00SR" length="271">
   <ORG_GROUP FIELD="*NONE" Header="ORG_GROUP1">
    <FIELD Header="A" length="4" Type="HEX"></FIELD>
    <FIELD Header="B" length="4" Type="HEX"></FIELD>
    <FIELD Header="c" length="4" Type="HEX"></FIELD>
  </ORG_GROUP>
   <ORG_GROUP FIELD="A" Header="my_custom_header">
    <FIELD Header="F" length="4" Type="HEX" description="address of F == address of A"></FIELD>
    <FIELD Header="G" length="4" Type="HEX"></FIELD>
    <FIELD Header="H" length="4" Type="HEX"></FIELD>
    <ORG_GROUP FIELD="*+4" Header="another_org">
      <FIELD Header="J" length="2" Type="HEX" description="address of J = current location +
4"></FIELD>
    </ORG_GROUP>
  </ORG_GROUP>
  <FIELD Header="R" length="4" Type="HEX"></FIELD>
  <FIELD Header="Z" length="4" Type="HEX"></FIELD>
</LAYOUT>
```

ここで、

- Header 属性は定義されたグループの名前で、構造体またはマップ・エレメントの Header 属性に似ています。

- FIELD の値は評価され、ORG_GROUP の開始アドレスとして使用されます。例えば、
  - FIELD="*NONE" または FIELD="*" は、マップでの現在位置を意味します。
  - FIELD="* +/- a +/- b ..." も有効です。 a および b はマップ内のフィールドの名前 (この項目は定義済みでなければなりません) であるか、a および b は整数であるかのいずれかです。
  - FIELD="NAME" は、NAME というマップ内の要素のアドレスを意味します。これは、FIELD="NAME" または FIELD="NAME +/- a_1 +/- a_2 .... +/- a_n" のような式の場合もあります。 a_i はそれぞれ、マップ内のフィールドの名前 (この項目は定義済みでなければなりません) または整数のいずれかです。

メモリー・レイアウトの編集

「メモリー」ビューからメモリー・レイアウトの編集を行う方法が 2 種類用意されています。方法の 1 つめは、レンダリングに使用している現在のマップに対するグループを 設定し、続いてマップをエクスポートする (マップをソース・レイアウトとして 同じディレクトリーにエクスポートすると、既存のマップを上書きします) ことができます。 マップ・レイアウト・フィールドのグループ化についての詳細は、関連トピックを参照してください。

もう 1 種類の方法としては、 現在レンダリングに使用しているマップ・ファイルを開いて編集し、再ビルドして、 使用できるようにすることができます。マップ・ファイルを開くには、マップ・ファイルのレンダリング内部で右クリックし、メニューから「**マップ・ファイルを開く**」を選択します。編集するマップ・ファイルが開きます。マップの編集が終了したら、保存します。変更したマップを現在のレンダリングに使用するには、「メモリー」ビューの「**レンダリング**」ペイン内を右クリックして、「**マップの再ビルド**」を選択します。

注 : 1 つのマップ・ファイルの ノードを右クリックして「**マップ・ファイルを開く**」と、 そのマップ・ファイルの XML ファイルが開きます。複数のマップのノードを右クリックした場合、 ポップアップ・メニューの「**マップ・ファイルを開く**」アクションを選択すると、 サブメニューが開き、選択したすべてのマップの XML ファイルがリストされます。この リストから、開く XML ファイルを選択します。

関連タスク

352 ページの『マップ・レイアウト・フィールドのグループ化』
マッピング・レイアウト内のフィールドを グループに編成すると、操作がしやすくなります。

「メモリー」ビューでのマップされたメモリーおよびフィールド記述の編集

「メモリー」ビューでマップされたメモリーの内容またはフィールド記述を変更するには、以下の手順を完了してください。

1. 「メモリー」ビューの「**レンダリング**」ペインで、変更を行うマップされたレンダリングを選択します。
2. 変更するフィールドまでスクロールダウンします。あるいは、レンダリングを右クリックして、「**フィールドの検索**」メニュー項目を選択します。これで「**フィールドの検索**」ダイアログ・ボックスが開き、ジャンプ先のフィールドを入力できます。
3. 以下のいずれかのタスクを実行して、メモリーの内容を変更します。
   a) フィールドまたはその値をダブルクリックします。これで、フィールド値は編集モードに入り、 そこでそのメモリー・ロケーションの有効値を入力できます。
   b) フィールドまたはその値を右クリックし、メニューから「**値の編集**」を選択します。これで、フィールド値は編集モードに入り、 そこでそのメモリー・ロケーションの有効値を入力できます。
4. 以下のいずれかのタスクを実行して、フィールド記述を変更します。
   a) フィールドの「**記述**」セルをダブルクリックします。 これによって記述が編集モードになり、記述を入力したり既存の記述を編集したりできるようになります。
   b) フィールドを右クリックして、メニューから「**記述の編集**」を選択します。これによって記述が編集モードになり、記述を入力したり既存の記述を編集したりできるようになります。
5. **Enter** を押して、変更を実行依頼します。 メモリーの内容を変更する場合は、デバッガーによって有効な値かどうかがチェックされます。

複数のフィールドの記述を一度に編集するには、キーボードの Ctrl キーまたは Shift キーを使用してフィールドを選択してから、右クリックしてメニューから「**記述の編集**」を選択します。これによって、「記述の

編集」ダイアログ・ボックスが開きます。ここで、フィールドの記述を編集して、選択されたすべてのフィールドに 1 つの記述を適用できます。

注 : 編集できるのはフィールド記述のみです。分割された要素の記述または編成グループの記述は編集できません。

「メモリー」ビューからのマップされたメモリーの除去

「メモリー」ビューの「**レンダリング**」ペインから現在のメモリー・マップを 除去するには、「**レンダリングの除去**」ボタン ( ) を クリックします。

マップ・レイアウト・フィールドのグループ化

マッピング・レイアウト内のフィールドを グループに編成すると、操作がしやすくなります。

1. 「メモリー」ビューの「**レンダリング**」ペインの内側で 右クリックし、「**グループの管理**」をクリックします。 そうすると、「グループの管理」 ダイアログ・ボックスが開きます。このダイアログ・ボックスで、現在のメモリー・マップに対する グループ名の追加と除去を行うことができます。
2. 必要なメモリー・グループが追加されると、次の手順で マップ・レイアウト・フィールドを追加することができます。
   a) グループに追加するフィールドを選択します。 複数のフィールドを選択するには、キーボードの Ctrl または Shift キーを使用します。
   b) 選択を右クリックして、ポップアップ・メニューから「**グループを設定**」 を選択します。このメニュー項目を選択すると、サブグループが展開します。そこで、 フィールドの追加先とするグループを選択するか、フィールドをすべてのグループに追加することを選択できます。
3. 操作に利用するグループを設定した後は、 それらのグループ設定を使用して、見やすくするために、マッピング・レイアウトから フィールドをフィルター操作で除外できます。これを行うには、ペイン内部で右クリックし、ポップアップ・メニューから「**グループを表示**」を選択します。このメニューはサブグループに展開します。ここでは 表示するグループを選択することができます。 グループを選択すると、そのグループに属さないフィールドがフィルターで除外されます。ペインにすべてのフィールドを表示 させる場合は、「**マップ全体を表示**」を選択してください。
4. 現在レンダリングされているマップにグループを 追加すると、変更をエクスポートすることができます。エクスポートするには、「**レンダリング**」ペイン内を右クリックして、ポップアップ・メニューから「**マップ・ファイルのエクスポート**」を選択します。マップ・ファイルにエラーがあると、 エラー・ダイアログが表示され、マップ・ファイルのエクスポートはできなくなります。マップ・ファイルに エラーがないと、マップを保存するロケーションを参照するダイアログ が表示されます。オリジナル・マップがあったロケーションにマップを保存する場合、マップはエクスポートされたマップにより上書されます。

   重要 : グループ化情報は、「メモリー・マップ設定」の「**グループおよび記述の編集時には必ず変更をファイルに保存する**」を選択するか、情報をファイルに明示的にエクスポートしない限り、メモリー・マップ・レイアウトには保存されません。それ以外の場合、レンダリングにグループ化情報を追加して、ファイルをエクスポートせずにレンダリングを除去すると、グループ化情報は廃棄されます。

マップに対してグループの変更を行い、この変更をすべて廃棄する場合は、 「**レンダリング**」ペインの内側で右クリックし、「**マップの再作成**」を選択します。これにより、マップの再作成の際に、保存されていないグループ化情報を 保存するかどうか確認するダイアログが表示されます。「**いいえ**」 をクリックすると、マップ・グループに対して行った変更はすべて廃棄されます。グループ化情報を保存する には、「**はい**」をクリックしてください。

フィールドの検索および展開

メモリー・マップに関する操作を行うときに、フィールドの検出を支援するアクションを利用できます。

デフォルトでは、マップを使用してメモリーをレンダリングするとき、 ルート要素のみが展開されます。他のすべての要素は縮小されます。マップ内のすべてのフィールド (マップ・タイプを 除く) を展開するには、マップ・レンダリングの内側で右クリックし、ポップアップ・メニューから 「**マップ全体を展開**」を選択します。ある 1 個のノードを展開し、すべての子を表示 するには、そのノードを右クリックし、「**<ノード名> を展開**」選択します。 ここで、**<ノード名>** は、展開しようと選択したノード です。このアクションを選択した場合、そのノード内のマップ・タイプは 展開されません。

「フィールドの検索」ダイアログ・ボックス を開くには、マップ・レンダリングの内側で右クリックし、ポップアップ・メニューから「**フィールドの検索**」を選択します。このダイアログ・ボックスでは、 ジャンプしたいフィールドを入力できます。「**検索フィルターの選択 (Choose a search filter)**」ドロップダウン選択ボックスで 検索フィルターを選択すれば、フィールド、記述、パス、またはグループで検索を実行できます。 その場合、検索に使用するストリングを「**検索ストリングの入力 (Enter a search string)**」フィールドに入力します。 例えば、グループ GroupA にあるフィールドを検索する場合は、「**グループ**」検索フィルターを 選択し、検索ストリング・フィールドに GroupA と入力します。 そのグループ内のフィールドだけがダイアログ・ボックス内のテーブルに表示されます。 そのフィールドから、検索するフィールドを選択できます。

ノードおよびマップは、展開される前は、 まだ「メモリー」ビューにビルド されていません。フィールドを検出しようとするとき、 「フィールドの検索」ダイアログ・ボックスに含まれているのは、ビルド済みのフィールドのみです。「メモリー」ビューで (表示目的のために) 分割された、長い要素を表示するには、「**分割された要素の表示**」チェック・ボックスを選択します。マップ全体をビルドして、 すべての要素 (タイプがマップのものを除く) がリスト・ボックスに表示されるようにするには、「**マップ全体をビルド**」チェック・ボックスを選択します。デフォルトでは、このチェック・ボックスは選択されます。 この設定 (「検索」ダイアログ・ボックスを開く前にマップをビルドする) は、 メモリー・マップ設定で行うこともできます。これについて詳しくは、関連トピック を参照してください。

「フィールドの検索」ダイアログ・ボックスが開くと、すべてのビルド済みフィールドが リスト・ボックスに表示されます。 「**列の選択 (Choose Columns)**」をクリックすれば、このリスト・ボックスに表示される列を制御できます。 単にフィールドを検索する場合は、「**フィールド**」検索フィルターを選択して、 検索するフィールド名 (またはフィールド名の一部) を検索ストリング・フィールドに入力します。 入力するに従って、 リスト・ボックス内の内容が絞り込まれ、フィールドに入力した文字で始まるフィールドのみが 表示されます。このような支援を利用して、検出したいフィールド名を 入力できます。このフィールドには、検出するフィールドに 対するフィルター (ゼロ個以上の文字を表す '*' ワイルドカード、 任意の 1 個の文字を表す '?' ワイルドカードを含む) を入力できます。ワイルドカードは、他のフィルターで検索を行うときも同様に使用されます。

「フィールドの検索」ダイアログ・ボックスにフィールドを 入力し、「**OK**」をクリックすると、そのフィールドがマップ内にあれば、メモリー・マップのレンダリングで そのフィールドが強調表示されます。

複数のメモリー・マップの追加
式、変数、またはレジスターを 「メモリー」ビューに追加する場合、 複数のマップに対する追加が可能です。

マップは、一度に 1 つずつ追加することもできますし、 マップを選択する際に、 キーボードの Shift キーまたは Ctrl キーを使用して、複数のマップを選択することもできます。 この結果、選択したマップごとに、メモリー・マップ・レンダリングが作成されます。 各レンダリングはタブで区切られます。

TXSeries または CICS TX on Cloud を使用したローカル CICS トランザクションのデバッグ

TXSeries または CICS TX on Cloud を使用してローカル CICS トランザクションをデバッグできます。

TXSeries または CICS TX on Cloud を使用して IBM Debug for Linux on x86 を構成するには、以下の手順に従います。

1. CICS 領域の Region Definition (RD) の AllowDebugging 属性を yes に変更します。以下のコマンドを使用して、デバッグについての領域構成を変更することができます。

   ```
   cicsupdate -r REGION_NAME -c rd AllowDebugging=yes
   ```

2. デバッグする CICS COBOL プログラムをコンパイルします。**cicstcl** を使用してコンパイルする場合は **-a** フラグを指定し、**cob2** コンパイラー・コマンドを使用してコンパイルする場合は **-g** オプションを追加します。例えば、次のように指定します。

   ```
   cicstcl -a -lIBMCOB prog.ccp
   ```

   注 : 文字 C で始まるトランザクションは、CICS の内部使用のために予約されているため、デバッグすることはできません。

3. 領域の環境ファイルで以下の環境変数を設定し、領域をコールド・スタートします。

```
DER_DBG_PATH=Path_to_source_files [To inform IDEBUG to pick the source file
                                   from the specified path]
CICS_IDEBUG_LIBPATH=/opt/ibm/cobol/debug/usr/lib/
```

4. 次のように、CDCN の提供するトランザクションを通して、分散デバッガーを使用するように領域を構成します。

   a. cicsterm クライアントを使用して領域に接続します (または、他の任意の 3270 ベース端末エミュレーターを使用できます)。

   b. CDCN トランザクションを実行します。CDCN トランザクションの最初の画面は、以下のようになります。

```
 CDCN              CICS Debugging Configuration Transaction

        DISPLAY :(         )                              DEBUG  : ON

 To configure for a ter
minal specify the TERMID            TERMID : (    )

 To configure for a system specify the SYSID              SYSID  : (    )

 To configure for a transaction specify the TRANSID       TRANSID: (    )

 To configure for a program specify the PROGRAM           PROGRAM: (    )


 ENTER:    COMMIT SELECTION
 PF1 : HELP                PF2 : DEBUG ON/OFF        PF3 : EXIT
 PF4 : MESSAGES            PF5 : UNDEFINED           PF6 : UNDEFINED
 PF7 : UNDEFINED           PF8 : UNDEFINED           PF9 : UNDEFINED
 PF10: UNDEFINED           PF11: UNDEFINED           PF12: UNDEFINED
```

   c. CDCN を使用して、特定のトランザクションや特定のプログラムなど、適切な CICS® リソースを構成します。 CDCN ではまた、指定の端末で稼働するすべてのプログラムをデバッグしたり、特定のシステムにルーティングされるすべてのプログラムをデバッグしたりすることもできます。 複数のリソースが指定された場合、CDCN は以下の優先順位で処理します。

      i) TERMID

      ii) SYSID

      iii) TRANSID

      iv) PROGRAM

      例えば、CUSTECIC という名前の PROGRAM リソースをデバッグするには、以下のステップを実行します。

      i) CDCN 画面で、「DISPLAY:」フィールドに、マシンの IP アドレスと、分散デバッガー・ユーザー・インターフェースが稼働するポートを設定します。

      ii) CUSTECIC プログラムのみをデバッグするには、「PROGRAM」フィールドに CUSTECIC を設定します。 以下の表示は、変更後の CDCN 画面の内容を示しています。

```
CDCN              CICS Debugging Configuration Transaction

       DISPLAY : 9.100.194.80:9005                        DEBUG  : ON

To configure for a terminal specify the TERMID           TERMID : (    )

To configure for a system specify the SYSID               SYSID  : (    )

To configure for a transaction specify the TRANSID       TRANSID: (    )
```

```
To configure for a program specify the PROGRAM            PROGRAM: CUSTECIC


ENTER:    COMMIT SELECTION
PF1 : HELP                PF2 : DEBUG ON/OFF        PF3 : EXIT
PF4 : MESSAGES            PF5 : UNDEFINED           PF6 : UNDEFINED
PF7 : UNDEFINED           PF8 : UNDEFINED           PF9 : UNDEFINED
PF10: UNDEFINED           PF11: UNDEFINED           PF12: UNDEFINED
```

iii) Enter を押します。 以下のメッセージが表示され、デバッガーが正常に構成されたことが示されます。

```
There are 2 messages:

ERZI04066I: Successfully configured debugging on program 'CUSTECIC'
ERZI04072I: The display to be used for the debugging information is
'9.100.194.80:9005 '
```

iv) Enter を押し、続いて F3 を押して CDCN トランザクションを終了します。

ワークステーション上で Eclipse IDE を使用して CICS プログラムのデバッグを開始するには、以下のステップに従います。

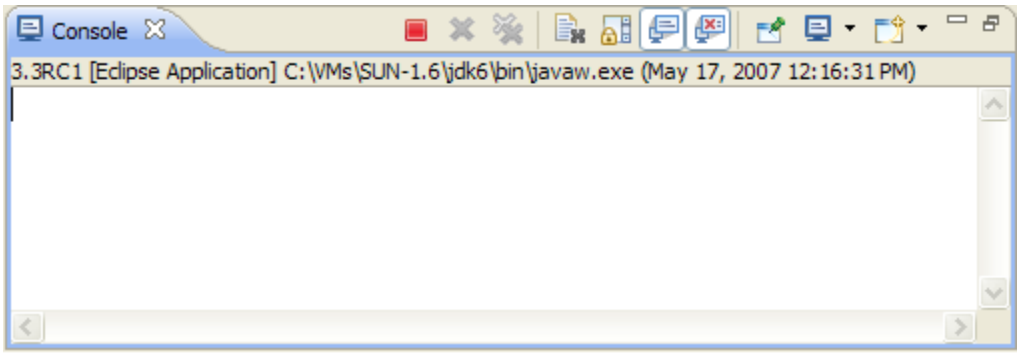
1. Eclipse IDE デバッグ・パースペクティブを開き、 をクリックして、 CDCN トランザクションを使用して CICS 領域からのデバッグ用に構成されたポートで listen を開始します。 デフォルトのポートは 8001 です。
2. トランザクションまたはプログラムを実行し、そのトランザクションまたはプログラムに、CDCN トランザクションを使用して既にデバッグのための構成が行われていることを確認します。
3. Eclipse IDE にプログラム・ソースが表示され、そのプログラムがデバッガーの制御下にあることが示されます。 デバッガー・オプションを使用して、CICS COBOL プログラムのデバッグを続行できます。

リファレンス

このセクションでは、IBM Debug for Linux on x86 のビューに関する参照情報を提供します。

「コンソール」ビュー

「コンソール」ビューには、開発のタイプおよび現在のユーザー設定の組み合わせに応じて、さまざまなコンソール・タイプが表示されます。



Eclipse プラットフォームでは、以下の 3 つのコンソールがデフォルトで提供されています。

- プロセス・コンソール
- スタック・トレース・コンソール
- CVS コンソール

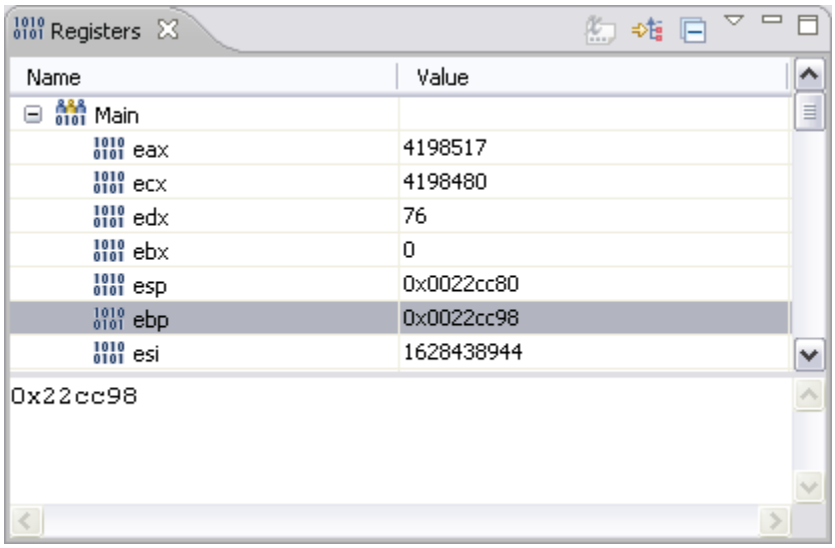
コンソールの設定は、「コンソール」設定ページで「実行/デバッグ」>「コンソール」を選択して変更することができます。

「コンソール」ビューで使用可能なコマンドを以下にリストします。

表 34. 「コンソール」ビューのコマンド			
コマンド	**名前**	**説明**	**可用性**
	コンソールのクリア	現在アクティブなコンソールがクリアされます。このコマンドは、ビュー・コマンドとコンテキスト・メニュー項目の両方として使用できるようになっています。	コンテキスト・メニューおよびアクションの表示
	選択されたコンソールの表示	現在のコンソールのリストが開き、そのリストから、表示するコンソールを選択することができます。	アクションの表示
	コンソールを開く	選択されたタイプの新しいコンソールが開きます。	アクションの表示
	ピン	現在のコンソールをピン留めして、常にその他のすべてのコンソールよりも手前に表示されるようにします。	アクションの表示
	スクロール・ロック	現在のコンソール内でスクロール・ロックを有効にするかどうかの設定を変更します。	コンテキスト・メニューおよびアクションの表示

「レジスター」ビュー

デバッグ・パースペクティブの「レジスター」ビューには、選択されているスタック・フレーム内のレジスターについての情報がリストされます。 プログラムが停止すると、変更された値が「レジスター」ビューで強調表示されます。



「レジスター」ビューのツールバーのオプション

以下の表に、「レジスター」ビューのツールバーに表示されるアイコンをリストします。

アイコン	名前	説明
	型名の表示	各レジスター値の横に型 (**int** など) を表示します。
	論理構造の表示	論理構造をビュー内に表示するかどうかを変更します。
	すべて省略表示	現在展開されているすべてのレジスターを省略表示します。
	「ビュー・メニュー」>「レイアウト」	「レジスター」ビューのさまざまなレイアウト・オプションを表示します。

「レジスター」ビューのコンテキスト・メニューのコマンド

「レジスター」ビューのコンテキスト・メニューのコマンドには、以下が含まれます。

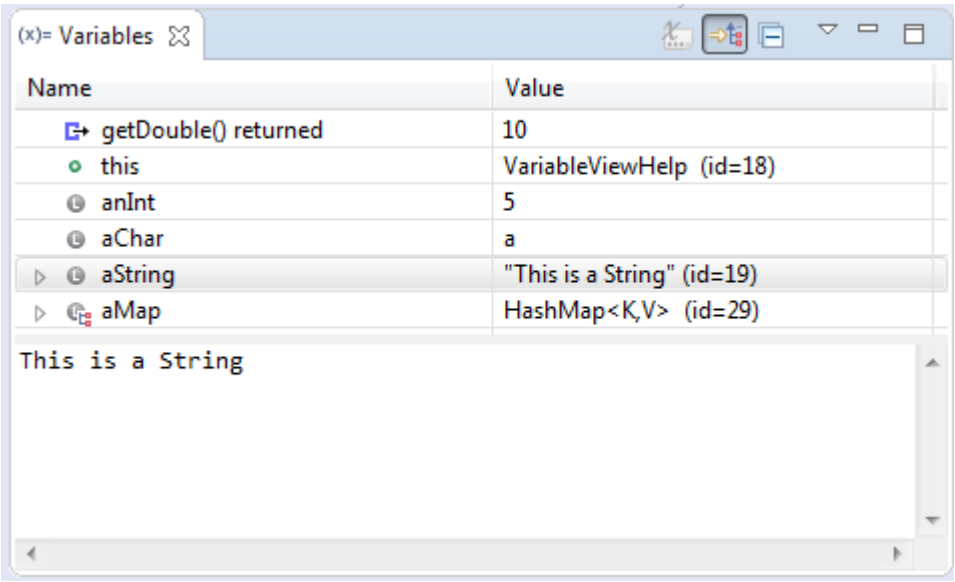
アイコン	名前	説明
	レジスター・グループの追加	「**レジスター・グループ**」ダイアログが開き、そこで「レジスター」ビューに表示されるレジスター・グループを定義することができます。
	値の代入	選択したレジスターに値を代入します。
	型にキャスト...	「**型にキャスト**」ダイアログを開きます。
	値の変更...	選択されているレジスター値の変更のために「**値の設定**」ダイアログを開きます。
	コンテンツ・アシスト	現行カーソル位置で「コンテンツ・アシスト」ダイアログを開きます。
	コピー	現在選択されているテキストまたは要素をクリップボードにコピーします。
	レジスターのコピー	レジスターの名前と内容をクリップボードにコピーします。
	監視式の作成	選択されているレジスターを監視式に変換します。
	切り取り	現在選択されているテキストまたは要素をクリップボードにコピーして、その要素を削除します。
	使用不可にする	選択されているレジスターを使用不可にします。
	配列として表示...	「**配列として表示**」ダイアログが開き、そこで配列の開始と長さを指定できます。

アイコン	名前	説明
	レジスター・グループの編集	選択されているレジスター・グループの編集のために「**レジスター・グループ**」ダイアログを開きます。
☑	使用可能にする	選択されているレジスターを使用可能にします。
	検索...	「**検索**」ダイアログが開き、そこでビュー内の特定の要素を検索できます。
	検索/置換	「**検索/置換**」ダイアログを開きます。
	フォーマット	フォーマット・タイプを選択できます。 選択肢には、デフォルト、10 進数、16 進数、8 進数、2 進数があります。
	最大長...	表示する文字の最大数の設定のために「**詳細ペインの構成**」ダイアログを開きます。デフォルトは 10000 です。
	貼り付け	現在のクリップボードの内容をテキストとして貼り付けます。
	レジスター・グループの除去	現在選択されているレジスター・グループを除去します。
	デフォルト・レジスター・グループの復元	オリジナルのレジスター・グループを復元します。
	オリジナル型の復元	選択したレジスターをオリジナルの型に戻します。
	すべて選択	エディターの内容をすべて選択します。
	テキストの折り返し	アクティブにすると、「レジスター」ビューの「詳細」ペインの可視領域内でテキスト・コンテンツが折り返されます。

「変数」ビュー

「変数」ビューには、「デバッグ」ビューで選択されているスタック・フレームに関連付けられた変数についての情報が表示されます。 Java™ プログラムをデバッグする場合は、変数を選択して、より詳しい情報を詳細ペインに表示できます。 さらに、Java オブジェクトを展開して、変数に含まれているフィールドを表示することができます。

「変数」ビューには列が表示されます。 詳細ペインは、テキストを表示するための、ビューの下部にある領域です。



「変数」ビューで使用できるコマンドは多数あります。

- ビューの表示コマンドは、ビューに表示される変数、およびその表示方法に影響します。
- 詳細ペインには、右クリックすると選択できる多くのコマンドがあります。
- ビューのレイアウト・コマンドは、詳細ペインの向き、および列を表示するかどうかに影響します。
- その他のコマンドを以下に示します。

表 35. 「変数」ビューのコマンド

コマンド	名前	説明	可用性
	すべてのインスタンス	選択された Java 型のすべてのインスタンスのリストを表示するポップアップ・ダイアログを開きます。 使用する Java 仮想マシンがインスタンスの取得をサポートしている必要があります。	コンテキスト・メニュー
	すべての参照	選択された変数への参照を持つすべての Java オブジェクトのリストを表示するポップアップ・ダイアログを開きます。使用する Java 仮想マシンが参照の取り出しをサポートしている必要があります。	コンテキスト・メニュー
	値の変更...	基礎になる選択された変数の値を変更できます。	コンテキスト・メニュー
	すべて省略表示	現在展開されているすべての変数を省略表示します。	アクションの表示

表 35. 「変数」ビューのコマンド (続き)

コマンド	名前	説明	可用性
	変数のコピー	選択されている変数をシステム・クリップボードにコピーします。	コンテキスト・メニュー
	監視式の作成	選択されている変数の監視式を作成できます。	コンテキスト・メニュー
	検索...	「変数」ビューで要素を検索するための検索ダイアログを開きます。	コンテキスト・メニュー
	インスペクション	選択された変数に対して新規のインスペクション・ステートメントを作成し、それを式ビューに追加します。	コンテキスト・メニュー
	インスタンス・ブレークポイント...	選択された変数インスタンスに対する既存のブレークポイントをフィルタリングできます。	コンテキスト・メニュー
	Java 設定...	ビューに影響を与えるオプションを含んでいる複数の設定ページを開きます。	アクションの表示
	新規詳細フォーマッター...	その型の変数のために独自の詳細フォーマッターを作成できます。	コンテキスト・メニュー
	実際の型を開く	選択された変数の実際の型を開きます。	コンテキスト・メニュー
	実際の型階層を開く	選択された変数の実際の型の実際の型階層を開きます。	コンテキスト・メニュー
	宣言された型を開く	選択された変数に対して宣言されている型を、新しいエディターで開きます。	コンテキスト・メニュー
	宣言された型階層を開く	選択されている変数の宣言された型の型階層を開きます。	コンテキスト・メニュー
	すべて選択	ビュー内のすべての変数を選択します。	コンテキスト・メニュー
	論理構造の表示	選択された論理構造型変数を表示するためのフォーマッターを選択できます。	コンテキスト・メニュー
	論理構造の編集	論理構造の編集のために設定ページを開きます。	サブコンテキスト・メニュー

表 35. 「変数」ビューのコマンド (続き)			
コマンド	**名前**	**説明**	**可用性**
	詳細の別名表示...	選択された変数に関する詳細情報の表示のために、別の詳細ペインを選択できます。	コンテキスト・メニュー
	監視ポイントの切り替え	現在選択されているフィールドに新規の監視ポイントを作成するか、または監視ポイントが既に存在している場合はそれを除去します。	コンテキスト・メニュー

リストの入手

コンパイラー・オプションを使用して適切なコンパイラー・リストを要求することによって、デバッグに必要な情報を 入手してください。

重要: コンパイラーによって作成されるリストは、プログラミング・インターフェースではなく、容易に変更できるものです。

表 36. **コンパイラー・オプションとリストの対応**			
用途	**リスト**	**内容**	**コンパイラー・オプション**
プログラムに有効なオプションのリスト、プログラムの内容に関する統計、およびコンパイルに関する診断メッセージを検査する。 コンパイル時に、有効なロケールを検査する。	短縮リスト	• プログラムに有効なオプションのリスト • プログラムの内容に関する統計 • コンパイルに関する診断メッセージ [1] 有効なロケールを示すロケール行	NOSOURCE、NOXREF、NOVBREF、NOMAP、NOLIST
プログラムのテストおよびデバッグを援助する。プログラムのデバッグ後にレコードを得る。	ソース・リスト	ソースのコピー	281 ページの『SOURCE』
特定のデータ項目を見つける。再入可能性または最適化を考慮した後の最終ストレージ割り振りを調べる。プログラムが定義されている場所を見つけ、その属性を検査する。	DATA DIVISION 項目のマップ	すべての DATA DIVISION 項目および暗黙的に宣言されたすべての項目 組み込みマップ要約 (DATA DIVISION 内の、データ宣言が含まれている行のリストの右マージン) ネストされたプログラム・マップ (ネストされたプログラムが含まれているプログラム)	271 ページの『MAP』 [2]

表 36. **コンパイラー・オプションとリストの対応** (続き)

用途	リスト	内容	コンパイラー・オプション
名前が定義、参照、または変更されている場所を調べる。プロシージャーが参照されているコンテキスト (例えば、ステートメントが PERFORM ブロックで使用されたかどうか) を判別する。コピーブックの取得元のファイルを判別する。	名前のソートされた相互参照リスト。COPY/BASIS ステートメントおよびコピーブック・ファイルのソートされた相互参照リスト	データ名、プロシージャー名、プログラム名。これらの名前への参照。 COPY/BASIS テキスト名とライブラリー名、および関連コピーブックを取得したファイル 埋め込まれた変更済み相互参照は、データ名およびプロシージャー名が定義された行番号を提供します。	290 ページの『XREF』 [2,3]
プログラム内で障害のあるステートメントを見つける。 または、プログラムの実行中に移動されるデータ項目のストレージ内でのアドレスを調べる。	コンパイラーによって生成される PROCEDURE DIVISION コードおよびアセンブラー・コード [3]	生成されたコード	270 ページの『LIST』 [2,4]
特定のステートメントのインスタンスを見つける。	アルファベット順のステートメント	使用されたそれぞれのステートメント、各ステートメントが使用された回数、各ステートメントが使用された行番号	289 ページの『VBREF』

1. メッセージを除去するには、コンパイル診断情報のレベルを左右するオプション (例えば、FLAG) をオフにしてください。
2. コンパイル済みプログラムの行番号を使用するには、NUMBER コンパイラー・オプションを使用してください。 コンパイラーは、ステートメントが読み込まれるときに、桁 1 から 6 にあるソース・ステートメント行番号のシーケンスを 検査します。 行番号が順序どおりになっていないことがわかると、コンパイラーは先行のステートメントの行番号より 1 だけ大きい値の番号を 割り当てます。 新しい値には、2 つのアスタリスクのフラグが付けられます。 シーケンス・エラーを示す診断メッセージがコンパイル・リストに入れられます。
3. プロシージャー参照のコンテキストは、行番号の前の文字で示されます。
4. アセンブラー・リストはリスト・ファイル (ソース・プログラムと同じ名前であるがサフィックスが .wlist であるファイル) に書き込まれます。

363 ページの『例: 短縮リスト』
364 ページの『例: SOURCE および NUMBER 出力』
365 ページの『例: MAP 出力』
366 ページの『例: 組み込みマップ要約』
368 ページの『例: ネストされたプログラム・マップ』
369 ページの『例: XREF 出力: データ名相互参照』
370 ページの『例: XREF 出力: プログラム名相互参照』
370 ページの『例: XREF 出力: COPY/BASIS 相互参照』
371 ページの『例: XREF 出力: 組み込み相互参照』
372 ページの『例: VBREF コンパイラー出力』

関連タスク
231 ページの『コンパイラー・メッセージのリストの生成』

関連参照
231 ページの『コンパイラー検出エラーに関するメッセージおよびリスト』

例: 短縮リスト

下記リストに示された括弧付きの番号は、リストに続く説明の番号と対応しています。 診断メッセージの原因となったエラーのいくつかは、説明を行うために故意に挿入されたものです。

```
PROCESS(CBL) statements:   (1)
 CBL    NOSOURCE,NOXREF,NOVBREF,NOMAP,NOLIST    (2)
Options in effect:    (3)
 NOADATA
   ADDR(32)
   QUOTE
   ARITH(COMPAT)
   CALLINT(NODESCRIPTOR)
   CHAR(NATIVE)
 NOCICS
   COLLSEQ(BINARY)
 NOCOMPILE(S)
 NOCURRENCY
 NODATEPROC
 NODIAGTRUNC
 NODYNAM
 NOEXIT
   FLAG(I,I)
 NOFLAGSTD
   FLOAT(NATIVE)
   LINECOUNT(60)
 NOLIST
   LSTFILE(LOCALE)
 NOMAP
   MAXMEM(2048K)
 NOMDECK
   NCOLLSEQ(BINARY)
   NSYMBOL(NATIONAL)
 NONUMBER
 NOOPTIMIZE
   PGMNAME(LONGUPPER)
   SEPOBJ
   SEQUENCE
   SIZE(8388608)
 NOSOSI
 NOSOURCE
   SPACE(1)
   SPILL(512)
 NOSQL
   SRCFORMAT(COMPAT)
 NOSSRANGE
   TERM
 NOTEST
 NOTHREAD
   TRUNC(STD)
 NOVBREF
 NOWSCLEAR
 NOXREF
   YEARWINDOW(1900)
   ZWB
```

```
LineID  Message code  Message text    (4)
        IGYDS0139-W   Diagnostic messages were issued during processing of compiler options.  These messages are
                    located at the beginning of the listing.
   193  IGYDS1050-E   File "LOCATION-FILE" contained no data record descriptions.  The file definition was discarded.
   889  IGYPS2052-S   An error was found in the definition of file "LOCATION-FILE".  The reference to this file
                    was discarded.
                      Same message on line:    983
   993  IGYPS2121-S   "WS-DATE" was not defined as a data-name.  The statement was discarded.
                      Same message on line:    994
   995  IGYPS2121-S   "WS-TIME" was not defined as a data-name.  The statement was discarded.
                      Same message on line:    996
   997  IGYPS2053-S   An error was found in the definition of file "LOCATION-FILE".  This input/output statement
                    was discarded.
                      Same message on line:   1009
  1008  IGYPS2121-S   "LOC-CODE" was not defined as a data-name.  The statement was discarded.
  1219  IGYPS2121-S   "COMMUTER-SHIFT" was not defined as a data-name.  The statement was discarded.
                      Same message on line:   1240
  1220  IGYPS2121-S   "COMMUTER-HOME-CODE" was not defined as a data-name.  The statement was discarded.
                      Same message on line:   1241
  1222  IGYPS2121-S   "COMMUTER-NAME" was not defined as a data-name.  The statement was discarded.
                      Same message on line:   1243
  1223  IGYPS2121-S   "COMMUTER-INITIALS" was not defined as a data-name.  The statement was discarded.
                      Same message on line:   1244
```

```
  1233  IGYPS2121-S   "WS-NUMERIC-DATE" was not defined as a data-name.  The statement was discarded.
Messages    Total    Informational    Warning    Error   Severe   Terminating    (5)
Printed:       21                        2          1       18
  * Statistics for COBOL program SLISTING:    (6)
* Source records = 1765
* Data Division statements = 277
* Procedure Division statements = 513
Locale = en_US.ISO8859-1                  (7)
End of compilation 1,  program SLISTING,  highest severity: Severe.   (8)
Return code 12
```

(1)
PROCESS (または CBL) ステートメントで指定されたオプションに関するメッセージ。 このメッセージは、オプションが指定されていない場合には表示されません。

(2)
PROCESS (または CBL) ステートメントの中にコーディングされたオプション。

(3)
このコンパイルの開始時のオプションの状況。

(4)
プログラム診断メッセージ。 最初のメッセージは、ライブラリー・フェーズ診断 (ある場合) に言及しています。 ライブラリー・フェーズの診断は、常時リストの先頭に示されます。

(5)
このプログラムの診断メッセージのカウントで、重大度レベルによってグループ化されたもの。

(6)
プログラム SLISTING のプログラム統計。

(7)
コンパイラーが使用したロケール。

(8)
コンパイル単位のプログラム統計。 バッチ・コンパイルを実行する (1 回のコンパイルで複数の最外部 COBOL プログラムを実行する) 場合、戻りコードは、コンパイル全体について最高レベルのメッセージ重大度です。

例: SOURCE および NUMBER 出力

次に示されているリストの部分では、プログラマーは 2 つのステートメントに順序どおりでない番号を付けています。 リスト内の注釈番号は、番号が付いた後続の説明と対応しています。

```
                    (1)
 LineID  PL SL  ----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7-|--+----8  Cross-
Reference
   (2)    (3)    (4)                                                                              |
                 087000/*****************************************************************         |
                 087100***                 D O   M A I N   L O G I C                   **         |
                 087200***                                                             **         |
                 087300*** Initialization. Read and process update transactions until **         |
                 087400*** EOE. Close files and stop run.                              **         |
                 087500*****************************************************************         |
                 087600 procedure division.                                                       |
                 087700   000-do-main-logic.                                                      |
                 087800     display "PROGRAM SRCOUT - Beginning"                                  |
                 087900     perform 050-create-stl-main-file.                                     |
                 088150     display "perform 050-create-stl-main-file finished".                  |
 088151**        088125     perform 100-initialize-paragraph                                      |
                 088200     display "perform 100-initialize-paragraph finished"                   |
                 088300     read update-transaction-file into ws-transaction-record               |
                 088400         at end                                                            |
              1  088500       set transaction-eof to true                                         |
                 088600     end-read                                                              |
                 088700     display "READ completed"                                              |
                 088800     perform until transaction-eof                                         |
              1  088900       display "inside perform until loop"                                 |
              1  089000       perform 200-edit-update-transaction                                 |
              1  089100       display "After perform 200-edit   "                                 |
              1  089200       if no-errors                                                        |
              2  089300         perform 300-update-commuter-record                                |
              2  089400         display "After perform 300-update "                               |
              1  089650       else                                                                |
 089651**     2  089600         perform 400-print-transaction-errors                              |
              2  089700         display "After perform 400-errors "                               |
              1  089800       end-if                                                              |
              1  089900       perform 410-re-initialize-fields                                    |
              1  090000       display "After perform 410-reinitialize"                            |
              1  090100       read update-transaction-file into ws-transaction-record             |
```

```
            1  090200           at end                                                |
            2  090300         set transaction-eof to true                             |
            1  090400       end-read                                                  |
            1  090500       display "After '2nd READ'   "                             |
               090600     end-perform                                                 |
```

(1)
スケール行では、区域 A、区域 B、およびソース・コード桁番号にラベルを付けます。

(2)
コンパイラーが割り当てるソース・コード行番号。

(3)
プログラム (PL) とステートメント (SL) のネスト・レベル。

(4)
プログラムの第 1 から 6 桁 (シーケンス番号域)。

例: MAP 出力

次の例は、MAP オプションからの出力を示しています。 その下の説明で使用されている番号は、出力に付けられている番号と対応しています。

```
 Data Division Map
 (1)
 Data Definition Attribute codes (rightmost column) have the following meanings:
     D = Object of OCCURS DEPENDING    G = GLOBAL                           LSEQ= ORGANIZATION LINE
SEQUENTIAL
     E = EXTERNAL                      O = Has OCCURS clause                SEQ= ORGANIZATION SEQUENTIAL
     VLO=Variably Located Origin       OG= Group has own length definition  INDX= ORGANIZATION INDEXED
     VL= Variably Located              R = REDEFINES                        REL= ORGANIZATION RELATIVE

 (2)      (3) (4)                                          (5)      (6)         (7)           (8)
 Source   Hierarchy and                                                                     Data Def
 LineID   Data Name                                  Length(Displacement) Data Type      Attributes
      4  PROGRAM-ID IGYTCARA----------------------------------------------------------------------------
*
    180   FD COMMUTER-FILE                                                  File            INDX
    182   1  COMMUTER-RECORD                                 80             Group
    183     2  COMMUTER-KEY                                 16(0000000)    Display
    184     2  FILLER                                       64(0000016)    Display
    186   FD COMMUTER-FILE-MST                                              File            INDX
    188   1  COMMUTER-RECORD-MST                             80             Group
    189     2  COMMUTER-KEY-MST                             16(0000000)    Display
    190     2  FILLER                                       64(0000016)    Display
    192   FD LOCATION-FILE                                                  File            SEQ
    203   FD UPDATE-TRANSACTION-FILE                                        File            SEQ
    208   1  UPDATE-TRANSACTION-RECORD                       80             Display
    216   FD PRINT-FILE                                                     File            SEQ
    221   1  PRINT-RECORD                                   121            Display
    228   1  WORKING-STORAGE-FOR-IGYCARA                      1            Display
```

(1)
データ定義属性コードの説明。

(2)
データ項目が定義されたソース行番号。

(3)
レベル定義または番号。 コンパイラーは、次の方法でこの番号を生成します。

- 階層の第 1 レベルは常に 01 です。 レベル 02 から 49 としてコーディングした項目のレベルごとに 1 を加えます。
- レベル番号の 66、77、および 88、そして標識 FD と SD は変更されません。

(4)
ソース・モジュールでソース順序で使用されるデータ名。

(5)
データ項目の長さ。 ベース・ロケーター値。

(6)
収容構造の先頭からの 16 進変位。

(7)
データ型および使用法。

(8)
データ定義属性コード。 定義は DATA DIVISION マップの先頭で説明されています。

366 ページの『例: 組み込みマップ要約』
368 ページの『例: ネストされたプログラム・マップ』

関連参照
367 ページの『MAP 出力で使用される用語および記号』

例: 組み込みマップ要約

次の例は、MAP オプションによって作成される組み込みマップ要約を示しています。 この要約は、DATA DIVISION の、データ宣言を含む行のリストの右マージンに現れます。

```
 000002                Identification Division.                                          |
 000003                                                                                  |
 000004                Program-id.    EMBMAP.                                          |
     . . .                                                                              |
 000176                Data division.                                                    |
 000177                File section.                                                     |
 000178                                                                                  |
 000179                                                                                  |
 000180                FD  COMMUTER-FILE                                                 |
 000181                    record 80 characters.                                         |      (1)   (2)
 000182                  01 commuter-record.                                             |       80
 000183                    05 commuter-key                PIC x(16).                     |
16(0000000)
 000184                    05 filler                      PIC x(64).                     |
64(0000016)
     . . .                                                                              |
 000221         IA1620   01 print-record                 pic x(121).                    |      121
     . . .                                                                              |
 000227                Working-storage section.                                          |
 000228                  01 Working-storage-for-EMBMAP    pic x.                         |        1
 000229                                                                                  |
 000230                  77 comp-code                     pic S9999 comp.                |        2
 000231                  77 ws-type                       pic x(3)   value spaces.       |        3
 000232                                                                                  |
 000233                                                                                  |
 000234                  01 i-f-status-area.                                             |        2
 000235                    05 i-f-file-status             pic x(2).                      |
2(0000000)
 000236                       88 i-o-successful           value zeroes.                  |IMP
 000237                                                                                  |
 000238                                                                                  |
 000239                  01 status-area.                                                 |        8
 000240                    05 commuter-file-status        pic x(2).                      |(3)
2(0000000)
 000241                       88 i-o-okay                 value zeroes.                  |IMP
     . . .                                                                              |
 000246                                                                                  |
 000247                  77 update-file-status           pic xx.                        |        2
 000248                  77 loccode-file-status          pic xx.                        |        2
 000249                  77 updprint-file-status         pic xx.                        |        2
     . . .                                                                              |
 000877                procedure division.                                               |
 000878                  000-do-main-logic.                                              |
 000879                    display "PROGRAM EMBMAP - Beginning".                         |
 000880                    perform 050-create-stl-main-file.                           |931
     . . .                                                                              |
```

(1)
データ項目の長さ (10 進数)

(2)
ベース・ロケーター値の先頭からの 16 進変位。

(3)
特殊定義記号:

UND
ユーザー名が未定義です。

DUP
ユーザー名が 1 回を超えて定義されています。

IMP
暗黙的に定義された名前 (特殊レジスターや形象定数など)。

IFN
組み込み関数参照。

EXT
外部参照。

MAP 出力で使用される用語および記号

次の表は、MAP コンパイラー・オプションによって作成されるリストで使用される用語および記号を説明しています。

表 *37.* ***MAP*** **出力で使用される用語および記号**

用語	説明
ALPHABETIC	英字 (PICTURE A)
ALPHA-EDIT	英字編集
AN-EDIT	英数字編集
BINARY	バイナリー (USAGE BINARY、COMPUTATIONAL、または COMPUTATIONAL-5)
COMP-1	単精度内部浮動小数点 (USAGE COMPUTATIONAL-1)
COMP-2	倍精度内部浮動小数点 (USAGE COMPUTATIONAL-2)
DBCS	DBCS (USAGE DISPLAY-1)
DBCS-EDIT	DBCS 編集
DISP-FLOAT	表示浮動小数点 (USAGE DISPLAY)
DISPLAY	英数字 (PICTURE X)
DISP-NUM	ゾーン 10 進数 (USAGE DISPLAY)
DISP-NUM-EDIT	数字編集 (USAGE DISPLAY)
FD	ファイル定義
FUNCTION-PTR	外部呼び出し可能関数を指すポインター (USAGE FUNCTION-POINTER)
GROUP	英数字固定長グループ
GRP-VARLEN	英数字可変長グループ
INDEX	指標 (USAGE INDEX)
INDEX-NAME	指標名
NATIONAL	カテゴリー国別 (USAGE NATIONAL)
NAT-EDIT	国別編集 (USAGE NATIONAL)
NAT-FLOAT	国別浮動小数点 (USAGE NATIONAL)

表 37. ***MAP*** **出力で使用される用語および記号** (続き)	
用語	**説明**
NAT-GROUP	国別グループ (GROUP-USAGE NATIONAL)
NAT-GRP-VARLEN	国別可変長グループ (GROUP-USAGE NATIONAL)
NAT-NUM	国別 10 進数 (USAGE NATIONAL)
NAT-NUM-EDIT	国別数字編集 (USAGE NATIONAL)
PACKED-DEC	内部 10 進数 (USAGE PACKED-DECIMAL または COMPUTATIONAL-3)
POINTER	ポインター (USAGE POINTER)
PROCEDURE-PTR	外部呼び出し可能プログラムを指すポインター (USAGE PROCEDURE-POINTER)
SD	ソート・ファイル定義
01 から 49、77	データ記述に関するレベル番号
66	RENAMES に関するレベル番号
88	条件名に関するレベル番号

例: ネストされたプログラム・マップ

この例は、MAP コンパイラー・オプションを指定することによって作成される、ネストされた プロシージャーのマップを示しています。 括弧内の番号は、後続の注釈に対応しています。

```
Nested Program Map

    (1)
Program Attribute codes (rightmost column) have the following meanings:
   C = COMMON
   I = INITIAL
   U = PROCEDURE DIVISION USING...

   (2)   (3)      (4)                                        (5)
Source Nesting                                             Program
LineID Level   Program Name from PROGRAM-ID paragraph      Attributes
     2         NESTED. . . . . . . . . . . . . . . . . .
    12     1     X1. . . . . . . . . . . . . . . . . . .
    20     2       X11 . . . . . . . . . . . . . . . . .
    27     2       X12 . . . . . . . . . . . . . . . . .
    35     1     X2. . . . . . . . . . . . . . . . . . .
```

(1)
プログラム属性コードの説明

(2)
プログラムが定義されたソース行番号

(3)
プログラムのネストの深さ

(4)
プログラム名

(5)
プログラム属性コード

例: XREF 出力: データ名相互参照

次の例は、XREF コンパイラー・オプションによって作成される、データ名のソート済み相互参照を示しています。 括弧内の番号は、後続の注釈に対応しています。

```
An "M" preceding a data-name reference indicates that the
data-name is modified by this reference.

     (1)        (2)                              (3)
  Defined   Cross-reference of data-names   References

      265   ABEND-ITEM1
      266   ABEND-ITEM2
      347   ADD-CODE . . . . . .  1102 1162
      381   ADDRESS-ERROR. . . . . .  M1126
      280   AREA-CODE. . . . . .  1236 1261 1324 1345
      382   CITY-ERROR . . . . . .  M1129

        (4)
 Context usage is indicated by the letter preceding a procedure-name
 reference. These letters and their meanings are:
     A = ALTER (procedure-name)
     D = GO TO (procedure-name) DEPENDING ON
     E = End of range of (PERFORM) through (procedure-name)
     G = GO TO (procedure-name)
     P = PERFORM (procedure-name)
     T = (ALTER) TO PROCEED TO (procedure-name)
     U = USE FOR DEBUGGING (procedure-name)

     (5)       (6)                              (7)
  Defined   Cross-reference of procedures   References

      877   000-DO-MAIN-LOGIC
      930   050-CREATE-STL-MAIN-FILE . . .  P879
      982   100-INITIALIZE-PARAGRAPH . . .  P880
     1441   1100-PRINT-I-F-HEADINGS. . . .  P915
     1481   1200-PRINT-I-F-DATA. . . . . .  P916
     1543   1210-GET-MILES-TIME. . . . . .  P1510
     1636   1220-STORE-MILES-TIME. . . . .  P1511
     1652   1230-PRINT-SUB-I-F-DATA. . . .  P1532
     1676   1240-COMPUTE-SUMMARY . . . . .  P1533
     1050   200-EDIT-UPDATE-TRANSACTION. .  P886
     1124   210-EDIT-THE-REST. . . . . .  P1116
     1159   300-UPDATE-COMMUTER-RECORD . .  P888
     1207   310-FORMAT-COMMUTER-RECORD . .  P1164 P1179
     1258   320-PRINT-COMMUTER-RECORD. . .  P1165 P1176 P1182 P1192
     1288   330-PRINT-REPORT . . . . . .  P1178 P1202 P1256 P1280 P1340 P1365 P1369
     1312   400-PRINT-TRANSACTION-ERRORS .  P890
```

データ名の相互参照:

(1)
その名前が定義されている行番号。

(2)
データ名。

(3)
その名前が使用されている行番号。M が行番号の前に置かれている場合は、データ項目がその位置で明示的に変更されたことを意味します。

プロシージャー参照の相互参照:

(4)
プロシージャー参照のコンテキスト取扱コードの説明。

(5)
そのプロシージャー名が定義されている行番号。

(6)
プロシージャー名。

(7)
そのプロシージャーが参照されている行番号およびそのプロシージャーのコンテキスト取扱コード。

370 ページの『例: XREF 出力: プログラム名相互参照』
370 ページの『例: XREF 出力: COPY/BASIS 相互参照』
371 ページの『例: XREF 出力: 組み込み相互参照』

例: XREF 出力: プログラム名相互参照

次の例は、XREF コンパイラー・オプションによって作成される、プログラム名のソート済み相互参照を示しています。 括弧内の番号は、後続の注釈に対応しています。

```
   (1)       (2)                          (3)
Defined   Cross-reference of programs     References

EXTERNAL   EXTERNAL1. . . . . . . . . . . 25
      2   X. . . . . . . . . . . . . . .  41
     12   X1 . . . . . . . . . . . . . .  33 7
     20   X11. . . . . . . . . . . . . .  25 16
     27   X12. . . . . . . . . . . . . .  32 17
     35   X2 . . . . . . . . . . . . . .  40 8
```

(1)
そのプログラム名が定義されている行番号。プログラムが外部の場合は、定義行番号の代わりに EXTERNAL という語が表示されます。

(2)
プログラム名。

(3)
そのプログラムが参照されている行番号。

例: XREF 出力: COPY/BASIS 相互参照

次の例は、 の XREF コンパイラー・オプションで 生成された関連コピーブックのライブラリー名およびファイル名に対する コピーブックのソート済み相互参照を示しています。 括弧内の番号は、後続の注釈に対応しています。

```
        COPY/BASIS cross-reference of text-names, library names and file names

   (1)                          (1)                   (2)
Text-name                    Library-name            File
name
                       (3)                       (3)  (4)
"realxrealyrealzlongxlo>    'thisislongdirecto>    <toryname/
realxrealyrealzlongxname.cpy
"realxrealyrealzlongxlo>    SYSLIB (default)    (5)./cbldir1/
realxrealyrealzlongxname.cpy
"copyA.cpy"                  SYSLIB (default)       ./cbldir1/copyA.cpy
'./copydir2/copyM.cbl'       SYSLIB (default)       ./copydir2/copyM.cbl
'/copyB.cpy'                 SYSLIB                 ./cbldir1/copyB.cpy
'/copydir/copyM.cbl'         SYSLIB                 ./cbldir1/copydir/copyM.cbl
'cbldir1/copyC.cpy'          ALTDD2                 ./cbldir1/copyC.cpy
'copydir/copyM.cbl'          SYSLIB                 ./cbldir1/copydir/copyM.cbl
'copydir2/copyM.cbl'         SYSLIB (default)       ./copydir2/copyM.cbl
'copydir3/stuff.cpy'         ALTDD2                 ./copydir3/stuff.cpy
'stuff.cpy'                  ALTDD                  ./copydir3/stuff.cpy
OTHERDD                      ALTDD2                 .//cbldir1/other.cob
REALXLONGXLONGYNAMEX         SYSLIB (default)       ./REALXLONGXLONGYNAMEX
. . .

  (5)
./ = /afs/stllp.sanjose.ibm.com/usr1/cobdev/tmross/stuff/subdir
Note: Some names were truncated.   > = truncated on right   < = truncated on left
```

(1)
ソースの COPY ステートメントからのものです。例えば、上の相互参照の 5 番目の項目に対応する COPY ステートメントは次のようなものです。

```
COPY '/copyB.cpy' Of SYSLIB
```

(2)
COPY メンバーをコピーした元のファイルの完全修飾パス

(3)
長いテキスト名またはライブラリー名の右側の切り捨ては、大なり記号 (>) で示されます。

(4)
長いファイル名の左側の切り捨ては、小なり記号 (<) で示されます。

(5)
相互参照では、ファイル名の現行作業ディレクトリー部分は **./** で表されます。 相互参照の下に、現行作業ディレクトリー名を展開した完全なディレクトリー名が示されます。

関連参照

290 ページの『XREF』

例: XREF 出力: 組み込み相互参照

次の例は、ソース・リストに組み込まれる変更済み相互参照を示しています。 この相互参照は XREF コンパイラー・オプションによって作成されます。

```
LineID  PL SL  ----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7-|--+----8  Map and Cross Reference
  . . .                                                                                 |
000878                procedure division.                                               |
000879                  000-do-main-logic.                                              |
000880                    display "PROGRAM IGYTCARA - Beginning".                       |
000881                    perform 050-create-stl-main-file.                             | 932 (1)
000882                    perform 100-initialize-paragraph.                             | 984
000883                    read update-transaction-file into ws-transaction-record       | 204 340
000884                        at end                                                    |
000885      1                    set transaction-eof to true                          | 254
000886                    end-read.                                                     |
  . . .                                                                               |
000984                  100-initialize-paragraph.                                     |
000985                    move spaces to ws-transaction-record                        | IMP 340 (2)
000986                    move spaces to ws-commuter-record                           | IMP 316
000987                    move zeroes to commuter-zipcode                             | IMP 327
000988                    move zeroes to commuter-home-phone                          | IMP 328
000989                    move zeroes to commuter-work-phone                          | IMP 329
000990                    move zeroes to commuter-update-date                         | IMP 333
000991                    open input update-transaction-file                          | 204
000992                        location-file                                           | 193
000993                        i-o commuter-file                                       | 181
000994                        output print-file                                       | 217
  . . .                                                                             |
001442                1100-print-i-f-headings.                                      |
001443                                                                              |
001444                    open output print-file.                                   | 217
001445                                                                              |
001446                    move function when-compiled to when-comp.                 | IFN 698 (2)
001447                    move when-comp (5:2) to compile-month.                    | 698 640
001448                    move when-comp (7:2) to compile-day.                      | 698 642
001449                    move when-comp (3:2) to compile-year.                     | 698 644
001450                                                                              |
001451                    move function current-date (5:2) to current-month.        | IFN 649
001452                    move function current-date (7:2) to current-day.          | IFN 651
001453                    move function current-date (3:2) to current-year.         | IFN 653
001454                                                                              |
001455                    write print-record from i-f-header-line-1                 | 222 635
001456                          after new-page.                                     | 138
  . . .                                                                             |
```

(1)
プログラム内のデータ名またはプロシージャー名の定義の行番号。

(2)
特殊定義記号:

UND
ユーザー名が未定義です。

DUP
ユーザー名が 1 回を超えて定義されています。

IMP
暗黙的に定義された名前 (特殊レジスターや形象定数など)。

IFN
組み込み関数参照。

EXT
外部参照。

*****
NOCOMPILE オプションが有効なため、プログラム名が未解決です。

例: VBREF コンパイラー出力

次の例は、プログラム内のすべてのステートメントのアルファベット順のリストと、各動詞が参照されている場所を示しています。 このリストは、VBREF コンパイラー・オプションによって作成されます。

```
 (1)     (2)                 (3)
2        ACCEPT . . . . . .  1010 1012
2        ADD. . . . . . . .  1290 1306
1        CALL . . . . . . .  1406
5        CLOSE. . . . . . .  898 945 970 1526 1535
20       COMPUTE. . . . . .  1506 1640 1644 1657 1660 1663 1664 1665 1678 1682 1686 1691 1696 1701 1709 1713
                             1718 1723 1728 1733
2        CONTINUE . . . . .  1062 1069
2        DELETE . . . . . .  964 1193
48       DISPLAY. . . . . .  878 906 917 918 919 933 940 942 947 953 960 966 972 996 997 998 999 1003 1006 1037
                             1090 1168 1171 1185 1195 1387 1388 1389 1390 1391 1392 1393 1401 1402 1403 1404
                             1405 1433 1485 1486 1492 1497 1498 1520 1521 1528 1529 1624
2        EVALUATE . . . . .  1161 1557
47       IF . . . . . . . .  887 905 932 939 946 952 959 965 971 993 1002 1036 1051 1054 1071 1074 1077 1089
                             1102 1111 1115 1125 1128 1131 1134 1137 1141 1145 1148 1151 1167 1184 1194 1240
                             1247 1265 1272 1289 1321 1330 1339 1351 1361 1484 1496 1519 1527
183      MOVE . . . . . . .  907 937 957 983 984 985 986 987 988 1004 1011 1013 1025 1038 1052 1055 1060 1067
                             1072 1075 1078 1079 1080 1081 1082 1083 1091 1103 1112 1126 1129 1132 1135 1139
                             1143 1146 1149 1152 1160 1163 1169 1175 1177 1180 1181 1186 1191 1196 1201 1208
                             1209 1210 1211 1212 1213 1214 1215 1216 1217 1218 1219 1220 1221 1222 1229 1230
                             1231 1232 1233 1234 1235 1239 1241 1244 1248 1250 1251 1253 1254 1255 1257 1258
                             1259 1260 1264 1266 1269 1273 1275 1276 1278 1279 1291 1294 1299 1301 1303 1307
                             1313 1314 1315 1316 1317 1318 1319 1320 1322 1323 1327 1328 1331 1333 1334 1336
                             1338 1341 1342 1343 1344 1348 1349 1352 1354 1355 1357 1362 1364 1368 1374 1375
                             1376 1377 1378 1379 1380 1381 1414 1417 1422 1425 1445 1446 1447 1448 1450 1451
                             1452 1457 1464 1489 1502 1507 1508 1509 1517 1551 1561 1566 1571 1576 1581 1586
                             1591 1596 1601 1606 1611 1616 1621 1626 1627 1679 1683 1688 1693 1698 1703 1710
                             1715 1720 1725 1730 1735
5        OPEN . . . . . . .  931 951 989 1443 1483
62       PERFORM. . . . . .  879 880 885 886 888 890 892 908 909 915 916 934 935 941 943 948 949 954 955 961
                             962 967 968 973 974 1000 1005 1008 1023 1039 1092 1093 1116 1164 1165 1170 1172
                             1176 1178 1179 1182 1187 1188 1192 1197 1198 1202 1246 1256 1271 1280 1329 1340
                             1350 1359 1365 1369 1504 1510 1511 1532 1533
8        READ . . . . . . .  881 893 958 1014 1026 1085 1490 1514
1        REWRITE. . . . . .  1183
4        SEARCH . . . . . .  1058 1065 1413 1421
46       SET. . . . . . . .  883 895 1016 1028 1041 1057 1064 1084 1087 1363 1412 1420 1493 1499 1516 1522 1548
                             1550 1559 1560 1564 1565 1569 1570 1574 1575 1579 1580 1584 1585 1589 1590 1594
                             1595 1599 1600 1604 1605 1609 1610 1614 1615 1619 1620 1639 1643
2        STOP . . . . . . .  920 1434
4        STRING . . . . . .  1236 1261 1324 1345
33       WRITE. . . . . . .  938 1166 1292 1293 1295 1296 1297 1298 1300 1302 1305 1454 1459 1462 1465 1467 1469
                             1471 1512 1654 1655 1667 1668 1669 1740 1742 1744 1745 1746 1747 1748 1749 1750
```

(1)
そのステートメントがプログラムで使用されている回数。

(2)
ステートメント

(3)
そのステートメントが使用されている行番号。

オフセット情報を含むメッセージによるデバッグ

いくつかの IWZ メッセージには、失敗したプログラムの特定の行の識別に使用できるオフセット情報が含まれています。

この情報を使用するには、次のようにします。

1. LIST オプションを使用してプログラムをコンパイルします。 次のステップにより、接尾部が .wlist のアセンブラー・リスト・ファイルが作成されます。
2. トレースバックを含むメッセージを取得する場合は、COBOL プログラムに対するオフセット情報を検索します。次の例では、プログラムは TEST で、対応する 16 進オフセットは 0x678 です (太字で強調表示されています)。

```
Traceback:
/opt/ibm/cobol/rte/usr/lib/libcob2_32r.so(+0x66b60)[0xf76f3b60]
/opt/ibm/cobol/rte/usr/lib/libcob2_32r.so(iwzWriteERRmsg+0x17)[0xf76f41f7]
/opt/ibm/cobol/rte/usr/lib/libcob2_32r.so(_iwzcBCD_CONV_Pckd_To_Int4+0x176)[0xf76be7b6]
test.out(TEST+0x678)[0x5655a844]
/lib/libc.so.6(__libc_start_main+0xf3)[0xf74b12d3]
    --- End of call chain ---
IWZ903S  The system detected a data exception.
IWZ901S  Program exits due to severe or critical error.
```

3. .wlist ファイル内で 16 進オフセットを確認します。 16 進オフセットの右側、命令バイトの後は、COBOL ソース行番号です。次の例では、0x678 に対応する行番号は 174 です (太字で強調表示されています)。

```
000174:            COMPUTE x = FUNCTION FACTORIAL(Packed-Dec-05).
    000669  EC83 6A08               000174        sub   esp, 0x00000008
    00066C  046A                    000174        push  0x00000004
    00066E  0068 0000 E800          000174        push  OFFSET FLAT:LEVEL-01-PACKED-DECIMAL
    000673  00E8 0000 8300          000174        call  OFFSET
FLAT:_iwzcBCD_CONV_Pckd_To_Int4
    000678  C483 8910               000174        add   esp, 0x00000010
```

4. プログラム・リスト内でこのステートメントを確認します。

関連参照
270 ページの『LIST』

アセンブラー・ルーチンのデバッグ

アセンブラー・ルーチンをデバッグするには、「Disassembly (分解)」ビューを使用します。 アセンブラー・ルーチンにはデバッグ情報がないため、デバッガーは自動的にこのビューに移動します。

「Disassembly (分解)」ビュー内で、逆アセンブルされたステートメントにブレークポイントを設定するには、接頭部域内をダブルクリックします。 デフォルトでは、デバッガーは開始時に、最初のデバッグ可能ステートメントを検出するまで実行します。 (デバッグ可能かどうかを問わず) アプリケーション内の最初の命令でデバッガーを停止させるには、 -i オプションを使用する必要があります。 次に例を示します。

```
irmtdbgc –i -qhost=myhost progname
```

第 4 部 特定の環境に合わせた COBOL プログラムの目標

© Copyright IBM Corp. 2021

第 17 章 Db2 環境用のプログラミング

一般に、COBOL プログラムのコーディングは、プログラムから Db2 データベースにアクセスするかどうかに関係なく同じになります。しかし、Db2 データの検索、更新、挿入、および削除を行い、さらにその他の Db2 サービスを使用するためには、SQL ステートメントを使用しなければなりません。

Db2 と通信するには、以下のステップを実行します。

- EXEC SQL および END-EXEC ステートメントで区切って、必要なすべての SQL ステートメントをコーディングします。
- Db2 を開始します (まだ開始されていない場合)。
- SQL コンパイラー・オプションを使用してコンパイルします。
- NODYNAM コンパイラー・オプションを使用してコンパイルします (アプリケーションが Db2 独立型プリコンパイラーを使用してコンパイルされている場合)。

  注: NODYNAM コンパイラー・オプションは、EXEC CICS ステートメントまたは EXEC SQL ステートメントを含むプログラムに必要です。

  COBOL 動的呼び出しによってロードされる COBOL ライブラリーで EXEC SQL ステートメントを使用する場合は、1 つ以上の EXEC SQL ステートメントがメインプログラムにはいっていなければなりません。(呼び出し元の Db2 API は COBOL 動的呼び出しではロードできません。)

関連概念
377 ページの『Db2 コプロセッサー』

関連タスク
143 ページの『Db2 ファイルの使用』

378 ページの『SQL ステートメントのコーディング』
379 ページの『コンパイル前の Db2 の起動』
379 ページの『SQL オプションを使用したコンパイル』

関連参照

264 ページの『DYNAM』
Db2 V11.1 for Linux, UNIX, and Windows の SQL 解説書

Db2 コプロセッサー

Db2 コプロセッサーを使用すると、コンパイラーは、別個のプリコンパイラーを使用することなく、組み込み SQL ステートメントを含むソース・プログラムを処理します。

Db2 コプロセッサーを使用するには、SQL コンパイラー・オプションを指定してください。

コンパイラーは、ソース・プログラムで SQL ステートメントを検出すると、Db2 コプロセッサーとインターフェースします。EXEC SQL ステートメントと END-EXEC ステートメントの間のテキストはすべてコプロセッサーに渡されます。 コプロセッサーが、SQL ステートメントに対して適切なアクションを取り、それらのために生成する固有 COBOL ステートメントをコンパイラーに指示します。

Db2 プリコンパイラーを使用するときに適用される COBOL 言語の使用に関する制約事項は、Db2 コプロセッサーを使用するときには適用されません。

- SQL ステートメントで使用されているホスト変数を識別するのに EXEC SQL BEGIN DECLARE SECTION および EXEC SQL END DECLARE SECTION ステートメントを使用する必要はありません。
- 複数のネストなし COBOL プログラムが入っているソース・ファイルをバッチでコンパイルできます。
- ソース・プログラムはネストされたプログラムを含むことができます。

© Copyright IBM Corp. 2021

- 拡張ソース形式 (Extended Source) は完全にサポートされます。

関連タスク

379 ページの『SQL オプションを使用したコンパイル』

関連参照

282 ページの『SQL』
283 ページの『SRCFORMAT』

SQL ステートメントのコーディング

SQL ステートメントは、EXEC SQL と END-EXEC で区切らなければなりません。 EXEC SQL および END-EXEC 区切り文字はそれぞれ 1 行の中で完結している必要があります。 複数行にわたって継続させることはできません。 EXEC SQL ステートメント内に COBOL ステートメントをコーディングしないでください。

さらに、以下の特別なステップを実行する必要があります。

- EXEC SQL INCLUDE ステートメントをコーディングして、最外部プログラムの WORKING-STORAGE SECTION または LOCAL-STORAGE SECTION に SQL 通信域 (SQLCA) を組み込んでください。 再帰的プログラムや の場合には、LOCAL-STORAGE をお勧めします。
- SQL ステートメントで使用するすべてのホスト変数を WORKING-STORAGE SECTION、LOCAL-STORAGE SECTION、または LINKAGE SECTION に定義する。ただし、EXEC SQL BEGIN DECLARE SECTION および EXEC SQL END DECLARE SECTION を指定する必要はありません。

ラージ・オブジェクト (BLOB や CLOB など) および複合 SQL の場合でも SQL ステートメントを使用できます。

関連タスク
145 ページの『同一プログラム内での Db2 ファイルと SQL ステートメントの使用』
378 ページの『Db2 コプロセッサーを用いた SQL INCLUDE の使用』
379 ページの『SQL ステートメントでのバイナリー項目の使用』
379 ページの『SQL ステートメントの成否の判断』

Db2 コプロセッサーを用いた SQL INCLUDE の使用

SQL コンパイラー・オプションを使用すると、SQL INCLUDE ステートメントは、ネイティブの COBOL COPY ステートメント (使用される検索パスおよびファイル・サフィックスを含む) とまったく同様に扱われます。

したがって、次の 2 行は同様に扱われます。 (EXEC SQL INCLUDE ステートメントの終了を示すピリオドが必要です。)

```
EXEC SQL INCLUDE name END-EXEC.
COPY name.
```

SQL INCLUDE ステートメント内の *name* は、COPY *text-name* と同じ規則に従い、REPLACING 句を持たない COPY *text-name* ステートメントとまったく同様に処理されます。

COBOL は、SQL INCLUDE 処理に対して Db2 環境変数 DB2INCLUDE を使用しません。SQL INCLUDE 処理に DB2INCLUDE 環境変数を使用する場合は、ホーム・ディレクトリーまたは Linux コマンド・シェルのプロンプトで、この環境変数を、.profile ファイルに入っている COBOL SYSLIB 環境変数の設定と連結することができます。例えば、次のように指定します。

```
export SYSLIB=$DB2INCLUDE:$SYSLIB
```

関連参照
293 ページの『第 14 章 コンパイラー指示ステートメント』
COPY ステートメント (*COBOL for Linux on x86* 言語解説書)

SQL ステートメントでのバイナリー項目の使用

EXEC SQL ステートメントで指定するバイナリー・データ項目の場合は、USAGE COMP-5 としてか、USAGE BINARY、COMP、または COMP-4 として定義することができます。

バイナリー・データ項目を USAGE BINARY、COMP、または COMP-4 として定義する場合は、TRUNC(BIN) オプションを使用します。(この技法は、個々のデータ項目で USAGE COMP-5 を使用するよりも、パフォーマンスへの効果が大きくなることがあります。) 代わりに、TRUNC(OPT) または TRUNC(STD) が有効な場合は、コンパイラーはその項目を受け入れますが、10 進数切り捨て規則のため、そのデータは無効なことがあります。 切り捨てがデータの妥当性に影響を与えないようにしなければなりません。

関連概念
39 ページの『数値データの形式』

関連参照

286 ページの『TRUNC』

SQL ステートメントの成否の判断

Db2 が SQL ステートメントの実行を終了すると、Db2 は戻りコードを SQLCA 構造体の SQLCODE フィールドおよび SQLSTATE フィールドに入れて送り、 操作が成功したか失敗したかを示します。 プログラムでは、戻りコードをテストし、必要なアクションがあればそれを実行します。

関連参照
SQL 連絡域 (SQLCA) 構造
SQLCODE、SQLSTATE、および SQLWARN 情報

コンパイル前の Db2 の起動

コンパイラーは Db2 コプロセッサーと連動するため、プログラムをコンパイルする前に Db2 を起動する必要があります。

コンパイルを行うためにターゲット・データベースに接続する場合は、コンパイルを開始する前に接続するか、 またはコンパイラーに接続を実行させることができます。 コンパイラーに接続を実行させるには、次のいずれかの方法でデータベースを指定します。

- SQL オプション内に DATABASE サブオプションを使用する。
- DB2DBDFT 環境変数でデータベース名を指定する。

SQL オプションを使用したコンパイル

SQL コンパイラー・オプションで指定されたオプション・ストリングは、Db2 コプロセッサーで使用可能になります。ストリングの内容を表示するのは Db2 コプロセッサーだけです。

例えば、次の cob2 コマンドは、データベース名 SAMPLE と Db2 オプション USER および USING をコプロセッサーに渡します。

```
cob2 -q"sql('database sample user myname using mypassword')" mysql.cbl. . .
```

次のオプションは、Db2 プリコンパイラーにとっては意味があるものであり、Db2 プリコンパイラーによって使用されていましたが、 コプロセッサーでは無視されます。

- MESSAGES

- NOLINEMACRO
- OPTLEVEL
- OUTPUT
- SQLCA
- TARGET
- WCHARTYPE

関連タスク
380 ページの『Db2 サブオプションの分離』
380 ページの『パッケージ名およびバインド・ファイル名の使用』

関連参照
282 ページの『SQL』
DB2 コマンド解説書 (プリコンパイル)

Db2 サブオプションの分離

複数の SQL オプション指定が連結されているので、(1 つの CBL ステートメントに収まらない可能性がある) 別々の Db2 サブオプションを複数の CBL ステートメントに分離できます。

サブオプション・ストリングに組み込まれるオプションは累積されます。 コンパイラーは、複数のソースからこれらのサブオプションを、指定された順に連結します。 例えば、ソース・ファイル mypgm.cbl に以下のコードが含まれているとします。

```
cbl . . . SQL("string2") . . .
cbl . . . SQL("string3") . . .
```

コマンド cob2 mypgm.cbl -q:"SQL('string1')" を発行すると、コンパイラーは次のサブオプション・ストリングを Db2 コプロセッサーに渡します。

```
"string1 string2 string3"
```

連結ストリングはシングル・スペースで区切られます。 コンパイラーが同じ SQL サブオプションの複数インスタンスを検出した場合は、連結ストリングの中の最後のサブオプション指定が有効になります。 コンパイラーは、連結 Db2 サブオプション・ストリングの長さを 4 KB に限定しています。

パッケージ名およびバインド・ファイル名の使用

SQL オプションで指定できるサブオプションは、package name と bind file name の 2 つです。 これらの名前を指定しない場合は、ソース・ファイル名 (非バッチ・コンパイルの場合) または最初のプログラム (バッチ・コンパイルの場合) に基づいてデフォルト名が作成されます。

バッチ・コンパイルの後続のネストなしプログラムについては、各プログラムの PROGRAM-ID に基づいて名前が設定されます。

パッケージ名の場合は、ベース名 (ソース・ファイル名または PROGRAM-ID) が次のように変更されます。

- 8 文字を超える名前は 8 文字に切り詰められる。
- 小文字は大文字に変換される。
- A から Z、0 から 9、または _ (下線) 以外の文字はすべて 0 に変更される。
- 先頭文字が英字でない場合は A に変換される。

したがって、ベース名が 9123aB-cd の場合、パッケージ名は A123AB0C となります。

バインド・ファイル名の場合は、ベース名に接尾部 .bnd が追加されます。 明示的に指定しない限り、ファイル名は現行ディレクトリーと相対します。

第 18 章 COBOL プログラムの開発 (CICS の場合)

CICS アプリケーションは COBOL で記述することができ、CICS TXSeries または CICS TX を使用して Linux ワークステーション上で実行することができます。

CICS で実行する COBOL アプリケーションを作成するには、以下の手順を実行します。

1. CICS 管理者が領域の環境ファイルを変更して、ランタイム・ディレクトリーを含むように環境変数 COBPATH、および LD_LIBRARY_PATH を設定してあることを確認します。また、ランタイム・ディレクトリーに対するアクセス権が CICS 領域に付与されていることを確認します。

   環境ファイルは `/var/cics_regions/`*`xxxxxxxx`*`/environment` です (*xxxxxxxx* は領域の名前です)。
2. エディターを使用して次のタスクを実行し、アプリケーションを作成します。
   - COBOL ステートメントと CICS コマンドを使用して、プログラムをコーディングします。
   - COBOL コピーブックを作成します。
   - プログラムが使用する CICS 画面マップを作成します。
3. コマンド `cicsmap` を使用して、画面マップを処理します。
4. `cicstcl` コマンドを使用して、組み込みの CICS 変換プログラムと連動して CICS コマンドを変換し、プログラムをコンパイルしてリンクします。ファイル拡張子を指定しないと、`cicstcl` ではデフォルトで COBOL ソース・ファイル拡張子 `.cbl` が使用されます。

   以下の例は、`cicstcl` コマンドを使用して、 TXSeries または CICS TX で実行されるサンプル COBOL プログラム APPLCOB の変換、コンパイル、リンクを行う方法を示しています。`cicstcl` コマンドは、拡張子が `.ibmcob` の出力モジュールを生成します。

   - CICS COBOL アプリケーションのコンパイル、変換、リンク・エディットを行うには、`-l IBMCOB` オプションを使用して、ソース言語を IBM COBOL として指定します。CICS COBOL プログラム・ファイルの拡張子は `.ccp` または `.cbl` にすることができます。

     ```
     cicstcl -l IBMCOB APPLCOB.ccp
     ```

     ```
     cicstcl -l IBMCOB APPLCOB.cbl
     ```
   - CEDF を使用したデバッグに使用される CICS COBOL アプリケーションのコンパイル、変換、リンク・エディットを行うには、`-e` オプションを使用します。CICS ステートメントの行番号を表示するには、`-d` オプションを使用します。

     ```
     cicstcl -e -l IBMCOB APPLCOB.cbl
     ```

     ```
     cicstcl -e -d -l IBMCOB APPLCOB.cbl
     ```
   - デバッグに使用される CICS COBOL アプリケーションのコンパイル、変換、リンク・エディットを行うには、以下のように `-a` オプションを使用します。

     ```
     cicstcl -a -l IBMCOB APPLCOB.cbl
     ```
   - COPYBOOK パスを指定して CICS COBOL アプリケーションのコンパイル、変換、リンク・エディットを行うには、以下のように、SYSLIB 環境変数を設定して COBOL ソースの COPYBOOK パスのディレクトリーを指定してから、`cicstcl` コマンドを使用します。

     ```
     export SYSLIB="/program_copybook_path"
     ```

     ```
     cicstcl -l IBMCOB APPLCOB.cbl
     ```

© Copyright IBM Corp. 2021

- COBOL モジュールを静的リンクして CICS COBOL アプリケーションのコンパイル、変換、リンク・エディットを行うには、以下のように、USERLIB 環境変数を設定してコンパイル済みのオブジェクト・モジュールのパスを指定してから、cicstcl コマンドを使用します。

```
export USERLIB="cobol_module.o"
```

```
cicstcl -l IBMCOB APPLCOB.cbl
```

注: cicstcl コマンドを使用せずに CICS COBOL プログラムのコンパイルとリンクを行う場合は、-qcics オプションを指定して cob2 コンパイラー・コマンドを使用します。以下の例は、cob2 コマンドを使用して CICS COBOL アプリケーションのコンパイルとリンクを行う方法を示しています。

```
cob2 -qNOTHREAD -I/opt/ibm/cics/include -qcics -o APPLCOB.ibmcob APPLCOB.cbl -L/opt/ibm/
cics/lib -lcicsprIBMCOB
```

cicstcl コマンドの使用法について詳しくは、TXSeriesfor Multiplatforms 資料の「cicstcl」か、CICSTX on Cloud 資料の「cicstcl」を参照してください。

5. CICS 領域に対するアプリケーションのリソース (トランザクション、アプリケーション・プログラム、ファイルなど) を定義します。

   これらのアクションを実行するには、CICS 管理者権限が必要です。
6. cicsterm コマンドを使用するなどして、CICS 領域にアクセスします。
7. アプリケーションと関連付けられた 4 文字のトランザクション ID を入力して、アプリケーションを実行します。

関連概念
387 ページの『組み込みの CICS 変換プログラム』

関連タスク
382 ページの『CICS のもとで実行する COBOL プログラムのコーディング』
386 ページの『CICS プログラムのコンパイルおよび実行』
388 ページの『CICS プログラムのデバッグ』
TXSeries for Multiplatforms の資料
IBM CICS TX on Cloud の資料

CICS のもとで実行する COBOL プログラムのコーディング

CICS 下で実行されるようにプログラムをコーディングするには、EXEC CICS コマンド・フォーマットを使用して CICS コマンドを PROCEDURE DIVISION 内にコーディングしてください。

```
EXEC CICS command-name command-options
END-EXEC
```

CICS コマンドの基本形式は上に示したようなものです。EXEC コマンド内では、スペースをワード分離文字として使用してください。 コンマやセミコロンは使用しないでください。 EXEC CICS コマンド内に COBOL ステートメントをコーディングしないでください。

一般に、CICS 環境では COBOL 言語がサポートされています。ただし、CICS TXSeries または CICS TX で実行する COBOL プログラムをコーディングする際には、注意すべき制約事項と考慮事項がいくつかあります。

制約事項:

- CICS TXSeries または CICS TX と相互協調処理を行う Db2 ファイルは、FILEMODE(SMALL) を有効にして作成される必要があります。
- オブジェクト指向プログラミング、および Java とのインターオペラビリティーはサポートされていません。COBOL クラス定義およびメソッドは、CICS 環境では実行できません。
- ソース・プログラムには、ネスト済みプログラムを含めてはなりません。
- CICS TXSeries または CICS TX で実行される COBOL プログラムは 32 ビットでなければなりません。

変数名として EXEC、CICS、END-EXEC を 使用しないでください。 また、メインプログラムに対してユーザー指定のパラメーターを使用しないでください。 さらに、次の COBOL 言語エレメントのいずれも使用しないことをお勧めします。

- ENVIRONMENT DIVISION の FILE-CONTROL 記入項目
- DATA DIVISION の FILE SECTION
- USE 宣言部分 (USE FOR DEBUGGING を除く)

また、CICS 環境では、次の COBOL ステートメントを使用することもお勧めできません。

- ACCEPT format 1
- CLOSE
- DELETE
- DISPLAY UPON CONSOLE、DISPLAY UPON SYSPUNCH
- MERGE
- OPEN
- READ
- REWRITE
- SORT
- START
- STOP *literal*
- WRITE

ACCEPT ステートメントの一部の形式を除き、メインフレーム CICS では上記に示す COBOL 言語エレメントをサポートしません。これらの言語エレメントを使用する場合は、次の制限に注意してください。

- プログラムをメインフレーム CICS 環境に完全に移植することはできません。
- CICS 障害が発生した場合は、上記のステートメントを使用して更新されたリソースに対してバックアウト (失敗したタスクに関連するリソースを復元すること) を実行することができません。

制約事項 : 分離または統合された CICS 変換プログラムによって変換され、CICS TXSeries または CICS TX 上で動作する COBOL プログラムでサポートされる IBM Z ホスト・データ形式はありません。

関連タスク
383 ページの『CICS のもとでのシステム日付の取得』
384 ページの『CICS での動的呼び出し』
386 ページの『SFS データへのアクセス』
386 ページの『CICS での COBOL および C/C++ 間の呼び出し』

関連参照
118 ページの『Db2 ファイル・システム』
252 ページの『ADDR』
529 ページの『付録 B IBM Z ホスト・データ形式についての考慮事項』

CICS のもとでのシステム日付の取得

CICS プログラムでシステム日付を検索するには、形式 2 の ACCEPT ステートメントまたは CURRENT-DATE 組み込み関数を使用してください。

以下の形式 2 の ACCEPT ステートメントを CICS で使用すると、システム日付を入手することができます。

- ACCEPT *identifier-2* FROM DATE (2 桁年)
- ACCEPT *identifier-2* FROM DATE YYYYMMDD
- ACCEPT *identifier-2* FROM DAY (2 桁年)
- ACCEPT *identifier-2* FROM DAY YYYYDDD

• ACCEPT *identifier-2* FROM DAY-OF-WEEK (1 桁の整数。1 は月曜日を表します。)

次に示す形式 2 の ACCEPT ステートメントを CICS で使用すると、システム時刻を入手することができます。

• ACCEPT *identifier-2* FROM TIME

あるいは、CURRENT-DATE 組み込み関数を使用できます。この組み込み関数も時刻を提供できます。

これらの方法は、CICS 環境と非 CICS 環境の両方で使用できます。

CICS プログラムでは、形式 1 の ACCEPT ステートメントは使用しないでください。

関連タスク
30 ページの『画面またはファイルからの入力の割り当て (ACCEPT)』

関連参照
CURRENT-DATE (*COBOL for Linux on x86* 言語解説書)

CICS での動的呼び出し

CICS 環境では、CALL*identifier* ステートメントを使用して、動的呼び出しをすることができます。ただし、COBPATH 環境変数を正しく設定する必要があります。 また、呼び出されたモジュールの名前が正しいことも確認する必要があります。

以下では、alpha が CICS ステートメントを含む COBOL プログラムである例を考えます。

```
WORKING-STORAGE SECTION.
01  WS-COMMAREA   PIC 9    VALUE ZERO.
77   SUBPNAME     PIC X(8) VALUE SPACES
. . .
PROCEDURE DIVISION.
    MOVE 'alpha' TO SUBPNAME.
    CALL SUBPNAME USING DFHEIBLK, DFHCOMMAREA, WS-COMMAREA.
```

CICS 制御ブロック DFHEIBLK および DFHCOMMAREA (上記に示される) を alpha に渡す必要があります。

alpha のソースはファイル alpha.ccp 内にあります。 コマンド cicstcl は、alpha.ccp の変換、コンパイル、およびリンクに使用されます。 COBOL でのデフォルトは大文字の名前です。 したがって、PGMNAME(MIXED) コンパイラー・オプションを使用してこのデフォルトを変更する場合以外は、ソース・ファイル ALPHA.ccp (alpha.ccp ではない) を指定して、ALPHA.ibmcob (alpha.ibmcob ではない) を作成する必要があります。

CICS 領域が green と呼ばれていると仮定します。その場合、ファイル ALPHA.ibmcob を /var/cics_regions/green/bin にコピーする必要があり、新しいリソース定義を追加する cicsadd コマンドを使用して、CICS プログラムとして ALPHA を定義する必要があります。インストール・スタッフは、ファイル /var/cics_regions/green/environment に次の行を追加する必要があります。

```
COBPATH=/var/cics_regions/green/bin
```

次に、スタッフは CICS の green 領域をシャットダウンしてから、再起動する必要があります。呼び出し先プログラムをその他のディレクトリー内に動的に入れる場合は、インストール・スタッフが COBPATH にそのディレクトリーを追加することと、CICS サーバーにそのディレクトリーへのアクセス権があることを確認してください。

関連タスク
385 ページの『CICS での共有ライブラリーの動的呼び出し』
385 ページの『CICS での動的呼び出しのパフォーマンスのチューニング』

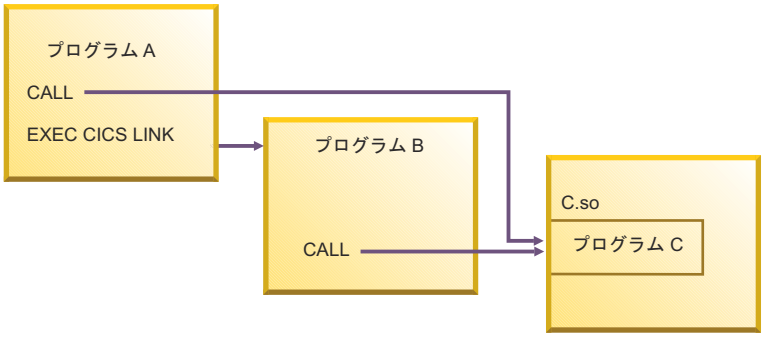
関連参照
216 ページの『コンパイラー環境変数とランタイム環境変数』

CICS での共有ライブラリーの動的呼び出し

1 つ以上の COBOL プログラムが含まれる共用ライブラリーは、同一 CICS トランザクション内の複数の実行単位で使用しないでください。さもないと、予測不能な結果が生じます。

次の図は、2 つの異なる実行単位から同一のサブプログラムが呼び出された場合の CICS トランザクションを示しています。

- プログラム A は (C.so 内の) プログラム C を呼び出します。
- プログラム A は EXEC CICS LINK コマンドを使用してプログラム B にリンクします。 この組み合わせが、同じトランザクション内の新しい実行単位になります。
- プログラム B は (C.so 内の) プログラム C を呼び出します。



プログラム A と B はプログラム C の同一コピーを共用するため、コピーの状態が変化すると両方のプログラムに影響を与えます。

CICS 環境では、実行単位内で初回呼び出し時にのみ、共有ライブラリー内のプログラムが初期化 (WSCLEAR コンパイラー・オプションまたは VALUE 文節のいずれかによる初期化) されます。1 つの COBOL サブプログラムが複数回呼び出される場合、呼び出し元のメインプログラムが同一であってもなくても、 サブプログラムは初回呼び出し時にのみ初期化されます。

それぞれのメインプログラムからの初回呼び出し時に、サブプログラムを初期化する必要がある場合は、サブプログラムの個々のコピーを各呼び出し側プログラムと静的にリンクします。 呼び出しのたびにサブプログラムを初期化する必要がある場合は、次のいずれかの方法を使用してください。

- 再初期化するデータを WORKING-STORAGE SECTION ではなく、サブプログラムの LOCAL-STORAGE SECTION に入れる。 これは、WSCLEAR による初期化ではなく、VALUE 文節による初期化にのみ影響します。
- CANCEL を使用して、サブプログラムの使用後ごとにサブプログラムをキャンセルする。これにより、次回の呼び出し時には、そのプログラムが初期状態になります。
- サブプログラムに INITIAL 属性を追加する。

関連タスク
463 ページの『第 24 章 共用ライブラリーの使用』

関連参照
289 ページの『WSCLEAR』

CICS での動的呼び出しのパフォーマンスのチューニング

COBOL for Linux アプリケーションで、永続的な CICS トランザクションのパフォーマンスが、デフォルトでモジュール・キャッシングによって向上しました。

モジュール・キャッシングを有効または無効にするには、以下の環境変数を設定します。

```
export COBOL_CPM_CACHE=0    ## To disable caching
export COBOL_CPM_CACHE=1    ## To enable caching
```

COBOL_CPM_CACHE 環境変数が指定されていない場合、デフォルトは以下のようになります。

- CICS トランザクションの場合、キャッシングはデフォルトで有効になっています。
- 非 CICS プログラムの場合、キャッシングは自動制御されます。キャッシングを有効にするかどうか、およびいつ有効にするかは COBOL ランタイムによって決定されます。

モジュールがキャッシュされると、その実行セマンティクスが変わることがあります。これは、VALUE 節を持たず、明示的な初期化ステートメントがない WORKING-STORAGE のデータ項目が、最後に使用された状態のままになるためです。 メモリー内のプログラムがキャッシュされず、代わりにディスクから再ロードされる場合、これらの未初期化の変数の最後に使用された状態は失われます。

未初期化のデータ項目の値に依存しないでください。 プログラムに入るたびに、このようなデータ項目を明示的に初期化するか、または一時的に WSCLEAR コンパイラー・オプションを使用して WORKING-STORAGE を 2 進ゼロにクリアしてください。

WSCLEAR は、プログラムの初期化に必要な時間、そしておそらくはスペースを増加させるため、パフォーマンスに影響を与える可能性があります。 モジュール・キャッシングが有効な場合に最高のパフォーマンスを得るには、アプリケーション・コードを検討して、変数を明示的に初期化するかどうか判断してください。

関連参照
289 ページの『WSCLEAR』

SFS データへのアクセス

ユーザーのプログラムが CICS で実行されていなければ、SFS ファイル・システム (CICS TXSeries または CICS TX によって使用されるデフォルトのファイル・システム) から SFS ファイルにアクセスできます。

関連タスク
113 ページの『ファイルの識別』
115 ページの『SFS ファイルの識別』
146 ページの『SFS ファイルの使用』

関連参照

120 ページの『SFS ファイル・システム』

CICS での COBOL および C/C++ 間の呼び出し

呼び出される側のプログラムに CICS コマンドが含まれていない場合に限り、COBOL から C/C++ プログラムへの呼び出し、または C/C++ プログラムから COBOL プログラムへの呼び出しを CICS で行うことができます。(呼び出し側プログラムに CICS コマンドを含めることはできます。)

COBOL プログラムは、C/C++ プログラムに CICS コマンドが含まれているかどうかに関係なく、EXEC CICS LINK または EXEC CICS XCTL コマンドを C/C++ プログラムに発行することができます。したがって、COBOL プログラムが、CICS コマンドを含む C/C++ プログラムを呼び出す場合は、COBOL CALL ステートメントではなく EXEC CICS LINK または EXEC CICS XCTL を使用してください。

関連タスク
438 ページの『COBOL および C/C++ プログラム間の呼び出し』

CICS プログラムのコンパイルおよび実行

COBOL for Linux CICS TXSeries または CICS TX プログラムをコンパイルするには、cob2 コマンドを使用します。

TRUNC(BIN) は、CICS で実行する COBOL プログラムに推奨されるコンパイラー・オプションです。ただし、BINARY、COMP、および COMP-4 データ項目の値が切り捨てられていない値で、PICTURE の指定に準拠することが確実な場合は、TRUNC(OPT) を使用することにより、プログラムのパフォーマンスが向上する可能性があります。

EXEC CICS コマンド引数として、BINARY、COMP、または COMP-4 データ項目の代わりに COMP-5 データ項目を使用することができます。 TRUNC(BIN) が 有効な場合、COMP-5 データ項目は、BINARY、COMP、または COMP-4 データ項目と同様に扱われます。

CICS Client を使用するプログラムには、PGMNAME(MIXED) コンパイラー・オプションを使用する必要があります。

COBOL プログラムを (分離型または組み込みの CICS 変換プログラムを使用して) 変換する際に、DYNAM または ADDR(64) コンパイラー・オプションを使用しないでください。それ以外の COBOL コンパイラー・オプションはすべてサポートされます。

ランタイム・オプション: ファイルに使用するファイル・システムが ASSIGN 文節によって指定されていない場合は、FILESYS ランタイム・オプションを使用してデフォルトのファイル・システムを指定します。

関連概念
387 ページの『組み込みの CICS 変換プログラム』

関連タスク
225 ページの『コマンド行からのコンパイル』
TXSeries for Multiplatforms の資料

関連参照

248 ページの『コンパイラー・オプション』
250 ページの『矛盾するコンパイラー・オプション』
300 ページの『FILESYS』

組み込みの CICS 変換プログラム

CICS コンパイラー・オプションを指定して COBOL プログラムをコンパイルする場合、COBOL コンパイラーは組み込みの CICS 変換プログラムと連動して、ソース・プログラム内のネイティブ COBOL ステートメントと組み込みの CICS ステートメントの両方を処理します。

コンパイラーは、CICS ステートメントを検出したとき、およびソース・プログラム内の重要な地点で、組み込みの CICS 変換プログラムとインターフェースをとります。EXEC CICS ステートメントと END-EXEC ステートメントの間のテキストはすべて変換プログラムに渡されます。 変換プログラムは適切な処置を行ってから、通常は、生成するネイティブ言語ステートメントを指示してコンパイラーに制御を戻します。

cicstcl コマンドを使用して COBOL プログラムをコンパイルする場合は、組み込みの CICS 変換プログラムを使用します。cicstcl コマンドは、CICS コンパイラー・オプションの適切なサブオプションでコンパイラーを呼び出します。

組み込みの CICS ステートメントは、分離型の変換も引き続き可能ですが、組み込みの CICS 変換プログラムを使用することを推奨します。分離型の変換プログラムを使用する場合に適用される一部の制約は、組み込みの変換プログラムを使用する場合には適用されません。組み込みの変換プログラムを使用することにはいくつかの利点があります。

- を使用して、分離型の CICS 変換プログラムによって生成される拡張ソースではなく、元のソースをデバッグすることができます。
- コピーブック内の EXEC CICS ステートメントステートメントを個別に変換する必要がありません。
- 変換済みで未コンパイル・バージョンのソース・プログラムに対応する中間ファイルが不要です。
- 出力リストは 2 つではなく 1 つだけ作成されます。
- REPLACE ステートメントを EXEC CICS ステートメントに影響させることができます。
- CICS ステートメントが含まれているプログラムをバッチ・コンパイル (一連のプログラムのコンパイル) でコンパイルすることができます。
- 拡張ソース形式 (Extended Source) は完全にサポートされます。

関連タスク

382 ページの『CICS のもとで実行する COBOL プログラムのコーディング』
386 ページの『CICS プログラムのコンパイルおよび実行』

関連参照
257 ページの『CICS』
283 ページの『SRCFORMAT』

CICS プログラムのデバッグ

統合変換プログラムを使用して CICS プログラムをコンパイルする場合、分離型の CICS 変換プログラムが提供する拡張ソースをデバッグするのではなく、元のソース・レベルでプログラムをデバッグできます。

分離型の CICS 変換プログラムを使用する場合は、まず CICS プログラムを COBOL に変換します。次いで結果の COBOL プログラムを、他の COBOL プログラムと同様の方法でデバッグすることができます。

COBOL for Linux に付属の IBM Debug for Linux on x86 を使用して、CICS プログラムをデバッグすることも可能です。この場合は、分散デバッガーが使用するシンボリック情報を生成するようコンパイラーに指示する必要があります。

関連概念
303 ページの『第 16 章 デバッグ』
387 ページの『組み込みの CICS 変換プログラム』

関連タスク

225 ページの『コマンド行からのコンパイル』
TXSeries for Multiplatforms の資料

第 5 部 XML および COBOL の併用

© Copyright IBM Corp. 2021

第 19 章 XML 入力の処理

XML PARSE ステートメントを使用すると、COBOL プログラム内で XML 入力を処理できます。

XML PARSE ステートメントは、COBOL ランタイムの一部である高速 XML パーサーとの間の COBOL 言語インターフェースです。

XML 入力の処理には、XML パーサーと、パーサー・イベントを処理する処理プロシージャーの間で制御を渡すことが含まれます。

XML 入力を処理するには、以下の COBOL 機能を使用します。

- XML PARSE ステートメントは、XML 構文解析を開始して、ソース XML 文書および処理プロシージャーを識別します。
- 処理プロシージャーは、構文解析を制御します。すなわち、XML イベントおよび関連文書フラグメントを受け取って処理し、パーサーに戻って処理を続行します。
- 特殊レジスターは、パーサーと処理プロシージャーの間で次のように情報を交換します。
  - XML-CODE は、XML 構文解析の状況を受け取り、時として、情報をパーサーに返します。
  - XML-EVENT は、各 XML イベントの名前をパーサーから受け取ります。
  - XML-NTEXT は、国別文字データとして返された XML 文書フラグメントを受け取ります。
  - XML-TEXT は、英数字データとして返された文書フラグメントを受け取ります。

関連概念
391 ページの『COBOL での XML パーサー』

関連タスク
392 ページの『XML 文書へのアクセス』
393 ページの『XML 文書の構文解析』
401 ページの『XML PARSE の例外処理』
404 ページの『XML 構文解析の終了』

関連参照

398 ページの『XML 文書のエンコード方式』
577 ページの『付録 E XML 参照資料』
Extensible Markup Language (XML)

COBOL での XML パーサー

COBOL for Linux ではイベント・ベースのインターフェースが提供されるため、これを使用して XML 文書を構文解析し、さらに COBOL データ構造に変換することができます。

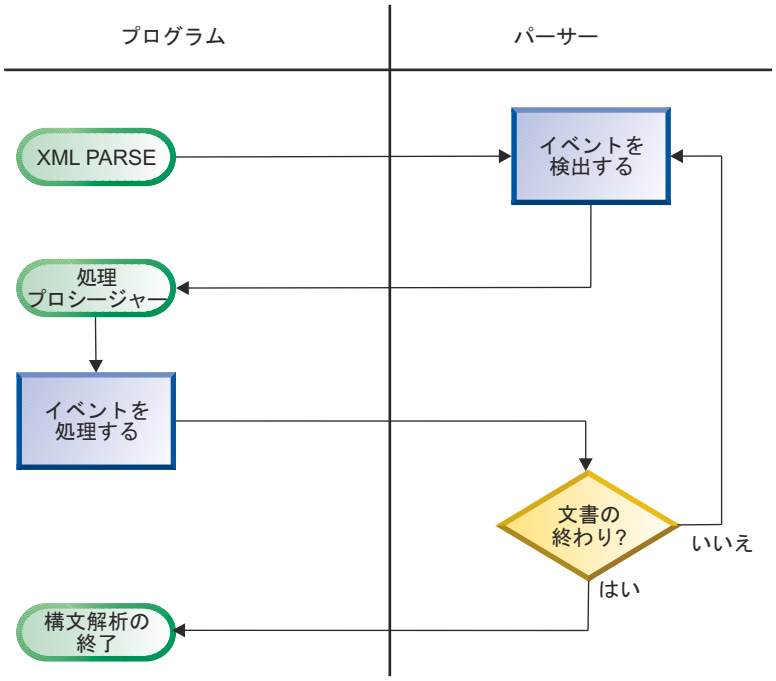
XML パーサーがソース XML 文書内のフラグメントを検出し、作成した処理プロシージャーによってそれらのフラグメントに対する操作が実行されます。 フラグメントは XML イベントに関連付けられます。それぞれの XML イベントを処理するために独自の処理プロシージャーをコーディングします。

XML PARSE ステートメントを実行すると、構文解析が開始されてパーサーでの処理プロシージャーが確立されます。 パーサーは、文書の処理中に検出した XML イベントごとに、処理プロシージャーに制御を渡します。 イベントの処理後、処理プロシージャーは自動的に制御をパーサーに返します。 処理プロシージャーから正常に制御が返されるごとに、パーサーは XML 文書の分析を続けて次のイベントに報告します。 この操作の間中、制御がパーサーと処理プロシージャーの間を行き来します。

XML PARSE ステートメントに、構文解析の終了時に制御を渡したい 2 つの命令ステートメントを指定することもできます。1 つは正常終了の場合、もう 1 つは例外条件が存在する場合のためのステートメントです。

次の図は、パーサーと COBOL プログラム間で行われる基本的な制御受け渡しの概要を示しています。

© Copyright IBM Corp. 2021



通常、構文解析は XML 文書全体が構文解析されるまで継続されます。

XML パーサーは、XML 文書のさまざまな側面が整形式であるかどうかを検査します。 文書が整形式 であるのは、*XML specification* に記載されている XML 構文規則に準拠し、その他のいくつかの規則 (終了タグの適切な使用、属性名が固有であることなど) に従っている場合です。

関連概念
398 ページの『XML 入力文書エンコード』

関連タスク
393 ページの『XML 文書の構文解析』
401 ページの『XML PARSE の例外処理』
404 ページの『XML 構文解析の終了』

関連参照
398 ページの『XML 文書のエンコード方式』
585 ページの『XML 準拠』
XML specification

XML 文書へのアクセス

XML PARSE ステートメントを使用して XML 文書を構文解析する前に、その文書をプログラムで使用できるようにしておく必要があります。 XML 文書を取得する一般的な方法としては、プログラムに対するパラメーターから取得する方法や、ファイルから文書を読み取る方法があります。

構文解析する XML 文書がファイル内に保管されている場合は、以下に示す通常の COBOL 機能を使用して文書をプログラムのデータ項目に入れてください。

- FILE-CONTROL 記入項目でプログラムに対してファイルを定義します。
- OPEN ステートメントでファイルをオープンします。
- READ ステートメントで、ファイルからすべてのレコードを読み取って、データ項目 (カテゴリー英数字またはカテゴリー国別の基本項目、あるいは英数字グループまたは国別グループ) に入れます。 WORKING-STORAGE SECTION または LOCAL-STORAGE SECTION でデータ項目を定義できます。
- (オプション) STRING ステートメントで、個別レコードすべてを 1 つの連続ストリームに結合したり、無関係なブランクを除去したり、可変長レコードを処理したりします。

XML 文書の構文解析

XML 文書を構文解析するには、次のコード・フラグメントで示すように、XML PARSE ステートメントを使用して、 構文解析する XML 文書、および構文解析時に発生する XML イベントを扱うための処理プロシージャーを指定します。

```
XML PARSE xml-document
    PROCESSING PROCEDURE xml-event-handler
  ON EXCEPTION
     DISPLAY 'XML document error ' XML-CODE
     STOP RUN
  NOT ON EXCEPTION
     DISPLAY 'XML document was successfully parsed.'
END-XML
```

最初に XML PARSE ステートメントで、XML 文書文字ストリームを含む構文解析データ項目 (上の例では xml-document) を識別します。 DATA DIVISION には、 文書のエンコードが Unicode UTF-16 である場合には、カテゴリー国別の基本データ項目または国別グループ項目として 構文解析データ項目を定義します。 それ以外の場合には、基本英数字データ項目または英数字グループ項目として構文解析データ項目を定義します。

- 構文解析データ項目が国別の場合、XML 文書は リトル・エンディアン形式の UTF-16 でエンコードする必要があります。
- 構文解析データ項目が英数字である場合には、そのコンテンツは、XML 文書のエンコードに関する、 関連参照に説明されている、サポートされているコード・ページのいずれかでエンコードする必要があります。

次に、文書の構文解析で発生した XML イベントを処理する処理プロシージャーの名前 (上記の例では xml-event-handler) を指定します。

さらに、以下のオプションで句の一方または両方を指定して (上記フラグメントを参照)、構文解析の終了後に行われるアクションを指示できます。

- ON EXCEPTION は、構文解析中に未処理の例外が発生した場合に制御を受け取ります。
- NOT ON EXCEPTION は、それ以外の場合に制御を受け取ります。

XML PARSE ステートメントを終了するには、明示範囲終了符号の END-XML を使用します。 END-XML を使用して、条件ステートメント内で ON EXCEPTION 句または NOT ON EXCEPTION 句を使用する XML PARSE ステートメントをネストできます。

パーサーは、XML イベントごとに処理プロシージャーに制御を渡します。 処理プロシージャーの終わりに到達すると、制御はパーサーに返されます。 XML パーサーと処理プロシージャー間での制御の受け渡しは、以下のイベントのいずれかが発生するまで継続します。

- XML 文書全体の構文解析が完了したことが、END-OF-DOCUMENT イベントによって示された場合。
- パーサーが文書内にエラーを検出し、EXCEPTION イベントをシグナル通知した場合。この場合、処理プロシージャーは、パーサーに制御を戻す前に特殊レジスター XML-CODE をゼロにリセットしません。
- パーサーに戻る前に XML-CODE 特殊レジスターを -1 に設定する、処理プロシージャー内のコードによって、構文解析プロセスは意図的に終了されます。

関連概念
395 ページの『XML イベント』
395 ページの『XML-CODE』

関連タスク

202 ページの『文字データのコード・ページの指定』
394 ページの『XML を処理するためのプロシージャーの作成』
401 ページの『UTF-8 でエンコードされた XML 文書の構文解析』

関連参照

398 ページの『XML 文書のエンコード方式』
577 ページの『XML PARSE 例外』
XML PARSE ステートメント (*COBOL for Linux on x86* 言語解説書)

XML を処理するためのプロシージャーの作成

処理プロシージャーには、XML イベントを処理するためのステートメントをコーディングします。

パーサーは、イベントを検出すると、特殊レジスター内の処理プロシージャーに情報を渡します。 これらの特殊レジスターのコンテンツは、COBOL データ構造の取り込みと、処理の制御に使用します。

XML-EVENT 特殊レジスターを検査して、パーサーが処理プロシージャーに渡したイベントを判別します。 XML-EVENT には、'START-OF-ELEMENT' などのイベント名が入ります。 XML-TEXT または XML-NTEXT の特殊レジスターからイベントに関連付けられたテキストを取得します。

XML 特殊レジスターがネストされたプログラムで使用された場合は、最外部のプログラムで GLOBAL として暗黙的に定義されます。

XML の特殊レジスターに関する追加の詳細については、以下の表を参照してください。

表 *38.* ***XML*** **パーサーの使用する特殊レジスター**

特殊レジスター	暗黙的な定義および使用法	内容
XML-EVENT[1]	PICTURE X(30) USAGE DISPLAY VALUE SPACE	XML イベントの名前
XML-CODE[2]	PICTURE S9(9) USAGE BINARY VALUE ZERO	各 XML イベント用の例外コードまたはゼロ
XML-TEXT[1、3]	可変長基本カテゴリー英数字項目。	XML PARSE ID として英数字項目を指定した場合は、XML 文書のテキスト (パーサーが検出したイベントに対応)
XML-NTEXT[1]	可変長基本カテゴリー国別項目。	XML PARSE ID として国別項目を指定した場合は、XML 文書のテキスト (パーサーが検出したイベントに対応)

1. この特殊レジスターを受け取りデータ項目として使用することはできません。
2. XML GENERATE ステートメントでも XML-CODE が使用されます。 したがって、処理プロシージャー内に XML GENERATE ステートメントがある 場合、XML GENERATE ステートメントの前に XML-CODE の値を 保存し、 XML GENERATE ステートメントの後に保存した値をリストアします。
3. XML-TEXT の内容にソース XML 文書のエンコードが 含まれます (CHAR(NATIVE) コンパイラー・オプションが有効な 場合は ASCII または UTF-8、CHAR(EBCDIC) が有効な場合は EBCDIC)。

制約事項:

- 処理プロシージャーで、XML PARSE ステートメントを直接実行してはなりません。 しかし、CALL ステートメントを使用して、処理プロシージャーから最外部のプログラムに制御が渡る場合、ターゲットとなるプログラムは同一または別の XML PARSE ステートメントを実行できます。複数のスレッドで実行されているプログラムから、同一または別の XML ステートメントを同時に実行することもできます。
- 処理プロシージャーの範囲で、GOBACK または EXIT PROGRAM ステートメントを実行してはいけません。ただし、制御がそれぞれ CALL ステートメントで渡されたプログラムから制御を返す場合を除きます。この場合は、処理プロシージャーの範囲で実行されます。

  処理プロシージャーに STOP RUN ステートメントをコーディングすると、実行単位を終了させることができます。

コンパイラーは、各処理プロシージャーの最後のステートメントの後に、戻り機構を挿入します。

405 ページの『例: XML の処理用プログラム』

関連概念
395 ページの『XML イベント』

395 ページの『XML-CODE』
397 ページの『XML-TEXT および XML-NTEXT』

関連タスク

404 ページの『XML 構文解析の終了』

関連参照
256 ページの『CHAR』
XML-EVENT (*COBOL for Linux on x86* 言語解説書)

XML イベント

XML イベント は、XML 文書の処理中に XML パーサーがさまざまな条件 (END-OF-INPUT や EXCEPTION など) を検出したり、文書フラグメント (CONTENT-CHARACTERS や START-OF-CDATA-SECTION など) を検出したりした場合に発生します。

XML 構文解析中に発生するイベントごとに、パーサーは XML-EVENT 特殊レジスターに関連イベント名を設定し、XML-EVENT 特殊レジスターを処理プロシージャーに渡します。 イベントによっては、パーサーは他の特殊レジスターを設定して、イベントに関する追加情報を含めます。

ほとんどの場合、パーサーは次のように XML-TEXT または XML-NTEXT の特殊レジスターに、イベントを引き起こした XML フラグメントを設定します。 XML 文書が国別データ項目である場合、またはパーサーが文字参照を検出した場合には XML-NTEXT、 それ以外の場合には XML-TEXT を設定します。

パーサーは、文書内でエンコード競合や整形式性エラーを検出すると、XML-EVENT を 'EXCEPTION' に 設定し、XML-CODE 特殊レジスターで例外に関する情報を追加します。

XML イベント・セットについて詳しくは、XML-EVENT に関する関連参照を参照してください。

関連概念
391 ページの『COBOL での XML パーサー』
395 ページの『XML-CODE』

397 ページの『XML-TEXT および XML-NTEXT』

関連タスク
394 ページの『XML を処理するためのプロシージャーの作成』

関連参照

577 ページの『XML PARSE 例外』
XML-EVENT (*COBOL for Linux on x86* 言語解説書)

XML-CODE

EXCEPTION イベント以外の各 XML イベントについて、 パーサーは、XML-CODE 特殊レジスターの値にゼロを設定します。 EXCEPTION イベントについては、パーサーは、 XML-CODE に特定の例外を識別する値を設定します。

可能な例外コードについては、関連参照を参照してください。

パーサーが処理プロシージャーから XML PARSE ステートメントに制御を戻したとき、 XML-CODE には通常、パーサーによって設定された最新の値が入っています。 ただし、EXCEPTION 以外のイベントでは、処理プロシージャーで XML-CODE を -1 に設定すると、 制御がパーサーに戻ったときに、ユーザー設定の例外条件で構文解析が終了し、 XML-CODE には値 -1 が保持されます。

EXCEPTION XML イベントでは、 制御がパーサーに戻る前に、処理プロシージャーが XML-CODE を意味がある値に設定できるケースがあります。 (詳細については、XML PARSE 例外の処理とエンコード競合の処理に関する関連タスクを参照してください。) XML-CODE を他のゼロ以外の値に設定した場合、または他の例外のために設定した場合には、パーサーは XML-CODE を元の例外コードに再設定します。

以下の表では、XML-CODE をさまざまな値に設定した場合の結果を示しています。 左端の列は、処理プロシージャーに渡された XML イベントのタイプを表し、 他の列の見出しは、処理プロシージャーによって設定された XML-CODE の値を表します。 それぞれの行と列が交差したところのセルに、 XML イベントと XML-CODE 値の特定の組み合わせで処理プロシージャーから戻ったときのパーサーのアクションが示されています。

表 *39.* ***XML-CODE*** **に対する処理プロシージャーの変更の結果**

XML イベント・タイプ	**-1**	**0**	**XML-CODE-100,000 (EBCDIC) XML-CODE-200,000 (ASCII)**	**その他のゼロ以外の値**
エンコード競合例外 (例外コード 50 から 99)	設定を無視。元の XML-CODE 値を保持	個別の例外コードに応じてエンコードを選択 [1]	設定を無視。元の XML-CODE 値を保持	設定を無視。元の XML-CODE 値を保持
エンコード選択例外 (例外コード > 100,000)	設定を無視。元の XML-CODE 値を保持	外部コード・ページを使用して構文解析 [2]	差 (上記) をエンコード値として使用して構文解析 [2]	設定を無視。元の XML-CODE 値を保持
その他の例外	設定を無視。元の XML-CODE 値を保持	例外コード 1 から 49 の場合にのみ限定的に継続 [3]	設定を無視。元の XML-CODE 値を保持	設定を無視。元の XML-CODE 値を保持
通常イベント	即時に終了。XML-CODE = -1[4]	[XML-CODE に明らかな変更はなし]	即時に終了。XML-CODE = -1	即時に終了。XML-CODE = -1

1. XML PARSE 例外に関する関連参照の例外コードを参照してください。
2. エンコード競合の処理に関する関連タスクを参照してください。
3. XML PARSE 例外の処理に関する関連タスクを参照してください。
4. XML 構文解析の終了に関する関連タスクを参照してください。

XML 生成でも XML-CODE 特殊レジスターが使用されます。 詳細に ついては、XML GENERATE 例外の処理に関する関連タスクを参照してください。

関連概念
402 ページの『XML パーサーによるエラーの処理方法』

関連タスク
394 ページの『XML を処理するためのプロシージャーの作成』
401 ページの『XML PARSE の例外処理』
403 ページの『エンコード競合の処理』
404 ページの『XML 構文解析の終了』
414 ページの『XML GENERATE 例外の処理』

関連参照

577 ページの『XML PARSE 例外』
587 ページの『XML GENERATE 例外』
XML-CODE (*COBOL for Linux on x86* 言語解説書)
XML-EVENT (*COBOL for Linux on x86* 言語解説書)

XML-TEXT および XML-NTEXT

ほとんどの XML イベントで、パーサーは XML-TEXT または XML-NTEXT を関連文書フラグメントに設定します。

通常、XML 文書が英数字データ項目である場合に、パーサーは XML-TEXT を設定します。 パーサーは次の場合に、XML-NTEXT を設定します。

- XML 文書が国別データ項目である
- XML 文書が英数字データ項目であり、ATTRIBUTE-NATIONAL-CHARACTER または CONTENT-NATIONAL-CHARACTER イベントが発生した

特殊レジスター XML-TEXT と XML-NTEXT は、相互に排他的です。 パーサーが XML-TEXT を設定した場合、XML-NTEXT は空で長さゼロになります。 パーサーが XML-NTEXT を設定した場合、XML-TEXT は空で長さゼロになります。

XML-NTEXT 内の文字エンコード・ユニットの数を決定するには、LENGTH 組み込み関数 (例: FUNCTION LENGTH(XML-NTEXT)) を使用します。 XML-NTEXT 内のバイト数を決定するには、特殊レジスター LENGTH OF XML-NTEXT を使用します。 文字エンコード・ユニット数は、バイト数とは異なります。

XML-TEXT 内のバイト数を決定するには、特殊レジスター LENGTH OF XML-TEXT または LENGTH 組み込み関数を 使用します (それぞれバイト数を返します)。

XML-TEXT および XML-NTEXT 特殊レジスターは、処理プロシージャー外では未定義です。

関連概念
395 ページの『XML イベント』
395 ページの『XML-CODE』

関連タスク
394 ページの『XML を処理するためのプロシージャーの作成』

関連参照

XML-TEXT (*COBOL for Linux on x86* 言語解説書)
XML-NTEXT (*COBOL for Linux on x86* 言語解説書)

XML テキストの COBOL データ項目への変換

XML データは固定長ではなく、固定形式でもないので、XML データを COBOL データ項目に移動するときは、特別な技法を使用する必要があります。

英数字項目の場合、XML データを COBOL 項目の左端 (デフォルト) に配置するか、右端に配置するかを決める必要があります。 データが右端に配置する場合は、項目の定義に JUSTIFIED RIGHT 節を指定します。

数字の XML 値、特に、'$1,234.00' または '$1234' といった「修飾」された通貨値には特別な配慮が必要です。 この 2 つのストリングは、XML で同じものを意味することがありますが、COBOL 送信フィールドとして使用された場合には、まったく別の定義を必要とします。

XML データを COBOL データ項目に移動するには、以下の技法のいずれかを使用します。

- 適度に規則性のあるフォーマットの場合は、MOVE を使用して、適切な数字編集項目として再定義した英数字項目に移動します。 次に、数字編集項目から移動して編集解除することにより、数字 (操作) 項目への最終的な移動を行います。 (規則性のあるフォーマットとは、例えば、小数点以下の桁数が同じで、999 より大きい値にはコンマ区切り文字が付くフォーマットです。)
- 英数字の XML データに対して以下の組み込み関数を使用すると、高度な柔軟性が簡単に実現できます。
  - NUMVAL を使用して、単純な数字を表現している XML データから単純な数値を抽出およびデコードします。
  - NUMVAL-C を使用して、通貨数量を表現している XML データから数値を抽出およびデコードします。

  ただし、これらの関数を使用するとパフォーマンスが下がります。

関連タスク
105 ページの『数値への変換 (NUMVAL、NUMVAL-C)』
180 ページの『COBOL での国別データ (Unicode) の使用』
394 ページの『XML を処理するためのプロシージャーの作成』

XML 文書のエンコード方式

XML 文書は、サポートされたコード・ページでエンコードする必要があります。

XML PARSE ステートメントで構文解析する XML 文書は エンコードが必要であり、XML GENERATE ステートメントで作成した XML 文書はエンコードされます。 エンコードは以下のように行います。

- 国別データ項目の文書: リトル・エンディアン形式の Unicode UTF-16
- ネイティブ英数字データ項目の文書: Unicode UTF-8、 または International Components for Unicode (ICU) 変換ライブラリーでサポートされる 1 バイト ASCII コード・ページ

  ネイティブ英数字データ項目 は、CHAR(NATIVE) コンパイラー・オプションを有効にして コンパイルしたカテゴリー英数字データ項目、または NATIVE 句が含まれるデータ記述記入項目を持つ カテゴリー英数字データ項目です。
- ホスト英数字データ項目の文書: ICU 変換ライブラリーでサポートされる 1 バイト EBCDIC コード・ページ

  ホスト英数字データ項目 は、CHAR(EBCDIC) コンパイラー・オプションを有効にして コンパイルしたカテゴリー英数字データ項目のうち、NATIVE 句が含まれないデータ記述記入項目を持つ カテゴリー英数字データ項目です。

ICU 変換ライブラリーでサポートされるエンコードについては、ICU コンバーター・エクスプローラーに関する 関連参照を参照してください。

関連概念
398 ページの『XML 入力文書エンコード』

関連タスク
400 ページの『エンコードの指定』
401 ページの『UTF-8 でエンコードされた XML 文書の構文解析』
409 ページの『第 20 章 XML 出力の生成』

関連参照

256 ページの『CHAR』
International Components for Unicode: Converter Explorer

XML 入力文書エンコード

XML PARSE ステートメントを使用して XML 文書を構文解析するには、文書はサポートされているエンコードでエンコードされている必要があります。

特定の構文解析操作に対してサポートされているエンコードは、XML 文書を含むデータ項目のタイプに依存します。 パーサーは、以下のデータ項目とエンコードのタイプをサポートします。

- リトル・エンディアン形式で、Unicode UTF-16 でエンコードされた内容を持つカテゴリー国別データ項目
- Unicode UTF-8 またはサポートされる 1 バイトの ASCII コード・ページのいずれかをエンコードされた内容を持つネイティブ英数字データ項目
- サポートされる 1 バイトの EBCDIC コード・ページのいずれかでエンコードされた内容を持つホスト英数字データ項目

サポートされているコード・ページについては、XML 文書のエンコードに関する関連参照を参照してください。

パーサーは、XML 文書の最初の数バイトを検査することによって実際の文書エンコードを判別します。 実際の文書エンコードが ASCII または EBCDIC の場合、パーサーは正しく構文解析するために特定のコード・ページ情報を必要とします。 この追加のコード・ページ情報は、文書エンコード宣言または外部コード・ページ情報から取得されます。

文書エンコード宣言は、XML 宣言のオプション部分で、文書の先頭にあります。 詳細については、エンコードの指定に関する関連タスクを参照してください。

ASCII XML 文書の 外部コード・ページ (外部 *ASCII* コード・ページ) は、現行のランタイム・ロケールによって示されるコード・ページです。 EBCDIC XML 文書の外部コード・ページ (外部 *EBCDIC* コード・ページ) は、次のいずれかになります。

- EBCDIC_CODEPAGE 環境変数で指定されたコード・ページ
- EBCDIC_CODEPAGE 環境変数を設定しなかった場合は、現行のランタイム・ロケールに対して選択されたデフォルトの EBCDIC コード・ページ

指定されたエンコードがサポートされているコード化文字セットのものではない場合、パーサーは構文解析操作を実行する前に XML 例外イベントをシグナル通知します。 実際の文書エンコードが指定されたエンコードに一致しない場合には、パーサーは構文解析操作の開始後に該当する XML 例外をシグナル通知します。

サポートされていないコード・ページでエンコードされた XML 文書を構文解析するには、NATIONAL-OF 組み込み関数を使用して、まず文書を国別文字データ (UTF-16) に変換します。 特殊レジスター XML-NTEXT で処理プロシージャーに渡されるそれぞれの文書テキスト部分は、DISPLAY-OF 組み込み関数を使用して元のコード・ページに変換することができます。

XML 宣言および空白文字 (white space)

XML 文書は、XML 宣言がない場合のみ、冒頭に空白文字 *(white space)* を置くことができます。

- XML 文書の冒頭が XML 宣言である場合、文書中の最初の不等号括弧 (<) がその文書の先頭文字である必要があります。
- XML 文書の冒頭が XML 宣言でない場合、文書中の最初の不等号括弧の前に置くことができるのは空白文字 (white space) のみです。

空白文字 (white-space characters) には、次の表に記載された 16 進値が含まれます。

表 *40.* **空白文字 *(white-space characters)* の *16* 進値**

空白文字 (white-space character)	**EBCDIC**	**Unicode / ASCII**
スペース	X'40'	X'20'
水平タブ	X'05'	X'09'
復帰	X'0D'	X'0D'
改行	X'25'	X'0A'
改行/次の行	X'15'	X'85'

関連タスク
188 ページの『国別 (Unicode) 表現との間の変換』
400 ページの『エンコードの指定』
401 ページの『UTF-8 でエンコードされた XML 文書の構文解析』
401 ページの『XML PARSE の例外処理』

関連参照
204 ページの『サポートされるロケールおよびコード・ページ』
398 ページの『XML 文書のエンコード方式』
400 ページの『XML マークアップ内の EBCDIC コード・ページ依存文字』
577 ページの『XML PARSE 例外』

エンコードの指定

英数字データ項目の XML 文書を構文解析するためのエンコードを指定する方法を選択できます。

推奨する方法としては、文書からエンコード宣言を省略し、代わりに外部コード・ページの指定に従います。

エンコード宣言を省略することにより、異機種システム間で XML 文書をより簡単に伝送できるようになります。 (エンコード宣言を組み込んだ場合、伝送プロセスで発生するコード・ページ変換を反映するためにエンコード宣言を 更新する必要があります。)

エンコード宣言を持たない英数字 XML 文書の構文解析に使用されるコード・ページは、ランタイム・コード・ページです。

代わりに XML 宣言内にエンコード宣言を指定できます。 多くの XML 文書は XML 宣言で始まります。 次に例を示します。

```
<?xml version="1.0" encoding="ibm-1140"?>
```

XML パーサーは、先頭バイトが XML 宣言で開始されていない XML 文書を検出すると例外を生成します。

エンコード宣言を指定する場合、ICU 変換ライブラリーでサポートされている いずれかの基本コード・ページ名または別名コード・ページ名を使用してください。 コード・ページ名については、ICU コンバーター・エクスプローラーに関する関連参照を参照してください。

XML 構文解析にサポートされている CCSID について詳しくは、XML 文書のエンコードに関する 関連参照を参照してください。

関連概念
398 ページの『XML 入力文書エンコード』

関連タスク
401 ページの『UTF-8 でエンコードされた XML 文書の構文解析』
403 ページの『エンコード競合の処理』

関連参照

204 ページの『サポートされるロケールおよびコード・ページ』
398 ページの『XML 文書のエンコード方式』
International Components for Unicode: Converter Explorer

XML マークアップ内の EBCDIC コード・ページ依存文字

XML マークアップで使用されるいくつかの特殊文字には、さまざまな EBCDIC コード・ページで異なる 16 進表記があります。

次の表に、各種 EBCDIC CCSID の特殊文字とそれぞれの 16 進値を示します。

表 *41.* **さまざまな *EBCDIC CCSID* 用特殊文字の *16* 進値**

文字	1047	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149
[	X'AD'	X'BA'	X'63'	X'9E'	X'B5'	X'90'	X'4A'	X'B1'	X'90'	X'4A'	X'AE'
]	X'BD'	X'BB'	X'FC'	X'9F'	X'9F'	X'51'	X'5A'	X'BB'	X'B5'	X'5A'	X'9E'
!	X'5A'	X'5A'	X'4F'	X'4F'	X'4F'	X'4F'	X'BB'	X'5A'	X'4F'	X'4F'	X'4F'
|	X'4F'	X'4F'	X'BB'	X'BB'	X'BB'	X'BB'	X'4F'	X'4F'	X'BB'	X'BB'	X'BB'
#	X'7B'	X'7B'	X'7B'	X'4A'	X'63'	X'B1'	X'69'	X'7B'	X'B1'	X'7B'	X'7B'

UTF-8 でエンコードされた XML 文書の構文解析

Unicode UTF-8 でエンコードされた XML 文書を、他の XML 文書と同じように構文解析することができます。ただし、追加でいくつかの要件が適用されます。

UTF-8 XML 文書を構文解析するには、通常どおり XML 文書の構文解析に対して以下のように XML PARSE ステートメントをコーディングします。

```
XML PARSE xml-document
    PROCESSING PROCEDURE xml-event-handler
    . . .
END-XML
```

ただし、以下の追加要件に従ってください。

- 構文解析データ項目 (上記の例では xml-document) カテゴリー英数字でなければなりません。 また、CHAR(EBCDIC) コンパイラー・オプションが有効であってはなりません。
- XML 文書が ASCII ではなく UTF-8 として構文解析されるようにするには、以下の条件のうち、少なくとも 1 つが当てはまるようにしてください。
  - ランタイム・ロケールは UTF-8 ロケールである。
  - UTF-8 を指定する XML エンコード宣言 (encoding="UTF-8") が文書に含まれている。
  - 文書は UTF-8 バイト・オーダー・マークで始まっている。
- x'FFFF' を超える Unicode スカラー値を持つ文字が文書に含まれていてはなりません。 そのような文字には、文字参照 ("&#x*hhhhh*;") を使用してください。

パーサーは、 英数字特殊 レジスター XML-TEXT で XML 文書のフラグメントを返します。

UTF-8 文字 1 文字は、可変個のバイト数でエンコードされます。 英数字データに対するほとんどの COBOL 操作では、1 バイトのエンコード (1 文字が 1 バイトでエンコードされるエンコード) を想定しています。 UTF-8 文字を英数字データとして操作する場合、データが正常に処理されるようにする必要があります。 マルチバイト文字のバイトを分割する可能性がある操作 (参照変更や切り捨てを呼び出す移動操作など) は避けてください。 英数字データのマルチバイト文字を処理するために、INSPECT などのステートメントを信頼して使用することはできません。

関連概念
397 ページの『XML-TEXT および XML-NTEXT』

関連タスク
197 ページの『UTF-16 (国別) データ・タイプを使用して UTF-8 データを処理』
393 ページの『XML 文書の構文解析』
400 ページの『エンコードの指定』

関連参照
256 ページの『CHAR』
398 ページの『XML 文書のエンコード方式』
XML PARSE ステートメント (*COBOL for Linux on x86* 言語解説書)

XML PARSE の例外処理

XML パーサーは、構文解析中に異常またはエラーを検出すると、例外コードを XML-CODE 特殊レジスターに設定し、 XML 例外イベントをシグナル通知します。

例外コードが特定範囲内にある場合、処理プロシージャーで例外イベントを処理して構文解析を再開できます。

処理プロシージャーで例外を処理するには、以下の手順に従ってください。

1. XML-CODE の内容を検査します。
2. 例外を適切に処理します。

3. XML-CODE を、例外が処理されたことを示すゼロに設定します。
4. パーサーに制御を戻します。

これにより、例外条件がなくなります。

このような方法で例外を処理できるのは、XML-CODE に入れて渡される例外コードが以下の範囲の 1 つに含まれる場合 (エンコードの競合が検出されたことを示します) のみです。

- 50 から 99
- 100,001 から 165,535
- 200,001 - 265,535

例外コード 1 から 49: 処理プロシージャーでは、 例外コードが 1 から 49 の範囲内にある例外に対して、限られた例外処理を行うことができます。 この範囲内の例外が発生した後、パーサーは、戻る前に XML-CODE がゼロに設定されている場合でも、END-OF-DOCUMENT イベントを除き、それ以上標準イベントをシグナル通知しません。 XML-CODE をゼロに設定した場合、パーサーは文書の構文解析を続行し、検出した例外をシグナル通知します (これは、文書内で複数のエラーを発見する方法として有効です。)

制約事項 : COBOL XML パーサーは、すべての追加例外イベントをシグナル通知できない場合があります。例外数は、XML PARSE イベント・トークン配列の残りスペース (おそらく 8192 イベント) に限定されます。

例外コードが 1 から 49 の範囲内の例外が発生した後の構文解析終了時には、 ON EXCEPTION 句に指定されたステートメントに制御が渡されます。 ON EXCEPTION 句がコーディングされていない場合、制御は XML PARSE ステートメントの終わりに渡されます。 XML-CODE には、パーサーが最新の例外に設定したコードが含まれます。

例外コードが上記の範囲内にないすべての例外で、パーサーはこれ以上のイベントをシグナル通知しませんが、 ON EXCEPTION 句に指定されたステートメントに制御を渡します。 パーサーに制御を返す前に処理プロシージャーで XML-CODE を設定したとしても、XML-CODE には元の例外コードが含まれます。

例外を処理する必要がない場合は、XML-CODE の値を変更しないでパーサーに制御を戻します。 パーサーは、ON EXCEPTION 句で指定されたステートメントに制御を移します。 ON EXCEPTION 句がコーディングされていない場合、制御は XML PARSE ステートメントの終わりに移動します。

構文解析の終了時点までに未処理の例外がなかった場合は、NOT ON EXCEPTION 句に指定されたステートメントに制御が渡されます。 NOT ON EXCEPTION 句をコーディングしなかった場合、制御は XML PARSE ステートメントの終わりに移動します。 XML-CODE には、ゼロが含まれます。

関連概念
395 ページの『XML-CODE』
398 ページの『XML 入力文書エンコード』
402 ページの『XML パーサーによるエラーの処理方法』

関連タスク
394 ページの『XML を処理するためのプロシージャーの作成』
403 ページの『エンコード競合の処理』

関連参照

398 ページの『XML 文書のエンコード方式』
577 ページの『XML PARSE 例外』

XML パーサーによるエラーの処理方法

XML 文書の中にエラーがあるのを検出すると、XML パーサーは XML 例外イベントを生成し、制御を処理プロシージャーに渡します。

パーサーは、以下の情報を特殊レジスターに入れて処理プロシージャーに渡します。

- XML-EVENT に 'EXCEPTION' が設定されている。

- XML-CODE に数値の例外コードが設定されている。

  例外コードは、 XML PARSE 例外に関する関連参照に記載されています。
- 致命的な例外に関しては、例外が検出されたポイントまでの (そのポイントを含む) 文書テキストが XML-TEXT または XML-NTEXT に含まれます。
- 宣言されていない接頭部を使用していたために発行された警告例外に関しては、 完全修飾の属性名またはエレメント名が XML-TEXT または XML-NTEXT に含まれます。 すなわち、これらの名前には、宣言されていない接頭部と、区切り記号のコロン (:) が含まれます。
- XML-TEXT または XML-NTEXT には、 例外検出ポイントまでの文書テキストが含まれています。

すべての他の XML 特殊レジスターは空で、長さゼロです。

例外コードが次のいずれかの範囲内にある場合には、構文解析を続行できるように処理プロシージャーが例外を処理できる場合があります。

- 1 から 99
- 100,001 から 165,535
- 200,001 - 265,535

例外コードがその他のゼロ以外の値である場合は、構文解析を続けることはできません。

エンコード競合: エンコード方式の矛盾の例外 (50 から 99 および 300 から 399) は、文書の構文解析が 開始される前にシグナル通知されます。 このような例外の場合、XML-TEXT または XML-NTEXT は長さがゼロになるか、文書のエンコード宣言値のみが入ります。

例外コード 1 から 49: 例外コードが 1 から 49 までの範囲にある例外は、 *XML* 仕様 によって致命的エラーになります。 したがって、処理プロシージャーが例外を処理しても、パーサーは通常の構文解析を続けません。ただし、パーサーは、文書の終わりに到達するまで、または既存の XML EVENT トークン配列が使い尽くされるまで、その他のエラーの走査を続けます。 このような例外の場合、パーサーでは、END-OF-DOCUMENT イベント以外については、追加の標準イベントをシグナル通知しません。

関連概念
395 ページの『XML イベント』
395 ページの『XML-CODE』
398 ページの『XML 入力文書エンコード』

関連タスク

401 ページの『XML PARSE の例外処理』
403 ページの『エンコード競合の処理』

404 ページの『XML 構文解析の終了』

関連参照

398 ページの『XML 文書のエンコード方式』
577 ページの『XML PARSE 例外』
XML specification

エンコード競合の処理

ご使用の処理プロシージャーが、文書エンコード競合の例外を処理できる場合があります。 構文解析データ項目が英数字であり、XML-CODE の例外コードが 100,001 から 165,535 または、200,001 から 265,535 の範囲にある例外イベントは、(エンコード宣言で指定された) 文書のコード・ページが、外部コード・ページ情報と矛盾していることを示しています。

この特殊ケースでは、XML-CODE の 値から 100,000 または 200,000 (コード・ページがそれぞれ EBCDIC か ASCII かによる) を 減算することで、文書のコード・ページを使用して構文解析を行うようにすることができます。 例えば、XML-CODE が 101,140 に設定されている場合、文書のコード・ページは 1140 で

す。 別の方法では、パーサーに戻る前に XML-CODE をゼロに設定して、外部コード・ページを使用して構文解析することもできます。

パーサーは、コード・ページ矛盾の例外イベント用の処理プロシージャーから戻ると、以下の 3 つの処置のいずれかをとります。

- XML-CODE をゼロに設定した場合、パーサーは、構文解析データ項目がネイティブ英数字項目かホスト英数字項目かによって、 外部 ASCII コード・ページまたは外部 EBCDIC コード・ページをそれぞれ使用します。
- XML-CODE に文書のコード・ページ (すなわち、元の XML-CODE 値から 100,000 または必要に応じて 200,000 を減算した値) を設定すると、パーサーは文書のコード・ページを使用します。

  処理プロシージャーからの戻り時に XML-CODE が非ゼロ値に設定されていたとき、パーサーが処理を続けるのはこのケースに該当する場合だけです。
- それ以外の場合、パーサーは文書の処理を停止し、例外条件とともに制御を XML PARSE ステートメントに戻します。 XML-CODE は、当初に例外イベントへ渡された例外コードに設定されます。

関連概念
395 ページの『XML-CODE』
398 ページの『XML 入力文書エンコード』
402 ページの『XML パーサーによるエラーの処理方法』

関連タスク
401 ページの『XML PARSE の例外処理』

関連参照

398 ページの『XML 文書のエンコード方式』
577 ページの『XML PARSE 例外』

XML 構文解析の終了

プロシージャーが通常の XML イベント (つまり、EXCEPTION 以外のイベント) からパーサーに戻る前に、処理プロシージャーで XML-CODE を -1 に設定すると、 残りの XML テキストを処理せずに直ちに構文解析を終了することができます。

この技法は、処理プロシージャーで文書を十分に検査済みである場合、あるいは処理プロシージャーが文書に何らかの異常を検出し、それ以上処理を続けても意味がない場合に使用できます。

このように構文解析を終了した場合、パーサーはそれ以上 XML イベント (例外イベントを含む) をシグナル通知しません。 制御は、XML PARSE ステートメントの ON EXCEPTION 句が指定されている場合にはその句に渡されます。

ON EXCEPTION 句の命令ステートメントでは、XML-CODE に -1 が入っているかどうかをテストすることによって、 構文解析が意図的に終了されたかどうかを判断します。 ON EXCEPTION 句が指定されていない場合、制御は XML PARSE ステートメントの終わりに渡されます。

また、XML-CODE の値を変更せずに処理プロシージャーからパーサーに戻ることによって、XML EXCEPTION イベントの発生後に構文解析を終了することができます。 この場合、結果は意図的に終了した場合と似ていますが、XML-CODE に元の例外コードが入った状態でパーサーが XML PARSE ステートメントに戻る点は除きます。

関連概念
395 ページの『XML-CODE』
402 ページの『XML パーサーによるエラーの処理方法』

関連タスク
394 ページの『XML を処理するためのプロシージャーの作成』
401 ページの『XML PARSE の例外処理』

XML PARSE の例

以下で参照される例では、XML PARSE ステートメントのさまざまな使用方法を説明します。

405 ページの『例: 単純な文書の構文解析』
405 ページの『例: XML の処理用プログラム』

例: 単純な文書の構文解析

この例では、単純な XML 文書の構文解析によるイベントのフローおよび特殊レジスター XML-TEXT の内容を示します。

COBOL プログラムには次のようなデータ項目 Doc の XML 文書が含まれていると仮定します。

```
<?xml version="1.0"?><msg type="short">Hello, World!</msg>
```

次のコードの断片では、Doc を構文解析する XML PARSE ステートメント、および XML イベントを処理する処理プロシージャー P を示します。

```
XML Parse Doc
   Processing procedure P
   . . .
P. Display XML-Event XML-Text.
```

処理プロシージャーでは、パーサーが構文解析中にシグナル通知する各イベントの XML-EVENT および XML-TEXT のコンテンツが表示されます。 次の表に、イベントおよびテキストを示します。

表 *42.* ***XML*** **イベントおよび特殊レジスター**

XML-EVENT	**XML-TEXT**
START-OF-DOCUMENT	
VERSION-INFORMATION	1.0
START-OF-ELEMENT	msg
ATTRIBUTE-NAME	type
ATTRIBUTE-CHARACTERS	short
CONTENT-CHARACTERS	Hello, World!
END-OF-ELEMENT	msg
END-OF-DOCUMENT	

関連概念
395 ページの『XML イベント』
397 ページの『XML-TEXT および XML-NTEXT』

例: XML の処理用プログラム

以下の例では、XML 文書の構文解析と、各種 XML イベントおよびその関連テキスト・フラグメントを報告する処理プロシージャーを示しています。

構文解析のフローを簡単に追えるように、プログラム・ソースに XML 文書が示されています。 例の後に、プログラムの出力が示されています。

パーサーと処理プロシージャーとの間の相互作用を理解し、イベントと文書フラグメントを突き合わせるために、 XML 文書とプログラムの出力を比較してください。

```
Process codepage(1047)
       Identification division.
         Program-id. XMLSAMPL.
       Data division.
        Working-storage section.
      ****************************************************************
      * XML document data, encoded as initial values of data items.    *
      ****************************************************************
         1 xml-document-data.
          2 pic x(39) value '<?xml version="1.0" encoding="UTF-8"'.
          2 pic x(19) value ' standalone="yes"?>'.
          2 pic x(39) value '<!--This document is just an example-->'.
          2 pic x(10) value '<sandwich>'.
          2 pic x(33) value '<bread type="baker&apos;s best"/>'.
          2 pic x(36) value '<?spread We'll use real mayonnaise?>'.
          2 pic x(29) value '<meat>Ham &amp; turkey</meat>'.
          2 pic x(34) value '<filling>Cheese, lettuce, tomato, '.
          2 pic x(32) value 'and that's all, Folks!</filling>'.
          2 pic x(25) value '<![CDATA[We should add a '.
          2 pic x(20) value '<relish> element!]]>'.
          2 pic x(28) value '<listprice>$4.99</listprice>'.
          2 pic x(25) value '<discount>0.10</discount>'.
          2 pic x(31) value '</sandwich>'.
      ****************************************************************
      * XML document, represented as fixed-length records.            *
      ****************************************************************
         1 xml-document redefines xml-document-data.
          2 xml-segment pic x(40) occurs 10 times.
         1 xml-segment-no comp pic s9(4).
         1 content-buffer pic x(100).
         1 current-element-stack.
          2 current-element pic x(30) occurs 10 times.
      ****************************************************************
      * Sample data definitions for processing numeric XML content.    *
      ****************************************************************
         1 element-depth comp pic s9(4).
         1 discount computational pic 9v99 value 0.
         1 display-price pic $$9.99.
         1 filling pic x(4095).
         1 list-price computational pic 9v99 value 0.
         1 ofr-ed pic x(9) justified.
         1 ofr-ed-1 redefines ofr-ed pic 999999.99.
       Procedure division.
        Mainline section.
           Move 1 to xml-segment-no
           Display 'Initial segment {' xml-segment(xml-segment-no) '}'
           Display ' '
           XML parse xml-segment(xml-segment-no)
               processing procedure XML-handler
             On exception
               Display 'XML processing error, XML-Code=' XML-Code '.'
               Move 16 to return-code
               Goback
             Not on exception
               Display 'XML document successfully parsed.'
           End-XML
      ****************************************************************
      * Process the transformed content and calculate promo price.     *
      ****************************************************************
           Display ' '
           Display '-----++++++++++ Using information from XML '
               '**********-----'
           Display ' '
           Move list-price to Display-price
           Display ' Sandwich list price: ' Display-price
           Compute Display-price = list-price * (1 - discount)
           Display ' Promotional price: ' Display-price
           Display ' Get one today!'
           Goback.
        XML-handler section.
           Evaluate XML-Event
      * ==> Order XML events most frequent first
             When 'START-OF-ELEMENT'
               Display 'Start element tag: {' XML-Text '}'
               Add 1 to element-depth
               Move XML-Text to current-element(element-depth)
             When 'CONTENT-CHARACTERS'
```

```
           Display 'Content characters: {' XML-Text '}'
      * ==> In general, a split can occur for any element or attribute
      * ==> data, but in this sample, it only occurs for "filling"...
           If xml-information = 2 and
               current-element(element-depth) not = 'filling'
             Display 'Unexpected split in content for element '
                 current-element(element-depth)
             Move -1 to xml-code
           End-if
      * ==> Transform XML content to operational COBOL data item...
           Evaluate current-element(element-depth)
             When 'filling'
      * ==> After reassembling separate pieces of character content...
               String xml-text delimited by size into
                   content-buffer with pointer tally
                 On overflow
                   Display 'content buffer ('
                       length of content-buffer
                       ' bytes) is too small'
                   Move -1 to xml-code
               End-string
               Evaluate xml-information
                 When 2
                   Display '  Character data for element "filling" '
                       'is incomplete.'
                   Display '  The partial data was buffered for '
                       'content assembly.'
                 When 1
                   subtract 1 from tally
                   move content-buffer(1:tally) to filling
                   Display '  Element "filling" data (' tally
                       ' bytes) is now complete:'
                   Display '  {' filling(1:tally) '}'
               End-evaluate
             When 'listprice'
      * ==> Using function NUMVAL-C...
               Move XML-Text to content-buffer
               Compute list-price =
                   function numval-c(content-buffer)
             When 'discount'
      * ==> Using de-editing of a numeric edited item...
               Move XML-Text to ofr-ed
               Move ofr-ed-1 to discount
           End-evaluate
         When 'END-OF-ELEMENT'
           Display 'End element tag: {' XML-Text '}'
           Subtract 1 from element-depth
         When 'END-OF-INPUT'
           Display 'End of input'
           Add 1 to xml-segment-no
           Display '  Next segment: {' xml-segment(xml-segment-no)
               '}'
           Display ' '
           Move 1 to xml-code
         When 'START-OF-DOCUMENT'
           Display 'Start of document'
           Move 0 to element-depth
           Move 1 to tally
         When 'END-OF-DOCUMENT'
           Display 'End of document.'
         When 'VERSION-INFORMATION'
           Display 'Version: {' XML-Text '}'
         When 'ENCODING-DECLARATION'
           Display 'Encoding: {' XML-Text '}'
         When 'STANDALONE-DECLARATION'
           Display 'Standalone: {' XML-Text '}'
         When 'ATTRIBUTE-NAME'
           Display 'Attribute name: {' XML-Text '}'
         When 'ATTRIBUTE-CHARACTERS'
           Display 'Attribute value characters: {' XML-Text '}'
         When 'ATTRIBUTE-CHARACTER'
           Display 'Attribute value character: {' XML-Text '}'
         When 'START-OF-CDATA-SECTION'
           Display 'Start of CData section'
         When 'END-OF-CDATA-SECTION'
           Display 'End of CData section'
         When 'CONTENT-CHARACTER'
           Display 'Content character: {' XML-Text '}'
         When 'PROCESSING-INSTRUCTION-TARGET'
           Display 'PI target: {' XML-Text '}'
         When 'PROCESSING-INSTRUCTION-DATA'
           Display 'PI data: {' XML-Text '}'
```

```
             When 'COMMENT'
               Display 'Comment: {' XML-Text '}'
             When 'EXCEPTION'
               Compute tally = function length (XML-Text)
               Display 'Exception ' XML-Code ' at offset ' tally '.'
             When other
               Display 'Unexpected XML event: ' XML-Event '.'
           End-evaluate
           .
       End program XMLSAMPL.
```

以下の出力では、構文解析中に発生したイベントに、どの文書フラグメントが関連付けられていたかを確認することができます。

```
Start of document
Version: {1.0}
Encoding: {UTF-8}
Standalone: {yes}
Comment: {This document is just an example}
Start element tag: {sandwich}
Content characters: {  }
Start element tag: {bread}
Attribute name: {type}
Attribute value characters: {baker}
Attribute value character: {'}
Attribute value characters: {s best}
End element tag: {bread}
Content characters: {  }
PI target: {spread}
PI data: {please use real mayonnaise  }
Content characters: {  }
Start element tag: {meat}
Content characters: {Ham }
Content character: {&}
Content characters: { turkey}
End element tag: {meat}
Content characters: {  }
Start element tag: {filling}
Content characters: {Cheese, lettuce, tomato, etc.}
End element tag: {filling}
Content characters: {  }
Start of CData: {<![CDATA[}
Content characters: {We should add a <relish> element in future!}
End of CData: {]]>}
Content characters: {  }
Start element tag: {listprice}
Content characters: {$4.99 }
End element tag: {listprice}
Content characters: {  }
Start element tag: {discount}
Content characters: {0.10}
End element tag: {discount}
End element tag: {sandwich}
End of document.
XML document successfully parsed

-----++++++++++++ Using information from XML ++++++++++-----

  Sandwich list price:  $4.99
  Promotional price:    $4.49
  Get one today!
```

関連概念
395 ページの『XML イベント』

関連参照

XML-EVENT (*COBOL for Linux on x86* 言語解説書)

第 20 章 XML 出力の生成

XML GENERATE ステートメントを使用して、COBOL プログラムから XML 出力を生成させることができます。

XML GENERATE ステートメントでは、ソースおよび出力データ項目を指定します。 オプションとして、次のものも指定できます。

- 生成された XML 文字のカウントを受け取るフィールド
- 生成された XML 文書のエンコード
- 生成された文書の名前空間
- 各エレメントの開始および終了タグを修飾する名前空間接頭部 (名前空間を指定した場合)
- 生成された XML 文書のユーザー定義の要素または属性名
- いくつかの指定済み条件に応じて抑止される属性または要素
- 生成された XML 出力で属性、要素、またはコンテンツとして指定される特定の項目。
- 例外発生時に制御を受け取るステートメント

オプションとして、文書で XML 宣言を生成して、適格なソース・データ項目を出力でエレメントとしてではなく属性として表すことができます。

XML-CODE 特殊レジスターを使用して、XML 生成の状況を判別できます。

COBOL データ項目を XML に変換した後、得られた XML 出力をさまざまな方法で使用できます。例えば、Web サービスにそれを配置したり、ファイルに記述したり、他のプログラムにパラメーターとして渡したりできます。

関連タスク
409 ページの『XML 出力の生成』
414 ページの『生成される XML 出力のエンコードの制御』
414 ページの『XML GENERATE 例外の処理』
419 ページの『XML 出力の拡張』

関連参照
Extensible Markup Language (XML)
XML GENERATE ステートメント (*COBOL for Linux on x86* 言語解説書)

XML 出力の生成

COBOL データを XML に変換するには、以下の例のように XML GENERATE ステートメントを使用してください。

```
XML GENERATE XML-OUTPUT FROM SOURCE-REC
       COUNT IN XML-CHAR-COUNT
   ON EXCEPTION
       DISPLAY 'XML generation error ' XML-CODE
       STOP RUN
   NOT ON EXCEPTION
       DISPLAY 'XML document was successfully generated.'
END-XML
```

XML GENERATE ステートメントでは、XML 出力を受け取るデータ項目 (上の例では XML-OUTPUT) をまず識別します。 データ項目は、生成された XML 出力を格納できる十分な大きさに定義します。通常、データ名の長さに応じて、COBOL ソース・データ・サイズの 5 倍から 10 倍に定義します。

DATA DIVISION では、受信 ID を、英数字 (英数字グループ項目またはカテゴリー英数字の基本項目のどちらか) として、あるいは国別 (国別グループ項目またはカテゴリー国別の基本項目のどちらか) として定義できます。

© Copyright IBM Corp. 2021

次に、XML フォーマットに変換されるソース・データ項目 (この例では SOURCE-REC) を識別します。 ソース・データ項目は、英数字グループ項目、国別グループ項目、あるいはクラス英数字または国別の基本データ項目にすることができます。

COBOL データ項目の中には、XML に変換されず無視されるものがあります。 XML に変換する英数字グループ項目または国別グループ項目の従属データ項目は、次のような場合は無視されます。

- REDEFINES 節を指定しているか、またはそのような再定義項目に従属しているもの。
- RENAMES 節を指定しているもの。

ソース・データ項目の中の次のような項目も、XML の生成時に無視されます。

- 基本 FILLER (または名前なしの) データ項目
- SYNCHRONIZED データ項目で挿入されている遊びバイト

XML を読みやすくするために余分な空白文字 (例えば、改行または字下げ) が挿入されることはありません。

必要に応じて、COUNT IN 句をコーディングして、XML 出力の生成時に充てんされる XML 文字エンコード・ユニット数を取得できます。 受け取る ID のカテゴリーが国別である場合、カウントは、UTF-16 文字エンコード・ユニット数です。 すべての他のエンコード (UTF-8 を含む) では、カウントはバイト数です。

カウント・フィールドを参照変更長として使用して、生成された XML 出力を含む受け取りデータ項目の一部のみを取得できます。 例えば、XML-OUTPUT(1:XML-CHAR-COUNT) は、XML-OUTPUT の最初の XML-CHAR-COUNT 文字位置を参照します。

次のプログラムの抜粋を検討します。

```
01  doc pic x(512).
01  docSize pic 9(9) binary.
01  G.
    05  A pic x(3) value "aaa".
    05  B.
        10  C pic x(3) value "ccc".
        10  D pic x(3) value "ddd".
    05  E pic x(3) value "eee".
    . . .
    XML Generate Doc from G
```

上のコードによって、次の XML 文書が生成されます。ここで、A、B、および E は、エレメント G の子エレメントとして表され、 C および D は、エレメント B の子エレメントになります。

```
<G><A>aaa</A><B><C>ccc</C><D>ddd</D></B><E>eee</E></G>
```

また、XML GENERATE ステートメントの ATTRIBUTES 句を指定することもできます。 ATTRIBUTES 句を使用すると、生成された XML 文書に含まれる適格なデータ項目はすべて、それを含む XML エレメントの子エレメントとしてではなく、それを含む XML エレメントの属性として表現されます。 データ項目が適格になるためには、データ項目は基本データ項目でなければならず、データ項目の名前は FILLER 以外でなければならず、そのデータ記述項目に OCCURS 節があってはなりません。 データ項目を含む XML エレメントは、基本データ項目のすぐ上位にあるグループ・データ項目に対応しています。 オプションで、TYPE OF 句を使用して、どのデータ項目を属性またはエレメントとして表現するか、より詳細な制御を指定できます。

例えば、上記プログラムの抜粋の XML GENERATE ステートメントが次のようにコーディングされたとします。

```
XML Generate Doc from G with attributes
```

このコードによって次の XML 文書が生成されます。ここで、A および E は、エレメント G の属性として表され、 C および D はエレメント B の属性になります。

```
<G A="aaa" E="eee"><B C="ccc" D="ddd"></B></G>
```

オプションとして、XML GENERATE ステートメントの ENCODING 句をコーディングして、生成される XML 文書のエンコードを指定できます。 ENCODING 句を使用しなかった場合、文書エンコードは受け取りデータ項目のカテゴリーによって決まります。 詳細については、生成された XML 出力のエンコードの制御に関する下記の関連タスクを参照してください。

オプションとして、XML-DECLARATION 句をコーディングして、生成された XML 文書にバージョン情報およびエンコード宣言を含んだ XML 宣言を組み込むことができます。 受け取りデータ項目のカテゴリーによって次のようになります。

- 国別の場合: エンコード宣言には値 UTF-16 が含まれます (encoding="UTF-16")。
- 英数字の場合: エンコード宣言は、ENCODING 句 (指定されている場合) またはランタイム・ロケールまたは EBCDIC_CODEPAGE 環境変数 (ENCODING 句が指定されていない場合) から派生します。

例えば、下記のプログラムの抜粋では、XML GENERATE の XML-DECLARATION 句を指定して、 エンコードを UTF-8 で指定しています。

```
01  Greeting.
    05 msg  pic x(80)  value 'Hello, world!'.
    . . .
    XML Generate Doc from Greeting
        with Encoding "UTF-8"
        with XML-declaration
    End-XML
```

上のコードによって、次の XML 文書が生成されます。

```
<?xml version="1.0" encoding="UTF-8"?><Greeting><msg>Hello, world!</msg></Greeting>
```

XML-DECLARATION 句をコーディングしなければ、XML 宣言は生成されません。

オプションとして、NAMESPACE 句をコーディングして、生成される XML 文書の名前空間を指定することもできます。 名前空間の値は有効な *URI (Uniform Resource Identifier)* (例: URL (Uniform Resource Locator)) である必要があります。 詳細については、下記の URI 構文に関する関連概念を参照してください。

名前空間は、カテゴリーが国別または英数字の ID またはリテラルで指定します。

名前空間を指定して名前空間接頭部 (下記) を指定しない場合には、その名前空間は文書のデフォルト名前空間になります。 つまり、ルート・エレメントで定義された名前空間が文書内のルート・エレメントを含む各エレメント名にデフォルトで適用されます。

例えば、次のデータ定義および XML GENERATE ステートメントを検討してみましょう。

```
01  Greeting.
    05  msg  pic x(80)  value 'Hello, world!'.
01  NS  pic x(20)   value 'http://example'.
    . . .
    XML Generate Doc from Greeting
        namespace is NS
```

次に示すとおり、結果として得られる XML 文書は、デフォルト名前空間 (http://example) を持ちます。

```
<Greeting xmlns="http://example"><msg>Hello, world!</msg></Greeting>
```

名前空間を指定しなければ、生成される XML 文書内のエレメント名は、どの名前空間にも属しません。

オプションとして、NAMESPACE-PREFIX 句をコーディングして、生成される文書内の各エレメントの開始および終了タグに適用する接頭部を指定することもできます。 接頭部は、上述のように名前空間を指定した場合にのみ指定できます。

XML GENERATE ステートメントを実行する場合、接頭部の値はコロン (:) なしの有効な XML 名でなければなりません。 詳しくは、名前空間に関する下記の関連参照を参照してください。 値には末尾スペースを含めることができますが、接頭部の使用前に除去されます。

名前空間接頭部は、カテゴリーが国別または英数字の ID またはリテラルで指定します。

接頭部は、各エレメントの開始および終了タグを修飾するため、短くすることを推奨します。

例えば、次のデータ定義および XML GENERATE ステートメントを検討してみましょう。

```
01  Greeting.
    05  msg  pic x(80)  value 'Hello, world!'.
01  NS  pic x(20)   value 'http://example'.
01  NP  pic x(5)    value 'pre'.
    . . .
    XML Generate Doc from Greeting
        namespace is NS
        namespace-prefix is NP
```

次のように、結果として得られる XML 文書は明示的な名前空間 (http://example) を持ち、接頭部 pre がエレメント Greeting および msg の開始および終了タグに適用されます。

```
<pre:Greeting xmlns:pre="http://example"><pre:msg>Hello, world!</pre:msg></
pre:Greeting>
```

オプションで、NAME 句をコーディングして、生成される XML 文書内に属性と要素の名前を指定することができます。 属性名と要素名は、英数字または国別リテラルでなければならず、XML 1.0 標準に従う正式名でなければなりません。

例えば、次のデータ構造および XML GENERATE ステートメントを検討してみましょう。

```
01 Msg.
    02 Msg-Severity pic 9 value 1.
    02 Msg-Date pic 9999/99/99 value "2012/04/12".
    02 Msg-Text pic X(50) value "Sell everything!".
01 Doc pic X(500).
```

```
    XML Generate Doc from Msg
        With attributes
        Name of Msg          is  "Message"
                Msg-Severity is  "Severity"
                Msg-Date     is  "Date"
                Msg-Text     is  "Text"
    End-XML
```

結果として生成される XML 文書は以下のとおりです。

```
<Message Severity="1" Date="2012/04/12" Text="Sell everything!"></Message>
```

オプションで、SUPPRESS 句をコーディングし、特定の基準を満たすかどうかに基づいて個々のデータ項目を生成するかどうかを指定できます。

例えば、スペースとゼロを抑止する次のデータ構造および XML GENERATE ステートメントを検討してみましょう。

```
01 G.
    02 SensitiveInfo.
        03 SSN pic x(11) value '123-45-6789'.
        03 HomeAddress pic x(50) value '123 Main St, Anytown, USA'.
    02 Aarray value spaces.
        03 A pic AAA occurs 5.
    02 Barray value spaces.
        03 B pic XXX occurs 5.
    02 Carray value zeros.
        03 C pic 999 occurs 5.
    Move 'abc' to A(1)
    Move 123 to C(3)
    XML Generate Doc from G
        Suppress SensitiveInfo
                 every nonnumeric element when space
                 every numeric element when zero
    End-XML
```

結果として生成される XML 文書は以下のとおりです。

```
<G>
   <Aarray><A>abc</A></Aarray>
   <Carray><C>123</C></Carray>
</G>
```

オプションで、TYPE OF 句を使用して、個々のデータ項目を属性、エレメント、または内容のいずれとして表現するかを指定できます。

例えば、次のデータ構造および XML GENERATE ステートメントを検討してみましょう。

```
01 Msg.
   02 Msg-Severity pic 9 value 1.
   02 Msg-Date pic 9999/99/99 value "2012/04/12".
   02 Msg-Text pic X(50) value "Sell everything!".
01 Doc pic X(500).
    XML Generate Doc from Msg
        With attributes
        Type of Msg-Severity is attribute
                Msg-Date     is attribute
                Msg-Text     is element
    End-XML
```

結果として生成される XML 文書は以下のとおりです。

```
<Msg Msg-Severity="1" Msg-Date="2012/04/12">
       <Msg-Text>Sell everything!</Msg-Text></Msg>
```

さらに、XML 文書の生成後に制御を受け取るために、以下の句のいずれかまたは両方を指定できます。

- ON EXCEPTION。XML 生成時にエラーが発生したときに制御を受け取る場合。
- NOT ON EXCEPTION。エラーが発生しなかったときに制御を受け取る場合。

XML GENERATE ステートメントを終了するには、明示範囲終了符号の END-XML を使用します。 条件ステートメントで ON EXCEPTION または NOT ON EXCEPTION 句を指定している XML GENERATE ステートメントをネストするには、END-XML をコーディングします。

XML への COBOL ソース・レコードの変換が完了するか、またはエラーが発生するまで、XML 生成は継続します。 エラーが発生した場合、結果は次のようになります。

- XML-CODE 特殊レジスターには、ゼロ以外の例外コードが含まれます。
- ON EXCEPTION 句が指定されている場合、これに制御が渡されます。指定されていない場合は、XML GENERATE ステートメントの最後に制御が渡されます。

XML 生成時にエラーが発生しなかった場合、XML-CODE 特殊レジスターにはゼロが入り、制御は、NOT ON EXCEPTION 句が指定されている場合はこの句に、指定されていない場合は XML GENERATE ステートメントの最後に渡されます。

415 ページの『例: XML の生成』

関連概念
Uniform Resource Identifier (URI): Generic Syntax

関連タスク
414 ページの『生成される XML 出力のエンコードの制御』
414 ページの『XML GENERATE 例外の処理』
197 ページの『UTF-16 (国別) データ・タイプを使用して UTF-8 データを処理』

関連参照

XML GENERATE ステートメント (*COBOL for Linux on x86* 言語解説書)
Extensible Markup Language (XML)
Namespaces in XML 1.0

生成される XML 出力のエンコードの制御

XML GENERATE ステートメントを使用して XML 出力を生成する場合、出力を受け取るデータ項目のカテゴリーによって、および XML GENERATE ステートメントの WITH ENCODING 句を使用して文書エンコードを指定することによって、出力のエンコードを制御することができます。

WITH ENCODING *codepage* 句を指定する 場合、*codepage* は、COBOL XML 処理に対してサポートされているいずれかのコード・ページを識別するものでなければなりません。 詳しくは、XML 文書のエンコードに関する下記の関連参照を参照してください。 整数となる場合の *codepage* は、有効な CCSID 番号でなければなりません。 クラス英数字または国別となる場合の *codepage* は、International Components for Unicode (ICU) 変換ライブラリーで サポートされるコード・ページ名を識別するものでなければなりません (下記の Converter Explorer 表を参照)。

WITH ENCODING 句をコーディングしなければ、生成される XML 出力は下表のとおりエンコードされます。

表 *43.* ***ENCODING*** **句を省略した場合に生成される** ***XML*** **のエンコード**

受信 XML ID の定義	**生成される XML 出力のエンコード**
ネイティブ英数字 (CHAR(EBCDIC) が有効になっていないか、またはデータ記述に NATIVE 句が含まれる)	有効なランタイム・ロケールによって指定された ASCII または UTF-8 コード・ページ
ホスト英数字 (CHAR(EBCDIC) が有効になっていて、 データ記述に NATIVE 句が含まれない)	有効な EBCDIC コード・ページ [1]
国別	リトル・エンディアン形式の UTF-16

1. EBCDIC_CODEPAGE 環境変数を使用することによって、EBCDIC コード・ページを設定できます。 環境変数が設定されていない場合には、エンコードは、現在のランタイム・ロケールに関連付けられたデフォルトの EBCDIC コード・ページになります。

バイト・オーダー・マークは生成されません。

データ項目が XML へ変換される方法および XML エレメント名および属性名が COBOL データ名から形成される方法に関する詳細については、XML GENERATE ステートメントの操作に関する以下の関連参照を参照してください。

関連タスク
201 ページの『第 11 章 ロケールの設定』
215 ページの『環境変数の設定』

関連参照
256 ページの『CHAR』
398 ページの『XML 文書のエンコード方式』
XML GENERATE ステートメント (*COBOL for Linux on x86* 言語解説書)
XML GENERATE の操作 (*COBOL for Linux on x86* 言語解説書)
International Components for Unicode: Converter Explorer

XML GENERATE 例外の処理

XML 出力の生成時にエラーが検出された場合、例外条件が存在します。 エラー・タイプを示す数値の例外コードが格納される、XML-CODE 特殊レジスターを検査するコードを記述することができます。

エラーを処理するには、XML GENERATE ステートメントの以下の句の一方または両方を使用してください。

- ON EXCEPTION
- COUNT IN

XML GENERATE ステートメントに ON EXCEPTION 句をコーディングした場合、指定された命令ステートメントに制御が転送されます。 命令ステートメントをコーディングして、例えば、XML-CODE 値を表示できます。 ON EXCEPTION 句がコーディングされていない場合、制御は XML GENERATE ステートメントの終わりに移動します。

エラーが発生する場合、XML 出力を受け取るデータ項目が十分大きくないという問題であることがあります。 この場合、XML 出力は不完全となり、XML-CODE 特殊レジスターにエラー・コード 400 が格納されます。

次のステップを実行することにより、生成された XML 出力を検査することができます。

1. XML GENERATE ステートメントに COUNT IN 句をコーディングしてください。

   指定するカウント・フィールドは、XML 生成時に充てんされる XML 文字エンコード・ユニットのカウントを保持します。 XML 出力を国別として定義した場合、カウントは UTF-16 文字エンコード・ユニット数であり、すべての他のエンコードの場合 (UTF-8 の場合を含む)、カウントはバイト数です。
2. カウント・フィールドを参照変更長として使用して、エラー発生時点までに生成された XML 文字を含んだデータ項目を受け取るサブストリングを参照してください。

   例えば、XML-OUTPUT が XML 出力を受け取るデータ項目であり、XML-CHAR-COUNT がカウント・フィールドである場合、XML-OUTPUT(1:XML-CHAR-COUNT) は XML 出力を指します。

XML-CODE の内容を使用して、行うべき修正アクションを判別してください。 XML 生成中に発生する可能性がある例外の一覧については、以下の関連参照を参照してください。

関連タスク
99 ページの『データ項目のサブストリングの参照』

関連参照

587 ページの『XML GENERATE 例外』
XML-CODE (*COBOL for Linux on x86* 言語解説書)

例: XML の生成

以下の例は、グループ・データ項目での購入注文の作成をシミュレートし、その購入注文の XML 版を生成します。

プログラム XGFX は XML GENERATE を使用して、ソース・レコードであるグループ・データ項目 purchaseOrder から基本データ 項目 xmlPO に XML 出力を生成します。 ソース・レコード内の基本データ項目は、必要に応じて文字形式に変換され、変換された文字は、ソース・レコードのデータ名から派生した名前を持つ XML 属性の値として挿入されます。

XGFX はプログラム Pretty を呼び出します。このプログラムは、処理プロシージャー p を指定した XML PARSE ステートメントを 使用しており、XML の内容を一層容易に検査できるよう改行とインデントを 含んだ XML 出力をフォーマット設定するようになっています。

プログラム XGFX

```
Identification division.
  Program-id. XGFX.
Data division.
 Working-storage section.
   01 numItems pic 99 global.
   01 purchaseOrder global.
     05 orderDate pic x(10).
     05 shipTo.
       10 country pic xx value 'US'.
       10 name pic x(30).
       10 street pic x(30).
       10 city pic x(30).
       10 state pic xx.
       10 zip pic x(10).
     05 billTo.
       10 country pic xx value 'US'.
```

```
       10 name pic x(30).
       10 street pic x(30).
       10 city pic x(30).
       10 state pic xx.
       10 zip pic x(10).
     05 orderComment pic x(80).
     05 items occurs 0 to 20 times depending on numItems.
       10 item.
         15 partNum pic x(6).
         15 productName pic x(50).
         15 quantity pic 99.
         15 USPrice pic 999v99.
         15 shipDate pic x(10).
         15 itemComment pic x(40).
   01 numChars comp pic 999.
   01 xmlPO pic x(999).
Procedure division.
  m.
    Move 20 to numItems
    Move spaces to purchaseOrder

    Move '1999-10-20' to orderDate

    Move 'US' to country of shipTo
    Move 'Alice Smith' to name of shipTo
    Move '123 Maple Street' to street of shipTo
    Move 'Mill Valley' to city of shipTo
    Move 'CA' to state of shipTo
    Move '90952' to zip of shipTo

    Move 'US' to country of billTo
    Move 'Robert Smith' to name of billTo
    Move '8 Oak Avenue' to street of billTo
    Move 'Old Town' to city of billTo
    Move 'PA' to state of billTo
    Move '95819' to zip of billTo
    Move 'Hurry, my lawn is going wild!' to orderComment

    Move 0 to numItems
    Call 'addFirstItem'
    Call 'addSecondItem'
    Move space to xmlPO
    Xml generate xmlPO from purchaseOrder  count in numChars
        with xml-declaration  with attributes
        namespace 'http://www.example.com'  namespace-prefix 'po'
    Call 'pretty' using xmlPO value numChars
    Goback
    .

Identification division.
  Program-id. 'addFirstItem'.
Procedure division.
    Add 1 to numItems
    Move '872-AA' to partNum(numItems)
    Move 'Lawnmower' to productName(numItems)
    Move 1 to quantity(numItems)
    Move 148.95 to USPrice(numItems)
    Move 'Confirm this is electric' to itemComment(numItems)
    Goback.
End program 'addFirstItem'.

Identification division.
  Program-id. 'addSecondItem'.
Procedure division.
    Add 1 to numItems
    Move '926-AA' to partNum(numItems)
    Move 'Baby Monitor' to productName(numItems)
    Move 1 to quantity(numItems)
    Move 39.98 to USPrice(numItems)
    Move '1999-05-21' to shipDate(numItems)
    Goback.
End program 'addSecondItem'.

End program XGFX.
```

プログラム Pretty

```
Identification division.
```

```
  Program-id. Pretty.
Data division.
 Working-storage section.
   01 prettyPrint.
     05 pose pic 999.
     05 posd pic 999.
     05 depth pic 99.
     05 inx pic 999.
     05 elementName pic x(30).
     05 indent pic x(40).
     05 buffer pic x(998).
     05 lastitem pic 9.
       88 unknown value 0.
       88 xml-declaration value 1.
       88 element value 2.
       88 attribute value 3.
       88 charcontent value 4.
 Linkage section.
  1 doc.
   2 pic x occurs 16384 times depending on len.
  1 len comp-5 pic 9(9).
Procedure division using doc value len.
  m.
    Move space to prettyPrint
    Move 0 to depth
    Move 1 to posd pose
    Xml parse doc processing procedure p
    Goback
    .
  p.
    Evaluate xml-event
      When 'VERSION-INFORMATION'
        String '<?xml version="' xml-text '"' delimited by size
            into buffer with pointer posd
        Set xml-declaration to true
      When 'ENCODING-DECLARATION'
        String ' encoding="' xml-text '"' delimited by size
            into buffer with pointer posd
      When 'STANDALONE-DECLARATION'
        String ' standalone="' xml-text '"' delimited by size
            into buffer with pointer posd
      When 'START-OF-ELEMENT'
        Evaluate true
          When xml-declaration
            String '?>' delimited by size into buffer
                with pointer posd
            Set unknown to true
            Perform printline
            Move 1 to posd
          When element
            String '>' delimited by size into buffer
                with pointer posd
          When attribute
            String '">' delimited by size into buffer
                with pointer posd
        End-evaluate
        If elementName not = space
          Perform printline
        End-if
        Move xml-text to elementName
        Add 1 to depth
        Move 1 to pose
        Set element to true
        String '<' xml-text delimited by size
            into buffer with pointer pose
        Move pose to posd
      When 'ATTRIBUTE-NAME'
        If element
          String ' ' delimited by size into buffer
              with pointer posd
        Else
          String '" ' delimited by size into buffer
              with pointer posd
        End-if
        String xml-text '="' delimited by size into buffer
            with pointer posd
        Set attribute to true
      When 'ATTRIBUTE-CHARACTERS'
        String xml-text delimited by size into buffer
            with pointer posd
      When 'ATTRIBUTE-CHARACTER'
        String xml-text delimited by size into buffer
```

```
              with pointer posd
        When 'CONTENT-CHARACTERS'
          Evaluate true
            When element
              String '>' delimited by size into buffer
                  with pointer posd
            When attribute
              String '">' delimited by size into buffer
                  with pointer posd
          End-evaluate
          String xml-text delimited by size into buffer
              with pointer posd
          Set charcontent to true
        When 'CONTENT-CHARACTER'
          Evaluate true
            When element
              String '>' delimited by size into buffer
                  with pointer posd
            When attribute
              String '">' delimited by size into buffer
                  with pointer posd
          End-evaluate
          String xml-text delimited by size into buffer
              with pointer posd
          Set charcontent to true
        When 'END-OF-ELEMENT'
          Move space to elementName
          Evaluate true
            When element
              String '/>' delimited by size into buffer
                  with pointer posd
            When attribute
              String '"/>' delimited by size into buffer
                  with pointer posd
            When other
              String '</' xml-text '>' delimited by size
                  into buffer with pointer posd
          End-evaluate
          Set unknown to true
          Perform printline
          Subtract 1 from depth
          Move 1 to posd
        When other
          Continue
      End-evaluate
      .
    printline.
      Compute inx = function max(0 2 * depth - 2) + posd - 1
      If inx > 120
        compute inx = 117 - function max(0 2 * depth - 2)
        If depth > 1
          Display indent(1:2 * depth - 2) buffer(1:inx) '...'
        Else
          Display buffer(1:inx) '...'
        End-if
      Else
        If depth > 1
          Display indent(1:2 * depth - 2) buffer(1:posd - 1)
        Else
          Display buffer(1:posd - 1)
        End-if
      End-if
      .
End program Pretty.
```

プログラム XGFX からの出力

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<po:purchaseOrder xmlns:po="http://www.example.com" orderDate="1999-10-20" orderComment="Hurry, my lawn
is going wild!">
  <po:shipTo country="US" name="Alice Smith" street="123 Maple Street" city="Mill Valley" state="CA"
zip="90952"/>
  <po:billTo country="US" name="Robert Smith" street="8 Oak Avenue" city="Old Town" state="PA"
zip="95819"/>
  <po:items>
    <po:item partNum="872-AA" productName="Lawnmower" quantity="1" USPrice="148.95" shipDate=" "
itemComment="Confirm...
  </po:items>
```

```
  <po:items>
    <po:item partNum="926-AA" productName="Baby Monitor" quantity="1" USPrice="39.98"
shipDate="1999-05-21" itemComme...
  </po:items>
</po:purchaseOrder>
```

関連タスク
391 ページの『第 19 章 XML 入力の処理』

関連参照

XML GENERATE の操作 (*COBOL for Linux on x86* 言語解説書)

XML 出力の拡張

XML フォーマットで表したい情報が既に DATA DIVISION のグループ項目に存在しているが、1 つ以上の要因のためその項目を使用して XML 文書を直接生成できないおそれがあります。

次に例を示します。

- 必要データのほかに、項目には、XML 出力文書とは無関係な値を含んでいる従属データ項目が含まれています。
- 必要データ項目の名前が、外部表示には不適当なものであり、プログラマーにしか意味のないものである可能性があります。
- 必要データ項目があまりに多くのコンポーネントに分割されており、収容グループの内容として出力する必要があります。

こうした状態を取り扱うことのできるさまざまな方法があります。 1 つの手法として考えられるのは、適切な特性を持つデータ項目を新しく定義し、作成した新しいデータ項目の適切なフィールドに必要なデータを移動することです。 しかし、この手法は多少面倒な作業で、元のデータ項目と新しいデータ項目の同期を維持するため注意深い保守作業が必要となります。

こうした問題の多くに対処するための手法として優れているのは、XML GENERATE ステートメントの新しいオプション句を使用して以下を行うことです。

- 選択された基本項目およびそれらの基本項目を含んでいるグループ項目に、もっと意味のある適切な名前を付けます。
- データ項目をその値に基づいて抑制することにより、生成された XML から不適切なデータ項目を除外します。

以下に示す例は、その方法を示しています。

419 ページの『例: XML 出力の拡張』

関連参照
XML GENERATE の操作 (*COBOL for Linux on x86* 言語解説書)

例: XML 出力の拡張

次の例は、どうすれば XML 出力を変更させることができるかを示しています。

以下のデータ構造について考慮してください。 構造から生成される XML には、訂正可能ないくつかの問題が含まれています。

```
01  CDR-LIFE-BASE-VALUES-BOX.
    15  CDR-LIFE-BASE-VAL-DATE    PIC X(08).
    15  CDR-LIFE-BASE-VALUE-LINE  OCCURS  2 TIMES.
        20  CDR-LIFE-BASE-DESC.
            25 CDR-LIFE-BASE-DESC1 PIC X(15).
            25  FILLER             PIC X(01).
            25  CDR-LIFE-BASE-LIT  PIC X(08).
            25  CDR-LIFE-BASE-DTE  PIC X(08).
        20  CDR-LIFE-BASE-PRICE.
            25  CDR-LIFE-BP-SPACE  PIC 9(08).
            25  CDR-LIFE-BP-DASH   PIC X.
```

```
            25  CDR-LIFE-BP-SPACE1 PIC X(02).
        20  CDR-LIFE-BASE-PRICE-ED  REDEFINES
             CDR-LIFE-BASE-PRICE  PIC $$$.$$.
        20  CDR-LIFE-BASE-QTY.
            25  CDR-LIFE-QTY-SPACE   PIC X(08).
            25  CDR-LIFE-QTY-DASH    PIC X.
            25  CDR-LIFE-QTY-SPACE1  PIC X(03).
            25  FILLER               PIC X(02).
        20  CDR-LIFE-BASE-VALUE   PIC $$$9.99
            BLANK WHEN ZERO.
    15  CDR-LIFE-BASE-TOT-VALUE   PIC X(15)
```

このデータ構造にいくつかのサンプル値を取り込み、XML をそれから直接生成し、その後プログラム Pretty (415 ページの『例: XML の生成』に示されています) を使用してフォーマット設定すると、結果は次のようになります。

```
<CDR-LIFE-BASE-VALUES-BOX>
  <CDR-LIFE-BASE-VAL-DATE>01/02/03</CDR-LIFE-BASE-VAL-DATE>
  <CDR-LIFE-BASE-VALUE-LINE>
    <CDR-LIFE-BASE-DESC>
      <CDR-LIFE-BASE-DESC1>First</CDR-LIFE-BASE-DESC1>
      <CDR-LIFE-BASE-LIT> </CDR-LIFE-BASE-LIT>
      <CDR-LIFE-BASE-DTE>01/01/01</CDR-LIFE-BASE-DTE>
    </CDR-LIFE-BASE-DESC>
    <CDR-LIFE-BASE-PRICE>
      <CDR-LIFE-BP-SPACE>23</CDR-LIFE-BP-SPACE>
      <CDR-LIFE-BP-DASH>.</CDR-LIFE-BP-DASH>
      <CDR-LIFE-BP-SPACE1>00</CDR-LIFE-BP-SPACE1>
    </CDR-LIFE-BASE-PRICE>
    <CDR-LIFE-BASE-QTY>
      <CDR-LIFE-QTY-SPACE>123</CDR-LIFE-QTY-SPACE>
      <CDR-LIFE-QTY-DASH>.</CDR-LIFE-QTY-DASH>
      <CDR-LIFE-QTY-SPACE1>000</CDR-LIFE-QTY-SPACE1>
    </CDR-LIFE-BASE-QTY>
    <CDR-LIFE-BASE-VALUE>$765.00</CDR-LIFE-BASE-VALUE>
  </CDR-LIFE-BASE-VALUE-LINE>
  <CDR-LIFE-BASE-VALUE-LINE>
    <CDR-LIFE-BASE-DESC>
      <CDR-LIFE-BASE-DESC1>Second</CDR-LIFE-BASE-DESC1>
      <CDR-LIFE-BASE-LIT> </CDR-LIFE-BASE-LIT>
      <CDR-LIFE-BASE-DTE>02/02/02</CDR-LIFE-BASE-DTE>
    </CDR-LIFE-BASE-DESC>
    <CDR-LIFE-BASE-PRICE>
      <CDR-LIFE-BP-SPACE>34</CDR-LIFE-BP-SPACE>
      <CDR-LIFE-BP-DASH>.</CDR-LIFE-BP-DASH>
      <CDR-LIFE-BP-SPACE1>00</CDR-LIFE-BP-SPACE1>
    </CDR-LIFE-BASE-PRICE>
    <CDR-LIFE-BASE-QTY>
      <CDR-LIFE-QTY-SPACE>234</CDR-LIFE-QTY-SPACE>
      <CDR-LIFE-QTY-DASH>.</CDR-LIFE-QTY-DASH>
      <CDR-LIFE-QTY-SPACE1>000</CDR-LIFE-QTY-SPACE1>
    </CDR-LIFE-BASE-QTY>
    <CDR-LIFE-BASE-VALUE>$654.00</CDR-LIFE-BASE-VALUE>
  </CDR-LIFE-BASE-VALUE-LINE>
  <CDR-LIFE-BASE-TOT-VALUE>Very high!</CDR-LIFE-BASE-TOT-VALUE>
</CDR-LIFE-BASE-VALUES-BOX>
```

生成されたこの XML にはいくつかの問題があります。

- エレメント名が長く、あまり意味のあるものではありません。
- エレメントであるいくつかのフィールド (CDR-LIFE-BASE-VAL-DATE や CDR-LIFE-BASE-DESC1 など) は属性にする必要があります。
- 不要なデータ、例えば、CDR-LIFE-BASE-LIT や CDR-LIFE-BASE-DTE があります。
- 必要データに、不必要な親があります。例えば、CDR-LIFE-BASE-DESC1 には親の CDR-LIFE-BASE-DESC があります。
- 他の必須フィールドが、あまりに多くのサブコンポーネントに分割されています。例えば、CDR-LIFE-BASE-PRICE には、1 つの金額に対して 3 つのサブコンポーネントがあります。

XML 出力のこのような特性は、以下のように XML GENERATE ステートメントの句を追加で使用することにより修正できます。

- NAME OF 句を使用して、適切なタグ名または属性名を指定する。
- TYPE OF … IS ATTRIBUTE 句を使用して、XML エレメントではなく XML 属性にすべきフィールドを選択する。
- TYPE OF … IS CONTENT 句を使用して、余分なサブコンポーネントのタグを抑制する。
- SUPPRESS … WHEN 句を使用して、関心のない値が含まれているフィールドを除外する。

以下は、これらの問題に対処するための XML GENERATE ステートメントの例です。

```
XML generate Doc from CDR-LIFE-BASE-VALUES-BOX
  Count in tally
  Name of
          CDR-LIFE-BASE-VALUES-BOX
       is 'Base_Values'
          CDR-LIFE-BASE-VAL-DATE
       is 'Date'
          CDR-LIFE-BASE-DTE
       is 'Date'
          CDR-LIFE-BASE-VALUE-LINE
       is 'BaseValueLine'
          CDR-LIFE-BASE-DESC1
       is 'Description'
          CDR-LIFE-BASE-PRICE
       is 'BasePrice'
          CDR-LIFE-BASE-QTY
       is 'BaseQuantity'
          CDR-LIFE-BASE-VALUE
       is 'BaseValue'
          CDR-LIFE-BASE-TOT-VALUE
       is 'TotalValue'
  Type of
          CDR-LIFE-BASE-VAL-DATE is attribute
          CDR-LIFE-BASE-DESC1 is attribute
          CDR-LIFE-BP-SPACE   is content
          CDR-LIFE-BP-DASH    is content
          CDR-LIFE-BP-SPACE1  is content
          CDR-LIFE-QTY-SPACE  is content
          CDR-LIFE-QTY-DASH   is content
          CDR-LIFE-QTY-SPACE1 is content
  Suppress every nonnumeric when space
           every numeric when zero
```

上記のステートメントから XML を生成およびフォーマット設定した結果は、より使用に適したものになっています。

```
<Base_Values Date="01/02/03">
  <BaseValueLine Description="First">
    <Date>01/01/01</Date>
    <BasePrice>23.00</BasePrice>
    <BaseQuantity>123.000</BaseQuantity>
    <BaseValue>$765.00</BaseValue>
  </BaseValueLine>
  <BaseValueLine Description="Second">
    <Date>02/02/02</Date>
    <BasePrice>34.00</BasePrice>
    <BaseQuantity>234.000</BaseQuantity>
    <BaseValue>$654.00</BaseValue>
  </BaseValueLine>
  <TotalValue>Very high!</TotalValue>
</Base_Values>
```

現在、出力では COBOL 予約語 DATE を XML タグ名として使用でき、COBOL データ名では使用できない他の文字 (下線 _ など) も使用できることに注意してください。

COBOL 予約語 DATE は現在、出力で XML タグ名として使用できます。 1 バイト・データ名では許可されていない、アクセント記号やピリオド . などの文字も使用できます。

関連参照
XML GENERATE の操作 (*COBOL for Linux on x86* 言語解説書)
REPLACE ステートメント (*COBOL for Linux on x86* 言語解説書)

第 6 部 複雑なアプリケーションを扱う作業

© Copyright IBM Corp. 2021

第 21 章 プラットフォーム間でのアプリケーションの移植

Linux on x86 システムには、IBM Z や IBM Power システムとは異なるハードウェアやオペレーティング・システムのアーキテクチャーが採用されています。 このようなアーキテクチャーの違いのために、COBOL プログラムをこれらのプラットフォーム環境間で移植する際には、いくつかの問題が発生する可能性があります。

IBM Power システムには、IBM Z や Linux on x86 システムとは異なるハードウェアやオペレーティング・システムのアーキテクチャーが採用されています。

以下の関連タスクと関連参照では、開発プラットフォーム間の違いと、移植性の問題を最小限に抑えるための方法について説明します。

関連タスク
425 ページの『IBM Enterprise COBOL for z/OS アプリケーションをコンパイルする』
426 ページの『実行する IBM Enterprise COBOL for z/OS アプリケーションの取得: 概要』
429 ページの『IBM Enterprise COBOL for z/OS で実行されるコード作成』

関連参照
521 ページの『付録 A IBM Enterprise COBOL for z/OS との違いのまとめ』

IBM Enterprise COBOL for z/OS アプリケーションをコンパイルする

Enterprise COBOL プログラムを IBM Z システムから Linux on x86 システムに移行し、IBM COBOL for Linux on x86 を使用してコンパイルする場合は、正しいコンパイラー・オプションを選択して、IBM Enterprise COBOL for z/OS とは異なる言語機能を意識する必要があります。COPY ステートメントを使用して、プログラムの移植に役立てることもできます。

正しいコンパイラー・オプションの選択: 移植性に影響する Enterprise COBOL コンパイラー・オプションについて詳しくは、 コンパイラー・オプションに関する関連参照を参照してください。

Enterprise COBOL の言語機能の考慮: Enterprise COBOL プログラムで複数の言語機能が有効になっている 場合に、COBOL for Linux を使用してコンパイルを行うと、エラーが発生したり、 予期しない結果が生じたりすることがあります。詳しくは、言語エレメントに関する関連参照を参照してください。

移植の問題に役立つ COPY ステートメントの使用: 多くの場合、COPY ステートメントを使用してプラットフォーム固有のコードを分離すると、移植性の問題を回避することができます。 例えば、あるプラットフォーム用のコンパイル時にプラットフォーム固有のコードを組み込み、別のプラットフォーム用のコンパイル時にはそのコードを除外することができます。 また、COPY REPLACING 句を使用して、ファイル名などの移植不可能なソース・コード・エレメントをグローバル変更することも可能です。

関連タスク

215 ページの『環境変数の設定』

関連参照
521 ページの『コンパイラー・オプション』
525 ページの『言語エレメント』
COPY ステートメント (*COBOL for Linux on x86* 言語解説書)

© Copyright IBM Corp. 2021

実行する IBM Enterprise COBOL for z/OS アプリケーションの取得: 概要

Enterprise COBOL プログラムをダウンロードして IBM COBOL for Linux on x86 を使用して正常にコンパイルしたら、次にそのプログラムを実行します。多くの場合、ソースを大幅に変更しなくても、IBM z/OS 上での結果と同じものを取得できます。

ソースに変更を加えるべきかどうかを判断するには、基本的なハードウェアまたはソフトウェア・アーキテクチャーによって異なる COBOL 言語のエレメントや動作の修正方法を理解する必要があります。

関連タスク
426 ページの『データ表現による違いの修正』
428 ページの『移植性に影響する環境の違いの修正』
428 ページの『言語エレメントによる違いの修正』

データ表現による違いの修正

プログラムに同じ動作をさせるためには、特定のデータ表現方法における違いを理解して、適切な処置を取る必要があります。

文字データの表現は、データ項目を記述する USAGE 文節と実行時に有効なロケールによって異なる可能性があります。COBOL では、符号付きパック 10 進数は Linux on x86 と IBM z/OS の両方で同じ方法で格納されます。しかし、2 進数データ、外部 10 進数データ、浮動小数点データ、および符号なしパック 10 進数データは、デフォルトでは異なる方法で表現されます。

大半のプログラムは、データ表現に関係なく、IBM z/OS 上でも、Linux on x86 上でも同じ動作をします。

関連タスク
426 ページの『ASCII SBCS 文字と EBCDIC SBCS 文字の違いの処理』
427 ページの『IEEE データと 16 進数データの違いの処理』
428 ページの『ASCII マルチバイト・ストリングと EBCDIC DBCS ストリングの違いの処理』

関連参照
521 ページの『データ表現』

ASCII SBCS 文字と EBCDIC SBCS 文字の違いの処理

ASCII 文字と EBCDIC 文字のデータ表現の違いによる問題を避けるには、`CHAR(EBCDIC)` コンパイラー・オプションを使用します。

COBOL for Linux on x86 は ASCII 文字セットを使用しますが、Enterprise COBOL for z/OS は EBCDIC 文字セットを使用します。 したがって、大半の文字が、次の表に示すように異なる 16 進値を持ちます。

表 *44.* ***ASCII*** **文字と** ***EBCDIC*** **文字との対比**

文字	**ASCII の場合の 16 進値**	**EBCDIC の場合の 16 進値**
'0' から '9'	X'30' から X'39'	X'F0' から X'F9'
'a'	X'61'	X'81'
'A'	X'41'	X'C1'
ブランク	X'20'	X'40'

また、次の表に示すように、文字データの EBCDIC 16 進値に依存するコードは、文字データが ASCII 値を持つ場合、ほとんどが失敗してしまいます。

表 45. *ASCII* での比較と *EBCDIC* での比較の対比		
比較	**ASCII の場合の評価**	**EBCDIC の場合の評価**
'a' < 'A'	偽	真
'A' < '1'	偽	真
x >= '0'	真の場合、*x* が数字であるかどうかは示されない	真である場合、*x* はおそらく数字
x = X'40'	*x* がブランクかどうかはテストされない	*x* かどうかがテストされる

このような違いがあるため、文字ストリングのソート結果は EBCDIC と ASCII では異なります。 多くのプログラムでは、これらの違いによる影響はありませんが、プログラムが一部の文字ストリングの正確なソート順序に依存する場合は、論理エラーの可能性にも注意する必要があります。 EBCDIC 照合シーケンスに依存するプログラムをワークステーションに移植する場合は、PROGRAM COLLATING SEQUENCE IS EBCDIC または COLLSEQ(EBCDIC) コンパイラー・オプションを使用して、EBCDIC 照合シーケンスを取得することができます。

関連参照
256 ページの『CHAR』
258 ページの『COLLSEQ』

IEEE データと 16 進数データの違いの処理

IEEE と 16 進数の浮動小数点データ間の表現の違いによる一般的な問題を回避するには、FLOAT(BE) コンパイラー・オプションを使用します。

COBOL for Linux on x86 は、IEEE 形式を使用して浮動小数点データを表現します。Enterprise COBOL for z/OS は、IBM Z 16 進形式を使用します。次の表に、USAGE COMP-1 データと USAGE COMP-2 データに対する、正規化浮動小数点 IEEE と正規化 16 進数の違いをまとめます。

表 46. *IEEE* と *16* 進数の対比				
仕様	**COMP-1 データの IEEE**	**COMP-1 データの 16 進数**	**COMP-2 データの IEEE**	**COMP-2 データの 16 進数**
範囲	1.17E-38* から 3.37E+38*	5.4E-79* から 7.2E+75*	2.23E-308* から 1.67E+308*	5.4E-79* から 7.2E+75*
指数表現	8 ビット	7 ビット	11 ビット	7 ビット
小数部表現	23 ビット	24 ビット	53 ビット	56 ビット
正確性のある桁数	6 桁	6 桁	15 桁	16 桁
* 値は正数でも負数でも構いません。				

大半のプログラムでは、これらの違いによる問題は発生しません。 ただし、データの 16 進数表現に依存するプログラムの場合は、移植時に注意が必要です。

パフォーマンスについての考慮事項: 一般に、IBM Z の浮動小数点表記を使用すると、ソフトウェアが IBM Z のハードウェア命令のセマンティクスをシミュレートする必要があるため、プログラムの実行速度が遅くなります。 これは、特に FLOAT(BE) コンパイラー・オプションが有効で、なおかつプログラムに多数の浮動小数点計算がある場合の考慮事項です。

41 ページの『例: 数値データおよび内部表現』

関連参照
269 ページの『FLOAT』

ASCII マルチバイト・ストリングと EBCDIC DBCS ストリングの違いの処理

DBCS 文字を含む英数字データ項目に対する Enterprise COBOL の動作を取得するには、CHAR(EBCDIC) および SOSI コンパイラー・オプションを使用します。ASCII DBCS 文字と EBCDIC DBCS 文字のデータ表現の違いによる問題を避けるには、CHAR(EBCDIC) コンパイラー・オプションを使用します。

英数字データ項目では、Enterprise COBOL の 2 バイト文字ストリング (EBCDIC DBCS 文字を含む) はシフト・コードで囲まれますが、COBOL for Linux on x86 のマルチバイト文字ストリング (ASCII DBCS、UTF-8、または EUC 文字を含む) はシフト・コードでは囲まれません。また、同じ文字の表現に使用される 16 進値も異なります。

DBCS データ項目で、Enterprise COBOL の 2 バイト文字ストリングはシフト・コードで囲まれませんが、文字を表すために使用される 16 進値は、COBOL for Linux on x86 のマルチバイト・ストリングで同じ文字を表すために使用される 16 進値とは異なります。

大半のプログラムでは、これらの違いがあっても移植に問題はありません。 ただし、プログラムが、マルチバイト・ストリングの 16 進値に依存する場合や、英数字ストリングに 1 バイト文字とマルチバイト文字が混在することを期待する場合は、コーディングの方法に注意してください。

関連参照
256 ページの『CHAR』
280 ページの『SOSI』

移植性に影響する環境の違いの修正

Linux on x86 と IBM z/OS プラットフォーム間におけるファイル名や制御コードの違いは、プログラムの移植に影響する可能性があります。

Linux on x86 のファイル命名規則は、IBM z/OS の命名規則とはかなり異なります。COBOL ソース・プログラム内でファイル名を使用する場合は、この違いが移植性に影響を与える可能性があります。 例えば、次のファイル名は、Linux on x86 上では有効ですが、IBM z/OS (z/OS UNIX ファイル・システム 以外) 上では無効です。

```
/users/joesmith/programs/cobol/myfile.cbl
```

大/小文字の区別: z/OS とは異なり、Linux は大/小文字を区別します。ソース・プログラムで使用される名前 (大文字のファイル名など) は、Linux ファイル・ディレクトリー内で適切に名前を付ける必要があります。

z/OS 上で特定の意味を持たない一部の文字は、Linux では制御文字として解釈されます。この違いにより、ASCII テキスト・ファイルが誤って処理される可能性があります。 ファイルには、次のどの文字も含めないようにしてください。

- X'0A' (LF: 改行)
- X'0D' (CR: 復帰)
- X'1A' (EOF: ファイルの終わり)

装置依存 (プラットフォーム固有) の制御コードをプログラムまたはファイル内に使用する場合は、これらの制御コードをサポートしていないプラットフォームにそのプログラムまたはファイルを移植しようとすると、問題が生じる可能性があります。 他のプラットフォーム固有コードについても同様ですが、このようなコードはできるだけ分離して、アプリケーションを別のプラットフォームに移行する際に、容易にコードを置き換えられるようにすることが得策です。

言語エレメントによる違いの修正

一般に、移植可能な COBOL プログラムは、Linux 上でも z/OS 上と同様に動作することが期待できます。ただし、I/O 処理で使用されるファイル状況値の違いには注意してください。

プログラムがファイル状況データ項目に応答する場合は、最初のファイル状況データ項目と 2 番目のファイル状況データ項目のどちらに応答するかによって、次の 2 つの問題に注意してください。

- プログラムが最初のファイル状況データ項目 (*data-name-1*) に応答する場合は、9*n* の範囲内で返される値がプラットフォームによって異なることに注意してください。 プログラムが特定の 9*n* 値 (例えば 97) の解釈に依存する場合は、その語の持つ意味が Linux 上と z/OS 上とでは異なる可能性があります。この場合は、一般的な I/O エラーとして任意の 9*n* 値にプログラムが応答するように修正してください。
- プログラムが 2 番目のファイル状況データ項目 (*data-name-8*) に応答する場合は、返される値がプラットフォームとファイル・システムの両方によって異なることに注意してください。 例えば、STL ファイル・システムは、Linux 上では、z/OS 上の VSAM ファイル・システムとはレコード構造の異なる値を返します。2 番目のファイル状況データ項目の解釈に依存するプログラムは、たいていは移植できません。

関連タスク
170 ページの『ファイル状況キーの使用』
172 ページの『ファイル・システム状況コードの使用』

関連参照
FILE STATUS 節 (*COBOL for Linux on x86* 言語解説書)
ファイル状況キー (*COBOL for Linux on x86* 言語解説書)

IBM Enterprise COBOL for z/OS で実行されるコード作成

IBM COBOL for Linux on x86 を使用して新しいアプリケーションを開発すると、Linux on x86 システムを使用した場合の生産性と柔軟性の向上を活用できます。ただし、COBOL プログラムの開発時は、IBM Enterprise COBOL for z/OS でサポートされていない機能は使用しないようにする必要があります。

言語機能: COBOL for Linux がサポートしている言語機能の中には、Enterprise COBOL では対応していないものもあります。z/OS 上で実行するコードを Linux on x86 上で作成する場合は、次の機能を使用しないでください。

- DISPLAY-OF および NATIONAL-OF 組み込み関数への引数としてのコード・ページ名
- PREVIOUS 句を使用した READ ステートメント
- KEY 句で <、<=、または NOT > を使用した START ステートメント
- >>CALLINTERFACE コンパイラー指示

コンパイラー・オプション: 下の各コンパイラー・オプションは、Enterprise COBOL では使用できません。コードを z/OS に移植する場合は、ソース・コードに次のどのコンパイラー・オプションも使用しないでください。

- BINARY(NATIVE)
- CALLINT (コメントとして扱われる)
- CHAR(NATIVE)
- FLOAT(NATIVE)

ファイル名: Linux とホスト・ファイル・システムではファイルの命名規約が異なることに注意してください。 ソース・プログラム内でファイル名をハードコーディングすることは避けてください。 この代わりに、各プラットフォーム上で定義する簡略名を使用して、これをメインフレームの DD 名または環境変数にマップします。 その後、プログラムをコンパイルして、ソース・コードに変更を加えずにファイル名の変更に対応することができます。

特に、次の言語エレメントでのファイル参照方法を検討してください。

- ACCEPT または DISPLAY ターゲット名
- ASSIGN 文節
- COPY ステートメント (*text-name* または *library-name*)

ファイル・サフィックス: COBOL for Linux では、cob2 コマンドの 1 つを使用してコンパイルすると、サフィックスが .cbl または .cob の COBOL ソース・ファイルがコンパイラーに渡されます。 メインフレーム COBOL では、z/OS UNIX ファイル・システム 内でコンパイルすると、サフィックスが .cbl のファイルのみがコンパイラーに渡されます。

ネストされたプログラム: メインフレーム上のマルチスレッド・プログラムは再帰的でなければなりません。 したがって、プログラムをメインフレームに移植して、それをマルチスレッド環境で実行できるようにしたい場合は、ネストされたプログラムをコーディングしないでください。

第 22 章 サブプログラムの使用

多くのアプリケーションは、ともにリンクされた別々にコンパイルされたプログラムから構成されます。各プログラムが相互に呼び出す場合は、これらは通信できる必要があります。 これらのプログラムは制御権を渡す必要があり、また通常は共通データに対するアクセス権を必要とします。

COBOL プログラムどうしが互いの内部にネストされている場合は、プログラム間でのやり取りも可能です。 1 つのアプリケーションに必要なサブプログラムをすべて 1 つのソース・ファイルに含めることができるため、必要なコンパイルは 1 回だけで済みます。

関連概念
431 ページの『メインプログラム、サブプログラム、および呼び出し』

関連タスク
431 ページの『メインプログラムまたはサブプログラムの終了と再入』
432 ページの『ネストされた COBOL プログラムの呼び出し』
436 ページの『ネストなし COBOL プログラムの呼び出し』
438 ページの『COBOL および C/C++ プログラム間の呼び出し』
444 ページの『再帰呼び出しの実行』

メインプログラム、サブプログラム、および呼び出し

COBOL プログラムが実行単位の最初のプログラムであれば、その COBOL プログラムはメインプログラムです。 それ以外の場合は、そのプログラムも実行単位内の他のすべての COBOL プログラムも、サブプログラム です。 特定のソース・コードのステートメントまたはオプションが、COBOL プログラムをメインプログラムまたはサブプログラムとして識別することはありません。

COBOL プログラムがメインプログラムであるかサブプログラムであるかは、次の 2 つの理由により重要になることもあります。

- プログラム終了処理ステートメントの影響
- 戻った後に再入する際のそのプログラムの状態

PROCEDURE DIVISION で、あるプログラムから別のプログラム (通常サブプログラム と呼ばれる) を呼び出すことができ、この呼び出し先プログラム自体からも別のプログラムを呼び出すことができます。 別のプログラムを呼び出すプログラムは呼び出し側 プログラムと呼ばれ、そのプログラムが呼び出すプログラムは呼び出し先 プログラムと呼ばれます。 呼び出し先プログラムの処理が完了すると、そのプログラムは制御権を呼び出し側プログラムに戻すか、実行単位を終了することができます。

呼び出し先 COBOL プログラムは、PROCEDURE DIVISION の先頭で実行を開始します。

関連タスク
431 ページの『メインプログラムまたはサブプログラムの終了と再入』
432 ページの『ネストされた COBOL プログラムの呼び出し』
436 ページの『ネストなし COBOL プログラムの呼び出し』
438 ページの『COBOL および C/C++ プログラム間の呼び出し』
444 ページの『再帰呼び出しの実行』

メインプログラムまたはサブプログラムの終了と再入

プログラムが最後に使用された状態になるのかまたは初期状態になるのか、および戻り先がどの呼び出し元になるのかは、使用するステートメントにより異なる可能性があります。

メインプログラムで実行を終了させるには、メインプログラムで、STOP RUN または GOBACK ステートメントをコーディングする必要があります。 STOP RUN は実行単位を終了して、メインプログラムおよびメインプログラムに呼び出されたサブプログラムによって開かれたすべてのファイルを閉じます。 制御は、メインプログラムの呼び出し元に返されます。この呼び出し元は多くの場合、オペレーティング・システ

© Copyright IBM Corp. 2021

ムです。 GOBACK は、メインプログラムで同じ効果を発揮します。 メインプログラムで実行された EXIT PROGRAM には効果がありません。

EXIT PROGRAM、GOBACK、または STOP RUN ステートメントを使用して、サブプログラムを終了することができます。 EXIT PROGRAM または GOBACK ステートメントを使用した場合には、制御は、実行単位を終了させることなく、サブプログラムの直接の呼び出し元に戻ります。 呼び出されるプログラムの中に次に実行可能なステートメントがない場合、暗黙の EXIT PROGRAM ステートメントが生成されます。 サブプログラムを STOP RUN ステートメントを使用して終了させた場合には、その効果は、メインプログラムの場合と同じです。 実行単位のすべての COBOL プログラムが終了し、制御はメインプログラムの呼び出し元に戻ります。

サブプログラムは通常、EXIT PROGRAM または GOBACK で 終了されると、最後に使用された状態になります。 次回にサブプログラムが実行単位内で呼び出されると、PERFORM ステートメントの戻り値が初期値にリセットされることを除けば、その内部値は終了時のまま残されます (それに対して、メインプログラムは呼び出されるたびに初期化されます)。

プログラムが初期状態になるのは、次のような場合です。

- 動的に呼び出され、その後取り消されるサブプログラムは、次に呼び出されると初期状態になります。
- PROGRAM-ID 段落に INITIAL 節を持つプログラムは、呼び出されるたびに初期状態になります。
- LOCAL-STORAGE SECTION で定義されているデータ項目は、プログラムが呼び出されるたびに、VALUE 節で指定されている初期状態にリセットされます。

関連概念
11 ページの『WORKING-STORAGE と LOCAL-STORAGE の比較』

関連タスク
432 ページの『ネストされた COBOL プログラムの呼び出し』
444 ページの『再帰呼び出しの実行』

ネストされた COBOL プログラムの呼び出し

ネストされたプログラムを呼び出すことによって、構造化プログラミング技法を使用したアプリケーションを作成することができます。 また、データ項目を不用意に変更しないようにするために、PERFORM プロシージャーの代わりに、ネストされたプログラムを呼び出すことができます。

ネストされたプログラムの呼び出しには、CALL *literal* ステートメントまたは CALL *identifier* ステートメントを使用します。

ネストされた プログラムは、 それを直接収容するプログラムからのみ呼び出すことができます。ただし、ネストされた プログラムをその PROGRAM-ID 段落で COMMON として識別している場合は別です。 その場合、共通プログラム は、共通プログラムと同じプログラム内の任意の (直接的または間接的) ネストされたプログラムから呼び出すことができます。 COMMON として識別できるのは、ネストされた プログラムだけです。 再帰呼び出しはできません。

ネストされたプログラム構造を使用する場合は、次のガイドラインに従ってください。

- 各プログラムに IDENTIFICATION DIVISION をコーディングします。 他の部はすべてオプションです。
- 状況に応じて、それぞれの ネストされた プログラムの名前を固有にしてください。 ネストされた プログラムの名前は (名前の有効範囲に関する関連参照で記述されているように) 固有である必要はありませんが、 名前を固有にしておくと、アプリケーションの保守が一層容易になります。 ネストされた プログラムの名前として、任意の有効なユーザー定義語または英数字リテラルを使用できます。
- 必要となる可能性のある CONFIGURATION SECTION 記入項目は、最外部のプログラムでコーディングします。 ネストされた プログラムが CONFIGURATION SECTION を持つことはできません。
- それぞれの ネストされた プログラムを収容プログラム内に組み込む場合、 収容プログラムの END PROGRAM マーカーの直前に組み込んでください。

- END PROGRAM マーカーを使用して、ネストされた プログラムと収容しているプログラムを終了させます。

関連概念
433 ページの『ネストされたプログラム』

関連参照

435 ページの『名前の有効範囲』

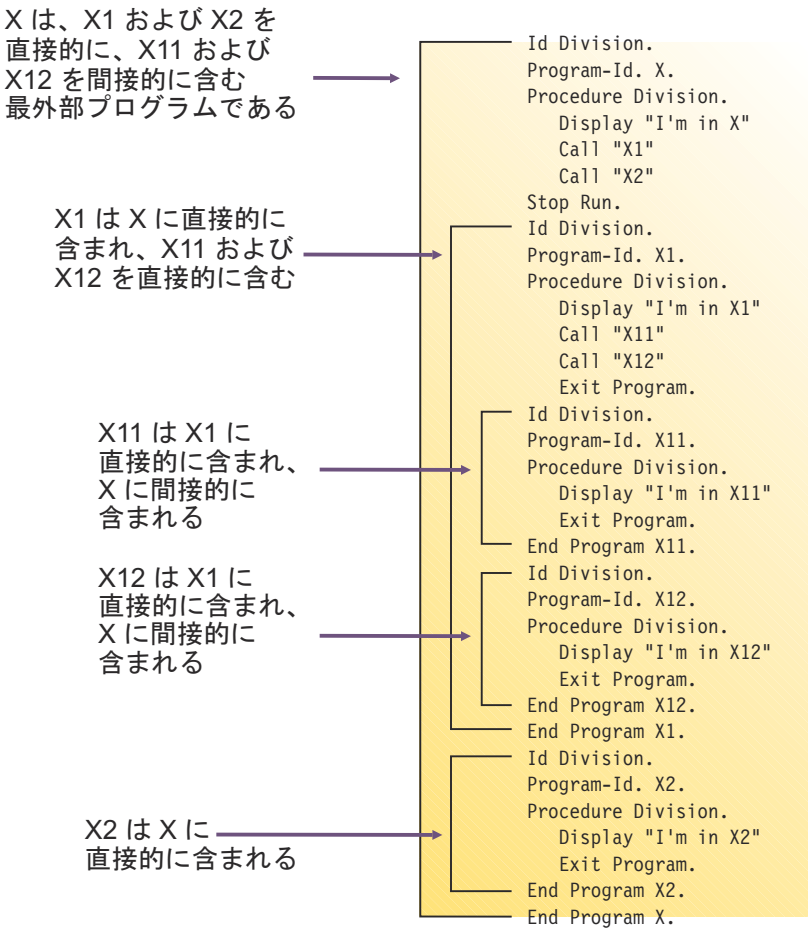
ネストされたプログラム

COBOL プログラムには、他の COBOL プログラムをネスト すること、つまり、他の COBOL プログラムを含めることができます。 ネストされたプログラム自体にも他のプログラムを含められます。 ネストされたプログラムは、プログラムに直接的に含めることも、間接的に含めることもできます。

呼び出されるプログラムをネストすることには、主に次の 4 つの長所があります。

- ネストされたプログラムは、モジュラー機能の作成と構造化プログラミング手法の保守を行う方法を提供します。 これらは、(PERFORM ステートメントを使用して) プロシージャーを実行するために同じように使用することができます。ただし、制御フローはより構造化されたものになり、ローカル・データ項目を保護することができます。
- ネストされたプログラムにより、プログラムをアプリケーションに組み込む前にデバッグすることができます。
- ネストされたプログラムを使用すると、コンパイラーを一度起動するだけでアプリケーションをコンパイルできます。
- COBOL CALL ステートメントのさまざまな形式の中で、ネストされたプログラムへの呼び出しは最高のパフォーマンスを発揮します。

次の例は、直接的および間接的に含まれたプログラムのあるネストされた構造を説明したものです。



434 ページの『例: ネストされたプログラムの構造』

関連タスク
432 ページの『ネストされた COBOL プログラムの呼び出し』

関連参照
435 ページの『名前の有効範囲』

例: ネストされたプログラムの構造

次の例は、一部のネストされたプログラムが COMMON として識別された、ネストされた構造を示しています。

```
Program-Id. A.
  Program-Id. A1.
    Program-Id. A11.
      Program-Id. A111.
      End Program A111.
    End Program A11.
    Program-Id. A12 is Common.
    End Program A12.
  End Program A1.
  Program-Id. A2 is Common.
  End Program A2.
  Program-Id. A3 is Common.
  End Program A13.
End Program A.
```

次の表に、上記の例で示した構造の呼び出し階層について説明しています。 プログラム A12、A2、および A3 は COMMON として識別されており、それらに関連する呼び出しはそれぞれ異なります。

対象となるプログラム	対象プログラムが呼び出せるプログラム	対象プログラムを呼び出せるプログラム
A	A1、A2、A3	なし
A1	A11、A12、A2、A3	A
A11	A111、A12、A2、A3	A1
A111	A12、A2、A3	A11
A12	A2、A3	A1、A11、A111
A2	A3	A、A1、A11、A111、A12、 A3
A3	A2	A、A1、A11、A111、A12、 A2

この例では、次の点に注目してください。

- A2 は A1 を呼び出すことはできません。A1 は共通ではなく、A2 に含まれていないからです。
- A1 は A2 を呼び出すことができます。A2 は共通であるからです。

名前の有効範囲

ネストされた構造における名前は、ローカルとグローバルの 2 つのクラスに分けられます。 名前を宣言しているプログラムの有効範囲を超えてその名前が知られているかどうかが、クラスによって判別されます。プログラム内で名前が参照された後で、特定の検索シーケンスで名前の宣言を見つけます。

ローカル名

他に宣言されていない限り、名前はローカルです (プログラム名を除く)。 ローカル名は、それが宣言されているプログラム内からのみ見る、またはアクセスすることができます。 含まれているプログラムおよび収容プログラムが、ローカル名を見たり、アクセスすることはできません。

グローバル名

グローバルである (GLOBAL 節を使用して示された) 名前は、その名前が宣言されているプログラムと、そのプログラムに直接的および間接的に含まれているすべてのプログラムから可視でありアクセス可能です。 したがって、含まれているプログラムは、単に項目の名前を参照するだけで、収容プログラムからの共通データおよびファイルを共用することができます。

グローバル項目に従属するすべての項目 (条件名と指標を含む) は、自動的にグローバルになります。

各宣言が異なるプログラムの中で現れるのであれば、GLOBAL 節を使用して同じ名前を複数回宣言することができます。 同じ収容構造の異なるプログラムに同じ名前を持たせることによって、ネストされた構造における名前のマスキングや隠蔽が可能であることに注意してください。 ただし、このようなマスキングによって、名前宣言の検索時に問題が生じることがあります。

名前の宣言の探索

プログラム内で名前が参照されると、その名前の宣言を見つける探索が行われます。 探索は、参照が含まれるプログラムで始まり、一致する名前が見つかるまで、順番に収容プログラムへ移って外側へ続けられます。 検索は次のプロセスに従います。

1. そのプログラム内の宣言が検索されます。
2. 一致する名前が見つからない場合は、連続する外側の収容プログラムの中で、グローバル宣言だけが検索されます。
3. 一致する最初の名前が検出されると、検索は終了します。 一致が検出されない場合、エラーが存在します。

検索はグローバル名に関するものであり、データ項目やファイル結合子などの名前に関連した特定の型のオブジェクトに関するものではありません。 オブジェクトの型に関係なく、何らかの一致する名前が見つ

かると探索は停止します。 宣言されたオブジェクトが予期されたものと違う場合は、エラー状態が存在します。

ネストなし COBOL プログラムの呼び出し

COBOL プログラムは、呼び出し元と同じ実行可能モジュールにリンクされた (スタティック・リンク) サブプログラムか、または 共用ライブラリー で提供された (ダイナミック・リンク) サブプログラムを呼び出すことができます。 COBOL for Linux では、共用ライブラリー からターゲット・サブプログラムを実行時に解決することができます。

ターゲット・プログラムを静的にリンクする場合は、プログラムが呼び出し元の実行可能モジュールの一部になり、呼び出し元とともにロードされます。 ターゲット・プログラムを動的にリンクするか、実行時に呼び出しを解決する場合は、ターゲット・プログラムがライブラリー内に提供され、呼び出し元のロード時かターゲット・プログラムの呼び出し時にロードされます。

COBOL の CALL *literal* に対してサブプログラムの動的または静的リンクが実行されます。 実行時の解決は、COBOL CALL *identifier* に対しては常に実行され、CALL *literal* に対しては DYNAM オプションが有効な場合に実行されます。

制限: 1 つのアプリケーションで 32 ビット COBOL プログラムと 64 ビット COBOL プログラムを混用することはできません。 アプリケーション内のプログラム・コンポーネントはすべて、同じ ADDR コンパイラー・オプション設定を使用してコンパイルしなければなりません。

関連概念
436 ページの『CALL identifier および CALL literal』
463 ページの『スタティック・リンクおよび共用ライブラリーの使用』

関連参照
252 ページの『ADDR』
264 ページの『DYNAM』
CALL ステートメント (*COBOL for Linux on x86* 言語解説書)

CALL identifier および CALL literal

CALL *identifier* を使用すると、呼び出し時に常にターゲット・プログラムがロードされます。この *identifier* には、実行時にネストなしサブプログラム名を含むデータ項目が入ります。 CALL *literal* は、静的または動的に解決することができます。この *literal* には、ネストなしターゲット・サブプログラムの明示的な名前が入ります。

CALL *identifier* については、実行可能モジュールまたは 共用ライブラリー の名前が、ターゲット入り口点の名前と一致している必要があります。

CALL *literal* については、NODYNAM コンパイラー・オプションが有効な場合は、スタティック・リンクとダイナミック・リンクのどちらでも実行できます。 DYNAM が有効な場合は、CALL *literal* が CALL *identifier* と同じ方法で解決されます。つまり、ターゲット・サブプログラムは呼び出し時にロードされ、実行可能モジュールの名前とターゲット入り口点の名前が一致していなければなりません。

これらの呼び出し定義は、COBOL プログラムがネストなしプログラムを呼び出す場合にのみ適用されます。 COBOL プログラムがネストされたプログラムを呼び出す場合は、この呼び出しは、システムが介入せずに、コンパイラーによって解決されます。

制約事項: アプリケーション内の 2 つ以上の別々にリンクされた実行可能モジュール は、同一のネストなしサブプログラムを静的に呼び出さないでください。

関連概念
463 ページの『スタティック・リンクおよび共用ライブラリーの使用』

関連参照
264 ページの『DYNAM』
CALL ステートメント (*COBOL for Linux on x86* 言語解説書)

例: CALL identifier を使用した動的呼び出し

次の例は、CALL *identifier* を使用する動的呼び出しを行う方法を示しています。

最初のプログラム dl1.cbl は 、CALL *identifier* を使用して 2 番目のプログラム dl1a.cbl を呼び出します。

dl1.cbl

```
* Simple dynamic call to dl1a

 Identification Division.
 Program-id.    dl1.
*
 Environment Division.
 Configuration Section.
 Input-Output Section.
 File-control.
*
 Data Division.
 File Section.
 Working-storage Section.
 01 var pic x(10).
 Linkage Section.
*
 Procedure Division.
     move "Dl1A" to var.
     display "Calling " var.
     call var.
     move "dl1a     " to var.
     display "Calling " var.
     call var.
     stop run.
 End program dl1.
```

dlla.cbl

```
* Called by dl1.cbl using CALL identifier.

 IDENTIFICATION DIVISION.
 PROGRAM-ID. dl1a.
*
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 OBJECT-COMPUTER. ANY-THING.
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 77 num pic 9(4) binary value is zero.
*
 PROCEDURE DIVISION.
 LA-START.
     display "COBOL DL1A function." upon console.
     add 11 to num.
     display "num = " num
     goback.
```

プロシージャー

上記の例を作成して実行するには、次の手順を実行します。

1. cob2 dl1.cbl -o dl1 と入力して、実行可能モジュール dl1 を生成します。
2. cob2 dl1a.cbl -o DL1A と入力して、実行可能モジュール DL1A を生成します。

   PGMNAME(MIXED) オプションを使用してコンパイルしない限り、実行可能モジュールのプログラム名は大文字に変換されます。
3. コマンド export COBPATH=. を入力して、現行ディレクトリーが動的呼び出しのターゲットの検索場所となるようにします。

4. dl1 と入力してプログラムを実行します。

CALL *identifier* のターゲットは呼び出し先プログラムの名前と一致する必要があるため、上の例の実行可能モジュールは、dl1a としてではなく、DL1A として生成されています。

関連参照
276 ページの『PGMNAME』

COBOL および C/C++ プログラム間の呼び出し

C/C++ で記述された関数は、COBOL プログラムから呼び出すことができ、また、COBOL プログラムを C/C++ 関数から呼び出すこともできます。

言語間アプリケーションでは、64 ビット COBOL プログラムを 64 ビット C/C++ 関数と 結合したり、32 ビット COBOL プログラムを 32 ビット C/C++ 関数と結合したりできます。

制約事項 :

- 1 つのアプリケーションで 32 ビット・コンポーネントと 64 ビット・コンポーネントを混用することはできません。
- ADDR(64) オプションと -q64 オプションは現在サポートされていません。現時点で作成できるのは、32 ビット COBOL プログラムのみです。

COBOL と C++ の間で行われる言語間通信: COBOL と C++ を混用する言語間アプリケーションでは、 以下のガイドラインに従ってください。

- C++ から呼び出される COBOL プログラムの関数プロトタイプと、COBOL から呼び出される C++ 関数に、extern "C" を指定します。
- COBOL では、通常の C++ パラメーター規約に適合するように BY VALUE パラメーターを使用します。
- C++ では、COBOL BY REFERENCE 規約に適合するように参照パラメーターを使用します。

以下で参照される規則とガイドラインでは、これらの言語間呼び出しの実行方法について詳しく説明します。

参照される 各セクションで「C/C++」を無条件で参照する場合、 参照先は GNU GCC コンパイラーです。

関連タスク
386 ページの『CICS での COBOL および C/C++ 間の呼び出し』
438 ページの『環境の初期設定』
439 ページの『COBOL と C/C++ 間でのデータの受け渡し』
439 ページの『スタック・フレームの縮小と実行単位またはプロセスの終了』
447 ページの『第 23 章 データの共用』

関連参照
252 ページの『ADDR』
440 ページの『COBOL および C/C++ のデータ型』

環境の初期設定

C/C++ と COBOL の間で呼び出しを行うには、ターゲット環境を適切に初期設定する必要があります。

C/C++ で記述されたメインプログラムが、COBOL プログラムに対して複数の呼び出しを行う場合は、次のいずれかの方法を使用します。

- COBOL プログラムを呼び出す前に、C/C++ プログラムに COBOL 環境を事前初期設定します。 これにより最適なパフォーマンスが得られるため、この方法をお勧めします。
- COBOL プログラムを、COBOL を呼び出す C/C++ ルーチンの一部ではない実行可能ファイル内に入れます。 その後で、COBOL メインプログラムを呼び出すたびに、C/C++ プログラムで以下のステップを実行します。

  1. プログラムをロードします。

2. プログラムを呼び出します。
3. プログラムをアンロードします。

関連概念
469 ページの『第 25 章 COBOL ランタイム環境の事前初期設定』

COBOL と C/C++ 間でのデータの受け渡し

COBOL データ型には、C/C++ データ型と同じものがありますが、同じでないものもあります。 COBOL プログラムと C/C++ の関数間でデータを受け渡す場合は、データ交換を適切なデータ型のみで行うようにしてください。

デフォルトでは、COBOL は、引数を BY REFERENCE で受け渡します。 引数 BY REFERENCE を 受け渡す場合、C/C++ はその引数を指すポインターを取得します。 引数を BY VALUE で受け渡す場合は、COBOL は実引数を渡します。 BY VALUE は、次のデータ型にしか使用できません。

- 英数字
- USAGE NATIONAL 文字
- BINARY
- COMP
- COMP-1
- COMP-2
- COMP-4
- COMP-5
- FUNCTION-POINTER
- POINTER
- PROCEDURE-POINTER

441 ページの『例: C 関数を呼び出す COBOL プログラム』
442 ページの『例: COBOL プログラムによって呼び出される C プログラムと COBOL を呼び出す C/C++ プログラム』
443 ページの『例: C++ 関数を呼び出す COBOL プログラム』

関連タスク
447 ページの『第 23 章 データの共用』

関連参照

440 ページの『COBOL および C/C++ のデータ型』

スタック・フレームの縮小と実行単位またはプロセスの終了

ある言語で関数を呼び出すと、別の言語のプログラム・スタック・フレームが縮小してしまうような呼び出しは避けてください。

このガイドラインに該当する状況例を次に示します。

- ある言語で記述された一部のアクティブ・スタック・フレームを縮小する場合に、縮小されるスタック・フレーム内に別の言語で記述されたアクティブ・スタック・フレーム (C/C++ longjmp()) があるような場合。
- ある言語で記述されたスタック・フレームがアクティブなときに、COBOL の STOP RUN や C/C++ の exit() または _exit() などを発行すると、別の言語で記述された実行単位またはプロセスが終了します。 このような場合は、プログラムの呼び出し側に戻ることで、呼び出されたプログラムが終了するような方法で、アプリケーションを構築してください。

C/C++ の longjmp()、COBOL の STOP RUN、C/C++ の exit() または _exit() 呼び出しが使用できるのは、操作を開始する言語以外の言語で記述されたアクティブ・スタック・フレームが縮小されない場合に限られます。 縮小や終了を開始しない言語の場合は、その他に次のような悪影響が生じる可能性があります。

- 当該言語の通常の cleanup または exit 関数 (実行単位終了時の COBOL によるファイルのクローズ、強制終了された言語による動的獲得リソースのクリーンアップなど) が実行されない可能性があります。
- ユーザー指定の出口または関数 (デストラクターや C/C++ の atexit() 関数など) が出口や終了に対して呼び出されない可能性があります。

一般に、スタック・フレームの実行中に発生した例外は、その例外を招いた言語の規則に従って処理されます。 COBOL のインプリメンテーションで COBOL 言語セマンティクスをサポートする場合は、システム・サービスを介した例外の代行受信に依存しないため、TRAP(OFF) ランタイム・オプションを指定して、非 COBOL 言語の例外処理セマンティクスを有効にすることができます。

COBOL for Linux は、COBOL ランタイム環境の初期設定時の例外環境を保存し、COBOL 環境の終了時にはその例外環境を復元します。 COBOL では、インターフェース言語とツールが同じ規約に従うことを期待します。

関連参照
302 ページの『TRAP』

COBOL および C/C++ のデータ型

次の表に、COBOL および C/C++ で使用可能なデータ型の対応を示します。

表 *47.* ***COBOL* および *C/C++* のデータ型**

C/C++ データ型	**COBOL データ・タイプ**
wchar_t	USAGE NATIONAL (PICTURE N)
char	PIC X
signed char	相当する COBOL データ型なし
unsigned char	相当する COBOL データ型なし
short signed int	PIC S9-S9(4) COMP-5。 TRUNC(BIN) コンパイラー・オプションを使用した場合は、COMP、COMP-4、または BINARY が使用可能。
short unsigned int	PIC 9-9(4) COMP-5。 TRUNC(BIN) コンパイラー・オプションを使用した場合は、COMP、COMP-4、または BINARY が使用可能。
long int	PIC 9(5)-9(9) COMP-5。 TRUNC(BIN) コンパイラー・オプションを使用した場合は、COMP、COMP-4、または BINARY が使用可能。
long long int	PIC 9(10)-9(18) COMP-5。 TRUNC(BIN) コンパイラー・オプションを使用した場合は、COMP、COMP-4、または BINARY が使用可能。
float	COMP-1
double	COMP-2
enumeration	レベル 88 に類似するが、同一ではない。
char(*n*)	PICTURE X(*n*)
array pointer (*) to type	相当する COBOL データ型なし
pointer(*) to function	PROCEDURE-POINTER または FUNCTION-POINTER

関連タスク
439 ページの『COBOL と C/C++ 間でのデータの受け渡し』

関連参照
286 ページの『TRUNC』

例: C 関数を呼び出す COBOL プログラム

以下の例では、CALL ステートメントを使用して C 関数を呼び出す COBOL プログラムを示しています。

この例では、以下の概念を示しています。

- この CALL ステートメントは、呼び出し先プログラムが COBOL と C のどちらで記述されるかを示すものではありません。
- COBOL では、大小混合名を持つプログラムの呼び出しが可能です。
- 引数は、さまざまな方法で C プログラムに渡すことができます (例: BY REFERENCE または BY VALUE)。
- 非 void の C 関数を呼び出す CALL ステートメントでは、関数の戻り値を宣言する必要があります。
- COBOL データ型を適切な C データ型と対応させる必要があります。

```
 CBL PGMNAME(MIXED)
* This compiler option allows for case-sensitive names for called programs.
*
 IDENTIFICATION DIVISION.
 PROGRAM-ID. "COBCALLC".
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01 N4              PIC 9(4)  COMP-5.
 01 NS4             PIC S9(4) COMP-5.
 01 N9              PIC 9(9)  COMP-5.
 01 NS9             PIC S9(9) COMP-5.
 01 NS18            USAGE COMP-2.
 01 D1              USAGE COMP-2.
 01 D2              USAGE COMP-2.
 01 R1.
     02 NR1         PIC 9(8) COMP-5.
     02 NR2         PIC 9(8) COMP-5.
     02 NR3         PIC 9(8) COMP-5.
 PROCEDURE DIVISION.
     MOVE 123 TO N4
     MOVE -567 TO NS4
     MOVE 98765432 TO N9
     MOVE -13579456 TO NS9
     MOVE 222.22 TO NS18
     DISPLAY "Call MyFun with n4=" N4 " ns4=" NS4 " N9=" n9
     DISPLAY "                 ns9="  NS9" ns18="  NS18
* The following CALL illustrates several ways to pass arguments.
*
     CALL "MyFun" USING N4 BY VALUE NS4 BY REFERENCE N9 NS9 NS18
     MOVE 1024 TO N4
* The following CALL returns the C function return value.
*
     CALL "MyFunR" USING BY VALUE N4 RETURNING NS9
     DISPLAY "n4=" N4 " and ns9= n4 times n4= " NS9
     MOVE -357925680.25 TO D1
     CALL "MyFunD" USING BY VALUE D1 RETURNING D2
     DISPLAY "d1=" D1 " and d2= 2.0 times d2= " D2
     MOVE 11111 TO NR1
     MOVE 22222 TO NR2
     MOVE 33333 TO NR3
     CALL "MyFunV" USING R1
     STOP RUN.
```

関連タスク
439 ページの『COBOL と C/C++ 間でのデータの受け渡し』

関連参照
CALL ステートメント (*COBOL for Linux on x86* 言語解説書)

例: COBOL プログラムによって呼び出される C プログラムと COBOL を呼び出す C/C++ プログラム

次の例は、呼び出し先の C 関数が、COBOL CALL ステートメントで渡された順に引数を受け取ることを示しています。

ファイル MyFun.c には、次のソース・コードが含まれています。このソース・コードは COBOL プログラム tprog1 を呼び出します。

```
#include <stdio.h>
extern void TPROG1(double *);
void
MyFun(short *ps1, short s2, long *k1, long *k2, double *m)
{
    double x;
    x = 2.0*(*m);
    printf("MyFun got s1=%d s2=%d k1=%d k2=%d x=%f¥n",
             *ps1, s2, *k1,*k2, x);
}

long
MyFunR(short s1)
{
    return(s1 * s1);
}

double
MyFunD(double d1)
{
    double z;
    /* calling COBOL */
    z = 1122.3344;
    (void) TPROG1(&z);
    /* returning a value to COBOL */
    return(2.0 * d1);
}

void
MyFunV(long *pn)
{
    printf("MyFunV got %d %d %d¥n", *pn, *(pn+1), *(pn+2));
}
```

MyFun.c は、次の関数で構成されています。

MyFun
さまざまな引数を渡す方法を示します。

MyFunR
long 型変数を渡して戻す方法を示します。

MyFunD
C が COBOL プログラムを呼び出す方法と、double 型変数を渡して戻す方法を示します。

MyFunV
ポインターをレコードに渡す方法と、C プログラム内のレコードの項目にアクセスする方法を示します。

例: C プログラムによって呼び出される COBOL プログラム

次の例に、C プログラムによって呼び出される COBOL プログラムの記述方法を示します。

COBOL プログラム tprog1 が プログラム MyFun.c 内の C 関数 MyFunD によって 呼び出されます (442 ページの『例: COBOL プログラムによって呼び出される C プログラムと COBOL を呼び出す C/C++ プログラム』を参照)。 呼び出される COBOL プログラムには、以下のソース・コードが含まれます。

```
*
 IDENTIFICATION DIVISION.
 PROGRAM-ID. TPROG1.
*
```

```
 DATA DIVISION.
 LINKAGE SECTION.
*
 01 X                USAGE COMP-2.
*
 PROCEDURE DIVISION USING X.
     DISPLAY "TPROG1 got x= " X
     GOBACK.
```

関連タスク
438 ページの『COBOL および C/C++ プログラム間の呼び出し』

例: コンパイルおよび実行の結果の例

この例は、COBOL プログラム cobcallc.cbl と tprog.cbl、 および C プログラム MyFun.c をコンパイル、リンク、および実行する方法を示し、プログラムの実行結果を表示しています。

cobcallc.cbl、tprog.cbl、および MyFun.c をコンパイルしてリンクするには、以下のコマンドを発行します。

1. `gcc -m32 -c MyFun.c`
2. `cob2 cobcallc.cbl MyFun.o tprog1.cbl -o cobcallc`

プログラムを実行するには、 コマンド `cobcallc` を発行します。 結果は次のようになります。

```
call MyFun with n4=00123 ns4=-00567 n9=0098765432
           ns9=-0013579456 ns18=.22222000000000000E 03
MyFun got s1=123 s2=-567 k1=98765432 k2=-13579456 x=444.440000
n4=01024 and ns9= n4 times n4= 0001048576
TPROG1 got x=  .11223344000000000E 04
d1=-.35792568025000000E 09 and d2= 2.0 times d2= -.71585136050000000E 09
MyFunV got 11111 22222 33333
```

441 ページの『例: C 関数を呼び出す COBOL プログラム』
442 ページの『例: COBOL プログラムによって呼び出される C プログラムと COBOL を呼び出す C/C++ プログラム』
442 ページの『例: C プログラムによって呼び出される COBOL プログラム』

例: C++ 関数を呼び出す COBOL プログラム

以下の例では、引数 BY REFERENCE を渡す CALL ステートメントを使用して、C++ 関数を呼び出す COBOL プログラムを示しています。

この例では、以下の概念を示しています。

- この CALL ステートメントは、呼び出し先プログラムが COBOL と C++ のどちらで記述されるかを示すものではありません。
- 非 void の C++ 関数を呼び出す CALL ステートメントでは、関数の戻り値を宣言する必要があります。
- COBOL データ型は、適切な C++ データ型にマップされなければなりません。
- C++ 関数は、`extern "C"` として宣言する必要があります。
- COBOL 引数は、BY REFERENCE で受け渡されます。 C++ 関数は、参照パラメーターを使用してその引数を受け取ります。

COBOL プログラム driver:

```
cbl pgmname(mixed)
Identification Division.
Program-Id. "driver".
Data division.
Working-storage section.
01 A pic 9(8) binary value 11111.
01 B pic 9(8) binary value 22222.
01 R pic 9(8) binary.
Procedure Division.
```

```
    Display "Hello World, from COBOL!"
    Call "sub" using by reference A B
      returning R
    Display R
    Stop Run.
```

C++ 関数 sub:

```
#include <iostream.h>
extern "C" long sub(long& A, long& B) {
  cout << "Hello from C++" << endl;
  return A + B;
}
```

出力:

```
Hello World, from COBOL!
Hello from C++
00033333
```

関連タスク
439 ページの『COBOL と C/C++ 間でのデータの受け渡し』

関連参照
CALL ステートメント (*COBOL for Linux on x86* 言語解説書)

再帰呼び出しの実行

呼び出し先プログラムは、その呼び出し側を直接または間接に実行することができます。 例えば、プログラム X がプログラム Y を呼び出し、プログラム Y がプログラム Z を呼び出し、そして、プログラム Z がプログラム X を呼び出します。このような呼び出しを再帰的 と言います。

再帰呼び出しを行うには、再帰的に呼び出されるプログラムの PROGRAM-ID 段落で RECURSIVE 節をコーディングする必要があります。PROGRAM-ID 段落で RECURSIVE 節がコーディングされていない COBOL プログラムを再帰的に呼び出そうとすると、実行単位が異常終了します。

関連タスク
4 ページの『プログラムを再帰的として識別する』

関連参照
PROGRAM-ID 段落 (*COBOL for Linux on x86* 言語解説書)

戻りコードの受け渡し

RETURN-CODE 特殊レジスターを使用して、別々にコンパイルされたプログラム間で情報を受け渡すことができます。

呼び出し側に戻る前に RETURN-CODE を呼び出し先プログラムに設定して、戻される値を呼び出し側プログラムでテストすることができます。 この手法は、通常は呼び出し先プログラムの成功レベルを示すのに使用します。 例えば、ゼロの RETURN-CODE を使用して、呼び出し先プログラムが正常に実行されたことを示すことができます。

正常終了: メインプログラムが正常に終了すると、RETURN-CODE の値がオペレーティング・システムにユーザー戻りコードとして渡されます。 ただし、Linux は、ユーザー戻りコード値を 0 から 255 までに制限します。このため、プログラムが終了するときに例えば RETURN-CODE に 258 が含まれている場合、Linux はサポートされている範囲内の値を循環し、結果としてユーザー戻りコードが 2 になります。

回復不能な例外: プログラムが回復不能な例外を検出すると、ユーザー戻りコードは 128 にシグナル番号を加えた値に設定されます。 スレッド化されていないプログラムでは、実行単位が終了します。スレッド化されたプログラムでは、実行単位ではなく、プログラムが実行されているスレッドが終了します。

関連タスク
455 ページの『戻りコード情報の引き渡し』

関連参照
RETURN-CODE (*COBOL for Linux on x86* 言語解説書)

第 23 章 データの共用

実行単位が互いに呼び出す、別々にコンパイルされた複数のプログラムで構成される場合、プログラムは互いに通信できなければなりません。 また、通常は共通データにアクセスする必要があります。

ここでは、他のプログラムとデータを共用できるプログラムの作成方法を説明します。 ここで言うサブプログラム は、別のプログラムが呼び出す任意のプログラムのことです。

関連タスク

12 ページの『別のプログラムからのデータの使用』

447 ページの『データの受け渡し』
450 ページの『LINKAGE SECTION のコーディング』
451 ページの『引数を受け渡すための PROCEDURE DIVISION のコーディング』
454 ページの『プロシージャー・ポインターと関数ポインターの使用』
455 ページの『戻りコード情報の引き渡し』
456 ページの『EXTERNAL 節によるデータの共用』
456 ページの『プログラム間でのファイルの共用 (外部ファイル)』
459 ページの『コマンド行引数の使用』

データの受け渡し

プログラム間のデータの受け渡し方法には 3 つあり、それらから選択できます。BY REFERENCE、 BY CONTENT、または BY VALUE。

BY REFERENCE
サブプログラムは、データのコピーを処理するのではなく、呼び出し側プログラムのストレージ内のデータ項目を参照して処理します。 BY REFERENCE は、パラメーターに関して 3 つの方法のどれも指定されておらず、暗黙指定されてもいない 場合の、パラメーターの想定引き渡しメカニズムです。

BY CONTENT
呼び出し側プログラムは、*literal* または *identifier* の内容だけを渡します。 呼び出し先プログラムは、呼び出し側プログラム内の *literal* または *identifier* の値を変更できません。たとえ、 *literal* または *identifier* を受け取ったデータ項目を変更する場合でも変更できません。

BY VALUE
呼び出し側プログラムまたはメソッドは、送り出しデータ項目を参照せず、*literal* または *identifier* の値を渡します。 呼び出されるプログラムまたはメソッドは、その中のパラメーターを変更できます。しかし、サブプログラムまたはメソッドは送り出しデータ項目の一時コピーへのアクセス権しか持っていないので、どの変更内容も呼び出し側プログラムの引数に影響を与えません。

プログラムでどのようにデータを処理したいかに基づいて、上記のデータ受け渡し方法のどれを使用するかを決定してください。

表 *48.* ***CALL*** **ステートメントでデータを渡す方法**

コード	目的	コメント
CALL . . . BY REFERENCE *identifier*	呼び出し側プログラムの CALL ステートメントの 引数の定義と呼び出されるプログラムのパラメーターの定義に、同じメモリーを共用させる。	サブプログラムがパラメーターに対して行う変更は、呼び出し側プログラムの引数に影響を与えます。
CALL . . . BY REFERENCE ADDRESS OF *identifier*	*identifier* のアドレスを呼び出し先プログラムに渡します。ここで、*identifier* は LINKAGE SECTION 内の項目です。	サブプログラムがアドレスに対して行う変更は、呼び出し側プログラム内のアドレスに影響を与えます。

© Copyright IBM Corp. 2021

表 48. **CALL ステートメントでデータを渡す方法** (続き)

コード	目的	コメント
CALL . . . BY CONTENT ADDRESS OF *identifier*	*identifier* のアドレスのコピーを、呼び出し先プログラムに渡します。	アドレスのコピーに任意の変更を加えても *identifier* のアドレスには影響しませんが、アドレスのコピーを使用する *identifier* を変更すると *identifier* が変更されます。
CALL . . . BY CONTENT *ID*	ID のコピーをサブプログラムに渡す。	サブプログラムによってパラメーターを変更しても、呼び出し側の ID には影響しません。
CALL . . . BY CONTENT *literal*	呼び出し先プログラムにリテラル値のコピーを渡す。	
CALL . . . BY CONTENT LENGTH OF *identifier*	データ項目の長さのコピーを渡す。	呼び出し側プログラムは、その LENGTH 特殊レジスターから *identifier* の長さを渡します。

次のような BY REFERENCE と BY CONTENT の組み合わせ:
`CALL 'ERRPROC'`
`  USING BY REFERENCE A`
`  BY CONTENT LENGTH OF A.`
	データ項目とその長さのコピーの両方をサブプログラムに渡す。	

CALL . . . BY VALUE *identifier*	C/C++ プログラムなど、BY VALUE パラメーター・リンケージ規約を使用する プログラムにデータを渡す。	ID のコピーがパラメーター・リストとして直接渡されます。
CALL . . . BY VALUE *literal*	C/C++ プログラムなど、BY VALUE パラメーター・リンケージ規約を使用する プログラムにデータを渡す。	リテラルのコピーがパラメーター・リストとして直接渡されます。
CALL . . . BY VALUE ADDRESS OF *identifier*	呼び出し先プログラムに *identifier* のアドレスを渡す。 データに対するポインターを必要とする C/C++ プログラムにデータを渡すのに推奨される方法。	アドレスのコピーに任意の変更を加えても *identifier* のアドレスには影響しませんが、アドレスのコピーを使用する *identifier* を変更すると *identifier* が変更されます。
CALL . . . RETURNING	関数戻り値を使用して C/C++ 関数を呼び出す。	

関連タスク
449 ページの『呼び出し側プログラムの中での引数の記述』
449 ページの『呼び出し先プログラムの中でのパラメーターの記述』
449 ページの『OMITTED 引数に関するテスト』
456 ページの『CALL . . . RETURNING の指定』
456 ページの『EXTERNAL 節によるデータの共用』
456 ページの『プログラム間でのファイルの共用 (外部ファイル)』

関連参照
CALL ステートメント (*COBOL for Linux on x86* 言語解説書)
USING 句 (*COBOL for Linux on x86* 言語解説書)

呼び出し側プログラムの中での引数の記述

呼び出し側プログラムでは、DATA DIVISION の他のデータ項目と同じ方法で、DATA DIVISION で引数を記述します。

引数のためのストレージは、最外部プログラムにおいてのみ割り振られます。 例えば、プログラム A がプログラム B を呼び出し、プログラム B がプログラム C を呼び出すとします。データ項目はプログラム A で割り振られ、プログラム B と C の LINKAGE SECTION で記述され、そのデータの集合を 3 つのすべてのプログラムで使用できます。

ファイルのデータを参照する場合、データが参照される時には、そのファイルはオープンされていなければなりません。

引数を渡すためには、CALL ステートメントの USING 句を コーディングしてください。 データ項目を BY VALUE で渡す場合、データ項目は基本項目でなければなりません。

関連タスク
450 ページの『LINKAGE SECTION のコーディング』
451 ページの『引数を受け渡すための PROCEDURE DIVISION のコーディング』

関連参照

USING 句 (*COBOL for Linux on x86* 言語解説書)

呼び出し先プログラムの中でのパラメーターの記述

どんなデータが呼び出し側プログラムから渡されるのか知っている必要があり、さらに呼び出し側プログラムが直接的または間接的に呼び出すそれぞれのプログラム の LINKAGE SECTION でそれを記述する必要があります。

呼び出し側プログラムから渡されるデータを受け取るパラメーターを指定するために、USING 句を PROCEDURE DIVISION ヘッダーの後にコーディングしてください。

引数がサブプログラムに BY REFERENCE で渡される場合、メインプログラムで引き渡しが行われ定義されているもの以外のパラメーターおよびフィールドの間の関係をサブプログラムが指定しても無効です。 サブプログラムでは、以下を行うことはできません。

- 対応する引数よりバイト総数が大きくなるようパラメーターを定義する。
- 呼び出し側プログラムから引数として渡されたテーブルの限度を超えるエレメントを参照するような 添え字参照を使用する。
- 定義されたパラメーターの長さを超えるデータにアクセスする参照変更を使用する。
- 呼び出し側プログラムで定義された以外のデータ項目にアクセスするためにパラメーターのアドレスを操作する。

これらの規則のいずれかに違反していると、予期しない結果となります。

関連タスク
450 ページの『LINKAGE SECTION のコーディング』

関連参照

USING 句 (*COBOL for Linux on x86* 言語解説書)

OMITTED 引数に関するテスト

CALL ステートメント内の引数の代わりに OMITTED キーワードをコーディングして、1 つ以上の BY REFERENCE 引数が、呼び出し先プログラムに渡されないように指定することができます。

例えば、プログラム sub1 を呼び出すときに、2 番目の引数を 省略するには、次のステートメントをコーディングします。

```
Call 'sub1' Using PARM1, OMITTED, PARM3
```

CALL ステートメントの USING 句の引数は、数および位置において呼び出し先プログラムのパラメーターと一致しなければなりません。

呼び出し先プログラムで、対応するパラメーターのアドレスを NULL と比較して、引数が OMITTED として渡されたかどうかをテストすることができます。 次に例を示します。

```
Program-ID. sub1.
. . .
Procedure Division Using RPARM1, RPARM2, RPARM3.
    If Address Of RPARM2 = Null Then
        Display 'No 2nd argument was passed this time'
    Else
        Perform Process-Parm-2
    End-If
```

関連参照

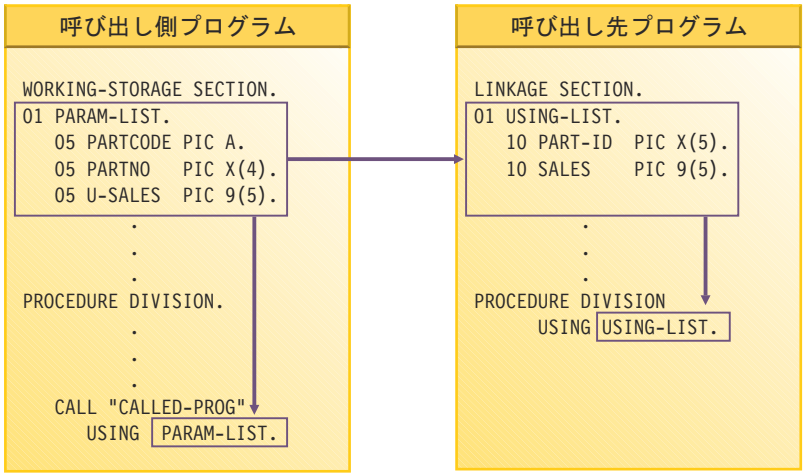
CALL ステートメント (*COBOL for Linux on x86* 言語解説書)
USING 句 (*COBOL for Linux on x86* 言語解説書)

LINKAGE SECTION のコーディング

呼び出し側プログラムの引数と同じ数のデータ名を、呼び出し先プログラムの ID リストにコーディングしてください。 位置で同期させます。なぜならコンパイラーは、呼び出し側プログラムの最初の引数を、呼び出し先プログラムの最初の identifier に渡し、以下同様に行うからです。

呼び出し先プログラムの identifier リストのデータ名の数が、呼び出し側プログラムから渡される引数の数よりも大きいと、エラーになります。コンパイラーは引数とパラメーターの突き合わせを試行しません。

次の図は、あるプログラムから別のプログラムにデータ項目が渡される様子を示しています (暗黙的に BY REFERENCE)。



呼び出し側プログラムでは、パーツ (PARTCODE) とパーツ・ナンバー (PARTNO) のコードは別のデータ項目です。それに対して、呼び出し先プログラムでは、パーツのコードとパーツ・ナンバーのコードが 1 つのデータ項目 (PART-ID) に結合されています。呼び出し先プログラムでは、PART-ID への参照は、これらの項目に対する唯一有効な参照です。

引数を受け渡すための PROCEDURE DIVISION のコーディング

引数 BY VALUE を渡す場合には、サブプログラムの PROCEDURE DIVISION ヘッダーにある USING BY VALUE 節をコーディングします。 引数を BY REFERENCE または BY CONTENT で渡す場合は、引数の受け渡し方法をヘッダーで指示する必要はありません。

```
PROCEDURE DIVISION USING BY VALUE. . .
PROCEDURE DIVISION USING. . .
PROCEDURE DIVISION USING BY REFERENCE. . .
```

上の最初のヘッダーは、データ項目は渡された BY VALUE を示します。2 番目または 3 番目のヘッダーは、項目が渡された BY REFERENCE または BY CONTENT を示します。

関連参照
手続き部ヘッダー (*COBOL for Linux on x86* 言語解説書)
USING 句 (*COBOL for Linux on x86* 言語解説書)
CALL ステートメント (*COBOL for Linux on x86* 言語解説書)

受け渡されるデータのグループ化

プログラム間で渡す必要があるすべてのデータ項目をグループ化し、それらを 1 つのレベル 01 項目に入れることを考慮してください。 そうすると、1 個のレベル 01 レコードを渡すことができます。

データ項目を BY VALUE で渡す場合、データ項目は基本項目でなければならないことに注意してください。

レコード突き合わせの間違いの可能性をより少なくするためには、レベル 01 レコードをコピー・ライブラリーの中に置き、両方のプログラムにそれをコピーするようにします。 すなわち、呼び出し側プログラムの WORKING-STORAGE SECTION と呼び出し先プログラムの LINKAGE SECTION の中でコピーします。

関連タスク
450 ページの『LINKAGE SECTION のコーディング』

関連参照
CALL ステートメント (*COBOL for Linux on x86* 言語解説書)

ヌル終了ストリングの処理

ヌル終了リテラルおよび 16 進リテラル X'00' と一緒にストリング処理ステートメントを使用する場合、COBOL はヌル終了ストリングをサポートします。

ヌル終了ストリング (例えば、C プログラムから渡された) は、次のコードのようなストリング処理メカニズムを使用して処理することができます。

```
01 L        pic X(20) value z'ab'.
01 M        pic X(20) value z'cd'.
01 N        pic X(20).
01 N-Length pic 99    value zero.
01 Y        pic X(13) value 'Hello, World!'.
```

ヌル終了ストリングの長さを決定して、そのストリングの値および長さを表示するには、次のようにコーディングします。

```
Inspect N tallying N-length for characters before initial X'00'
Display 'N: ' N(1:N-length) ' Length: ' N-length
```

ヌル終了ストリングを英数字ストリングに移動するが、ヌルを削除するには、次のようにコーディングします。

```
Unstring N  delimited by X'00' into X
```

ヌル終了ストリングを作成するには、次のようにコーディングします。

```
String Y      delimited by size
       X'00'  delimited by size
       into N.
```

2 つのヌル終了ストリングを連結するには、次のようにコーディングします。

```
String L      delimited by x'00'
       M      delimited by x'00'
       X'00'  delimited by size
       into N.
```

関連タスク
98 ページの『ヌル終了ストリングの取り扱い』

関連参照

ヌル終了英数字リテラル (*COBOL for Linux on x86* 言語解説書)

ポインターによるチェーン・リストの処理

レコード域のアドレスを渡したり受信する必要がある場合、ポインター・データ項目を使用できます。これは、USAGE IS POINTER 節を使用して定義されるデータ項目、または ADDRESS OF 特殊レジスターであるデータ項目のいずれかです。

ポインター・データ項目の代表的な適用は、チェーン・リスト (各レコードがそれぞれ次のレコードを指し示す一連のレコード) の処理です。

プログラム相互間のアドレスをチェーン・リストに入れて渡す場合は、NULL を使用して、以下の 2 つの方法のいずれかで、無効なアドレスの値 (非数値 0) をポインター項目に割り当てることができます。

- データ定義の中で VALUE IS NULL 節を使用する。
- SET ステートメントの中で送信フィールドとして NULL を使用する。

最後のレコードのポインター・データ項目がヌル値を含んでいるチェーン・リストの場合、このコードを使用して、リストの終わりを検査できます。

```
IF PTR-NEXT-REC = NULL
. . .
   (logic for end of chain)
```

リストの終わりに達していない場合、プログラムはレコードを処理して次のレコードに移ることができます。

呼び出し側プログラムから渡されるデータには、無視したいヘッダー情報が含まれていることがあります。ポインター・データ項目は数値ではないので、それらに対して直接に算術演算を行うことはできません。しかし、ヘッダー情報をバイパスするために、SET ステートメントを使用して、渡されたアドレスを増分することができます。

453 ページの『例: チェーン・リストを処理するためのポインターの使用』

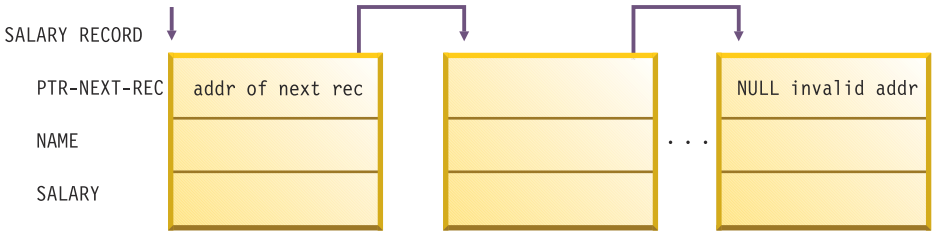
関連タスク
450 ページの『LINKAGE SECTION のコーディング』
451 ページの『引数を受け渡すための PROCEDURE DIVISION のコーディング』

関連参照
252 ページの『ADDR』
SET ステートメント (*COBOL for Linux on x86* 言語解説書)

例: チェーン・リストを処理するためのポインターの使用

次の例は、リンク・リスト、つまりデータ項目のチェーン・リストを処理する方法を示しています。

この例では、個々の給与レコードで構成されるデータのチェーン・リストを示しています。 次の図は、これらのレコードがストレージの中でどのようにリンクされているかを 視覚化する 1 つの方法を示しています。 最後のレコードを除いて、各レコードの最初の項目は次のレコードを指し示しています。 最後のレコードの最初の項目は、それが最後のレコードであることを示すために、(有効なアドレスではなく) ヌル値を含んでいます。



これらのレコードを処理するアプリケーションの高水準擬似コードは、次のようになります。

```
Obtain address of first record in chained list from routine
Check for end of the list
Do until end of the list
   Process record
   Traverse to the next record
End
```

以下のコードには、このチェーン・リスト処理例で使用される呼び出し側プログラム LISTS の概要が含まれています。

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. LISTS.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
******
 WORKING-STORAGE SECTION.
 77  PTR-FIRST         POINTER  VALUE IS NULL.                  (1)
 77  DEPT-TOTAL        PIC 9(4) VALUE IS 0.
******
 LINKAGE SECTION.
 01  SALARY-REC.
     02  PTR-NEXT-REC    POINTER.                               (2)
     02  NAME            PIC X(20).
     02  DEPT            PIC 9(4).
     02  SALARY          PIC 9(6).
 01  DEPT-X            PIC 9(4).
******
 PROCEDURE DIVISION USING DEPT-X.
******
* FOR EVERYONE IN THE DEPARTMENT RECEIVED AS DEPT-X,
* GO THROUGH ALL THE RECORDS IN THE CHAINED LIST BASED ON THE
* ADDRESS OBTAINED FROM THE PROGRAM CHAIN-ANCH
* AND ACCUMULATE THE SALARIES.
* IN EACH RECORD, PTR-NEXT-REC IS A POINTER TO THE NEXT RECORD
* IN THE LIST; IN THE LAST RECORD, PTR-NEXT-REC IS NULL.
* DISPLAY THE TOTAL.
******
     CALL "CHAIN-ANCH" USING PTR-FIRST                          (3)
     SET ADDRESS OF SALARY-REC TO PTR-FIRST                     (4)
******
     PERFORM WITH TEST BEFORE UNTIL ADDRESS OF SALARY-REC = NULL (5)

      IF DEPT = DEPT-X
        THEN ADD SALARY TO DEPT-TOTAL
        ELSE CONTINUE
      END-IF
      SET ADDRESS OF SALARY-REC TO PTR-NEXT-REC                 (6)

     END-PERFORM
******
```

```
      DISPLAY DEPT-TOTAL
      GOBACK.
```

(1)
PTR-FIRST は、NULL の初期値を持つポインター・データ項目として定義されます。 CHAIN-ANCH への呼び出しから正常に戻ると、PTR-FIRST には、チェーン・リスト内の最初のレコードのアドレスが入れられます。 呼び出しで何か間違いが起こり、PTR-FIRST がチェーンの最初のレコードのアドレスの値を受け取っていないと、PTR-FIRST はヌル値のままであり、プログラムのロジックに従って、レコードは処理されません。

(2)
呼び出し側プログラムの LINKAGE SECTION には、チェーン・リストのレコードの記述が入っています。 さらに、CALL ステートメントの USING 節で渡される部門コードの記述も含まれています。

(3)
最初の SALARY-REC レコード域のアドレスを取得するために、LISTS プログラムはプログラム CHAIN-ANCH を呼び出します。

(4)
SET ステートメントのレコード記述 SALARY-REC の基礎となっているのは、PTR-FIRST に含まれているアドレスです。

(5)
この例のチェーン・リストは、最後のレコードに無効アドレスが含まれるようにセットアップされます。 チェーン・リスト終了に対するこの検査は、do-while 構造を使用して行われます (最後のレコードのポインター・データ項目には値 NULL が割り当てられる構造になっています)。

(6)
LINKAGE-SECTION 内のレコードのアドレスは、SALARY-REC の最初のフィールドとして送信されるポインター・データ項目によって、次のレコードのアドレスに等しく設定されます。 レコード処理ルーチンが繰り返され、チェーン・リスト内の次のレコードが処理されます。

別のプログラムから受け取ったアドレスを増分するには、LINKAGE SECTION および PROCEDURE DIVISION を次のようにセットアップすることができます。

```
LINKAGE SECTION.
01  RECORD-A.
    02  HEADER          PIC X(12).
    02  REAL-SALARY-REC PIC X(30).
. . .
01  SALARY-REC.
    02  PTR-NEXT-REC    POINTER.
    02  NAME            PIC X(20).
    02  DEPT            PIC 9(4).
    02  SALARY          PIC 9(6).
. . .
PROCEDURE DIVISION USING DEPT-X.
. . .
    SET ADDRESS OF SALARY-REC TO ADDRESS OF REAL-SALARY-REC
```

この時点では、SALARY-REC のアドレスは、REAL-SALARY-REC、または RECORD-A + 12 のアドレスを基底にしています。

関連タスク
452 ページの『ポインターによるチェーン・リストの処理』

プロシージャー・ポインターと関数ポインターの使用

プロシージャー・ポインターとは、USAGE IS PROCEDURE-POINTER 文節によって定義されるデータ項目です。 関数ポインターとは、USAGE IS FUNCTION-POINTER 節によって定義されるデータ項目です。

ここでは、「ポインター」は、プロシージャー・ポインター・データ項目または関数ポインター・データ項目のどちらかを指します。 プロシージャー・ポインター・データ項目または関数ポインター・データ項目は、以下の入り口点の入り口アドレス (ポインター) が入るように設定することができます。

- ネストされていない別の COBOL プログラム。
- 別の言語で作成されているプログラム。 例えば、C 関数の入り口アドレスを受け取るためには、CALL ステートメントの CALL RETURNING 形式を使用して関数を呼び出してください。 この結果、SET ステートメントの形式を使用してプロシージャー・ポインターに変換できるポインターが戻されます。
- 別の COBOL プログラムの代替入り口点 (ENTRY ステートメントで定義されたもの)。

ポインター・データ項目を設定するには、SET ステートメントを使用する必要があります。 次に例を示します。

```
CALL 'MyCFunc' RETURNING ptr.
SET  proc-ptr  TO ptr.
CALL proc-ptr  USING dataname.
```

ポインター項目を、CALL *identifier* ステートメントで呼び出されるロード・モジュールの入り口アドレスに設定し、後でプログラムが呼び出されたモジュールをキャンセルしたとします。 この場合、ポインター項目は未定義の状態になり、後でその項目を参照しても、結果は信頼できないものになります。

関連参照
252 ページの『ADDR』

PROCEDURE-POINTER 句 (*COBOL for Linux on x86* 言語解説書)
SET ステートメント (*COBOL for Linux on x86* 言語解説書)

戻りコード情報の引き渡し

プログラム間で戻りコードを渡すには、RETURN-CODE 特殊レジスターを使用します。

関連タスク
444 ページの『戻りコードの受け渡し』

RETURN-CODE 特殊レジスターの使用

COBOL プログラムがその呼び出し側に戻ると、RETURN-CODE 特殊レジスターの内容が、呼び出されたプログラムの RETURN-CODE 特殊レジスターの値によって設定されます。

呼び出し先プログラムによる RETURN-CODE の設定は、COBOL プログラム間の呼び出しに制限されています。 そのため、COBOL プログラムが C プログラムを呼び出しても、COBOL プログラムの RETURN-CODE 特殊レジスターが設定されることは期待できません。

COBOL プログラムと C プログラムで同じように機能させるためには、COBOL プログラムが RETURNING 句を使用して C プログラムを呼び出すようにしなければなりません。 C プログラム (関数) が関数値を正しく宣言していれば、呼び出し COBOL プログラムの RETURNING 値が設定されます。

PROCEDURE DIVISION RETURNING . . .の使用

呼び出し側プログラムに情報を戻すには、プログラムの PROCEDURE DIVISION ヘッダーで RETURNING 句を使用してください。

```
PROCEDURE DIVISION RETURNING dataname2
```

例の呼び出し先プログラムが正常にその呼び出し側に戻ると、*dataname2* の値が、CALL ステートメントの RETURNING 句で指定された ID に保管されます。

```
CALL . . . RETURNING dataname2
```

CALL . . . RETURNING の指定

C/C++ 関数または COBOL サブルーチンへの呼び出しでは、CALL ステートメントの RETURNING 句を指定することができます。

RETURNING 句のフォーマットは次のとおりです。

```
CALL . . . RETURNING dataname2
```

呼び出し先プログラムの戻り値は *dataname2* に保管されます。 *dataname2* は、呼び出し側プログラムの DATA DIVISION で定義しなければなりません。 ターゲット関数で宣言される戻り値のデータ型は、*dataname2* のデータ型と同じでなければなりません。

EXTERNAL 節によるデータの共用

EXTERNAL 節を使用すると、別々にコンパイルされたプログラム (バッチ・シーケンスの プログラムを含む) がデータ項目を共用できるようになります。WORKING-STORAGE SECTION のレベル 01 データ記述に EXTERNAL をコーディングします。

次の規則が適用されます。

- EXTERNAL グループ項目に従属する項目はそれ自体が EXTERNAL です。
- EXTERNAL データ項目の名前を、同じプログラムの中で別の EXTERNAL 項目の名前として使用することはできません。
- VALUE 文節は、EXTERNAL であるいずれの基本項目、グループ項目、従属項目についても コーディングできません。

実行単位内で、ある COBOL プログラムまたはメソッドの項目のデータ記述が、その項目を含むプログラムのデータ記述と同じ場合は、そのプログラムはその項目にアクセスして処理できます。例えば、プログラム A に次のデータ記述があるとします。

```
01 EXT-ITEM1     EXTERNAL     PIC 99.
```

プログラム B は、WORKING-STORAGE SECTION に同じデータ記述が存在する場合、そのデータ項目にアクセスすることができます。

EXTERNAL データ項目にアクセス権を持っているプログラムはすべて、その項目の値を変更できます。 そのため、保護しなければならないデータ項目には、この節を使用しないでください。

関連参照

プログラム間でのファイルの共用 (外部ファイル)

実行単位の別々にコンパイルされたプログラムが共用ファイルとしてファイルにアクセスできるようにするには、そのファイルに EXTERNAL 節を使用します。

以下の指針に従うことをお勧めします。

- ファイル状況コードを検査するすべてのプログラムの FILE STATUS 節で、同じデータ名を使用する。
- 同じファイル状況フィールドを検査するすべてのプログラムについて、ファイル状況フィールドのレベル 01 データ定義で EXTERNAL 節をコーディングする。

外部ファイルを使用すると、次のような利点があります。

- メインプログラムに入出力ステートメントが含まれていなくても、ファイルのレコード域を参照することができます。
- それぞれのサブプログラムが、OPEN や READ のような単一の入出力機能を制御することができます。

- それぞれのプログラムがファイルにアクセスすることができます。

457 ページの『例: 外部ファイルの使用』

関連タスク
9 ページの『入出力操作でのデータの使用』

関連参照
EXTERNAL 節 (*COBOL for Linux on x86* 言語解説書)

例: 外部ファイルの使用

次の例は、いくつかのプログラムでの外部ファイルの使用を示しています。 それぞれのサブプログラムにファイルの同じ記述が含まれていることを保証するために、COPY ステートメントを使用します。

次の表で、メインプログラムとサブプログラムについて説明します。

名前	機能
ef1	メインプログラム。すべてのサブプログラムを呼び出し、レコード域の内容を検査する。
ef1openo	出力用に外部ファイルをオープンし、ファイル状況コードを検査する。
ef1write	外部ファイルにレコードを書き込み、ファイル状況コードを検査する。
ef1openi	入力用に外部ファイルをオープンし、ファイル状況コードを検査する。
ef1read	外部ファイルからレコードを読み取り、ファイル状況コードを検査する。
ef1close	外部ファイルをクローズし、ファイル状況コードを検査する。

各プログラムは、次の 3 つのコピーブックを使用します。

- efselect は FILE-CONTROL 段落に入れられます。

```
Select ef1
Assign To ef1
File Status Is efs1
Organization Is Sequential.
```

- effile は FILE SECTION に入れられます。

```
Fd ef1 Is External
          Record Contains 80 Characters
          Recording Mode F.
01  ef-record-1.
    02 ef-item-1  Pic X(80).
```

- efwrkstg は WORKING-STORAGE SECTION に入れられます。

```
01  efs1          Pic 99 External.
```

外部ファイルを使用する入出力

```
 IDENTIFICATION DIVISION.
 Program-Id.
     ef1.
*
* This main program controls external file processing.
*
 ENVIRONMENT DIVISION.
 Input-Output Section.
 File-Control.
     Copy efselect.
 DATA DIVISION.
```

```
 FILE SECTION.
     Copy effile.
 WORKING-STORAGE SECTION.
     Copy efwrkstg.
 PROCEDURE DIVISION.
     Call "ef1openo"
     Call "ef1write"
     Call "ef1close"
     Call "ef1openi"
     Call "ef1read"
     If ef-record-1 = "First record" Then
       Display "First record correct"
     Else
       Display "First record incorrect"
       Display "Expected: " "First record"
       Display "Found   : " ef-record-1
     End-If
     Call "ef1close"
     Goback.
 End Program ef1.
 IDENTIFICATION DIVISION.
 Program-Id.
     ef1openo.
*
* This program opens the external file for output.
*
 ENVIRONMENT DIVISION.
 Input-Output Section.
 File-Control.
     Copy efselect.
 DATA DIVISION.
 FILE SECTION.
     Copy effile.
 WORKING-STORAGE SECTION.
     Copy efwrkstg.
 PROCEDURE DIVISION.
     Open Output ef1
     If efs1 Not = 0
       Display "file status " efs1 " on open output"
       Stop Run
     End-If
     Goback.
 End Program ef1openo.
 IDENTIFICATION DIVISION.
 Program-Id.
     ef1write.
*
* This program writes a record to the external file.
*
 ENVIRONMENT DIVISION.
 Input-Output Section.
 File-Control.
     Copy efselect.
 DATA DIVISION.
 FILE SECTION.
     Copy effile.
 WORKING-STORAGE SECTION.
     Copy efwrkstg.
 PROCEDURE DIVISION.
     Move "First record" to ef-record-1
     Write ef-record-1
     If efs1 Not = 0
       Display "file status " efs1 " on write"
       Stop Run
     End-If
     Goback.
 End Program ef1write.
 Identification Division.
 Program-Id.
     ef1openi.
*
* This program opens the external file for input.
*
 ENVIRONMENT DIVISION.
 Input-Output Section.
 File-Control.
     Copy efselect.
 DATA DIVISION.
 FILE SECTION.
     Copy effile.
 WORKING-STORAGE SECTION.
     Copy efwrkstg.
```

```
 PROCEDURE DIVISION.
     Open Input ef1
     If efs1 Not = 0
       Display "file status " efs1 " on open input"
       Stop Run
     End-If
     Goback.
 End Program ef1openi.
 Identification Division.
 Program-Id.
     ef1read.
*
* This program reads a record from the external file.
*
 ENVIRONMENT DIVISION.
 Input-Output Section.
 File-Control.
     Copy efselect.
 DATA DIVISION.
 FILE SECTION.
     Copy effile.
 WORKING-STORAGE SECTION.
     Copy efwrkstg.
 PROCEDURE DIVISION.
     Read ef1
     If efs1 Not = 0
       Display "file status " efs1 " on read"
       Stop Run
     End-If
     Goback.
 End Program ef1read.
 Identification Division.
 Program-Id.
     ef1close.
*
* This program closes the external file.
*
 ENVIRONMENT DIVISION.
 Input-Output Section.
 File-Control.
     Copy efselect.
 DATA DIVISION.
 FILE SECTION.
     Copy effile.
 WORKING-STORAGE SECTION.
     Copy efwrkstg.
 PROCEDURE DIVISION.
     Close ef1
     If efs1 Not = 0
       Display "file status " efs1 " on close"
       Stop Run
     End-If
     Goback.
 End Program ef1close.
```

コマンド行引数の使用

コマンド行でメインプログラムに引数を渡すことができます。 オペレーティング・システムはメインプログラムを呼び出す際に、引数を含むヌル終了ストリングを使用します。

入力された引数が、その引数を受け取る COBOL データ名より短い場合は、ヌル終了ストリングの操作に関して 下記の関連タスクで示されている手法で、その引数を分離します。

引数の処理方法は、 cob2 コマンドの -host オプションを使用しているかどうかによって異なります。

-host オプションを指定しない場合、コマンド行引数がネイティブのデータ形式で受け渡され、Linux が次の引数を含むすべてのメインプログラムを呼び出します。

- コマンド行引数の数に 1 を加えた数
- プログラムの名前へのポインター
- 最初の引数へのポインター
- 2 番目の引数へのポインター
- . . .

- *n* 番目の引数へのポインター

-host オプションを指定すると、Linux では、ストリングの長さを含むハーフワード接頭部がある EBCDIC 文字ストリングを使用して、すべてのメインプログラムを呼び出します。コマンド行引数は、引用符 (") で囲んだ単一ストリングとして入力する必要があります。 ストリング内の引用符文字を渡すには、引用符の前に円記号 (¥) のエスケープ文字を付けます。

460 ページの『例: -host オプションを使用しないコマンド行引数』
461 ページの『例: -host オプションを使用したコマンド行引数』

関連タスク
98 ページの『ヌル終了ストリングの取り扱い』
451 ページの『ヌル終了ストリングの処理』

関連参照
232 ページの『cob2 オプション』

例: -host オプションを使用しないコマンド行引数

この例は、 -host オプションを使用していない場合のコマンド行引数の読み取り方法を示しています。

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. "targlinux".
*
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*
 LINKAGE SECTION.
 01  PARM-LEN PIC S9(9) COMP.
 01  OS-PARM.
     02 PARMPTR-TABLE OCCURS 1 TO 100 TIMES DEPENDING ON PARM-LEN.
        03 PARMPTR  POINTER.
 01 PARM-STRING PIC XX.
*
 PROCEDURE DIVISION USING BY VALUE PARM-LEN BY REFERENCE OS-PARM.
     display "parm-len=" parm-len
     SET ADDRESS OF PARM-STRING TO PARMPTR(2).
     display "parm-string= '" PARM-STRING "'";
     EVALUATE PARM-STRING
         when "01"  display "case one"
         when "02"  display "case two"
         when "95"  display "case ninety-five"
         when other display "case unknown"
     END-EVALUATE
     GOBACK.
```

次のプログラムをコンパイルし、実行するとします。

```
cob2 targlinux.cbl
a.out 95
```

結果は次のとおりです。

```
parm-len=000000002
parm-string= '95'
case ninety-five
```

例: -host オプションを使用したコマンド行引数

この例は、-host オプションを使用している場合のコマンド行引数の読み取り方法を示しています。

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. "testarg".
*
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*
 linkage section.
 01  os-parm.
     05 parm-len         pic s999 comp.
     05 parm-string.
         10 parm-char    pic x occurs 0 to 100 times
                         depending on parm-len.
*
 PROCEDURE DIVISION using os-parm.
     display "parm-len=" parm-len
     display "parm-string='" parm-string "'"
     evaluate parm-string
       when "01"  display "case one"
       when "02"  display "case two"
       when "95"  display "case ninety-five"
       when other display "case unknown"
     end-evaluate
     GOBACK.
```

以下のように、このプログラムをコンパイルして実行するとします。

```
cob2 -host testarg.cbl
a.out "95"
```

結果の出力は、次のとおりです。

```
parm-len=002
parm-string='95'
case ninety-five
```

第 24 章 共用ライブラリーの使用

共用ライブラリーは、アプリケーションをパッケージ化するための便利で効率的な手段を提供し、Linux で広く使用されています。

COBOL 用語における共用ライブラリーとは、1 つ以上の最外部プログラムの集合を意味します。 複数の COBOL プログラムを 1 つの実行可能ファイルにコンパイルおよびリンクできるのと同様に、1 つ以上のコンパイル済み最外部 COBOL プログラムをリンクし、共用ライブラリーを作成することができます。 共用ライブラリーは通常、頻繁に呼び出される関数の集合として使用します。

共有ライブラリー内の最外部プログラムにはネストされたプログラムを含めることができますが、共用ライブラリーの外部にあるプログラムは、共有ライブラリー内の最外部プログラム (入り口点 とも呼ばれる) しか呼び出すことができません。 共有ライブラリー内の各プログラムは、サブプログラムと呼ばれます。

COBOL 共有ライブラリー、または COBOL と C/C++ 混合の共有ライブラリーを呼び出せます。

464 ページの『例: サンプルの共用ライブラリーの作成』

関連概念
431 ページの『メインプログラム、サブプログラム、および呼び出し』
463 ページの『スタティック・リンクおよび共用ライブラリーの使用』
464 ページの『共用ライブラリーへの参照をリンカーが解決する方法』

スタティック・リンクおよび共用ライブラリーの使用

スタティック・リンク は、呼び出し側プログラムと、1 つ以上の呼び出し先プログラムを単一の実行可能モジュールにリンクすることです。 プログラムのロード時には、オペレーティング・システムによって、実行可能コードおよびデータが含まれる 1 つのファイルがメモリーに格納されます。

スタティック・リンクの主な利点は、これを使用して必要なものを完備した、独立した実行可能モジュールを作成できることです。

ただし、スタティック・リンクには、次のよう な欠点があります。

- 外部プログラムが実行可能ファイルに組み込まれるため、実行可能ファイルのサイズが大きくなります。
- 実行可能ファイルの動作を変更するには、ファイルを再コンパイルしてリンクし直す必要があります。
- 複数の呼び出し側プログラムが呼び出し先プログラムにアクセスする必要がある場合は、メモリー内に呼び出し先プログラムの重複コピーをロードする必要があります。

これらの欠点に対処するため、共用ライブラリーを使用できます。

- 1 つ以上のサブプログラムを 1 つの共有ライブラリー内にビルドできます。また、複数のプログラムが、共有ライブラリー内にあるサブプログラムを呼び出せます。 共有ライブラリーのコードは呼び出し側プログラムのコードとは別のため、呼び出し側プログラムのサイズを小さくできます。
- 共有ライブラリー内のサブプログラムを変更しても、呼び出し側プログラムを再コンパイルまたは再リンクする必要はありません。
- メモリー内に必要な共有ライブラリーのコピーは 1 つのみです。

通常、共用ライブラリーは、多数のプログラムで使用できる共通関数を提供します。 例えば、共用ライブラリーを使用して、サブプログラム・パッケージ、サブシステム、および他のプログラムのインターフェースをインプリメントすることや、オブジェクト指向のクラス・ライブラリーを作成することが可能です。

464 ページの『例: サンプルの共用ライブラリーの作成』

関連タスク
436 ページの『ネストなし COBOL プログラムの呼び出し』

© Copyright IBM Corp. 2021

共用ライブラリーへの参照をリンカーが解決する方法

プログラムをコンパイルするときには、コンパイラーによって、プログラム内のコードに対応するオブジェクト・モジュールが生成されます。 外部オブジェクト・モジュールに含まれる任意のサブプログラム (C/C++ では関数、他の言語ではサブルーチン) を呼び出すと、コンパイラーは外部プログラム参照をターゲットのオブジェクト・モジュールに追加します。

リンカーは、共用ライブラリーへの外部参照を解決するために、実行可能ファイルのロード時に共用ライブラリー・コードの場所をローダーに通知する情報を実行可能ファイルに追加します。

COBOL CALL ステートメントによって行われる共用ライブラリーへの参照がすべて、リンカーで解決されるとは限りません。 DYNAM コンパイラー・オプションが有効な場合は、CALL *literal* ステートメントが実行されるときに、COBOL がこれらの呼び出しを解決します。 CALL *identifier* 呼び出しも動的に解決されます。

464 ページの『例: サンプルの共用ライブラリーの作成』

関連概念
463 ページの『スタティック・リンクおよび共用ライブラリーの使用』

関連タスク
385 ページの『CICS での共有ライブラリーの動的呼び出し』

関連参照
236 ページの『リンカーの入出力ファイル』
264 ページの『DYNAM』

例: サンプルの共用ライブラリーの作成

次の例は、3 つの COBOL プログラムを示しています。そのうちの 1 つ (alpha) は、他の 2 つ (beta および gamma)を呼び出します。 プログラムの後のプロシージャーは、2 つの呼び出し先プログラムを含む共有ライブラリーを作成し、その共有ライブラリーを含むアーカイブ・ライブラリーを作成し、アーカイブ・ライブラリー内の呼び出し先プログラムにアクセスするモジュールに呼び出し側プログラムをコンパイルおよびリンクする方法を示しています。

例 *1: alpha.cbl*

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. alpha.
*
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01 hello1   pic x(30)  value is "message from alpha to beta".
 01 hello2   pic x(30)  value is "message from alpha to gamma".
*
 PROCEDURE DIVISION.
     display "alpha begins"
     call "beta" using hello1
     display "alpha after beta"
     call "gamma" using hello2
     display "alpha after gamma"
     goback.
```

例 *2: beta.cbl*

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. beta.
*
 ENVIRONMENT DIVISION.
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
```

```
*
 Linkage section.
 01 msg   pic x(30).
*
 PROCEDURE DIVISION using msg.
     DISPLAY "beta gets msg=" msg.
     goback.
```

例 *3: gamma.cbl*

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. gamma.
*
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*
 Linkage section.
 01 msg            pic x(30).
*
PROCEDURE DIVISION using msg.
    DISPLAY "gamma gets msg=" msg.
    goback.
```

プロシージャー

これら 3 つのプログラムを結合する最も単純な方法は、それらのプログラムを次のコマンドを使用して単一の実行可能モジュール内にコンパイルおよびリンクすることです。

```
cob2 -o m_abg alpha.cbl beta.cbl gamma.cbl
```

その後、コマンド m_abg を発行してプログラムを実行できます。 出力結果は次のようになります。

```
alpha begins
beta gets msg=message from alpha to beta
alpha after beta
gamma gets msg=message from alpha to gamma
alpha after gamma
```

プログラムを単一の実行可能モジュール内にリンクする代わりに、共有ライブラリー (以下のプロシージャーの sh_bg) に beta と gamma を入れ、共有ライブラリー内の beta と gamma にアクセスする実行可能モジュールに alpha をコンパイルおよびリンクすることができます。 これを行うには、以下のステップを実行します。

1. 共有ライブラリーがエクスポートする必要のあるシンボルを指定するバージョン・スクリプトを作成します。

   ```
   {
   global:
        GAMMA;
        BETA;
   local:
        *;
   };
   ```

   COBOL はデフォルトで外部シンボルに大文字の名前を使用するため、上記のエクスポート・ファイル bg.version 内のシンボル名は大文字です。大/小文字混合の名前が必要な場合は、PGMNAME(MIXED) コンパイラー・オプションを使用します。

   エクスポート・ファイル bg.version を指定する場合は、次の手順で示すとおり、共有ライブラリーを作成するときにオプション -Wl,--version-script,bg.version を使用する必要があります。

2. 次のコマンドを使用して、beta と gamma を sh_bg という共有ライブラリー・オブジェクト内に結合します。

```
cob2 -o sh_bg.so beta.cbl gamma.cbl -Wl,--version-script,bg.version
```

このコマンドは、コンパイラーおよびリンカーに以下の情報を提供します。

- -o sh_bg beta.cbl gamma.cbl は、beta.cbl と gamma.cbl をコンパイルおよびリンクして、結果の出力モジュールに sh_bg.so という名前を付けます。
- -Wl,--version-script,bg.version は、エクスポート・ファイル bg.version に指定されたシンボルをエクスポートするよう、リンカーに指示します。

3. 次のコマンドを発行して alpha.cbl を再コンパイルし、sh_bg に解決される外部参照を持つ実行可能モジュール m_alpha を生成します。

```
cob2 -o m_alpha alpha.cbl sh_bg.so
```

その後、コマンド m_alpha を発行してプログラムを実行できます。これにより、上記のステップの前に示した出力と同じ出力が生成されます。

m_alpha の実行時には、sh_bg.so が LD_LIBRARY_PATH 環境変数内になければならないことに注意してください。例えば、次のコマンドを発行して、現行ディレクトリーを LD_LIBRARY_PATH に追加できます。

```
export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH
```

最後の 2 つのステップでコマンドを発行する代わりに、これらのコマンドを含む Make ファイルを作成することができます。

466 ページの『例: サンプルの共用ライブラリー用の Make ファイルの作成』

関連タスク
235 ページの『リンカーへのオプションの引き渡し』

関連参照
232 ページの『cob2 オプション』
236 ページの『リンカーの入出力ファイル』
276 ページの『PGMNAME』

例: サンプルの共用ライブラリー用の Make ファイルの作成

次の例は、共有オブジェクト sh_bg.so 用の Make ファイルの作成方法を示しています。これには beta および gamma という 2 つの呼び出し先プログラムがあります。

この Make ファイルは、バージョン・スクリプト bg.version により共有ライブラリーがエクスポートするシンボルが定義されると仮定しています。

(共用ライブラリーの作成は、464 ページの『例: サンプルの共用ライブラリーの作成』に示されています。)

```
#
#
all:     m_abg libbg.a m_alpha

#  Create m_abg containing alpha, beta, and gamma

m_abg:   alpha.cbl beta.cbl gamma.cbl
    cob2 -o m_abg alpha.cbl beta.cbl gamma.cbl

#  Create sh_bg.so containing beta and gamma
#  sh_bg.so is a shared object that exports the symbols defined in bg.version
#    contains both beta and gamma

sh_bg.so: beta.cbl gamma.cbl bg.version
```

```
    rm -f sh_bg.so
    cob2 -o sh_bg.so beta.cbl gamma.cbl -Wl,--version-script,bg.version

#  Create m_alpha containing alpha and using shared object sh_bg.so
m_alpha: alpha.cbl
    cob2 -o m_alpha alpha.cbl sh_bg.so

clean:
    rm -f m_abg m_alpha sh_bg.so *.lst
```

コマンド m_abg またはコマンド m_alpha を実行すると、同じ出力が得られます。

第 25 章 COBOL ランタイム環境の事前初期設定

事前初期設定を使用すると、アプリケーションが COBOL ランタイム環境を一度初期化した後、その環境を使用して複数の処理を実行してから、その環境を明示的に終了することができます。

事前初期設定を使用すると、C/C++ などの非 COBOL 環境から COBOL プログラムを複数回呼び出すことができます。

事前初期設定には、主に次のような利点があります。

- COBOL 環境を、いつでもプログラム呼び出し可能な状態にすることができる。

  COBOL 実行単位内の最初の COBOL プログラムから戻っても、その実行単位は終了しないため、COBOL プログラムが非 COBOL 環境から呼び出される場合でも、最後に使われた状態で呼び出すことができます。
- パフォーマンスが高い。

  COBOL ランタイム環境の作成と解除を何度も行うとオーバーヘッドが生じ、アプリケーションの処理速度が遅くなる可能性があります。

非 COBOL プログラムが、COBOL プログラムを最後に使われた状態で使用する必要がある場合は、複数言語アプリケーションに対して事前初期設定サービスを使用します。 例えば、COBOL プログラムに対する最初の呼び出し時にファイルをオープンした場合、呼び出し側プログラムは、同じプログラムに対する後続の呼び出し時でも、そのファイルがオープンしていることを期待します。

制約事項 : CICS では事前初期設定を行うことができません。

永続的な COBOL ランタイム環境の初期設定と終了を行うには、関連タスクで説明するインターフェースを使用します。事前初期設定された環境で使用される COBOL プログラムが共用ライブラリーに含まれている場合は、事前初期設定された環境が終了するまで、その共用ライブラリーを削除できません。

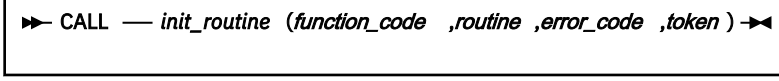
471 ページの『例: COBOL 環境の事前初期設定』

関連タスク
469 ページの『永続的な COBOL 環境の初期設定』
470 ページの『事前初期設定された COBOL 環境の終了』

永続的な COBOL 環境の初期設定

以下のインターフェースを使用して、永続的な COBOL 環境を初期設定します。

CALL init_routine の構文



CALL
init_routine の呼び出し。呼び出しの作成元言語に適した言語エレメントを使用します。

init_routine
初期設定ルーチンの名前: `_iwzCOBOLInit` または `IWZCOBOLINIT`

***function_code* (入力)**
値で渡される 4 バイトの 2 進数。 *function_code* は次のようになります。

1
この関数呼び出しの後に呼び出される最初の COBOL プログラムは、サブプログラムとして扱われます。

© Copyright IBM Corp. 2021

***routine* (入力)**
実行単位が終了した場合に呼び出されるルーチンのアドレス。 この関数に渡されるトークン引数は、実行単位の終了出口ルーチンに渡されます。 このルーチンは、実行単位の終了時に呼び出された際に、ルーチンの呼び出し側へ戻らないで、代わりに longjmp() または exit() を使用する必要があります。

出口ルーチン・アドレスを指定しない場合は、事前初期設定が失敗したことを示す *error_code* が生成されます。

***error_code* (出力)**
4 バイトの 2 進数。 *error_code* は次のようになります。

0
事前初期設定は正常に行われました。

1
事前初期設定は失敗しました。

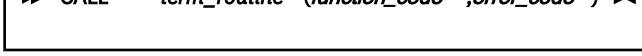
***token* (入力)**
上の実行単位の終了時にこのルーチンが呼び出された際に指定された出口ルーチン に渡される 4 バイトのトークン。

関連タスク
470 ページの『事前初期設定された COBOL 環境の終了』

事前初期設定された COBOL 環境の終了

以下のインターフェースを使用して、事前初期設定された永続的な COBOL 環境を終了します。

CALL term_routine の構文



CALL
term_routine の呼び出し。呼び出しの作成元言語に適した言語エレメントを使用します。

term_routine
終了ルーチンの名前: `_iwzCOBOLTerm` または `IWZCOBOLTERM`

***function_code* (入力)**
値で渡される 4 バイトの 2 進数。 *function_code* は次のようになります。

1
COBOL の `STOP RUN` ステートメントが実行されたかのように、事前初期設定された COBOL ランタイム環境をクリーンアップします。例えば、すべての COBOL ファイルがクローズされます。 ただし、制御権はこのサービスの呼び出し元に戻ります。

***error_code* (出力)**
4 バイトの 2 進数。 *error_code* は次のようになります。

0
終了は正常に行われました。

1
終了は失敗しました。

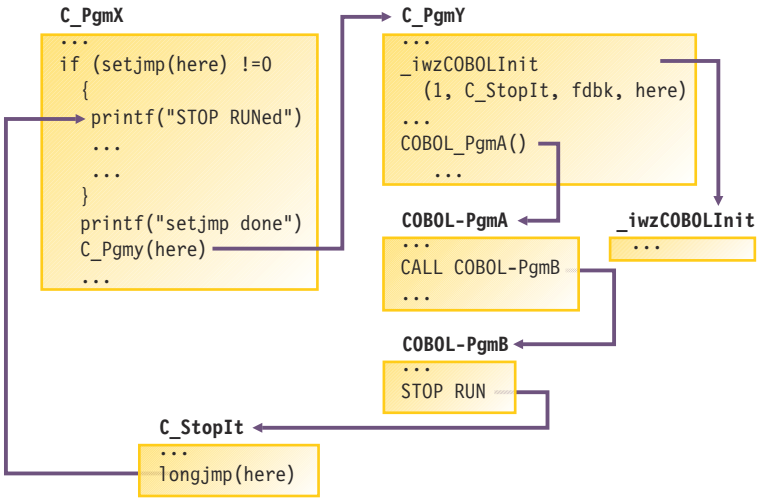
事前初期設定ルーチンの呼び出し後に呼び出される最初の COBOL プログラムは、サブプログラムとして扱われます。 したがって、この (初期) プログラムからの `GOBACK` は、ファイルのクローズなどの実行単位の終了セマンティクスをトリガーしません。 (`STOP RUN` などを使用した) 実行単位の終了では、実行単位の出口ルーチンが呼び出される前に、事前初期設定された COBOL 環境が解放されます。

COBOL 環境がアクティブでない場合: プログラムが終了ルーチンを呼び出し、COBOL 環境が既にアクティブでない場合、呼び出しは実行に何も影響を与えず、制御はエラー・コード 0 とともに呼び出し側に戻ります。

471 ページの『例: COBOL 環境の事前初期設定』

例: COBOL 環境の事前初期設定

次の図は、事前初期設定された COBOL 環境の仕組みを示しています。 この例では、C プログラムが COBOL 環境を初期設定し、COBOL プログラムを呼び出した後、COBOL 環境を終了します。



次の例は、COBOL 事前初期設定の使用法を示しています。 C メインプログラムは、COBOL プログラム XIO を数回呼び出します。 XIO の最初の呼び出しでファイルをオープンし、2 回目の呼び出しでレコードを 1 つ書き込みます (以下同様)。 そして最後の呼び出しで、ファイルをクローズします。 この後、C プログラムは C ストリーム I/O を使用して、このファイルをオープンし、読み取ります。

このプログラムをテストおよび実行するには、コマンド・シェルから次のコマンドを入力します。

```
xlc -c testinit.c
cob2 testinit.o xio.cbl
a.out
```

結果は次のとおりです。

```
_iwzCOBOLinit got 0
xio entered with x=0000000000
xio entered with x=0000000001
xio entered with x=0000000002
xio entered with x=0000000003
xio entered with x=0000000004
xio entered with x=0000000099
StopArg=0
_iwzCOBOLTerm expects rc=0 and got rc=0
FILE1 contains ----
11111
22222
33333
---- end of FILE1
```

この例では、実行単位が COBOL の STOP RUN で終了するのではなく、メインプログラムが _iwzCOBOLTerm を呼び出したときに終了している点に注意してください。

次の C プログラムは、ファイル testinit.c に入っています。

```
#ifdef _Linux
typedef int (*PFN)();
#define LINKAGE
#else
#include <windows.h>
#define LINKAGE _System
#endif

#include   <stdio.h>
#include   <setjmp.h>
```

```
extern void _iwzCOBOLInit(int fcode, PFN StopFun, int *err_code, void *StopArg);
extern void _iwzCOBOLTerm(int fcode, int *err_code);
extern void LINKAGE XIO(long *k);

jmp_buf Jmpbuf;
long StopArg = 0;

int LINKAGE
StopFun(long *stoparg)
{
        printf("inside StopFun¥n");
        *stoparg = 123;
        longjmp(Jmpbuf,1);
}

main()
{
        int rc;
        long k;
        FILE *s;
        int c;

        if (setjmp(Jmpbuf) ==0) {
                _iwzCOBOLInit(1, StopFun, &rc, &StopArg);
                printf( "_iwzCOBOLinit got %d¥n",rc);
                for (k=0; k <= 4; k++) XIO(&k);
                k = 99; XIO(&k);
        }
        else printf("return after STOP RUN¥n");
        printf("StopArg=%d¥n", StopArg);
        _iwzCOBOLTerm(1, &rc);
        printf("_iwzCOBOLTerm expects rc=0 and got rc=%d¥n",rc);
        printf("FILE1 contains ---- ¥n");
        s = fopen("FILE1", "r");
        if (s) {
                while (  (c = fgetc(s) ) != EOF ) putchar(c);
        }
        printf("---- end of FILE1¥n");
}
```

次の COBOL プログラムは、ファイル xio.cbl に入っています。

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.     xio.
*****************************************************************
 ENVIRONMENT    DIVISION.
 CONFIGURATION   SECTION.
 INPUT-OUTPUT    SECTION.
 FILE-CONTROL.
     SELECT file1 ASSIGN TO FILE1
       ORGANIZATION IS LINE SEQUENTIAL
       FILE STATUS IS file1-status.
 . . .
 DATA           DIVISION.
 FILE SECTION.
 FD FILE1.
 01 file1-id pic x(5).
 . . .
 WORKING-STORAGE SECTION.
 01 file1-status  pic xx    value is zero.
 . . .
 LINKAGE SECTION.
*
 01 x               PIC S9(8) COMP-5.
 . . .
 PROCEDURE DIVISION using x.
 . . .
     display "xio entered with x=" x
     if x = 0 then
       OPEN output FILE1
     end-if
     if x = 1  then
        MOVE ALL "1" to file1-id
        WRITE file1-id
     end-if
     if x = 2 then
        MOVE ALL "2" to file1-id
        WRITE file1-id
```

```
      end-if
      if x = 3 then
         MOVE ALL "3" to file1-id
         WRITE file1-id
      end-if
      if x = 99 then
        CLOSE file1
      end-if
      GOBACK.
```

第 26 章 2 桁年の日付の処理

2000 年言語拡張 (MLE) を使用すれば、COBOL プログラムの簡単な変更を行うだけで日付フィールドを定義できます。 コンパイラーは、世紀ウィンドウを使用して整合性を保証したうえで、これらの日付を認識し、作用します。

COBOL プログラムで日付の自動認識が行われるようにするには、以下のステップを実行してください。

1. プログラム内のデータ項目のうち、日付を含むデータ項目のデータ記述記入項目に DATE FORMAT 節を追加します。 比較で使用されないものを含め、すべての日付を DATE FORMAT 節を使用して識別しなければなりません。
2. 日付を拡張する場合は、MOVE または COMPUTE ステートメントを使用して、ウィンドウ表示日付フィールドの内容を拡張日付フィールドにコピーします。
3. 必要であれば、DATEVAL および UNDATE 組み込み関数を使用して、日付フィールドと非日付の間で変換を行います。
4. YEARWINDOW コンパイラー・オプションを使用して、世紀ウィンドウを固定ウィンドウまたはスライディング・ウィンドウとして設定します。
5. DATEPROC(FLAG) コンパイラー・オプションを使用してプログラムをコンパイルし、診断メッセージを検討して、日付処理によって予期しない副次作用が起こっていないかどうかを調べます。
6. コンパイルの結果、通知レベルの診断メッセージだけしかなければ、DATEPROC(NOFLAG) コンパイラー・オプションを使用してプログラムを再コンパイルして、簡潔なリストを作成することができます。

特定のプログラミング手法を使用して、日付処理を利用したり、日付フィールド使用の効果を制御することができます (例えば、日付の比較、日付によるソートとマージ、および日付を伴う算術演算の実行など)。 2000 年言語拡張は、比較、移動と保管、増減など、日付フィールドに関してよく使われる操作について、年先頭型、年単独型、年末尾型の日付フィールドをサポートします。

関連概念
475 ページの『2000 年言語拡張 (MLE)』

関連タスク
477 ページの『日付に関連したロジック問題の解決』
482 ページの『年先行型、年単独型、および年末尾型の日付フィールドの使用』
484 ページの『リテラルを日付として操作する』
487 ページの『日付フィールドに対する算術の実行』
489 ページの『日付処理の明示的制御』
491 ページの『日付関連診断メッセージの分析および回避』
493 ページの『日付処理上の問題の回避』

関連参照
261 ページの『DATEPROC』
291 ページの『YEARWINDOW』
DATE FORMAT 節 (*COBOL for Linux on x86* 言語解説書)

2000 年言語拡張 (MLE)

2000 年言語拡張 とは、西暦 2000 年以降の日付が関係するロジック問題に対処するのに役立つ、DATEPROC コンパイラー・オプションによってアクティブにされる COBOL for Linux の機能のことです。

使用可能にされた場合、言語拡張には以下のものが含まれます。

- DATE FORMAT 節。 日付フィールドを識別し、日付における年部分の位置を指定するには、DATA DIVISION 内の項目にこの節を追加します。

  DATE FORMAT 節にはいくつかの使用上の制約事項があります。例えば、USAGE NATIONAL を持つ項目にこの節を指定することはできません。 詳細は、下記の関連参照を参照してください。

© Copyright IBM Corp. 2021

- 以下の組み込み関数について、関数からの戻り値を日付フィールドとして再解釈すること。
  - DATE-OF-INTEGER
  - DATE-TO-YYYYMMDD
  - DAY-OF-INTEGER
  - DAY-TO-YYYYDDD
  - YEAR-TO-YYYY
- 次の形式の ACCEPT ステートメントにおける概念上のデータ項目 DATE、DATE YYYYMMDD、DAY、および DAY YYYYDDD を日付フィールドとして再解釈すること。
  - ACCEPT *identifier* FROM DATE
  - ACCEPT *identifier* FROM DATE YYYYMMDD
  - ACCEPT *identifier* FROM DAY
  - ACCEPT *identifier* FROM DAY YYYYDDD
- 組み込み関数 UNDATE と DATEVAL。これらは、日付フィールドおよび非日付の選択的再解釈に使用されます。
- 組み込み関数 YEARWINDOW。これは、YEARWINDOW コンパイラー・オプションによって設定された世紀ウィンドウの開始年を検索します。

DATEPROC コンパイラー・オプションは、識別された日付フィールドの特殊な日付中心処理を使用可能にします。 YEARWINDOW コンパイラー・オプションは、2 桁のウィンドウ表示西暦年を解釈するために使用する 100 年ウィンドウ (世紀ウィンドウ) を指定します。

関連概念
476 ページの『この拡張の原則と目標』

関連参照
261 ページの『DATEPROC』
291 ページの『YEARWINDOW』
日付フィールドの使用に関する制約事項 (*COBOL for Linux on x86* 言語解説書)

この拡張の原則と目標

2000 年言語拡張から最大のメリットが得られるようにするには、COBOL 言語にこれが導入された理由を理解する必要があります。

2000 年言語拡張が焦点を当てているのは、次の原則だけです。

- 日付のセマンティクスを使用して再コンパイルされるプログラムは、十分にテストされ、企業にとって価値のある資産です。 関連する制限事項は、プログラムの 2 桁の年号が 1900-1999 の範囲に制限されることだけです。
- 日付の年号以外の部分には特別な処理は行われません。 このため、サポートされる日付形式の年号以外の部分は X で表されます。 それ以外の表記を使用すると、既存のプログラムの意味が変わってしまうことがあります。 提供されている唯一の日付依存型セマンティクスは、プログラムの世紀ウィンドウに関して、日付の 2 桁の年部分を自動的に拡張 (および短縮) するということです。
- 4 桁の年号部分を持つ日付が重要になるのは、通常、ウィンドウ表示日付と組み合わせて使用される場合です。 それ以外の場合、4 桁年号日付と非日付の間に違いはほとんどありません。

上記の原則に基づき、2000 年言語拡張はいくつかの目標を達成するよう設計されています。 日付処理問題を解決するために満足しなければならない目標を評価し、それらを 2000 年言語拡張の目標と比較して、アプリケーションがそこからどのように利益を得ることができるかを判別しなければなりません。 新規アプリケーション、または既存アプリケーションへの拡張では、もっと後になるまで拡張できない古いデータをアプリケーションで使用している場合を除いて、拡張部分の使用を考慮してはなりません。

2000 年言語拡張の目標は、次のとおりです。

- 現在指定されているアプリケーション・プログラムの実用的な存続期間を拡張します。

- ソース変更を最小限にとどめ、可能であれば、DATA DIVISION の日付フィールドの宣言の増加だけに限定します。 世紀ウィンドウ・ソリューションをインプリメントするためには、PROCEDURE DIVISION のプログラム・ロジックを変更する必要がないようにします。
- 日付フィールドを追加するときは、プログラムの既存のセマンティクスを保持しなければなりません。 例えば、次のステートメントの場合のように、日付がリテラルとして表される場合、そのリテラルは、比較対象の日付フィールドと互換性があると見なされます (ウィンドウ操作または拡張されます)。

```
If Expiry-Date Greater Than 980101 . . .
```

  既存のプログラムは、リテラルとして表された 2 桁年号の日付が 1900-1999 の範囲内にあると想定するので、拡張でこの想定が変更されることはありません。
- ウィンドウ操作機能は長期間の使用を目的としたものではありません。 後で実施できる長期の解決策が採用されるまで、アプリケーションの実用的な存続期間を拡張することを意図しています。
- 拡張日付フィールド機能は、ファイルおよびデータベースの日付フィールドの拡張を支援するものとして、長期間の使用を意図しています。

拡張部分は、完全指定または完全な日付中心のデータ型に、認識できるセマンティクス (例えばグレゴリオ暦の月と日の部分) を与えません。 拡張部分は、日付の年号部分に特別なセマンティクスを与えるだけです。

日付に関連したロジック問題の解決

日付処理問題を解決するための助けとして、世紀ウィンドウ、内部ブリッジング、 または全フィールド拡張を使用する、いずれかのアプローチを採用することができます。

世紀ウィンドウ
世紀ウィンドウを定義し、ウィンドウ表示日付を含むフィールドを指定します。 コンパイラーは、これらのデータ・フィールドの 2 桁の年号を世紀ウィンドウに従って解釈します。

内部ブリッジング
ファイルおよびデータベースはまだ 4 桁年の日付に移行されていないが、プログラムでは 4 桁に拡張した年のロジックを使用したい場合、そのような日付を 4 桁年の日付として処理するための内部ブリッジング手法を使用することができます。

全フィールド拡張
このソリューションは、2 桁の年の日付フィールドを、ファイルおよびデータベースの中で完全な 4 桁の年になるように明示的に拡張した後、そのフィールドをプログラムの中で拡張形式で使用する方法です。 これは、すべてのアプリケーションについて信頼できる日付処理がなされる唯一の方法です。

それぞれの方法で 2000 年言語拡張を使用して解決することができますが、以下に示すようにそれぞれに利点と欠点があります。

表 *49.* ***2000*** **年問題のソリューションの利点および欠点**

局面	世紀ウィンドウ	内部ブリッジング	全フィールド拡張
インプリメンテーション	速くて容易ですが、すべてのアプリケーションに合うわけではありません。	データ破壊のリスクがあります。	データベース、コピーブック、およびプログラムに対する変更をすべて同期させる必要があります。
テスト	プログラム・ロジックの変更はないので、テストはほとんど必要ありません。	プログラム・ロジックに行われる変更が直接的なものなので、テストは簡単です。	
修正期間	プログラムは 2000 年を過ぎても機能しますが、長期的なソリューションとはなり得ません。	プログラムは 2000 年を過ぎても機能しますが、永続的なソリューションとはなり得ません。	永続的なソリューションです。

表 *49.* ***2000*** **年問題のソリューションの利点および欠点** (続き)			
局面	**世紀ウィンドウ**	**内部ブリッジング**	**全フィールド拡張**
パフォーマンス	パフォーマンスが低下する可能性があります。	良好なパフォーマンスが得られます。	最適なパフォーマンスが得られます。
保守			保守が容易になります。

479 ページの『例: 世紀ウィンドウ』
479 ページの『例: 内部ブリッジング』
480 ページの『例: 拡張日付形式へのファイルの変換』

関連タスク
478 ページの『世紀ウィンドウの使用』
479 ページの『内部ブリッジングの使用』
480 ページの『完全フィールド拡張への移行』

世紀ウィンドウの使用

世紀ウィンドウは、任意の 2 桁年が固有となる 100 年の間隔 (1950 から 2049 など) です。 ウィンドウ表示日付フィールドについては、YEARWINDOW コンパイラー・オプションを使用して、世紀ウィンドウ開始日を指定することができます。

DATEPROC オプションが有効であると、コンパイラーは、プログラムの 2 桁の日付フィールドにこのウィンドウを適用します。 例えば、1930-2029 の世紀ウィンドウの場合、COBOL は 2 桁の年号を次のように解釈します。

- 00 から 29 の年の値は、2000 から 2029 として解釈されます。
- 30 から 99 の年の値は、1930 から 1999 として解釈されます。

この世紀ウィンドウをインプリメントするには、プログラム内で DATE FORMAT 節を使用して日付フィールドを識別し、 YEARWINDOW コンパイラー・オプションを使用して世紀ウィンドウを固定ウィンドウまたはスライディング・ウィンドウとして定義します。

- 固定ウィンドウの場合は、YEARWINDOW オプションの値として 1900 から 1999 の 4 桁の年号を指定します。

  例えば、YEARWINDOW(1950) は、1950-2049 の固定ウィンドウを定義します。
- スライディング・ウィンドウの場合は、YEARWINDOW オプションの値として -1 から -99 の負の整数を指定します。

  例えば、YEARWINDOW(-50) は、プログラムが稼働する年の 50 年前に始まるスライディング・ウィンドウを定義します。 そのため、プログラムが 2010 年に稼働中である場合、世紀ウィンドウは 1960 から 2059 であり、2011 年には世紀ウィンドウは自動的に 1961 から 2060 になります。以下同様です。

コンパイラーは、識別されている日付フィールドに対する操作に、世紀ウィンドウを自動的に適用します。ウィンドウ操作をインプリメントするための余分のプログラム・ロジックは必要ありません。

479 ページの『例: 世紀ウィンドウ』

関連参照
261 ページの『DATEPROC』
291 ページの『YEARWINDOW』
DATE FORMAT 節 (*COBOL for Linux on x86* 言語解説書)
日付フィールドの使用に関する制約事項 (*COBOL for Linux on x86* 言語解説書)

例: 世紀ウィンドウ

次の例は、DATE FORMAT 節を使用してプログラムを変更して自動日付ウィンドウ操作を使用する方法を (太字で) 示しています。

```
CBLQUOTE,NOOPT,DATEPROC(FLAG),YEARWINDOW(-60)
. . .
01  Loan-Record.
    05  Member-Number   Pic X(8).
    05  DVD-ID          Pic X(8).
    05  Date-Due-Back   Pic X(6) Date Format yyxxxx.
    05  Date-Returned   Pic X(6) Date Format yyxxxx.
. . .
    If Date-Returned > Date-Due-Back Then
       Perform Fine-Member.
```

PROCEDURE DIVISION に対する変更はありません。 2 つの日付フィールドに DATE FORMAT 節を追加した意味は、コンパイラーがそれらをウィンドウ表示日付フィールドとして認識し、したがって、IF ステートメントを処理するときには世紀ウィンドウを適用するということです。 例えば、Date-Due-Back が 100102 (2010 年 1 月 2 日) を含み、Date-Returned が 091231 (2009 年 12 月 31 日) を含んでいる場合、Date-Returned は Date-Due-Back より小さいので (以前なので)、プログラムは Fine-Member 段落を実行しません。 (プログラムは、時間通りに DVD が戻されたかどうかを検査します。)

内部ブリッジングの使用

内部ブリッジングの場合、プログラムを適切に構成する必要があります。

以下の手順を 実行します。

1. 2 桁年号の日付を持つ入力ファイルを読み取る。
2. これらの 2 桁の日付をウィンドウ表示日付フィールドとして宣言し、コンパイラーがこれらの日付を自動的に 4 桁年号の日付に拡張できるように、これらを拡張日付フィールドに移動する。
3. プログラムの本体では、すべての日付処理に 4 桁年号の日付を使用する。
4. ウィンドウ操作により日付を 2 桁の年号に戻す。
5. 2 桁年号の日付を出力ファイルに書き込む。

このプロセスは、完全拡張日付ソリューションへの便利な移行パスになり、ウィンドウ表示日付を使用するよりもパフォーマンス上の利点もあるかもしれません。

この手法を使用すると、プログラム・ロジックへの変更は最小で済みます。 単に、日付を拡張および短縮するステートメントを追加し、また日付を参照するステートメントを、レコードの中の 2 桁年号のフィールドではなく WORKING-STORAGE の中の 4 桁年号の日付フィールドを使用するように変更するだけです。

出力のために日付を変換して 2 桁の年に戻しているので、年が世紀ウィンドウの範囲外になる可能性を考慮しておいてください。 例えば、日付フィールドに 2020 年が入っているが、世紀ウィンドウが 1920 から 2019 である場合、日付は世紀ウィンドウの外側になります。 年を 2 桁年フィールドに移動させるだけでは誤りになります。 この問題が生じないようにするには、COMPUTE ステートメントを使用して日付を保管し、ON SIZE ERROR 句を指定して日付が世紀ウィンドウの外側であるかどうかを検出することができます。

479 ページの『例: 内部ブリッジング』

関連タスク

478 ページの『世紀ウィンドウの使用』
487 ページの『日付フィールドに対する算術の実行』
480 ページの『完全フィールド拡張への移行』

例: 内部ブリッジング

次の例は、プログラムを変更して内部ブリッジングをインプリメントする方法を (太字で) 示しています。

```
CBL  DATEPROC(FLAG),YEARWINDOW(-60)
```

```
    . . .
    File Section.
    FD  Customer-File.
    01  Cust-Record.
        05  Cust-Number     Pic 9(9) Binary.
        . . .
        05  Cust-Date       Pic 9(6) Date Format yyxxxx.
    Working-Storage Section.
    77  Exp-Cust-Date       Pic 9(8) Date Format yyyyxxxx.
    . . .
    Procedure Division.
        Open I-O Customer-File.
        Read Customer-File.
        Move Cust-Date to Exp-Cust-Date.
        . . .
   *=====================================================*
   * Use expanded date in the rest of the program logic  *
   *=====================================================*
        . . .
        Compute Cust-Date = Exp-Cust-Date
             On Size Error
                Display "Exp-Cust-Date outside century window"
        End-Compute
        Rewrite Cust-Record.
```

完全フィールド拡張への移行

2000 年言語拡張を使用すれば、日付フィールドを完全に拡張するソリューションに向けて段階的に移行することができます。

以下の手順を 実行します。

1. 世紀ウィンドウ・ソリューションを適用し、より永続的なソリューションをインプリメントするためのリソースが用意されるまで、このソリューションを使用する。
2. 内部ブリッジング・ソリューションを適用する。 これは、ファイルでは 2 桁年号の形式で日付を保持した状態のまま、プログラムの中では拡張日付を使用する方法です。 プログラムの本体の中ではロジックにそれ以上の変更を行わないので、全フィールド拡張ソリューションにより容易に進むことができます。
3. ファイル設計およびデータベース定義を、4 桁年号の日付を使用するように変更する。
4. COBOL コピーブックを、4 桁年号の日付フィールドを反映するように変更する。
5. ユーティリティー・プログラム (または特殊な目的の COBOL プログラム) を実行して、古い形式のファイルから新しい形式にコピーする。
6. プログラムを再コンパイルし、レグレッション・テストおよび日付テストを行う。

最初の 2 つのステップを完了したら、それ以降のステップは何回でも繰り返すことができます。 すべてのファイルのすべての日付フィールドを同時に変更する必要はありません。 この方法では、段階的に変換するファイルを、業務上の必要性または他のアプリケーションとのインターフェースなどを基準にして選択することができます。

この方法を使用する場合、特別目的のプログラムを作成してファイルを拡張日付形式に変換する必要があります。

480 ページの『例: 拡張日付形式へのファイルの変換』

例: 拡張日付形式へのファイルの変換

次の例は、日付フィールドを拡張しながら、あるファイルから別のファイルにコピーする簡単なプログラムを示しています。 日付が拡張されるため、出力ファイルのレコード長は入力ファイルのレコード長より長くなっています。

```
CBL  QUOTE,NOOPT,DATEPROC(FLAG),YEARWINDOW(-80)
      ************************************************
      ** CONVERT - Read a file, convert the date    **
      **           fields to expanded form, write   **
      **           the expanded records to a new    **
      **           file.                            **
      ************************************************
```

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  CONVERT.

       ENVIRONMENT DIVISION.

       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT INPUT-FILE
                  ASSIGN TO INFILE
                  FILE STATUS IS INPUT-FILE-STATUS.

           SELECT OUTPUT-FILE
                  ASSIGN TO OUTFILE
                  FILE STATUS IS OUTPUT-FILE-STATUS.

       DATA DIVISION.
       FILE SECTION.
       FD  INPUT-FILE
           RECORDING MODE IS F.
       01  INPUT-RECORD.
           03  CUST-NAME.
               05  FIRST-NAME  PIC X(10).
               05  LAST-NAME   PIC X(15).
           03  ACCOUNT-NUM     PIC 9(8).
           03  DUE-DATE        PIC X(6) DATE FORMAT YYXXXX.    (1)
           03  REMINDER-DATE   PIC X(6) DATE FORMAT YYXXXX.
           03  DUE-AMOUNT      PIC S9(5)V99 COMP-3.

       FD  OUTPUT-FILE
           RECORDING MODE IS F.
       01  OUTPUT-RECORD.
           03  CUST-NAME.
               05  FIRST-NAME  PIC X(10).
               05  LAST-NAME   PIC X(15).
           03  ACCOUNT-NUM     PIC 9(8).
           03  DUE-DATE        PIC X(8) DATE FORMAT YYYYXXXX.  (2)
           03  REMINDER-DATE   PIC X(8) DATE FORMAT YYYYXXXX.
           03  DUE-AMOUNT      PIC S9(5)V99 COMP-3.

       WORKING-STORAGE SECTION.

       01  INPUT-FILE-STATUS   PIC 99.
       01  OUTPUT-FILE-STATUS  PIC 99.

       PROCEDURE DIVISION.

           OPEN INPUT INPUT-FILE.
           OPEN OUTPUT OUTPUT-FILE.

       READ-RECORD.
           READ INPUT-FILE
                AT END GO TO CLOSE-FILES.
           MOVE CORRESPONDING INPUT-RECORD TO OUTPUT-RECORD.   (3)
           WRITE OUTPUT-RECORD.

           GO TO READ-RECORD.

       CLOSE-FILES.
           CLOSE INPUT-FILE.
           CLOSE OUTPUT-FILE.

           EXIT PROGRAM.

       END PROGRAM CONVERT.
```

注:

(1)
入力レコード内のフィールド DUE-DATE と REMINDER-DATE は 2 桁年コンポーネントを持つグレゴリオ日付です。 これらのフィールドは DATE FORMAT 節で定義されているので、コンパイラーはこれらをウィンドウ表示日付フィールドとして認識します。

(2)
出力レコードには、同じ 2 つのフィールドが拡張日付形式で入れられます。 これらのフィールドは DATE FORMAT 節で定義されているので、コンパイラーはこれらを 4 桁年号の日付フィールドとして扱います。

(3)
MOVE CORRESPONDING ステートメントは、INPUT-RECORD 内の各項目を OUTPUT-RECORD 内の対応する項目に移動します。 2 つのウィンドウ表示日付フィールドが対応する拡張日付フィールドに移動されると、コンパイラーは現行の世紀ウィンドウを使用して年号値を拡張します。

年先行型、年単独型、および年末尾型の日付フィールドの使用

年先行型または年単独型のどちらかのタイプの 2 つの日付フィールドを比較するとき、その 2 つの日付には互換性がなければなりません。すなわち、その 2 つは、年以外の文字の数は同じでなければなりません。年部分の桁数は同じである必要はありません。

年先行型日付フィールドは、DATE FORMAT 指定が、YY または YYYY とその後に続く 1 つ以上の X で構成される日付フィールドです。 年単独型 日付フィールドの日付形式は YY または YYYY だけです。 年末尾型日付フィールドは、DATE FORMAT 節で 1 つ以上の X とその後に YY または YYYY を指定している日付フィールドです。

年末尾型の日付形式は日付の表示によく使われますが、コンピューターにとってはあまり便利ではありません。それは、日付の中で最も比重の大きい部分である年が日付表現における比重の小さい部分になるからです。

機能的な年末尾型日付フィールドのサポートは、等しいか等しくないかの比較とある種の代入操作のみに限られます。 オペランドは、日付形式が同一 (年末尾型) の日付であるか、または日付と非日付でなければなりません。 コンパイラー自体には、年末尾型日付の操作のための自動ウィンドウ操作機能は含まれていません。 年末尾型日付の算術演算など、サポートされていない方法で使用されると、コンパイラーはエラー・レベル・メッセージを提供します。

年末尾型日付のためのより一般的な日付処理が必要な場合は、日付の年部分を分離して処理する必要があります。

483 ページの『例: 年先行型日付フィールドの比較』

関連概念
482 ページの『互換性のある日付』

関連タスク
483 ページの『その他の日付形式の使用』

互換性のある日付

互換性のある日付 という用語の意味は、日付が DATA DIVISION で使用される場合と PROCEDURE DIVISION で使用される場合とでは異なります。

DATA DIVISION で使用する場合、日付フィールドの宣言、および従属データ項目や REDEFINES 節などの COBOL 言語エレメントを制御する規則 が関係します。 次の例では、Review-Year は Review-Date への従属データ項目として宣言することができるので、Review-Date と Review-Year には互換性があります。

```
01  Review-Record.
    03  Review-Date              Date Format yyxxxx.
        05  Review-Year Pic XX    Date Format yy.
        05  Review-M-D  Pic XXXX.
```

PROCEDURE DIVISION での使用は、比較、移動、算術式などの演算で日付フィールドを一緒に使用する方法に関連しています。 年先行型および年単独型フィールドに互換性があるとみなされるには、日付フィールドに同じ数の年以外の文字が含まれている必要があります。 例えば、DATE FORMAT YYXXXX を持つフィールドは、同じ日付形式の別のフィールドや YYYYXXXX フィールドと互換性がありますが、YYXXX フィールドとの互換性はありません。

年末尾型日付フィールドの場合は、DATE FORMAT 節が同じである必要があります。 特に、ウィンドウ表示日付フィールドと拡張年末尾型日付フィールドとの相互演算は許されません。 例えば、日付形式が

XXXXYY の日付フィールドを別の XXXXYY 日付フィールドへ移動させることはできますが、 XXXXYYYY 形式の日付フィールドへ移動させることはできません。

演算の中の日付フィールドに互換性があれば、日付フィールドに対して、または日付フィールドと非日付が組み合わされたものに対して演算を実行できます。 例えば、次のように定義されているとします。

```
01  Date-Gregorian-Win  Pic 9(6) Packed-Decimal Date Format yyxxxx.
01  Date-Julian-Win     Pic 9(5) Packed-Decimal Date Format yyxxx.
01  Date-Gregorian-Exp  Pic 9(8) Packed-Decimal Date Format yyyyxxxx.
```

2 つのフィールドの年以外の数字の数が異なるので、次のステートメントは成立しません。

```
If Date-Gregorian-Win Less than Date-Julian-Win . . .
```

2 つのフィールドの年以外の数字の数が同じなので、次のステートメントは受け入れられます。

```
If Date-Gregorian-Win Less than Date-Gregorian-Exp . . .
```

この場合は、比較が確実に意味のあるものになるようにするために、世紀ウィンドウがウィンドウ表示日付フィールド (Date-Gregorian-Win) に適用されます。

非日付を日付フィールドと一緒に使用した場合、非日付は日付フィールドと互換性があると見なされるか、または単純な数値として扱われます。

例: 年先行型日付フィールドの比較

次の例では、ウィンドウ化日付フィールドが拡張日付フィールドと比較されます。

```
77  Todays-Date         Pic X(8) Date Format yyyyxxxx.
01  Loan-Record.
    05  Date-Due-Back   Pic X(6) Date Format yyxxxx.
. . .
    If Date-Due-Back > Todays-Date Then . . .
```

世紀ウィンドウが Date-Due-Back に適用されます。 Todays-Date に DATE FORMAT 節を指定して、それを拡張日付フィールドとして定義しなければなりません。 そうしないと、そのフィールドは非日付フィールドとして扱われ、その結果、そのフィールドは年の桁数が Date-Due-Back と同じであるものと見なされます。 コンパイラーは 1900-1999 の仮定による世紀ウィンドウを適用するため、矛盾した比較が作成されます。

その他の日付形式の使用

自動ウィンドウ操作に適格であるためには、日付フィールドは、フィールドの最初の部分または唯一の部分として 2 桁年を含んでいる必要があります。 フィールドの残り (ある場合) は、1 から 4 文字を含んでいる必要がありますが、どんな内容であるかは重要ではありません。

アプリケーションの中に、このような基準に合わない日付フィールドがある場合は、DATE FORMAT 節によって日付の年部分だけを日付フィールドとして定義するためのコード変更が必要かもしれません。 このようなタイプの日付形式の例として、次のようなものがあります。

- 2 桁年、月の省略語を含む 3 文字、および日を表す 2 桁で構成される 7 文字フィールド。 日付フィールドには 1 から 4 文字の年以外の文字しか使えないため、この形式はサポートされません。
- 形式 DDMMYY のグレゴリオ暦日付。 年の部分が日付の最初の部分ではないので、自動ウィンドウ操作は適用されません。 このような年末尾型日付は、SORT または MERGE ステートメントではウィンドウ表示キーとして全面的にサポートされており、また一部の COBOL 操作でもサポートされています。

このような場合に日付のウィンドウ操作を使用する必要があるなら、日付の年部分を分離するために何らかのコードを追加する必要があります。

例: 年の分離

次の例は、DDMMYY という形式である日付フィールドの年部分を切り離すにはどうすればよいとかを示しています。

```
03  Last-Review-Date Pic 9(6).
03  Next-Review-Date Pic 9(6).
. . .
Add 1 to Last-Review-Date Giving Next-Review-Date.
```

上記のコードでは、Last-Review-Date が 230110 (2010 年 1 月 23 日) を含んでいる場合、ADD ステートメントの実行後に Next-Review-Date には 230111 (2011 年 1 月 23 日) が入ることになります。 これは、次の年次検査 (review) 日付を設定するための簡単な方法です。 ただし、Last-Review-Date に 230199 が含まれる場合に 1 を加算すると 230200 になりますが、 これは要求する結果ではありません。

これらの日付フィールドは年が日付フィールドの最初の部分ではないので、年の部分を分離するための何らかのコードを追加しないと、DATE FORMAT 節を適用することはできません。 次の例では、2 つの日付フィールドの年部分が分離されており、COBOL は世紀ウィンドウを適用して矛盾のない結果を維持できます。

```
03  Last-Review-Date Date Format xxxxyy.
    05  Last-R-DDMM  Pic 9(4).
    05  Last-R-YY    Pic 99 Date Format yy.
03  Next-Review-Date Date Format xxxxyy.
    05  Next-R-DDMM  Pic 9(4).
    05  Next-R-YY    Pic 99 Date Format yy.
. . .
Move Last-R-DDMM to Next-R-DDMM.
Add 1 to Last-R-YY Giving Next-R-YY.
```

リテラルを日付として操作する

ウィンドウ表示日付フィールドに、関連したレベル 88 条件名がある場合、 VALUE 節のリテラルは、想定された世紀ウィンドウ 1900-1999 では なく、コンパイル単位の世紀ウィンドウでウィンドウ操作が行われます。

例えば、次のようなデータ定義があるとします。

```
05  Date-Due        Pic 9(6)  Date Format yyxxxx.
    88  Date-Target           Value 101220.
```

世紀ウィンドウが 1950-2049 で、Date-Due の内容が 101220 (2010 年 12 月 20 日) であれば、以下の最初の条件は真に評価され、2 番目の条件は偽に評価されます。

```
If Date-Target. . .
If Date-Due = 101220
```

リテラル 101220 は非日付として扱われるため、1900-1999 の仮定による世紀ウィンドウに対してウィンドウ操作され、1909 年 12 月 20 日を表すことになります。 しかし、レベル 88 の条件名の VALUE 節にリテラルが指定された場合には、そのリテラルは、それが付加されるデータ項目の一部になります。 このデータ項目はウィンドウ表示日付フィールドなので、そのデータ項目が参照されるときには、必ず世紀ウィンドウが適用されます。

比較式の中で DATEVAL 組み込み関数を使用してリテラルを日付フィールドに変換することも可能です。 結果の日付フィールドは、ウィンドウ表示日付フィールドまたは拡張日付フィールドとして扱われるので、比較に矛盾は生じません。 例えば、上記の定義を使用すれば、以下の条件はどちらも真に評価されます。

```
If Date-Due = Function DATEVAL (101220 "YYXXXX")
If Date-Due = Function DATEVAL (20101220 "YYYYXXXX")
```

レベル 88 条件名では、VALUE 節に THRU オプションを指定できますが、スライディング・ウィンドウではなく固定世紀ウィンドウを YEARWINDOW コンパイラー・オプションで指定する必要があります。 次に例を示します。

```
05  Year-Field  Pic 99  Date Format yy.
    88 In-Range         Value 98 Thru 06.
```

この形式の場合、範囲内の 2 番目の項目のウィンドウ操作値は、1 つ目の項目のウィンドウ操作値より大きくなければなりません。 ただし、コンパイラーがこの差を検査できるのは、YEARWINDOW コンパイラー・オプションで固定世紀ウィンドウが指定されている場合のみです (例えば、YEARWINDOW(-70) ではなく YEARWINDOW(1940) が指定されている場合)。

ウィンドウ操作の順序に関する要件は、年末尾型日付フィールドには適用されません。 年末尾型日付フィールドに対して THROUGH 句を含む条件名 VALUE 節を指定する場合、2 つのリテラルは通常の COBOL 規則に従っていなければなりません。 つまり、最初のリテラルは 2 番目のリテラルより小さくなければなりません。

関連概念
485 ページの『仮定による世紀ウィンドウ』
486 ページの『非日付の処理』

関連タスク
489 ページの『日付処理の明示的制御』

仮定による世紀ウィンドウ

プログラムがウィンドウ表示日付フィールドを使用すると、コンパイラーは YEARWINDOW コンパイラー・オプションによって定義された世紀ウィンドウをコンパイル単位に適用します。 ウィンドウ化日付フィールドを非日付と一緒に使用する場合で、コンテキスト上、非日付はウィンドウ化日付として扱う必要がある場合、コンパイラーは仮定による世紀ウィンドウを使用して非日付フィールドを解決します。

仮定による世紀ウィンドウは 1900-1999 であり、これは、一般にはコンパイル単位の世紀ウィンドウと同じではありません。

多くの場合、特にリテラルの非日付の場合、この仮定による世紀ウィンドウは正しい選択です。 次の構成では、リテラルは、世紀ウィンドウが例えば 1975-2074 の場合でも、1972 年 1 月 1 日という元の意味を保たなければならず、2072 に変わるべきではありません。

```
01  Manufacturing-Record.
    03  Makers-Date Pic X(6) Date Format yyxxxx.
. . .
    If Makers-Date Greater than "720101" . . .
```

この仮定が正しくても、DATEVAL 組み込み関数を使用することによって年を明示し、警告レベルの診断メッセージ (仮定による世紀ウィンドウを適用することから生じるメッセージ) が出ないようにするのが適切です。

```
If Makers-Date Greater than
    Function Dateval("19720101" "YYYYXXXX") . . .
```

時には、仮定が正しくない場合があります。 次の例の場合、Project-Controls が COPY メンバーの中にあり、このメンバーが、西暦 2000 年処理用にまだアップグレードされていない他のアプリケーションによって使用され、このため Date-Target では DATE FORMAT 節を指定することができないものとします。

```
01  Project-Controls.
    03  Date-Target     Pic 9(6).
. . .
01  Progress-Record.
    03  Date-Complete   Pic 9(6) Date Format yyxxxx.
```

```
. . .
    If Date-Complete Less than Date-Target . . .
```

この例で、Date-Complete が Date-Target より前の日付 (より小) になるようにするには、次の 3 つの条件が真でなければなりません。

- 世紀ウィンドウが 1910-2009 である。
- Date-Complete が 991202 (グレゴリオ日付 1999 年 12 月 2 日) である。
- Date-Target が 000115 (グレゴリオ日付 2000 年 1 月 15 日) である。

ただし、Date-Target は、DATE FORMAT 節を持っていないので、非日付です。 したがって、これに適用される世紀ウィンドウは、仮定による世紀ウィンドウ 1900-1999 であり、1900 年 1 月 15 日として処理されることになります。 この 結果、Date-Complete は Date-Target より大きいことになりますが、これは要求する結果ではありません。

この場合には、DATEVAL 組み込み関数を使用して、この比較のために Date-Target を日付フィールドに変換すべきです。 例えば、次のように指定します。

```
If Date-Complete Less than
    Function Dateval (Date-Target "YYXXXX") . . .
```

関連タスク
489 ページの『日付処理の明示的制御』

非日付の処理

コンパイラーが非日付をどのように扱うかはコンテキストによって異なります。

以下の項目は非日付です。

- リテラル値。
- データ記述に DATE FORMAT 節が含まれていないデータ項目。
- 一部の算術式の結果 (中間または最終的)。 例えば、2 つの日付フィールドの差は非日付ですが、日付フィールドと非日付の和は日付フィールドです。
- UNDATE 組み込み関数からの出力。

非日付を日付フィールドと一緒に使用する場合、コンパイラーは非日付を形式が日付フィールドと互換性のある日付、または単純な数値と解釈します。 この解釈は、次に示すように、日付フィールドおよび非日付が使われたコンテキストによって異なります。

- 比較

  日付フィールドを非日付と比較する場合、非日付は、年と年以外の文字の文字数の点で日付フィールドと互換性があると見なされます。 次の例では、971231 という非日付リテラルをウィンドウ表示日付フィールドと比較します。

  ```
  01  Date-1    Pic 9(6) Date Format yyxxxx.
  . . .
      If Date-1 Greater than 971231 . . .
  ```

  非日付リテラル 971231 は、Date-1 と同じ DATE FORMAT を持っているかのように処理されますが、開始年 (base year) は 1900 です。

- 算術演算

  サポートされるすべての算術演算で、非日付フィールドは単純数値として処理されます。 次の例では、Date-2 のグレゴリオ日付に数値 10000 を加算しています。つまり、日付に 1 年が加えられます。

  ```
  01  Date-2    Pic 9(6) Date Format yyxxxx.
  ```

```
. . .
    Add 10000 to Date-2.
```

- MOVE ステートメント

  日付フィールドを非日付に移動することはサポートされません。 しかし、UNDATE 組み込み関数を使用してこれを行うことができます。

  非日付を日付フィールドに移動する場合、送信フィールドは、年と年以外の文字の文字数の点で、受信フィールドと互換性があると見なされます。 例えば、非日付をウィンドウ表示日付フィールドに移動する場合、非日付フィールドには 2 桁年と互換性のある日付が入っているものと想定されます。

符号条件の使用

アプリケーションの中には、日付フィールドで、トリガーとして機能する (つまり、ある特殊な処理が必要であることを知らせる) ゼロのような特殊値を使用するものがあります。

例えば、Orders ファイルで Order-Date に 0 の値を使用すると、レコードはオーダー・レコードではなく顧客合計レコードであることを意味するという場合です。 プログラムは、次のように日付を 0 と比較します。

```
01  Order-Record.
    05  Order-Date      Pic S9(5) Comp-3 Date Format yyxxx.
. . .
    If Order-Date Equal Zero Then . . .
```

しかし、この比較は有効ではありません。リテラル値の Zero は非日付であるため、仮定による世紀ウィンドウに対してウィンドウ操作されて値は 1900000 になります。

あるいは、次のように、リテラル比較の代わりに符号条件を使用できます。 符号条件を使用すると、Order-Date は非日付として扱われ、世紀ウィンドウは考慮に入れられません。

```
If Order-Date Is Zero Then . . .
```

このアプローチが適用されるのは、符号条件内のオペランドが算術式ではなく単純な ID である場合だけです。 式を指定した場合は、適宜世紀ウィンドウが適用されてその式が最初に評価されます。 そのあと、その式の結果に対して符号条件が比較されます。

代わりに UNDATE 組み込み関数を使用しても、同じ結果が得られます。

関連概念
486 ページの『非日付の処理』

関連タスク
489 ページの『日付処理の明示的制御』

関連参照

261 ページの『DATEPROC』

日付フィールドに対する算術の実行

任意の数値データ項目と同様に、数値日付フィールドにも算術操作を実行できます。 必要に応じて、世紀ウィンドウが計算に使用されます。

しかし、算術式やステートメントのどこで日付フィールドを使用できるかについては制限があります。 日付フィールドが含まれる算術演算は、次のものに限定されます。

- 日付フィールドへの非日付データの加算
- 日付フィールドからの非日付データの減算

- 互換性のある日付フィールドから日付フィールドを減算し、非日付の結果を与えること

以下の算術演算は使用できません。

- 互換性のない日付フィールド間の演算
- 2 つの日付フィールドの加算
- 非日付データからの日付フィールドの減算
- 日付フィールドに適用される単項減算
- 日付フィールドの関係した乗算、除算、またはべき乗計算
- 年末尾型日付フィールドを指定する算術式
- 年末尾型日付フィールドを指定する算術式 (送信フィールドが非日付の場合に、受け取りデータ項目として指定する場合を除く)

日付フィールドの年部分については日付のセマンティクスが提供されていますが、年以外の部分については提供されていません。 例えば、値 980831 を含むグレゴリオ暦のウィンドウ操作日付フィールドに 1 を加算すると、980901 ではなく 980832 の結果になります。

関連タスク
488 ページの『ウィンドウ表示日付フィールドのオーバーフローの考慮』
489 ページの『評価の順序の指定』

ウィンドウ表示日付フィールドのオーバーフローの考慮

(年末尾型でない) ウィンドウ表示日付フィールドが算術演算に関与するときは、世紀ウィンドウに応じて、まずフィールドの 年号構成要素の値が 1900 または 2000 だけ増分されたかのように処理されます。

```
01 Review-Record.
    03 Last-Review-Year Pic 99 Date Format yy.
    03 Next-Review-Year Pic 99 Date Format yy.
. . .
    Add 10 to Last-Review-Year Giving Next-Review-Year.
```

上の例で、世紀ウィンドウが 1910-2009 で、Last-Review-Year の値が 98 である場合は、1998 を与えるため、まず Last-Review-Year が 1900 だけ増分されたかのようにして計算が進められます。 次に、ADD 演算が実行され、2008 の結果が与えられます。 この結果は、08 として Next-Review-Year に保管されます。

しかし、次のステートメントを使用すると、2018 という結果になります。

```
Add 20 to Last-Review-Year Giving Next-Review-Year.
```

この結果は、世紀ウィンドウの範囲の外側になります。 結果が Next-Review-Year に保管されると、それ以降に Next-Review-Year が参照された場合に 1918 として解釈されるので、誤りになります。 この場合、演算の結果は、ON SIZE ERROR 句が ADD ステートメントに指定されているかどうかによって異なります。

- SIZE ERROR が指定されている場合、受信フィールドは変更されず、SIZE ERROR 命令ステートメントが実行されます。
- SIZE ERROR が指定されていない場合、結果は、左端から桁が切り捨てられて受信フィールドに保管されます。

この考慮事項は、内部ブリッジングを使用するときには大切です。 出力ファイルに書き出すために 4 桁年号の日付フィールドを 2 桁に短縮するときは、日付が世紀ウィンドウの範囲内になるようにする必要があります。 そうすると、2 桁年号の日付がフィールドで正しく表されるようになります。

適切な計算が行われるようにするには、短縮を行うための COMPUTE ステートメントに、ウィンドウ外条件を処理するための SIZE ERROR 句を指定しなければなりません。 次に例を示します。

```
Compute Output-Date-YY = Work-Date-YYYY
On Size Error Perform CenturyWindowOverflow.
```

ウィンドウ表示日付受け取り側についての SIZE ERROR 処理は、世紀ウィンドウの範囲外となるすべての年号値を認識します。 すなわち、世紀ウィンドウの開始年より小さい年号値も、世紀ウィンドウの終了年より大きい年号値の場合と同様に、SIZE ERROR 条件を引き起こします。

関連タスク
479 ページの『内部ブリッジングの使用』

評価の順序の指定

算術式の中の日付フィールドに関する制限のために、以前は正常にコンパイルされたプログラムが、今では一部のデータ項目が日付フィールドに変更された結果、診断メッセージが出るようになることがあります。

```
01 Dates-Record.
    03 Start-Year-1 Pic 99 Date Format yy.
    03 End-Year-1   Pic 99 Date Format yy.
    03 Start-Year-2 Pic 99 Date Format yy.
    03 End-Year-2   Pic 99 Date Format yy.
. . .
    Compute End-Year-2 = Start-Year-2 + End-Year-1 - Start-Year-1.
```

上の例では、評価される最初の算術式は次のものです。

```
Start-Year-2 + End-Year-1
```

しかし、2 つの日付フィールドを加算することは許可されません。 これらの日付フィールドを解決するためには、括弧を使用して、許可される算術式の部分を分離しなければなりません。 次に例を示します。

```
Compute End-Year-2 = Start-Year-2 + (End-Year-1 - Start-Year-1).
```

この場合、評価される最初の算術式は次のものです。

```
End-Year-1 - Start-Year-1
```

ある日付フィールドから別の日付フィールドを減算することは許可され、非日付の結果が与えられます。 次に、この非日付の結果が日付フィールド End-Year-1 に加算され、日付フィールドの結果が与えられ、これが End-Year-2 に保管されます。

日付処理の明示的制御

ある特定条件下でのみ、あるいはプログラムの特定部分でのみ、COBOL データ項目を日付フィールドとして処理することがあります。 あるいは、アプリケーションに含まれる 2 桁年の日付フィールドを、他のソフトウェア・プロダクトとの対話があるためにウィンドウ化日付フィールドとして宣言できないことがあるかもしれません。

例えば、日付フィールドが、特に解釈のための操作をすることなく純粋な 2 進数の内容によってのみ認識されるコンテキストでは、そのフィールドの日付をウィンドウ操作することはできません。 このような日付フィールドには、以下のものがあります。

- SdU ファイルのキー
- Db2 のようなデータベース・システムの検索フィールド

- CICS コマンドのキー・フィールド

逆に日付フィールドを、プログラムの特定の部分では非日付として扱いたいときがあるかもしれません。

COBOL は、このような条件を扱うために 2 つの組み込み関数を提供します。

DATEVAL
非日付を日付フィールドに変換します。

UNDATE
日付フィールドを非日付に変換します。

関連タスク
490 ページの『DATEVAL の使用』
490 ページの『UNDATE の使用』

DATEVAL の使用

DATEVAL 組み込み関数を使用して、非日付を 日付フィールドに変換できます。その結果、COBOL により関連する日付処理がそのフィールドに適用されることになります。

この関数の最初の引数には変換対象の非日付を指定し、2 番目の引数には日付形式を指定します。 2 番目の引数は、DATE FORMAT 節の日付パターンの指定に類似した指定を持つリテラル・ストリングです。

ほとんどの場合コンパイラーは、非日付の解釈について正しい仮定をしますが、その仮定に伴って警告レベルの診断メッセージを出します。 一般にこのメッセージは、ウィンドウ表示日付がリテラルと比較された場合に発生します。

```
03  When-Made        Pic x(6) Date Format yyxxxx.
. . .
If When-Made = "850701" Perform Warranty-Check.
```

リテラルは、互換性のあるウィンドウ表示日付と見なされますが、1900 から 1999 の世紀ウィンドウが使われているので、これは 1985 年 7 月 15 日を表しています。 DATEVAL 組み込み関数を使用してリテラルの日付の年を明示して、警告メッセージを出さないようにすることができます。

```
If When-Made = Function Dateval("19850701" "YYYYXXXX")
    Perform Warranty-Check.
```

490 ページの『例: DATEVAL』

UNDATE の使用

UNDATE 組み込み関数を使用して、日付フィールドを非日付に変換し、日付処理なしでそれを参照することができます。

重要: UNDATE を使用するのは最後の手段の場合を除いて、できる限り使わないようにしてください。プログラムでの日付フィールドのフローをコンパイラーが見失ってしまうからです。 もし見失った場合、日付の比較が正しくウィンドウ操作されないことがあります。

MOVE および COMPUTE では、関数 UNDATE の代わりに DATE FORMAT 節を使用するようにしてください。

491 ページの『例: UNDATE』

例: DATEVAL

この例は、フィールドを非日付としておいたまま、比較ステートメントの中で DATEVAL 組み込み関数を使用するのが適切である事例を示しています。

プログラム内でフィールド Date-Copied が何回も参照されるが、その参照の大半はレコード間でその値を移動したり、印刷のために再フォーマットしたりするだけのものであると想定します。 1 つの参照箇所においてのみ、(他の日付との比較の目的で) その内容を日付であると見なしています。 この場合、このフ

ィールドは非日付としておいたまま、比較ステートメントの中で DATEVAL 組み込み関数を使用するのが適切です。 次に例を示します。

```
03  Date-Distributed Pic 9(6) Date Format yyxxxx.
03  Date-Copied      Pic 9(6).
. . .
If Function DATEVAL(Date-Copied "YYXXXX") Less than Date-Distributed . . .
```

この例では、DATEVAL は Date-Copied を日付フィールドに変換し、比較を意味のあるものにしています。

関連参照
DATEVAL (*COBOL for Linux on x86* 言語解説書)

例: UNDATE

次の例は、日付フィールドを非日付に変換する事例を示しています。

フィールドは Invoice-Date は、ウィンドウ操作される年間通算日の日付フィールドです。 レコードによっては、00999 という値を含んでいるものがあり、これはレコードが真の送り状レコードではなく、ファイル制御情報を含んでいるレコードであることを示します。

Invoice-Date には DATE FORMAT 節があります。プログラム内でのこのフィールドへの参照のほとんどが日付固有であるからです。 しかし、制御レコードの有無が検査される場合には、年部分の値が 00 になっていると、何らかの混乱を招きます。 Invoice-Date の 00 という年の値は、世紀ウィンドウに応じて、1900 または 2000 のいずれかを表すことがあります。 これは、非日付 (例の中ではリテラル 00999) と比較されますが、それは、常に仮定による世紀ウィンドウに対してウィンドウ操作されるため、常に 1900 年を表します。

比較が矛盾した結果にならないようにするためには、UNDATE 組み込み関数を使用して Invoice-Date を非日付に変換しなければなりません。 したがって、IF ステートメントがどの日付フィールドも比較しない場合には、ウィンドウ操作を適用させる必要はありません。 次に例を示します。

```
01  Invoice-Record.
    03  Invoice-Date    Pic x(5) Date Format yyxxx.
. . .
    If FUNCTION UNDATE(Invoice-Date) Equal "00999" . . .
```

関連参照

UNDATE (*COBOL for Linux on x86* 言語解説書)

日付関連診断メッセージの分析および回避

DATEPROC(FLAG) コンパイラー・オプションが有効である場合、日付フィールドを定義または参照しているすべてのステートメントについて、コンパイラーは診断メッセージを作成します。

コンパイラー生成のすべてのメッセージと同じく、日付関連の各メッセージも以下の重大度レベルの 1 つを持っています。

- 通知レベル。これは、日付フィールドの定義または使用に注意を喚起するためのものです。
- 警告レベル。プログラムにコーディングされている情報が不十分なために、日付フィールドまたは非日付に関してコンパイラーがなんらかの仮定を設定したことを示すため、または正しいことを手作業で検査しなければならない日付ロジックの場所を示すためのものです。 コンパイルは続行し、仮定は適用されます。
- エラー・レベル。日付フィールドの使い方が誤っていることを表します。 コンパイルは継続されますが、ランタイムの結果は予測不能です。
- 重大レベル。日付フィールドの使い方が誤っていることを表します。 このエラーが生成される原因となったステートメントは、コンパイルから廃棄されます。

MLE メッセージを最も簡単に使用するには、FLAG オプション設定を使用してコンパイルします。これによって、ソース・リストの中でメッセージの参照する行の直後にメッセージが出力されるようになります。すべての MLE メッセージを表示したり、重大度によって選択したりできます。

すべての MLE メッセージを表示するには、FLAG(I,I) および DATEPROC(FLAG) コンパイラー・オプションを指定します。 自分のプログラム内の日付フィールドが MLE でどのように処理されるかを理解するため、最初はすべてのメッセージを表示することをお勧めします。 例えば、コンパイル・リストを使用してプログラム内のデータ使用の静的分析をしたい場合は、FLAG (I,I) を使用してください。

しかし、MLE 固有のコンパイルでは FLAG(W,W) を指定することをお勧めします。 重大レベル (S レベル) エラー・メッセージとエラー・レベル (E レベル) メッセージはすべて訂正する必要があります。 警告レベル (W レベル) のメッセージについては、各メッセージを調べ、以下の指針に従って、メッセージを除去するか、または回避不能なメッセージの場合はコンパイラーが正しい仮定を行うようにしなければなりません。

- 診断メッセージが、DATE FORMAT 節を持っていなければならない日付データ項目を示すことがあります。 これらの項目に DATE FORMAT 節を追加するか、またはこれらの項目への参照で DATEVAL 組み込み関数を使用してください。
- 日付フィールドを対象にする比較条件の中、または日付フィールドを含む算術式の中でリテラルを使用する場合には、特別な注意が必要です。 リテラル (および非日付データ項目) に DATEVAL 関数を使用して、使用する DATE FORMAT パターンを指定することができます。 UNDATE 関数は、最後の手段として、日付中心の動作を望まないコンテキストで使用される日付フィールドを使用可能にするために使用することができます。
- REDEFINES および RENAMES 節が指定されている場合、日付フィールドと非日付が同じ保管場所を占有していると、コンパイラーは警告レベルの診断メッセージを出すことがあります。 このような場合には注意深く調べて、種々の別名の付いたデータ項目のすべての使い方が正しいこと、および非日付と認識された再定義が実際にどれも日付でないこと、またはプログラム内の日付ロジックに悪影響を与えていないことを確認してください。

W レベル・メッセージが出ても気にしなければいいのですが、コードを変更して戻りコード = 0 のコンパイルを達成するのも良い方法です。

警告レベルの診断メッセージを回避するには、以下の簡単な指針に従ってください。

- 日付が入るデータ項目には DATE FORMAT 節を追加してください。その項目が比較で使用されない場合でもそのようにします。 ただし、日付フィールドの使用上の制約事項に関しては、以下の関連参照を参照してください。 例えば、暗黙的または明示的に USAGE NATIONAL として記述されているデータ項目では DATE FORMAT 節を使用できません。
- 日付フィールドが意味をなさないコンテキスト、例えば FILE STATUS、PASSWORD、ASSIGN USING、LABEL RECORD、または LINAGE 項目などでは、日付フィールドを指定しない。 指定すると、警告レベルのメッセージが出されて、日付フィールドが非日付として扱われます。
- 日付フィールドの暗黙の別名または明示された別名が、日付フィールドだけからなるグループ項目などの中で互換性があることを確認する。
- 日付フィールドが VALUE 節を定義されている場合、その値が日付フィールド定義と互換性があることを確認する。
- 非日付を日付フィールドとして扱いたい場合 (例えば、非日付を日付フィールドに移動する場合や、ウィンドウ表示日付を非日付と比較する場合など) で、ウィンドウ表示日付比較を行う場合は、DATEVAL 組み込み関数を使用する。 DATEVAL を使わないと、コンパイラーは非日付の使用に関してある仮定を行い、警告レベルの診断メッセージを作成します。 その仮定が正しくても、DATEVAL を使用すればメッセージを排除できます。
- 日付フィールドが非日付として処理されるようにしたい場合は、UNDATE 組み込み関数を使用してください。 例えば、日付フィールドを非日付へ移動させる場合や、ウィンドウ表示比較を行いたくないときに非日付とウィンドウ表示日付フィールドを比較する場合などです。

関連タスク
489 ページの『日付処理の明示的制御』
COBOL 2000 年言語拡張の手引き (日付関連診断メッセージの分析)

関連参照
日付フィールドの使用に関する制約事項 (*COBOL for Linux on x86* 言語解説書)

日付処理上の問題の回避

COBOL プログラムを変更して 2000 年言語拡張を使用する場合、 予想外の振る舞いが生じるのを解決するために、プログラムの一部に特別の注意を向けなければならないことがあります。 例えば、パック 10 進フィールドの問題、および拡張フィールドからウィンドウ日付フィールドに移行させるときに発生する問題を回避する必要が生じることがあります。

関連タスク
493 ページの『パック 10 進数フィールドの問題の回避』
493 ページの『拡張日付フィールドからウィンドウ表示日付フィールドへの移動』

パック 10 進数フィールドの問題の回避

COMPUTATIONAL-3 フィールド (パック 10 進数形式) は、奇数桁数を持つものとして定義されることがよくあります。 フィールドがその大きさの数値を保持しない場合でもそうです。 それは、内部表現ではパック 10 進数の桁数が常に奇数になっているからです。

例えば、6 桁のグレゴリオ暦日付を入れるフィールドを PIC S9(6) COMP-3 として宣言できます。 この宣言によって、4 バイトのストレージが予約されることになります。 しかし、プログラマーは、最高次の桁が常に 0 の 4 バイトが予約されることを考慮して、 PIC S9(7) としてフィールドを宣言している可能性があります。

このフィールドに DATE FORMAT YYXXXX という節を追加した場合、 コンパイラーから診断メッセージが出ます。PICTURE 節内の桁数が日付形式指定のサイズに一致しないからです。 その場合、フィールド使用を 1 つずつ慎重に調べる必要があります。 高位桁を使わない場合は、単にフィールド定義を PIC S9(6) に変更することができます。 使用する場合 (例えば同じフィールドに日付以外の値も入れることがある場合)、他のなんらかのアクションが必要になります。

- REDEFINES 節を使用して、日付および非日付の両方としてフィールドを定義する (この場合も、警告レベルの診断メッセージが出ます)。
- 日付を入れる別の WORKING-STORAGE フィールドを定義し、数値フィールドを新規フィールドに移動する。
- データ項目に DATE FORMAT 節を追加せず、それを日付フィールドとして参照するところで DATEVAL 組み込み関数を使用する。

拡張日付フィールドからウィンドウ表示日付フィールドへの移動

拡張英数字日付フィールドをウィンドウ表示日付フィールドへ移動させる場合、この移動は、英数字移動に関する通常の COBOL 規約に従いません。 送信フィールドと受信フィールドのどちらも日付フィールドである場合、移動は通常の左寄せではなく右寄せになります。 拡張からウィンドウ操作への (縮小) 移動の場合、年の先頭の 2 桁が切り捨てられることになります。

送信フィールドの内容によっては、このような移動は結果的に誤りを生じることがあります。 次に例を示します。

```
77  Year-Of-Birth-Exp  Pic x(4) Date Format yyyy.
77  Year-Of-Birth-Win  Pic xx   Date Format yy.
. . .
    Move Year-Of-Birth-Exp to Year-Of-Birth-Win.
```

Year-Of-Birth-Exp が '1925' を含んでいる場合、Year-Of-Birth-Win は '25' が含まれます。 しかし、世紀ウィンドウが 1930-2029 の場合、Year-Of-Birth-Win のあとの参照は 2025 と見なされます。しかし、それは誤っています。

第 7 部 パフォーマンスおよび生産性の向上

© Copyright IBM Corp. 2021

第 27 章 プログラムのチューニング

プログラムが分かりやすいものであってこそ、パフォーマンスの評価を行うことができます。 制御フローが複雑であると、プログラムの理解や保守が困難になり、コードの最適化が妨げられます。

プログラムのパフォーマンスを向上させるには、少なくとも以下の点を調査してください。

- 基になるアルゴリズム: 最高のパフォーマンスを得るには、健全なアルゴリズムを使用することが不可欠です。 例えば、次のように指定します。
  - 百万個の品目をソートするような洗練されたアルゴリズムは、単純なアルゴリズムよりも何百万倍も高速になります。
  - プログラムが頻繁にデータにアクセスする場合は、データにアクセスするステップの数を減らします。
- データ構造: アルゴリズムに適したデータ構造を使用することが不可欠です。

より優れたコード・シーケンスを生成し、システム・サービスをより効率的に活用するようなプログラムを作成することができます。 さらに、以下の点がパフォーマンスに影響します。

- コーディング技法: 最適化プログラムが効率的なデータ型を選択し、表を効率的に処理できるようにするプログラミング・スタイルを使用します。
- 最適化: `OPTIMIZE` コンパイラー・オプションを使用して コードを最適化することができます。
- コンパイラー・オプションおよび `USE FOR DEBUGGING ON ALL PROCEDURES`: 一部のコンパイラー・オプションと言語は、プログラムの効率に影響を与えます。
- ランタイム環境: ランタイム・オプションの選択を考慮します。
- **CICS**: トランザクションの応答時間を改善するには、`EXEC CICS LINK` ステートメントのインスタンスを `CALL` ステートメントに変換します。

  CICS での動的呼び出しのパフォーマンスを改善する方法については、関連タスクを参照してください。

関連概念
504 ページの『最適化』

関連タスク
385 ページの『CICS での動的呼び出しのパフォーマンスのチューニング』
497 ページの『最適なプログラミング・スタイルの使用』
499 ページの『効率的なデータ型の選択』
501 ページの『テーブルの効率的処理』
504 ページの『コードの最適化』
505 ページの『パフォーマンスを向上させるコンパイラー機能の選択』
149 ページの『SFS パフォーマンスの向上』

関連参照

299 ページの『第 15 章 ランタイム・オプション』
505 ページの『パフォーマンスに関連するコンパイラー・オプション』

最適なプログラミング・スタイルの使用

使用するコーディング・スタイルは、最適化プログラムがコードを処理する方法に影響を与えることがあります。 構造化プログラミング手法の使用、 一括表示表現、シンボリック定数の使用、および定数と重複計算のグループ化によって最適化を向上させることができます。

関連タスク
498 ページの『構造化プログラミングの使用』
498 ページの『一括表示表現』
498 ページの『シンボリック定数の使用』

© Copyright IBM Corp. 2021

498 ページの『定数計算のグループ化』
499 ページの『重複計算のグループ化』

構造化プログラミングの使用

構造化プログラミング・ステートメント (EVALUATE やインライン PERFORM) を使用すると、プログラムが一層分かりやすいものになり、より直線的な制御フローができあがります。 すると、最適化プログラムはプログラムのより多くの領域に作用することができるため、より効率のよいコードが与えられます。

トップダウン・プログラミング構成を使用してください。 ライン外の PERFORM ステートメントは、トップダウン・プログラミングを行う本来の手段です。 ライン外 PERFORM ステートメントがインラインの PERFORM ステートメントと同じくらい効率的になることがよくあります。これは、最適化プログラムがリンケージ・コードを簡略化または除去するためです。

次の構成は使用しないでください。

- ALTER ステートメント
- 逆方向ブランチ (PERFORM が不適当であるループに必要な場合を除く)。
- 変則的な制御フローを伴う PERFORM プロシージャー。例えば、プロシージャーの終わりに制御が渡されないために、PERFORM ステートメントに戻れないなど。

一括表示表現

プログラム内の式を因数処理することによって、多数の不要な計算を除去できる可能性があります。

例えば、次のコードの最初のブロックは、2 番目のブロックより効率的になっています。

```
MOVE ZERO TO TOTAL
PERFORM VARYING I FROM 1 BY 1 UNTIL I = 10
  COMPUTE TOTAL = TOTAL + ITEM(I)
END-PERFORM
COMPUTE TOTAL = TOTAL * DISCOUNT
```

```
MOVE ZERO TO TOTAL
PERFORM VARYING I FROM 1 BY 1 UNTIL I = 10
  COMPUTE TOTAL = TOTAL + ITEM(I) * DISCOUNT
END-PERFORM
```

最適化プログラムは式の因数処理を行いません。

シンボリック定数の使用

プログラム全体で最適化プログラムがデータ項目を定数として認識するようにさせるには、データ項目を VALUE 節で初期化し、プログラム内のどの場所でもそれを変更しないでください。

データ項目を BY REFERENCE によってサブプログラムに渡すと、最適化プログラムはその項目を外部データ項目と見なして、サブプログラムを呼び出すたびに、その項目が変更されるものと想定します。

リテラルをデータ項目に移動すると、最適化プログラムはそのデータ項目を定数と見なしますが、それは、MOVE ステートメントに続くプログラムの限られた領域内においてのみです。

定数計算のグループ化

式のいくつかの項目が定数である場合、最適化プログラムがそれらの項目を最適化できることを確認してください。 コンパイラーは COBOL の左から右への評価規則に従います。 したがって、すべての定数を式の左側に移動させるか、または括弧で囲んでグループ化します。

例えば、V1、V2、および V3 が変数で、C1、C2、および C3 が定数の場合、下記の左側にある式の方が、右側の対応する式より優れています。

効率がよい	効率がよくない
`V1 * V2 * V3 * (C1 * C2 * C3)`	`V1 * V2 * V3 * C1 * C2 * C3`
`C1 + C2 + C3 + V1 + V2 + V3`	`V1 + C1 + V2 + C2 + V3 + C3`

量産用プログラミングでは、式の右側に定数因子を置く傾向がよく見られます。 しかしその結果、最適化が行われないために、効率のよくないコードが生成される可能性があります。

重複計算のグループ化

さまざまな式のコンポーネントが重複している場合は、コンパイラーがそれらを最適化できることを確認してください。 算術式の場合、コンパイラーは、左から右への COBOL の評価規則に従います。 したがって、すべての重複を式の左側に移動させるか、または括弧で囲んでグループ化します。

例えば、V1 から V5 が変数の場合、`V2 * V3 * V4` の計算は、次の 2 つのステートメントで重複します (共通の副次式と呼ばれます)。

```
COMPUTE A = V1 * (V2 * V3 * V4)
COMPUTE B = V2 * V3 * V4 * V5
```

次の例では、`V2 + V3` が共通の副次式です。

```
COMPUTE C = V1 + (V2 + V3)
COMPUTE D = V2 + V3 + V4
```

次の例には、共通の副次式はありません。

```
COMPUTE A = V1 * V2 * V3 * V4
COMPUTE B = V2 * V3 * V4 * V5
COMPUTE C = V1 + (V2 + V3)
COMPUTE D = V4 + V2 + V3
```

最適化プログラムは重複する計算を除去できます。 人工的な一時計算を取り入れる必要はありません。 そのようなものがなくても、多くの場合、プログラムはずっと分かりやすく高速になります。

効率的なデータ型の選択

適切なデータ型および PICTURE 節を選択すると、より効率的なコードを得ることができますが、それは `USAGE DISPLAY` および `USAGE NATIONAL` データ項目を、計算に頻繁に使用される領域で使用しない場合に効率的なコードが得られるのと同様です。

データ項目を長くしすぎると、パフォーマンスが低下するおそれがあります。 ただし、データ項目の長さが 2 バイトの小さい累乗 (1、2、4、8、または 16) であり、そのデータ項目が、 そのデータ項目のサイズに適合する 2 バイトの累乗の境界に位置合わせされている場合、 通常、そのデータ項目は、奇数の長さや位置合わせのデータ項目と比較して、より素早く、より少ない命令で、初期化したり移動したりできます。

算術計算の速さは、2 進数、パック 10 進数、ゾーン 10 進数または DISPLAY、国別 10 進数の順番で高速です。

型に影響するオプションはパフォーマンスにも影響を及ぼす可能性があります。 例えば、`FLOAT(BE)` は `FLOAT(NATIVE)` よりもコストがかかります。

一貫性のあるデータ型を使用すると、データ項目に演算を実行する際に変換を行う必要性を減らすことができます。 また、固定小数点データ型と浮動小数点データ型をいつ使用するかを慎重に判別することで、プログラム・パフォーマンスを向上させることができます。

関連概念
39 ページの『数値データの形式』

関連タスク
500 ページの『効率的な計算データ項目の選択』
500 ページの『一貫性のあるデータ型の使用』
500 ページの『算術式の効率化』
501 ページの『指数計算の効率化』

効率的な計算データ項目の選択

データ項目を主に算術に、または添え字として使用する場合、その項目のデータ記述項目に USAGE BINARY をコーディングしてください。 2 進データを処理する操作は、10 進データを処理する操作よりも高速で行われます。

しかし、固定小数点算術ステートメントの中間結果で精度 (有効数字の数) が大きい場合、コンパイラーは、オペランドをパック 10 進数形式に変換したうえで 10 進数演算を使用します。 固定小数点算術ステートメントについては、コンパイラーは通常、精度が 8 桁以下にとどまる場合には、2 進オペランドを 使用した簡単な計算に 2 進数算術演算を使用します。 18 桁を超えると、コンパイラーは常に 10 進数算術演算を使用します。 9 から 18 桁の精度では、コンパイラーはいずれの形式でも使用できます。

BINARY データ項目について最も効率的なコードを作成するには、以下の特性を持たせるようにしてください。

- 符号 (PICTURE 節の S)。
- 8 桁以下。

8 桁より大きいデータ項目、あるいは DISPLAY または NATIONAL データ項目と一緒に使用されるデータ項目の場合は、PACKED-DECIMAL を使用してください。

PACKED-DECIMAL データ項目について最も効率的なコードを作成するには、以下の特性を持たせるようにしてください。

- 符号 (PICTURE 節の S)。
- ハーフ・バイトを残さずに正確なバイト数を占めるように、奇数の桁数 (PICTURE 節の 9 の数)。

一貫性のあるデータ型の使用

種々の型のオペランドに対する操作では、オペランドの 1 つを残りのものと同じ型に変換する必要があります。 各変換ごとにいくつかの命令が必要になります。 例えば、いずれかのオペランドは小数点以下の桁数が適切な数になるように位取りを指定する必要があるかもしれません。

一貫性のあるデータ型を使用し、両方のオペランドに同じ使用法を与え、さらに適切な PICTURE 指定を与えることで、変換を大部分は回避できます。 つまり、比較、加算、または減算を行う 2 つの数値は、同じ使用法を持つだけでなく、さらに小数部の桁数 (PICTURE 節の V の後の 9 の数) も同じでなければなりません。

算術式の効率化

オペランドをほとんど変換する必要がない場合、浮動小数点で評価される算術式の計算は最も効率的です。 COMP-1 または COMP-2 であるオペランドを使用すると、最も効率のよいコードが作成されます。

浮動小数点データに高速変換を行うには、整数項目を BINARY または PACKED-DECIMAL (9 桁以下) として定義します。さらに、COMP-1 または COMP-2 の項目から、9 桁以下の固定小数点整数への変換は (SIZE ERROR が有効ではない場合)、COMP-1 または COMP-2 項目の値が 1,000,000,000 未満の場合に効率がよくなります。

指数計算の効率化

評価をもっと速く行い、結果をもっと正確にするには、大きな指数に対しては指数の浮動小数点を使用してください。

例えば、以下に示す最初のステートメントは、2 番目のステートメントより速くかつより正確に計算されます。

```
COMPUTE fixed-point1 = fixed-point2 ** 100000.E+00
COMPUTE fixed-point1 = fixed-point2 ** 100000
```

これは、浮動小数点の指数があるため、べき乗計算の計算に浮動小数点演算が使用されるからです。

テーブルの効率的処理

いくつかの手法を使用して、テーブル処理演算の効率を上げたり、最適化プログラムに影響を及ぼしたりすることができます。 努力の成果が十分報われる可能性があります。 テーブル処理操作がアプリケーションの主要部分である場合には特に当てはまります。

以下の 2 つのガイドラインは、テーブル・エレメントの参照方法を選択する際に影響を与えるものです。

- 添え字付けではなく索引付けを使用する。

  コンパイラーは重複する指標や添え字を除去できますが、テーブル・エレメントへの元の参照は、指標を使用することで (たとえ添え字が BINARY であっても) より効率的になります。 これは、指標の値には既にエレメント・サイズが加味されているのに対して、添え字の値は使用時にエレメント・サイズを乗算しなければならないためです。 指標には既にテーブルの先頭からの変位が含まれており、実行時にこの値を計算する必要はありません。 ただし、添え字の方が理解しやすく維持するのが簡単かもしれません。
- 相対索引付けを使用する。

  相対指標参照 (つまり、符号なし数値リテラルが指標名に加えられるか、または指標名から引かれる参照) は、少なくとも直接指標参照と同じ位の速さで、時にはより高速で実行されます。 オフセットを含めた代替索引を保管してもメリットはありません。

指標または添え字のいずれを使用する場合でも、以下のコーディング指針は、より良いパフォーマンスを得る助けとなります。

- 定数および重複する指標または添え字を左側に置く。

  このように実行時の計算を削減または除去することができます。 すべての指標または添え字が可変であっても、プログラム内で互いに近接している参照に関して、右端の添え字がもっとも頻繁に変化するように、テーブルを使用してみてください。 この方法を使用すると、ページングだけでなくストレージ参照のパターンも改善されます。 すべての指標または添え字が重複している場合、指標または添え字の計算全体が共通副次式になります。
- 関連するテーブルの長さと一致するようなエレメントの長さを指定する。

  異なるテーブルに添え字または指標を付けるときは、すべてのテーブルのエレメント長が同じ場合に、最も効率がよくなります。 このように、テーブルの最後の次元のストライドが同じであるため、最適化プログラムは、1 つのテーブルで計算された右端の指標または添え字を再利用できるようになります。 エレメントの長さおよび各次元での出現回数が同じである場合、最後の次元以外は、次元のストライドもまた等しくなり、その結果、添え字計算相互間の共通性はより大きくなります。 最適化プログラムは、右端以外の添え字または指標を再使用することができます。
- 指標および添え字検査をプログラムにコーディングすることによって、参照エラーを回避する。

  指標および添え字を妥当性検査する必要がある場合は、SSRANGE コンパイラー・オプションを使用するよりも、独自の検査をコーディングする方が速い場合があります。

以下のガイドラインに従うことによって、テーブルの効率を改善することもできます。

- すべての添え字に 2 進数データ項目を使用する。

添え字を使用してテーブルをアドレッシングする場合は、8 桁以下の BINARY 符号付きデータ項目を使用してください。 さらに場合によっては、データ項目の桁数を 4 桁以下にすると、処理時間を短縮できます。

- 可変長テーブル項目に 2 進数データ項目を使用する。

  可変長項目を持つテーブルの場合は、OCCURS DEPENDING ON (ODO) のコードを改善することができます。 可変長項目が参照されるたびに不要な変換が行われないようにするには、OCCURS . . . DEPENDING ON オブジェクトを対象に BINARY を指定します。

- 可能であれば固定長データ項目を使用する。

  可変長データ項目を使用する場合、それらの使用頻度が高くなる前に、固定長データ項目にコピーすると、オーバーヘッドを緩和することができます。

- 使用する探索メソッドのタイプに従ってテーブルを編成する。

  テーブルが順次に探索される場合には、検索基準を満たす可能性が最も高いデータ値を テーブルの始まりに置くようにします。 テーブルが二分探索アルゴリズムを使用して 探索される場合は、検索キー・フィールドに基づいてアルファベット順にソートされたテーブルに、データ値を入れてください。

関連概念
502 ページの『テーブル参照の最適化』

関連タスク
62 ページの『テーブル内の項目の参照』
499 ページの『効率的なデータ型の選択』

関連参照
284 ページの『SSRANGE』

テーブル参照の最適化

COBOL コンパイラーは、テーブル参照を、いくつかの方法で最適化します。

テーブル・エレメント参照 ELEMENT(S1 S2 S3) (S1、S2、および S3 は添え字) の場合、コンパイラーは次の式を評価します。

```
comp_s1 * d1 + comp_s2 * d2 + comp_s3 * d3 + base_address
```

ここで、comp_s1 は 2 進数に変換された後の S1 の値、comp-s2 は 2 進数に変換された後の S2 の値 (以下同様) です。 それぞれの次元のストライドは d1、d2、および d3 です。 ある特定次元のストライドは、その次元での出現番号が 1 だけ違い、かつ他の出現番号が等しいようなテーブル・エレメント相互間の距離 (バイト単位) です。 例えば、上記の例の 2 次元のストライド d2 は、ELEMENT(S1 1 S3) と ELEMENT(S1 2 S3) との間の距離 (バイト単位) です。

指標計算は添え字計算に類似していますが、指標値ではそれらの中にストライドを含めているので、 乗算をする必要がないという点が異なります。 指標計算には、指標をレジスターに ロードすることも含まれます。これらのデータ転送は、個々の添え字計算の項を 最適化する場合とほぼ同様に、最適化することができます。

コンパイラーは式を左から右へと評価していくので、定数または重複する添え字が左端にあると、最適化プログラムが計算を除去する可能性が最も高くなります。

定数項目と変数項目の最適化

C1、C2、..... は、定数データ項目であり、V1、V2、..... は変数データ項目です。 したがって、テーブル・エレメント参照 ELEMENT(V1 C1 C2) の場合、コンパイラーは個々の項 comp_c1 * d2 および comp_c2 * d3 だけしか、式から定数として除去できません。

```
comp_v1 * d1 + comp_c1 * d2 + comp_c2 * d3 + base_address
```

しかし、テーブル・エレメント参照 ELEMENT(C1 C2 V1) の場合、コンパイラーは、副次式 comp_c1 * d1 + comp_c2 * d2 全体を、式から定数として除去することができます。

```
comp_c1 * d1 + comp_c2 * d2 + comp_v1 * d3 + base_address
```

テーブル・エレメント参照 ELEMENT(C1 C2 C3) では、添え字はすべて定数なので、実行時に添え字計算は行われません。 式は次のとおりです。

```
comp_c1 * d1 + comp_c2 * d2 + comp_c3 * d3 + base_address
```

最適化プログラムを使用すると、この参照は、スカラー (テーブルでない) 項目への参照と同じくらい効率的になります。

重複項目の最適化

テーブル・エレメント参照 ELEMENT(V1 V3 V4) および ELEMENT(V2 V3 V4) では、個々の項 comp_v3 * d2 および comp_v4 * d3 だけが、テーブル・エレメントの参照に必要な式における共通副次式です。

```
comp_v1 * d1 + comp_v3 * d2 + comp_v4 * d3 + base_address
comp_v2 * d1 + comp_v3 * d2 + comp_v4 * d3 + base_address
```

しかし、2 つのテーブル・エレメント参照 ELEMENT(V1 V2 V3) および ELEMENT(V1 V2 V4) の場合は、副次式 comp_v1 * d1 + comp_v2 * d2 全体が、テーブル・エレメントの参照に必要な 2 つの式間で共通となります。

```
comp_v1 * d1 + comp_v2 * d2 + comp_v3 * d3 + base_address
comp_v1 * d1 + comp_v2 * d2 + comp_v4 * d3 + base_address
```

2 つの参照 ELEMENT(V1 V2 V3) および ELEMENT(V1 V2 V3) では、式は同じです。

```
comp_v1 * d1 + comp_v2 * d2 + comp_v3 * d3 + base_address
comp_v1 * d1 + comp_v2 * d2 + comp_v3 * d3 + base_address
```

最適化プログラムを使用すると、同じエレメントへの 2 度目 (およびそれ以降) の参照は、スカラー (テーブルでない) 項目への参照と同じ効率になります。

可変長項目の最適化

従属 OCCURS DEPENDING ON データ項目の入っているグループ項目は可変長です。 プログラムは、可変長データ項目が参照されるたびに特殊コードを実行しなければなりません。

このコードはライン外のものなので、最適化の妨げになる可能性があります。 さらに、可変長データ項目を処理するためのコードは、固定サイズ・データ項目を処理するコードよりかなり効率が下がり、処理時間も大幅に増加することがあります。 例えば、可変長データ項目を比較したり移動したりするためのコードには、ライブラリー・ルーチンの呼び出しが必要なため、固定長データ項目の場合の同じコードよりかなり低速になります。

直接指標付けと間接指標付けの比較

相対指標参照は、直接指標参照と同じくらい迅速に実行されます。

ELEMENT (I5, J3, K2) の直接索引付けには、次のプリプロセスが必要になります。

```
SET I5 TO I
SET I5 UP BY 5
SET J3 TO J
SET J3 DOWN BY 3
```

```
SET K2 TO K
SET K2 UP BY 2
```

この処理のため、直接索引付けは、ELEMENT (I + 5, J - 3, K + 2) の相対索引付けよりも効率が悪くなります。

関連概念
504 ページの『最適化』

関連タスク
501 ページの『テーブルの効率的処理』

コードの最適化

プログラムの最終テストの準備ができたなら、OPTIMIZE コンパイラー・オプションを指定して、テスト・コードと実動コードが同一になるようにしてください。IBM では、最高のパフォーマンスを得るためにすべてのユーザーが OPT(FULL) を使用することをお勧めしています。

開発中に再コンパイルしないでプログラムを頻繁に実行する場合にも、OPTIMIZE を使用できます。ただし、頻繁に再コンパイルする場合は、アセンブラー言語の拡張部分 (LIST コンパイラー・オプション) を使用してプログラムの微調整を行う場合を除き、OPTIMIZE のオーバーヘッドが利点を上回ることがあります。

プログラムのユニット・テストについては、最適化されていないコードをデバッグする方が簡単です。

最適化プログラムがプログラムに対してどのように機能するかを確認するには、OPTIMIZE を使用した場合としない場合でのコンパイルを行い、生成されたコードを比較します。 (生成されたコードのアセンブラー・リストを要求するには、LIST コンパイラー・オプションを使用します。)

関連概念
504 ページの『最適化』

関連参照
270 ページの『LIST』
275 ページの『OPTIMIZE』

最適化

生成されたコードの効率を向上させるには、OPTIMIZE コンパイラー・オプションを使用できます。

OPTIMIZE を使用すると、COBOL 最適化プログラムが以下の最適化を行います。

- 不必要な制御権移動および非効率的な分岐を除去します。ソース・プログラムを見るだけではわからない、コンパイラーが生成する分岐も含みます。
- 可能であれば、最適化プログラムはステートメントをインラインに設定し、リンケージ・コードがなくて済むようにします。 この最適化は、プロシージャー統合 と呼ばれます。
- プログラムの結果に何の影響も与えない重複計算 (添え字計算や繰り返しのステートメントなど) を除去します。
- プログラムのコンパイル時に定数計算を実行することによって、定数計算を除去します。
- 定数条件式を除去します。
- 連続した項目 (MOVE CORRESPONDING の使用によって頻繁に発生するような) の移動を集約して、単一の移動にします。移動を集約するには、移動元も移動先も連続していなければなりません。
- 参照されないデータ項目を DATA DIVISION から廃棄し、これらのデータ項目をその VALUE 節に初期化するコードの生成を抑止します。 (最適化プログラムがこのアクションを取るのは、FULL サブオプションが指定された場合だけです。)

パフォーマンスを向上させるコンパイラー機能の選択

どんなパフォーマンス関連コンパイラー・オプションを選択するか、また USE FOR DEBUGGING ON ALL PROCEDURES ステートメントを使用するかどうかは、プログラムがどの程度うまく最適化されるかに影響を与えます。

カスタマイズ済みシステムには、最適なパフォーマンスを得るために特定のオプションが必要なものもあります。 以下の手順を 実行します。

1. システム・デフォルトの内容を調べるには、プログラムの短縮リストを入手し、リストされたオプション設定値を検討する。
2. プログラムをコンパイルするためのパフォーマンス関連オプションを選択する。

   重要: COBOL プログラムの調整方法については、システム・プログラマーと相談してください。 そうすれば、選択したオプションがインストール先のプログラムに適したものであるかどうかを確認できます。

考慮する必要のあるもう 1 つのコンパイラー機能として、USE FOR DEBUGGING ON ALL PROCEDURES ステートメントがあります。 これは、コンパイラーの最適化プログラムに大きな影響を与える可能性があります。 ON ALL PROCEDURES オプションは、プロシージャー名に移動するたびに、余分のコードを生成します。 これはデバッグには大変便利ですが、プログラムは非常に大型になり、実質上最適化を抑制する可能性があります。

関連概念
504 ページの『最適化』

関連タスク
504 ページの『コードの最適化』
361 ページの『リストの入手』

関連参照

505 ページの『パフォーマンスに関連するコンパイラー・オプション』

パフォーマンスに関連するコンパイラー・オプション

以下の表には、それぞれのオプションの目的、パフォーマンス上の利点と欠点、および使用上の注意 (該当する場合) が示されています。

表 *50.* **パフォーマンスに関連するコンパイラー・オプション**

コンパイラー・オプション	目的	パフォーマンス上の長所	パフォーマンス上の欠点	使用上の注意
ARITH(EXTEND) (253 ページの『ARITH』 を参照)	10 進数で許可される最大桁数を増やす	なし	ARITH(EXTEND) を使用すると、中間結果が大きくなるため、すべての 10 進数データ型でパフォーマンスがいくらか低下します。	どれほど低下するかは、使用する 10 進数データの量に直接左右されます。
DYNAM (264 ページの『DYNAM』を参照)	サブプログラム (CALL ステートメントによって呼び出された) が実行時に動的にロードされるようにする	サブプログラムが変更されても、アプリケーションをリンク・エディットする必要がないので、サブプログラムの保守が容易になります。	呼び出しはライブラリー・ルーチンを介して行う必要があるので、ちょっとしたパフォーマンス・ペナルティーがあります。	不要になった仮想記憶域を解放するには、CANCEL ステートメントを出してください。
OPTIMIZE(STD) (275 ページの『OPTIMIZE』を参照)	パフォーマンスがよくなるように、生成されるコードを最適化する	一般に、もっと効率的な実行時コードが得られます。	コンパイル時間が長くなります。 OPTIMIZE は NOOPTIMIZE に比べて、コンパイルのためにより多くの処理時間を必要とします。	NOOPTIMIZE は一般に、頻繁にコンパイルを行う必要があるプログラム開発時に使用します。 これにより、シンボリック・デバッグも可能になります。 実稼働には、OPTIMIZE の使用をお勧めします。

表 50. **パフォーマンスに関連するコンパイラー・オプション** (続き)

コンパイラー・オプション	目的	パフォーマンス上の長所	パフォーマンス上の欠点	使用上の注意
OPTIMIZE(FULL)	生成されたコードをパフォーマンス向上に関して最適化し、さらに DATA DIVISION も最適化する	一般に、より効率的なランタイム・コードが得られ、ストレージ使用量が少なくなります。	コンパイル時間が長くなること。OPTIMIZE は、NOOPTIMIZE に比べ、コンパイルにかかる処理時間が長くなります。	OPT(FULL) は未使用データ項目を削除しますが、それは、ダンプ読み取り用マーカーとしてのみ使用されるタイム・スタンプまたはデータ項目の場合には、望ましくないことがあります。
SSRANGE (284 ページの『SSRANGE』 を参照)	すべてのテーブル参照および参照変更式が適切な範囲内にあるかどうかを検査する	SSRANGE は、テーブル参照を検査するための追加コードを生成します。NOSSRANGE を使用すると、コードは生成されません。	SSRANGE を指定すると、有効範囲の検査はコンパイラーのパフォーマンスに影響を与えます。	一般に、テーブル参照のたびに検査する必要はなく、数度の検査だけで済む場合には、独自の検査をコーディングする方が、SSRANGE を使用するより速くなります。 CHECK(OFF) ランタイム・オプションを使用して、実行時に SSRANGE をオフにすることができます。 パフォーマンスを重視するアプリケーションについては、NOSSRANGE の使用をお勧めします。
NOTEST (285 ページの『TEST』 を参照)	デバッガーをフルに活用するために必要な追加のオブジェクト・コードを回避する。	なし	TEST はデバッグ情報を追加するため、オブジェクト・ファイルが大幅に拡張します。 プログラムをリンクするとき、デバッグ情報を除外するようリンカーに指示することができます。そうすると、実行可能モジュールのサイズは、モジュールが NOTEST でコンパイルされた場合に作成されるサイズとほぼ同じになります。 デバッグ情報が組み込まれると、実行可能モジュールが大きくなるほどロード時間が長くなり、ページングが増える可能性があるため、パフォーマンスが多少低下することがあります。	実稼働用には、NOTEST の使用をお勧めします。
TRUNC(OPT) (286 ページの『TRUNC』 を参照)	算術演算の受信フィールドを切り捨てるためのコードの生成を回避する	余分のコードを生成しないので、一般にパフォーマンスは向上します。	TRUNC(BIN) と TRUNC(STD) はともに、BINARY データ項目が変更されるたびに、余分のコードを生成します。 TRUNC(BIN) は通常、最も低速のオプションです。	TRUNC(STD) は 85 COBOL 標準に準拠しますが、TRUNC(BIN) および TRUNC(OPT) は準拠しません。 TRUNC(OPT) を使用すると、コンパイラーは、データが PICTURE および USAGE の仕様に従っていると見なします。 可能な場合は、TRUNC(OPT) を使用することをお勧めします。

関連概念
504 ページの『最適化』

関連タスク
231 ページの『コンパイラー・メッセージのリストの生成』
505 ページの『パフォーマンスを向上させるコンパイラー機能の選択』
501 ページの『テーブルの効率的処理』

関連参照
47 ページの『ゾーンおよびパック 10 進数データのサイン表記』
248 ページの『コンパイラー・オプション』

パフォーマンスの評価

プログラムのパフォーマンスの評価に役立つ次のワークシートに記入してください。 各質問に「はい」と答える場合は、おそらくパフォーマンスは向上しています。

パフォーマンスのトレードオフを比較検討する際には、各オプションの機能およびパフォーマンスの利点と欠点を十分に理解するようにしてください。 パフォーマンスの向上よりも、機能を重視する場合も多くあります。

表 *51.* **パフォーマンス調整のワークシート**

コンパイラー・オプション	考慮事項	はい
DYNAM	NODYNAM を使用できますか。 パフォーマンスのトレードオフを比較検討してください。	
OPTIMIZE	実稼働に OPTIMIZE を使用していますか。OPTIMIZE(FULL) を使用できますか。	
SSRANGE	NOSSRANGE を実稼働に使用していますか。	
TEST	実稼働に NOTEST を使用しますか。	
TRUNC	可能な場合、TRUNC(OPT) を使用していますか。	

関連タスク
505 ページの『パフォーマンスを向上させるコンパイラー機能の選択』

関連参照
505 ページの『パフォーマンスに関連するコンパイラー・オプション』

第 28 章 コーディングの単純化

コーディング技法を使用して、生産性を向上させることができます。 COPY ステートメント、COBOL 組み込み関数、および呼び出し可能サービスを使用することにより、反復コーディングや、多数の算術計算または他の複雑なタスクをコーディングする必要性を回避することができます。

プログラムに頻繁に使用されるコード・シーケンス (共通データ項目のブロック、入出力ルーチン、エラー・ルーチン、または COBOL プログラム全体) が含まれている場合は、それらのコード・シーケンスを一度作成し、それらを COBOL コピー・ライブラリーに入れてください。 COPY ステートメントを使用してこれらのコード・シーケンスを取り出し、 コンパイル時にプログラムに含めることができます。 このようにコピーブックを使用することによって、コーディングの繰り返しがなくなります。

COBOL は、ストリングおよび数値を扱うためのさまざまな機能を提供します。 これらの機能がコーディングの単純化に役立ちます。

日時呼び出し可能サービスは、日付をフルワード・バイナリー整数として保管し、タイム・スタンプを長精度 (64 ビット) 浮動小数点値として保管します。 これらの形式を使用することにより、算術計算を日時の値に基づいて単純かつ効率的に行うことができます。 このような計算を実行するために、言語ライブラリーの外側でサービスを使用する特別なサブルーチンを書く必要はありません。

関連タスク
50 ページの『数字組み込み関数の使用』
509 ページの『反復コーディングの除去』
104 ページの『データ項目の変換 (組み込み関数)』
106 ページの『データ項目の評価 (組み込み関数)』
510 ページの『日時の取り扱い』

反復コーディングの除去

保管されているソース・ステートメントをプログラムに組み込むには、COPY ステートメントをプログラムの任意の部、また任意のコード・シーケンス・レベルで使用します。 COPY ステートメントは任意の深さにネストできます。

複数のコピー・ライブラリーを指定するには、環境変数 SYSLIB を、 コロン (:) で区切った複数のパス名に設定するか、自身の環境変数を定義して、COPY ステートメントに次の句を組み込みます。

```
IN/OF library-name
```

次に例を示します。

```
COPY MEMBER1 OF COPYLIB
```

この修飾句を省略した場合、デフォルトは SYSLIB です。

COPY とデバッグ行: コピーされたテキストをデバッグ行として (例えば、7 桁目に D が挿入されているかのように) 扱わせるには、COPY ステートメントの最初の行に D を入れてください。 COPY ステートメント自体をデバッグ行にすることはできません。これに D が入っていても、WITH DEBUGGING モードが指定されていなければ、COPY ステートメントが処理されることはありません。

510 ページの『例: COPY ステートメントの使用』

関連参照

293 ページの『第 14 章 コンパイラー指示ステートメント』

© Copyright IBM Corp. 2021

例: COPY ステートメントの使用

これらの例は、COPY ステートメントを使用してライブラリー・テキストをプログラムに組み込む方法を示しています。

ライブラリー項目 CFILEA が以下の FD 項目から構成されているものとします。

```
    BLOCK CONTAINS 20 RECORDS
    RECORD CONTAINS 120 CHARACTERS
    LABEL RECORDS ARE STANDARD
    DATA RECORD IS FILE-OUT.
01  FILE-OUT       PIC X(120).
```

次のようにソース・プログラムで COPY ステートメントを使用すれば、テキスト名 CFILEA を取得することができます。

```
FD FILEA
         COPY CFILEA.
```

このライブラリー記入項目はプログラムにコピーされ、その結果生じるプログラム・リストは次のようになります。

```
FD FILEA
         COPY CFILEA.
C    BLOCK CONTAINS 20 RECORDS
C    RECORD CONTAINS 120 CHARACTERS
C    LABEL RECORDS ARE STANDARD
C    DATA RECORD IS FILE-OUT.
C    01  FILE-OUT   PIC X(120).
```

コンパイラー・ソース・リストで COPY ステートメントは別個の行に印刷され、コピーされた行の前には C が付けられます。

テキスト名 DOWORK を持つコピーブックが、以下のステートメントによって保管されているとします。

```
COMPUTE QTY-ON-HAND = TOTAL-USED-NUMBER-ON-HAND
MOVE QTY-ON-HAND to PRINT-AREA
```

DOWORK として識別されたコピーブックを取り出すには、次のようにコーディングします。

```
paragraph-name.
    COPY DOWORK.
```

DOWORK プロシージャー内のステートメントは *paragraph-name* の後に置かれます。

EXIT コンパイラー・オプションを使用して LIBEXIT モジュールを指定すると、結果がこの章で示されるものと異なることがあります。

関連タスク
509 ページの『反復コーディングの除去』

関連参照
293 ページの『第 14 章 コンパイラー指示ステートメント』

日時の取り扱い

日付または時刻の呼び出し可能サービスを呼び出すには、このサービス用の正しいパラメーターを指定した CALL ステートメントを使用してください。 その サービスによって要求されるデータ定義を使用して DATA DIVISION 内の CALL ステートメントに対してデータ項目を定義します。

```
77  argument          pic  s9(9)  comp.
```

```
01  format.
    05 format-length  pic  s9(4)  comp.
    05 format-string  pic  x(80).
77  result           pic  x(80).
77  feedback-code    pic  x(12)  display.
. . .
    CALL "CEEDATE" using argument, format, result, feedback-code.
```

上の例では、呼び出し可能サービス CEEDATE によって、データ項目 argument にリリアン日付で表された数値が、データ項目 result に書き込まれる文字形式の日付に変換されます。 データ項目 format に含まれるピクチャー・ストリングは、変換形式を制御します。 呼び出しの成功/失敗に関する情報は、データ項目 feedback-code に戻されます。

日時の呼び出し可能サービスを呼び出すための CALL ステートメントでは、ID ではなくプログラム名のリテラルを使用する必要があります。

プログラムは、標準のシステム・リンケージ規約を使用して、日時の呼び出し可能サービスを呼び出します。

512 ページの『例: 日付の操作』

関連概念
541 ページの『付録 D 日時呼び出し可能サービス』

関連タスク
511 ページの『日時呼び出し可能サービスからのフィードバックの取得』
511 ページの『日時の呼び出し可能サービスからの条件の処理』

関連参照

513 ページの『フィードバック・トークン』
514 ページの『ピクチャー文字項およびストリング』
CALL ステートメント (*COBOL for Linux on x86* 言語解説書)

日時呼び出し可能サービスからのフィードバックの取得

日時の呼び出し可能サービスでは、フィードバック・コード・パラメーター (オプション) を指定することができます。 このサービスが呼び出しの成功/失敗に関する情報を返さないようにするには、このパラメーターに対して OMITTED を指定します。

ただし、このパラメーターを指定せずに、呼び出し可能サービスが失敗した場合は、プログラムが異常終了します。

日時の呼び出し可能サービスの呼び出し時に、フィードバック・コードに OMITTED を指定すると、サービスが成功した場合は RETURN-CODE 特殊レジスターが 0 に設定されますが、サービスが失敗した場合は特殊レジスターが変更されません。 フィードバック・コードが OMITTED でない場合は、サービスが成功したかどうかに関係なく、RETURN-CODE 特殊レジスターは常に 0 に設定されます。

512 ページの『例: 出力用の日付形式』

関連参照
513 ページの『フィードバック・トークン』

日時の呼び出し可能サービスからの条件の処理

COBOL for Linux による条件処理は、 ホスト上の IBM 言語環境プログラムによって提供される条件処理とは大幅に異なります。COBOL for Linux は、ネイティブの COBOL 条件処理スキームに従っており、言語環境プログラムにあるサポートのレベルには対応していません。

フィードバック・トークンを引数として渡す場合は、適切な情報が埋め込まれると単に返されます。 呼び出しルーチンにロジックをコーディングして、内容を検証し、必要に応じてアクションを実行することができます。 条件はシグナル通知されません。

関連参照
513 ページの『フィードバック・トークン』

例: 日付の操作

次の例では、日時 の呼び出し可能サービスを使用して日付を別の形式に変換し、このフォーマットされた日付で単純な計算を行う方法を示します。

```
CALL CEEDAYS USING dateof_hire, 'YYMMDD', doh_lilian, fc.
CALL CEELOCT USING todayLilian, today_seconds, today_Gregorian, fc.
COMPUTE servicedays  = today_Lilian - doh_Lilian.
COMPUTE serviceyears = service_days / 365.25.
```

上の例では、YYMMDD 形式の雇用日を元にして、従業員の就労年数を計算します。 計算は次のようになります。

1. CEEDAYS (日付をリリアン形式へ変換) を呼び出して、日付をリリアン形式に変換します。
2. CEELOCT (現在の現地時間の取得) を呼び出して、現在の現地時間を取得します。
3. today_Lilian から doh_Lilian を減算して (グレゴリオ暦の開始から現地時間までの日数)、従業員の雇用日数を計算します。
4. この日数を 365.25 で除算して、就労年数を求めます。

例: 出力用の日付形式

次のサンプルは、日時の呼び出し可能サービスを使用して、ACCEPT ステートメントから取得された日付をフォーマットして表示します。

多くの呼び出し可能サービスには、サービスを使用しない場合には大量のコーディングを必要とする機能が備わっています。 それが CEEDAYS および CEEDATE というサービスです。日付をフォーマットする際には、これらを効率的に利用することができます。

```
CBL QUOTE
       ID DIVISION.
       PROGRAM-ID. HOHOHO.
      *************************************************************
      * FUNCTION:  DISPLAY TODAY'S DATE IN THE FOLLOWING FORMAT: *
      *            WWWWWWWWW, MMMMMMMM DD, YYYY                  *
      *                                                          *
      *            For example:  MONDAY, OCTOBER 18, 2010        *
      *                                                          *
      *************************************************************
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.

       01   CHRDATE.
            05 CHRDATE-LENGTH     PIC S9(4) COMP VALUE 10.
            05 CHRDATE-STRING     PIC X(10).
       01   PICSTR.
            05 PICSTR-LENGTH      PIC  S9(4) COMP.
            05 PICSTR-STRING      PIC  X(80).

       77   LILIAN PIC           S9(9) COMP.
       77   FORMATTED-DATE       PIC X(80).

       PROCEDURE DIVISION.
      ****************************************************************
      *    USE  DATE/TIME CALLABLE SERVICES TO PRINT OUT             *
      *    TODAY'S DATE FROM COBOL ACCEPT STATEMENT.                 *
      ****************************************************************
           ACCEPT CHRDATE-STRING FROM DATE.

           MOVE "YYMMDD" TO PICSTR-STRING.
           MOVE 6 TO PICSTR-LENGTH.
           CALL "CEEDAYS" USING CHRDATE , PICSTR , LILIAN , OMITTED.

           MOVE " WWWWWWWWWZ, MMMMMMMMMZ DD, YYYY " TO PICSTR-STRING.
           MOVE 50 TO PICSTR-LENGTH.
           CALL "CEEDATE" USING LILIAN , PICSTR , FORMATTED-DATE ,
```

```
                OMITTED.

           DISPLAY "******************************".
           DISPLAY FORMATTED-DATE.
           DISPLAY "******************************".

           STOP RUN.
```

フィードバック・トークン

フィードバック・トークンには、フィードバック情報が条件トークンの形式で含まれています。 呼び出し可能サービスによって設定される条件トークンは呼び出しルーチンに戻され、サービスが正常に完了したかどうかを示します。

COBOL for Linux では、言語環境プログラムと同じフィードバック・トークンを使用します。これは次のように定義されます。

```
01  FC.
    02  Condition-Token-Value.
    COPY  CEEIGZCT.
        03  Case-1-Condition-ID.
            04  Severity    PIC S9(4) COMP.
            04  Msg-No      PIC S9(4) COMP.
        03  Case-2-Condition-ID
                  REDEFINES Case-1-Condition-ID.
            04  Class-Code  PIC S9(4) COMP.
            04  Cause-Code  PIC S9(4) COMP.
        03  Case-Sev-Ctl    PIC X.
        03  Facility-ID     PIC XXX.
    02  I-S-Info            PIC S9(9) COMP.
```

各フィールドの内容とホスト上の IBM 言語環境プログラムとの違いは、以下のとおりです。

Severity
重大度数を表します。次の値を持ちます。

0
情報のみ (あるいは、トークン全体がゼロの場合は情報なし)。

1
警告: サービスは、ほぼ正常に完了しました。

2
エラーの検出: 修正が試みられましたが、サービスはおそらく正常には完了しませんでした。

3
重大エラー: サービスは完了しませんでした。

4
クリティカル・エラー: サービスは完了しませんでした。

Msg-No
関連するメッセージ番号です。

Case-Sev-Ctl
このフィールドには、常に値 1 が入ります。

Facility-ID
このフィールドには、常に文字 CEE が入ります。

I-S-Info
このフィールドには、常に値 0 が入ります。

サンプルのコピーブック CEEIGZCT.CPY は、条件トークンを定義します。 ファイル内の条件トークンは、言語環境プログラムに備わっている条件トークンと同じです。ただし､文字表現は EBCDIC ではなく ASCII です。 条件トークンを言語環境プログラムに備わっているものと比較する場合は、違いを考慮する必要があります。

各呼び出し可能サービスの記述には、シンボリック・フィードバック・コードのリストが含まれます。これらのコードは、サービスの呼び出し時に指定されたフィードバック・コード出力フィールドに戻される

場合があります。 このほかに、任意の呼び出し可能サービスに対してシンボリック・フィードバック・コード CEE0PD が戻される場合があります。 詳細については、メッセージ IWZ0813S を参照してください。

日時呼び出し可能サービスはすべて、グレゴリオ暦に基づいています。 グレゴリオ暦に関連する日付変数には、アーキテクチャー上の制限があります。 これらの制限を次に示します。

リリアン日付の開始
リリアン日付範囲の 1 日目は、グレゴリオ暦の 1582 年 10 月 15 日 (金曜日) と同じです。 この日付よりも前のリリアン日付は定義されていません。 したがって、次のようになります。

- 0 日目 = 1582 年 10 月 14 日 00:00:00
- 1 日目 = 1582 年 10 月 15 日 00:00:00

日付の入力として有効となるのは、1582 年 10 月 15 日の 00:00:00 以降です。

リリアン日付の終了
リリアン日付の終了日は、9999 年 12 月 31 日に設定されます。 この日付よりも後のリリアン日付は、9999 年が 4 桁で可能な最大の年であるため定義されていません。

関連参照
603 ページの『付録 G ランタイム・メッセージ』

ピクチャー文字項およびストリング

いくつかの日時の 呼び出し可能サービスには、ピクチャー・ストリング (入力データの形式または出力データの必須形式を示すテンプレート) を 使用します。

表 *52.* **ピクチャー文字項およびストリング**

ピクチャー項	説明	有効な値	Notes®
Y	1 桁の年号	0 から 9	Y は出力に対してのみ有効です。 YY は CEESCEN によって設定 された範囲を前提とします。 YYY/ZYY は、<JJJJ>、<CCCC>、および <CCCCCCCC> とともに使用されます。
YY	2 桁の年号	00 から 99	
YYY	3 桁の年号	000 から 999	
ZYY	特定元号での 3 桁の年号	1 から 999	
YYYY	4 桁の年号	1582 から 9999	
<JJJJ>	UTF-16 の 16 進数エンコード方式を使用した漢字の日本元号	令和 (NX'E44E8C54')	YY フィールドに影響します。 <JJJJ> が指定されている場合、YY は特定の日本元号の年号を意味します。 例えば、1988 年は昭和 63 年に相当します。
		平成 (NX'735E1062')	
		昭和 (NX'2D668C54')	
		大正 (NX'2759636B')	
		明治 (NX'0E66BB6C')	
MM	2 桁の月数	01 から 12	出力の場合は、先行ゼロが抑制されます。入力の場合は、ZM も MM と同じように扱われます。
ZM	1 または 2 桁の月数	1 から 12	
RRRR	ローマ数字の月数	I*bbb*-XII*b* (左揃え)	入力の場合は、ソース・ストリングが大文字変換されます。 出力の場合は、大文字のみで行われます。 I=Jan、II=Feb、...、XII=Dec
RRRZ			

表 52. **ピクチャー文字項およびストリング** (続き)

ピクチャー項	説明	有効な値	Notes®
MMM	3 文字の月名、大文字	JAN から DEC	入力の場合は、ソース・ストリングが必ず大文字変換されます。出力の場合は、M が大文字、m が小文字を生成します。 出力には、ブランク (*b*) が埋め込まれるか (Z が指定されていない場合)、M の数 (最大 20) まで切り詰められます。
Mmm	3 文字の月名、大/小文字混合	Jan から Dec	
MMMM...M	3–20 文字の月名、大文字	JANUARY*bb*-DECEMBER*b*	
Mmmm...m	3–20 文字の月名、大/小文字混合	January*bb*-December*b*	
MMMMMMMMMMZ	末尾ブランクは抑制されます。	JANUARY から DECEMBER	
Mmmmmmmmmmz	末尾ブランクは抑制されます。	January から December	
DD	2 桁の日付	01 から 31	出力の場合は、先行ゼロが常に抑制されます。 入力の場合は、ZD も DD と同じように扱われます。
ZD	1 または 2 桁の日付	1 から 31	
DDD	年間通算日 (ユリウス日付)	001 から 366	
HH	2 桁の時間数	00 から 23	出力の場合は、先行ゼロが抑制されます。入力の場合は、ZH も HH と同じように扱われます。 AP が指定されている場合の有効値は 01 から 12 です。
ZH	1 または 2 桁の時間数	0 から 23	
MI	分数	00 から 59	
SS	2 番目		
9	10 分の 1 の秒数	0 から 9	丸めなし
99	100 分の 1 の秒数	00 から 99	
999	1000 分の 1 の秒数	000 から 999	
AP	AM/PM 標識	AM または PM	AP は HH/ZH フィールドに影響します。 入力の場合は、ソース・ストリングが必ず大文字変換されます。 出力の場合は、AP が大文字、ap が小文字を生成します。
ap		am または pm	
A.P.		A.M. または P.M.	
a.p.		a.m. または p.m.	
W	1 文字の曜日名	S、M、T、W、T、F、S	入力の場合は、W が無視されます。出力の場合は、W が大文字、w が小文字を生成します。 出力には、ブランクが埋め込まれるか (Z が指定されていない場合)、W の数 (最大 20) まで切り詰められます。
WWW	3 文字の曜日名、大文字	SUN から SAT	
Www	3 文字の曜日名、大/小文字混合	Sun から Sat	
WWW...W	3–20 文字の曜日名、大文字	SUNDAY*bbb*-SATURDAY*b*	
Www...w	3–20 文字の曜日名、大/小文字混合	Sunday*bbb*-Saturday*b*	
WWWWWWWWWZ	末尾ブランクは抑制されます。	SUNDAY から SATURDAY	
Wwwwwwwwwz	末尾ブランクは抑制されます。	Sunday から Saturday	

表 52. **ピクチャー文字項およびストリング** (続き)			
ピクチャー項	**説明**	**有効な値**	**Notes®**
それ以外	区切り文字	X'01'から X'FF' (X'00' は 日時 呼び出し可能サービスが内部使用するために予約済み)	入力の場合は、月、日、年、時間、分、秒、秒の小数部の間の区切り文字として扱われます。 出力の場合は、現状のままターゲット・ストリングにコピーされます。
注: ブランク文字は、シンボル *b* で示されます。			

次の表に、<JJJJ> が指定されている場合に日時サービスによって使用される日本元号の定義を示します。

表 53. **日本元号**			
各日本元号の 1 日目	**年号名**	**UTF-16 の 16 進数エンコード方式を使用した漢字の元号名**	**有効な年数値**
1868-09-08	明治	NX'0E66BB6C'	01 から 45
1912-07-30	大正	NX'2759636B'	01 から 15
1926-12-25	昭和	NX'2D668C54'	01 から 64
1989-01-08	平成	NX'735E1062'	01 から 31
2019-05-01	令和	NX'E44E8C54'	01-999 (01 = 2019)

516 ページの『例: 日時のピクチャー・ストリング』

例: 日時のピクチャー・ストリング

日時サービスによって認識されるピクチャー・ストリングの例を次に示します。

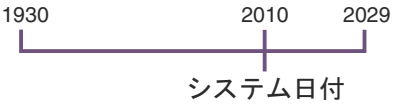
表 54. **日時のピクチャー・ストリングの例**		
ピクチャー・ストリング	**例**	**コメント**
YYMMDD	880516	
YYYYMMDD	19880516	
YYYY-MM-DD	1988-05-16	1988-5-16 も有効な入力になります。
<JJJJ> YY.MM.DD	昭和 63.05.16	昭和 は日本元号名です。 昭和 63 年は 1988 年に相当します。
MMDDYY	050688	1 桁の年号形式 (Y) は、出力に対してのみ有効です。
MM/DD/YY	05/06/88	
ZM/ZD/YY	5/6/88	
MM/DD/YYYY	05/06/1988	
MM/DD/Y	05/06/8	

表 *54.* **日時のピクチャー・ストリングの例** *(続き)*		
ピクチャー・ストリング	**例**	**コメント**
DD.MM.YY	09.06.88	Z は、ゼロおよびブランクを抑制します。
DD-RRRR-YY	09-VI -88	
DD MMM YY	09 JUN 88	
DD Mmmmmmmmm YY	09 June 88	
ZD Mmmmmmmmmz YY	9 June 88	
Mmmmmmmmmz ZD, YYYY	June 9, 1988	
ZDMMMMMMMMzYY	9JUNE88	
YY.DDD	88.137	ユリウス日付
YYDDD	88137	
YYYY/DDD	1988/137	
YYMMDDHHMISS	880516204229	タイム・スタンプは、CEESECS および CEEDATM に対してのみ有効です。 CEEDATE とともに使用した場合は、時刻の位置にゼロが埋め込まれます。 CEEDAYS とともに使用した場合は、HH、MI、SS、999 の各フィールドが無視されます。
YYYYMMDDHHMISS	19880516204229	
YYYY-MM-DD HH:MI:SS.999	1988-05-16 20:42:29.046	
WWW, ZM/ZD/YY HH:MI AP	MON, 5/16/88 08:42 PM	
Wwwwwwwwwz, DD Mmm YYYY, ZH:MI AP	Monday, 16 May 1988, 8:42 PM	
注: 小文字を使用できるのは、英字のピクチャー項のみです。		

世紀ウィンドウ

2000 年以降の 2 桁年号を処理するために、日時の呼び出し可能サービスではスライド方式を使用しています。この方式では、2 桁年号はすべて、現在のシステム日付よりも 80 年前から始まる 100 年間隔 (世紀ウィンドウ) に属するものと想定されます。

例えば 2010 年では、1930 から 2029 の 100 年間が、日時の呼び出し可能サービスに使用されるデフォルトの世紀ウィンドウとなります。 したがって 2010 年の場合、30 から 99 が 1930 年から 1999 年、00 から 29 が 2000 年から 2029 年と認識されます。



2080 年までは、2 桁年号はすべて 20*nn* 年と認識されます。 2081 年では、00 は 2100 年と認識されます。

アプリケーションによっては、別の 100 年間隔を設定する必要があります。 例えば、銀行で取り扱っている 30 年債の期限が 01/31/30 に設定されているとします。 2 桁の年 30 は、 上で説明した世紀ウィンドウが有効である場合は、1930 として認識されます。

CEESCEN 呼び出し可能サービスを使用すると、世紀ウィンドウを変更できます。 もう一方のサービス CEEQCEN は、現行の世紀ウィンドウを照会します。

CEEQCEN および CEESCEN を使用すると、例えばサブルーチンに、親ルーチンとは異なる日付処理間隔を使用させることができます。 サブルーチンは、戻る前にこの間隔を前の値にリセットする必要があります。

518 ページの『例: 世紀ウィンドウの照会および変更』

例: 世紀ウィンドウの照会および変更

CEEQCEN および CEESCEN サービスを使用して、世紀ウィンドウの開始点を照会、設定、および復元する方法の例を次に示します。

この例では、CEEQCEN を呼び出して整数 (OLDCEN) を取得し、現行の世紀ウィンドウが何年前に始まったかを示します。 次に、新しい値 (TEMPCEN) で CEESCEN を呼び出して、現行の世紀ウィンドウの開始点を新しい値に一時的に変更します。 この世紀ウィンドウは 30 に設定されているため、CEESCEN 呼び出しに続く 2 桁年号は、現行のシステム日付より 30 年前から始まる 100 年間隔に属するものと想定されます。

最後に、一時的な世紀ウィンドウを使用して日付 (例示なし) を処理した後、再度 CEESCEN を呼び出して、世紀ウィンドウの開始点を元の値にリセットします。

```
 WORKING-STORAGE SECTION.
 77 OLDCEN  PIC S9(9) COMP.
 77 TEMPCEN PIC S9(9) COMP.
 77 QCENFC  PIC X(12).
 . . .
 77 SCENFC1 PIC X(12).
 77 SCENFC2 PIC X(12).
 . . .
 PROCEDURE DIVISION.
     . . .
** Call CEEQCEN to retrieve and save current century window
     CALL "CEEQCEN" USING OLDCEN, QCENFC.
** Call CEESCEN to temporarily change century window to 30
     MOVE 30 TO TEMPCEN.
     CALL "CEESCEN" USING TEMPCEN, SCENFC1.
** Perform date processing with two-digit years
     . . .
** Call CEESCEN again to reset century window
     CALL "CEESCEN" USING OLDCEN, SCENFC2.
     . . .
     GOBACK.
```

関連参照

541 ページの『付録 D 日時呼び出し可能サービス』

形式 2 SORT ステートメントを使用したテーブルのソート

テーブルをソートするには、形式 2 SORT ステートメントの使用をお勧めします。 この方法は、形式 1 SORT ステートメントに比べて以下の利点があります。

表 *55.* 形式 *1* と形式 *2* の *SORT* ステートメントの比較

特性	**形式 1 SORT ステートメント**	**形式 2 SORT ステートメント**
ファイルまたはテーブルのソートに使用できる	はい	いいえ、テーブル専用です。
DFSORT または同等のソート・プログラムが必要	はい	いいえ
CICS でサポートされる	制限付き	はい
UNIX システム・サービスでサポートされる	いいえ	はい
1 つの SORT ステートメントを使用してテーブルをソートできる (コーディングが簡単)	いいえ、SELECT 節、レコード記述のある SD 項目、および入出力プロシージャーが必要です。	はい

表 *55.* 形式 *1* と形式 *2* の *SORT* ステートメントの比較 *(*続き*)*		
特性	**形式 1 SORT ステートメント**	**形式 2 SORT ステートメント**
ソートのキーをテーブル定義の一部として指定できる (SEARCH ALL ステートメントでも使用可能)	いいえ、キーは SORT ステートメントで指定しなければなりません。 SEARCH ALL を使用したテーブル検索も行う場合は、テーブル定義の一部としてキーを重複して指定することも必要です。	はい、必要であれば SORT ステートメントでキーを指定することもできます。
ソート処理中にテーブル・エレメントをフィルターに掛けるか、前処理を行うことができる	はい、入出力プロシージャーを使用します。	いいえ、テーブル・エレメントはすべてそのまま SORT に渡されます。
SORT-CONTROL、SORT-CORE-SIZE、SORT-FILE-SIZE、SORT-MESSAGE、SORT-MODE-SIZE、SORT-RETURN などの特殊レジスターを使用する	はい	いいえ
入出力プロシージャーの範囲内で実行できる	いいえ	はい

注 : ストレージが制約されている環境で形式 2 SORT を大きなテーブルとともに使用しないでください。形式 2 SORT ではソートの実行にヒープ・ストレージが使用されるためです。

関連参照
SORT ステートメント (*COBOL for Linux on x86* 言語解説書)

付録 A IBM Enterprise COBOL for z/OS との違いのまとめ

COBOL for Linux on x86 は、Enterprise COBOL for z/OS とは別の方法で 特定の項目を実装します。 詳細については、以下の関連参照を参照してください。

関連タスク
425 ページの『第 21 章 プラットフォーム間でのアプリケーションの移植』

関連参照
521 ページの『コンパイラー・オプション』
521 ページの『データ表現』
523 ページの『ランタイム環境変数』
523 ページの『ファイル指定』
524 ページの『言語間通信 (ILC)』
524 ページの『入出力』
525 ページの『ランタイム・オプション』
525 ページの『ソース・コードの行サイズ』
525 ページの『言語エレメント』

コンパイラー・オプション

COBOL for Linux on x86 は、Enterprise COBOL コンパイラー・オプションの一部をサポートしません。

サポートされないオプションは以下のとおりです。

ADATA、ADV、AFP、ARCH、AWO、BLOCK0、BUFSIZE、CODEPAGE、COPYLOC、COPYRIGHT、DATA、DBCS、DECK、DISPSIGN、DLL、DUMP、EXPORTALL、FASTSRT、HGPR、INITCHECK、INITIAL、INLINE、INTDATE、LANGUAGE、LP、MAXPCF、NAME、NUMCHECK、NUMPROC、OBJECT、OFFSET、OPTFILE、OUTDD、PARMCHECK、QUALIFY、RENT、RMODE、RULES、SERVICE、SQLCCSID、SQLIMS、STGOPT、SUPPRESS、THREAD、VLR、VSAMOPENFS、WORD、XMLPARSE、ZONECHECK、および ZONEDATA

関連タスク
425 ページの『IBM Enterprise COBOL for z/OS アプリケーションをコンパイルする』

関連参照

232 ページの『cob2 オプション』
256 ページの『CHAR』

データ表現

データの表示は、IBM Enterprise COBOL と COBOL for Linux on x86 の間で異なることがあります。

2 進数データ

デフォルトで COBOL for Linux on x86 は、バイナリー・データにリトル・エンディアン・フォーマットを使用し、Enterprise COBOL はビッグ・エンディアン・フォーマットを使用します。

ゾーン 10 進数データ

ゾーン 10 進数データの符号表現が ASCII と EBCDIC のどちらに基づくかは、CHAR コンパイラー・オプションの設定 (NATIVE または EBCDIC) と、USAGE 文節が NATIVE 句で指定されているかどうかによって決

© Copyright IBM Corp. 2021

まります。 COBOL for Linux on x86 では、ゾーン 10 進数データの符号表現の処理は、コンパイラー・オプション NUMPROC(NOPFD) が有効な場合に z/OS 上で行われる処理と整合性が取れています。

パック 10 進データ

符号なしパック 10 進数のサイン表記は、COBOL for Linux on x86 と Enterprise COBOL で異なります。 COBOL for Linux on x86 は、符号なしパック 10 進数の符号ニブル x'C' を使用します。 Enterprise COBOL では、符号なしパック 10 進数に対して符号ニブル x'F' が使用されます。 Linux および z/OS 間のパック 10 進数を含むデータ・ファイルを共用する場合は、符号なしパック 10 進数の代わりに符号ありパック 10 進数を使用することをお勧めします。

浮動小数点データの表示

FLOAT(BE) コンパイラー・オプションを使用すると、浮動小数点データ項目の表示がネイティブ (IEEE) 形式ではなく IBM Z データ表現 (16 進数) であることを指定できます。

IBM Z 形式の浮動小数点項目の表示を、INVOKE ステートメントの引数またはメソッドのパラメーターとして指定しないでください。これらの引数やパラメーターは、Java データ型と相互運用可能にするために、ネイティブ形式で指定する必要があります。

国別データ

デフォルトで COBOL for Linux on x86 では国別データに UTF-16 リトル・エンディアン形式を使用し、Enterprise COBOL では国別データに UTF-16 ビッグ・エンディアン形式を使用します。

EBCDIC および ASCII データ

英数字データ項目に EBCDIC 照合シーケンスを指定する際には、次の言語エレメントを使用することができます。

- ALPHABET 文節
- PROGRAM COLLATING SEQUENCE 文節
- SORT または MERGE ステートメントの COLLATING SEQUENCE 句

CHAR(EBCDIC) コンパイラー・オプションを指定すると、DISPLAY データ項目が IBM Z データ表現 (EBCDIC) であることを指定できます。

データ変換用のコード・ページの決定

英字、英数字、DBCS、および国別データ項目の場合は、実行時に有効なロケールから、ネイティブ文字の暗黙変換に使用されるソース・コード・ページが決定されます。

英数字、DBCS、および国別リテラルの場合は、コンパイル時に有効なロケールから、文字の暗黙変換に使用されるソース・コード・ページが決定されます。

DBCS 文字ストリング

COBOL for Linux on x86 では、ASCII DBCS 文字ストリングの区切りにシフトイン/シフトアウト文字は使用されません。ただし例外として、次に説明するようにダミーのシフトイン/シフトアウト文字を使用することは可能です。

SOSI コンパイラー・オプションを使用すると、Linux ワークステーションのシフトアウト (X'1E') およびシフトイン (X'1F') 制御文字が、ソース・プログラム内の DBCS 文字ストリング (ユーザー定義語、DBCS リテラル、英数字リテラル、国別リテラル、コメントを含む) を区切るように指定することができます。 ホストのシフトアウト (X'0E') およびシフトイン (X'0F') 制御文字は一般に、COBOL for Linux on x86 ソース・コードのダウンロード時に、使用するダウンロード方法に応じてワークステーションのシフトアウトおよびシフトイン制御文字に変換されます。

英数字リテラル内に制御文字 X'00' から X'1F' を使用すると、予測不能な結果が生じる可能性があります。

関連タスク
201 ページの『第 11 章 ロケールの設定』
426 ページの『データ表現による違いの修正』

関連参照
256 ページの『CHAR』
280 ページの『SOSI』

ランタイム環境変数

COBOL for Linux on x86 は、次に示すように、Enterprise COBOL では使用されないランタイム環境変数をいくつか認識します。

- CICS_TK_SFS_SERVER
- COBPATH
- COBRTOPT
- EBCDIC_CODEPAGE
- CICS_SFS_DATA_VOLUME
- CICS_SFS_INDEX_VOLUME
- CICS_VSAM_AUTO_FLUSH
- CICS_VSAM_CACHE
- CICS_SFS_CACHE_<*filename*>
- CICS_SFS_RDM_CACHE
- CICS_SFS_PREALLOC_<*filename*>
- COBCORE
- COBOUTDIR
- PATH
- SYSIN、 SYSIPT、 SYSOUT、 SYSLIST、 SYSLST、 CONSOLE、 SYSPUNCH、 SYSPCH

ファイル指定

COBOL for Linux on x86 によるファイルの処理方法と、Enterprise COBOL によるファイルの処理方法には、異なる点が少しあります。

COBOL for Linux on x86 および Enterprise COBOL との間でのファイル処理における相違点は、以下の領域にあります。

- 単一ボリューム・ファイル
- ソース・ファイルの接尾部
- 世代別データ・グループ (GDG)
- ファイルの連結

単一ボリューム・ファイル: COBOL for Linux on x86 は、すべてのファイルを単一ボリューム・ファイルとして扱います。 それ以外のファイル指定はすべてコメントとして扱われます。 この違いは、`REEL`、`UNIT`、`MULTIPLE FILE TAPE` 文節、および `CLOSE. . .UNIT/REEL` に影響します。

ソース・ファイルの接尾部 COBOL for Linux on x86 では、cob2 コマンドの 1 つを使用してコンパイルすると、接尾部が .cbl または .cob のいずれかである COBOL ソース・ファイルがコンパイラーに渡されます。 Enterprise COBOL では、z/OS UNIX ファイル・システム 内でコンパイルすると、接尾部が .cbl のファイルのみがコンパイラーに渡されます。

世代別データ・グループ (GDG): GDG サポートは、Enterprise COBOL での GDG サポートとほぼ同じです。 ただし、COBOL for Linux on x86 では以下の相違点があります。

- 世代別データ・セット (GDS) (または本書では生成ファイル と呼ぶ) は、サポートされているすべてのファイル・システムにおいて、すべてのファイル編成およびアクセス・モードでサポートされます。
- GDG サポートはファイル・システムに統合されていません。 独立型ユーティリティー gdgmgr を使用して、GDG の作成および削除、GDG 項目の管理および照会、限界処理 (limit processing)の実施、および既存ファイルに対する GDG カタログの調整を行います。
- 生成ファイル名の解決は、ジョブの初期化時ではなく、ファイルのオープン時に行われます。
- 限界処理 (limit processing) は、ジョブの終了時ではなく、新しい世代がグループに追加された時に行われます。
- 特定のエポック内の世代範囲は、1000 に制限されるのではなく、1 から 9999 までの間です。 したがって、世代 0001 および 9999 が、同じエポック内に存在することが可能です。
- 1 つのグループに含めることが可能な世代数は、255 ではなく 1000 です。
- バージョン管理はサポートされていません。 自動生成されるバージョンは常に v00 です。

ファイルの連結: COBOL for Linux では、ファイル ID をコロン (:) で区切ることにより、複数のファイルを連結できます。連結される COBOL ファイルは、順次編成または行順次編成である必要があり、順次アクセスでアクセスされる必要があります。また、入力用のみでオープン可能です。

関連概念
124 ページの『世代別データ・グループ』

関連タスク
131 ページの『ファイルの連結』
225 ページの『コマンド行からのコンパイル』

関連参照
130 ページの『世代別データ・グループの限界処理 (limit processing)』

言語間通信 (ILC)

ILC は、C/C++ プログラムで使用可能です。

言語環境プログラム を持つ z/OS 上で ILC を使用した場合と比較した、Linux on x86 上での ILC の動作の違いを示します。

- COBOL の STOP RUN C の exit() を使用した場合は、終了動作に違いがあります。
- 調整された条件処理は COBOL for Linux にはありません。COBOL プログラム間で C の longjmp() を使用することは避けてください。
- Enterprise COBOL では、プロセス内で呼び出され、なおかつ 言語環境プログラム に対応している最初のプログラムが、「メイン」プログラムと見なされます。COBOL for Linux 上では、プロセス内で呼び出される最初のプログラムが、COBOL による「メイン」プログラムと見なされます。この違いは、実行単位 (メインプログラムで始まる実行単位) の定義に依存する言語セマンティクスに影響します。例えば、STOP RUN を使用すると、制御権はメインプログラムの呼び出し側に戻りますが、混合言語環境では、前述とは異なる結果になる可能性があります。

関連概念
469 ページの『第 25 章 COBOL ランタイム環境の事前初期設定』

入出力

COBOL for Linux on x86 は、Db2、SdU、SFS、および STL ファイル・システムでの順次ファイル、相対ファイル、および索引付きファイルの入出力をサポートしています。また、QSAM ファイル・システムと RSD ファイル・システムでの順次ファイルの入出力もサポートしています。

行順次の入出力は、オペレーティング・システムのネイティブのバイト・ストリーム・ファイル・サポートでサポートしています。

返されるファイル状況情報のサイズおよび値は、使用しているファイル・システムによって異なります。

COBOL for Linux on x86 では、磁気テープ・ドライブやディスケット・ドライブを直接はサポートしていません。

関連概念
117 ページの『ファイル・システム』
122 ページの『行順次ファイル編成』

関連タスク
170 ページの『ファイル状況キーの使用』
172 ページの『ファイル・システム状況コードの使用』

ランタイム・オプション

COBOL for Linux on x86 は、 Enterprise COBOL ランタイム・オプション AIXBLD、ALL31、CBLPSHPOP、CBLQDA、COUNTRY、HEAP、MSGFILE、NATLANG、SIMVRD、 および STACK を認識せずに無効として扱います。

Enterprise COBOL では、STORAGE ランタイム・オプションを使用して、COBOL の WORKING-STORAGE を初期化することができます。COBOL for Linux on x86 では、WSCLEAR コンパイラー・オプションを使用します。

関連参照
289 ページの『WSCLEAR』

ソース・コードの行サイズ

COBOL for Linux on x86 では、COBOL ソース行の長さが可変です。 ソース行は、改行制御文字がある場所、または行の最大長に達した場所で終了します。

Enterprise COBOL では、各ソース行の長さは同じです。

関連参照
283 ページの『SRCFORMAT』

言語エレメント

次の表では、Enterprise COBOL コンパイラーと COBOL for Linux on x86 コンパイラーの間で異なる言語エレメントをリストし、可能であれば、COBOL for Linux on x86 プログラムにおいてそのような相違に対処する方法についてアドバイスしています。

Enterprise COBOL で有効な COBOL の文節および句の多くは構文チェックされますが、これらは COBOL for Linux on x86 プログラムの実行には影響しません。これらの文節や句は、ダウンロードした既存のアプリケーションにわずかな影響を与えます。 COBOL for Linux on x86 では、その構文が機能的な影響を持たない場合でも、Enterprise COBOL 言語構文を認識します。

表 56. ***Enterprise COBOL for z/OS* と *COBOL for Linux on x86* 間の言語の違い**

言語エレメント	**COBOL for Linux on x86 インプリメンテーションまたは制約事項**
ACCEPT ステートメント	Enterprise COBOL プログラムが ACCEPT ステートメントのターゲットとして DD 名を期待する場合は、値が適切なファイル名に設定された同等の環境変数を使用して、これらのターゲットを定義します。COBOL for Linux on x86 では、*environment-name* および関連する環境変数値 (設定されている場合) によって、ファイル ID が決まります。
APPLY WRITE-ONLY 文節	構文が検査されますが、COBOL for Linux on x86 のプログラムの実行には影響しません
ASSIGN 文節	COBOL for Linux on x86 は、*assignment-name* に基づいたシステム・ファイル名に対し、異なる構文およびマッピングを使用します。 ASSIGN . . .USING *data-name* は、Enterprise COBOL ではサポートされません。

表 56. ***Enterprise COBOL for z/OS* と *COBOL for Linux on x86* 間の言語の違い** (続き)	
言語エレメント	**COBOL for Linux on x86 インプリメンテーションまたは制約事項**
BLOCK CONTAINS 文節	構文が検査されますが、COBOL for Linux on x86 のプログラムの実行には影響しません
CALL ステートメント	COBOL for Linux on x86 では、ファイル名を CALL 引数として使用できません。
CLOSE ステートメント	FOR REMOVAL 句、WITH NO REWIND 句、 および UNIT/REEL 句は、COBOL for Linux on x86 で構文が検査されますが、プログラムの実行には影響しません。 移植可能にすることを 目的とするプログラムで、これらの句を使用しないでください。
CODE-SET 文節	構文が検査されますが、COBOL for Linux on x86 のプログラムの実行には影響しません
DATA RECORDS 文節	構文が検査されますが、COBOL for Linux on x86 のプログラムの実行には影響しません
DISPLAY ステートメント	Enterprise COBOL プログラムが DISPLAY ステートメントのターゲットとして DD 名を期待する場合は、値が適切なファイル名に設定された同等の環境変数を使用して、これらのターゲットを定義します。 COBOL for Linux on x86 では、*environment-name* および関連する環境変数値 (設定されている場合) によって、ファイル ID が決まります。
DYNAMIC LENGTH 節	動的長基本項目は、現時点では COBOL for Linux on x86 でサポートされていません。
ファイル状況 *data-name-1*	ファイル状況 9*x* の一部の値と意味は、Enterprise COBOL と COBOL for Linux on x86 とで異なります。
ファイル状況 *data-name-8*	プラットフォームおよびファイル・システムによって、形式と値が異なります。
INDEX データ項目	Enterprise COBOL では、INDEX データ項目は 4 バイトとして暗黙的に定義されます。ADDR(32) でコンパイルされた COBOL for Linux on x86 プログラムの場合、サイズは 4 バイトです。 コンパイル時に ADDR(64) が使用された場合、サイズは 8 バイトです。
JSON GENERATE ステートメントおよび JSON PARSE ステートメント	JSON は、現時点では COBOL for Linux on x86 でサポートされていません。
LABEL RECORDS 文節	LABEL RECORD IS *data-name* 句、USE. . .AFTER. . .LABEL PROCEDURE 句、 GO TO MORE-LABELS 句は、構文検査されますが、COBOL for Linux on x86 のプログラムの実行には影響しません。これらのいずれかの句を使用すると、警告が発行されます。 ユーザー・ラベルの宣言は実行時には呼び出されません。 z/OS QSAM 対応のユーザー・ラベル処理に依存するプログラムは移植できません。
MULTIPLE FILE TAPE	構文が検査されますが、COBOL for Linux on x86 のプログラムの実行には影響しません。Linux ワークステーションでは、すべてのファイルが単一ボリューム・ファイルとして扱われます。
OBJECT REFERENCE データ項目	OBJECT REFERENCE データ項目は COBOL for Linux on x86 ではサポートされていません。
OPEN ステートメント	REVERSED 句 および WITH NO REWIND 句は、COBOL for Linux on x86 で構文が検査されますが、プログラムの実行には影響しません。
PASSWORD 文節	構文が検査されますが、COBOL for Linux on x86 のプログラムの実行には影響しません

表 56. ***Enterprise COBOL for z/OS* と *COBOL for Linux on x86* 間の言語の違い** *(続き)*	
言語エレメント	**COBOL for Linux on x86 インプリメンテーションまたは制約事項**
POINTER、PROCEDURE-POINTER、および FUNCTION-POINTER データ項目	Enterprise COBOL では、POINTER データ項目と FUNCTION-POINTER データ項目は、 特殊レジスター ADDRESS OF と同様に、4 バイトとして暗黙的に定義されます。 PROCEDURE-POINTER データ項目は、8 バイトとして暗黙的に定義されます。 ADDR(32) でコンパイルされた COBOL for Linux on x86 プログラムの場合、これらの各データ項目のサイズは 4 バイトです。 コンパイル時に ADDR(64) が使用された場合、サイズは 8 バイトです。
READ . . .PREVIOUS	COBOL for Linux on x86 でのみ、DYNAMIC アクセス・モードを使用して、相対ファイルまたは索引付きファイルの前のレコードを読み取ることができます。
RECORD CONTAINS 文節	RECORD CONTAINS *n* CHARACTERS 文節は受け入れられますが、例外が 1 つあります。 RECORD CONTAINS 0 CHARACTERS は、COBOL for Linux on x86 で構文が検査されますが、プログラムの実行には影響しません。
RECORDING MODE 文節	相対ファイル、索引付きファイル、および行順次ファイルについては、COBOL for Linux on x86 で構文が検査されますが、プログラムの実行には影響しません。 順次ファイルの場合、RECORDING MODE U は構文が検査されますが、プログラムの実行には影響しません。
RERUN 文節	構文が検査されますが、COBOL for Linux on x86 のプログラムの実行には影響しません
RESERVE 文節	構文が検査されますが、COBOL for Linux on x86 のプログラムの実行には影響しません
SAME AREA 文節	構文が検査されますが、COBOL for Linux on x86 のプログラムの実行には影響しません
SAME SORT 文節	構文が検査されますが、COBOL for Linux on x86 のプログラムの実行には影響しません
SHIFT-IN、SHIFT-OUT 特殊レジスター	COBOL for Linux on x86 コンパイラーは、CHAR(EBCDIC) コンパイラー・オプションが有効でないと、これらのレジスターが検出されたときに E レベルのメッセージを出します。
SORT-CONTROL 特殊レジスター	この特殊レジスターの暗黙的な定義および内容は、ホスト COBOL とワークステーション COBOL とでは異なります。
SORT-CORE-SIZE 特殊レジスター	この特殊レジスターの内容は、ホスト COBOL とワークステーション COBOL とでは異なります。
SORT-FILE-SIZE 特殊レジスター	構文が検査されますが、COBOL for Linux on x86 のプログラムの実行には影響しません。この特殊レジスターの値は使用されません。
SORT-MESSAGE 特殊レジスター	構文が検査されますが、COBOL for Linux on x86 のプログラムの実行には影響しません
SORT-MODE-SIZE 特殊レジスター	構文が検査されますが、COBOL for Linux on x86 のプログラムの実行には影響しません。この特殊レジスターの値は使用されません。
SORT MERGE AREA 文節	構文が検査されますが、COBOL for Linux on x86 のプログラムの実行には影響しません
START...	COBOL for Linux on x86 では、IS LESS THAN、IS <、IS NOT GREATER THAN、IS NOT >、IS LESS THAN OR EQUAL TO、IS <= の各関係演算子を使用できます。

表 56. ***Enterprise COBOL for z/OS* と *COBOL for Linux on x86* 間の言語の違い** *(続き)*	
言語エレメント	**COBOL for Linux on x86 インプリメンテーションまたは制約事項**
STOP RUN	COBOL for Linux on x86 の マルチスレッド・プログラムではサポートされません。マルチスレッド・プログラム内で、C exit() 関数に対する呼び出しと置き換えることができます。
USAGE 節の UTF-8 句および 'U' PICTURE 記号	UTF-8 データ・クラスおよび UTF-8 データ・カテゴリーは、現時点では COBOL for Linux on x86 でサポートされていません。
WRITE ステートメント	COBOL for Linux on x86 では、WRITE. . .ADVANCING を C01 から C12 までの 環境名または S01 から S05 までの環境名とともに指定した場合、1 行拡張されます。
XML PARSE ステートメント	ホスト専用オプション XMLPARSE(XMLSS) を使用してコンパイルされた Enterprise COBOL プログラムでは、COBOL for Linux on x86 では利用できない 名前空間処理用の追加構文 (ENCODING 句および RETURNING NATIONAL 句) や特殊レジスターが 使用可能です。
プラットフォーム環境に既知の名前	*program-name*、*text-name*、*library-name*、*assignment-name*、SORT-CONTROL 特殊レジスター内のファイル名、*basis-name*、DISPLAY または ACCEPT ターゲット識別、およびシステム依存名は、それぞれ異なる方法で識別されます。

付録 B IBM Z ホスト・データ形式についての考慮事項

IBM Z ホスト・データ内部表現を使用するときに当てはまる 考慮事項、制約事項、および制限について、以下で説明します。

CHAR および FLOAT コンパイラー・オプションは、IBM Z ホスト・データ形式またはネイティブのデータ形式が使用されているかどうかを判別します (COMP-5 項目、または USAGE 文節内の NATIVE 句で定義された項目以外)。 (この情報の用語「ホスト・データ形式」と「ネイティブのデータ形式」は、データ項目の内部表現を指します。)

CICS アクセス

分離または統合された CICS 変換プログラムによって変換され、CICS TXSeries または CICS TX 上で動作する COBOL プログラムでサポートされる IBM Z ホスト・データ形式はありません。

日時呼び出し可能サービス

日時の呼び出し可能サービスでは、IBM Z ホスト・データ形式の内部表現で使用することができます。 呼び出し可能サービスに渡されるパラメーターはすべて、IBM Z ホスト・データ形式にする必要があります。日時サービスに対する同一の呼び出しで、ネイティブ・データの内部表現とホスト・データの内部表現を混用することはできません。

浮動小数点のオーバーフロー例外

Linux ワークステーション、および IBM Z ホスト上では、浮動小数点データ表現の制限に違いがあるため、FLOAT(BE) が有効な場合は、2 つの形式間での変換中に、浮動小数点のオーバーフロー例外が発生する可能性があります。例えば、ホスト上で正常に実行できるプログラムをワークステーション上で実行すると、次のようなメッセージを受け取る可能性があります。

```
IWZ053S An overflow occurred on conversion to floating point
```

この問題を避けるため、どちらのプラットフォームでも、データ型ごとにサポートされる最大浮動小数点値に注意する必要があります。 次の表に、それぞれの制限を示します。

表 *57.* **最大浮動小数点値**

データ型	最大ワークステーション値	最大 IBM Z ホスト値
COMP-1	*(2**128 - 2**4) (約 *3.4028E+38)	*(16**63 - 16**57) (約 *7.2370E+75)
COMP-2	*(2**1024 - 2**971) (約 *1.7977E+308)	*(16**63 - 16**49) (約 *7.2370E+75)
* 値は正数でも負数でも構いません。		

上記のとおり、ホストの COMP-1 値はワークステーションよりも大きくすることができ、ワークステーションの COMP-2 値はホストよりも大きくすることができます。

Db2

IBM Z ホスト・データ形式のコンパイラー・オプションは、Db2 プログラムで使用することができます。

© Copyright IBM Corp. 2021

分散コンピューティング環境アプリケーション

IBM Z ホスト・データ形式のコンパイラー・オプションは、分散コンピューティング環境プログラムで使用しないでください。

ファイル・データ

- EBCDIC データおよび 16 進数/2 進数データは、任意の順次ファイル、相対ファイル、索引付きファイルで読み取りおよび書き込みを行うことができます。 自動変換は行われません。
- ホスト・データを含むファイルにアクセスする場合は、コンパイラー・オプション BINARY(BE)、COLLSEQ(EBCDIC)、CHAR(EBCDIC)、および FLOAT(BE) を使用して、これらのファイルから得られる 2 進数データ、EBCDIC 文字データ、および 16 進数浮動小数点データを処理します。

SORT

IBM Z ホスト・データ形式 DBCS (USAGE DISPLAY-1 を除く) は、すべてソート・キーとして使用できます。

関連概念
39 ページの『数値データの形式』

関連参照

248 ページの『コンパイラー・オプション』

付録 C 中間結果および算術精度

コンパイラーは、算術ステートメントを、演算子優先順位に従って実行される一連の操作として処理し、それらの操作の結果を入れる中間フィールドをセットアップします。 コンパイラーはいくつかのアルゴリズムを使用して、確保する整数および小数部の桁数を判別します。

中間結果は、次の場合に得ることができます。

- 動詞の直後に複数のオペランドが入った ADD または SUBTRACT ステートメント。
- 一連の算術演算または複数の結果フィールドを指定する COMPUTE ステートメント。
- 条件ステートメントまたは参照変更指定に含まれている算術式。
- GIVING オプションおよび複数の結果フィールドを使用する ADD、SUBTRACT、MULTIPLY、または DIVIDE ステートメント。
- 組み込み関数をオペランドとして使用するステートメント。

532 ページの『例: 中間結果の計算』

中間結果の精度は、デフォルト・オプション ARITH(COMPAT) (互換モード と呼ばれる) を使用してコンパイルするか、または以下に説明しているように ARITH(EXTEND) (拡張モード と呼ばれる) を使用してコンパイルするかによって異なります。

互換モードでの算術演算の計算は、 IBM COBOL Set for Linux の場合と変わりません。

- 固定小数点の中間結果の場合は、最大 30 桁が使用されます。
- 浮動小数点組み込み関数は、長精度 (64 ビットの) 浮動小数点結果を戻します。
- 浮動小数点オペランド、分数指数、または浮動小数点組み込み関数を含む式は、浮動小数点でないすべてのオペランドが長精度浮動小数点に変換され、式の計算に浮動小数点演算が使用されるかのように評価されます。
- 浮動小数点リテラルおよび外部浮動小数点データ項目は、長精度浮動小数点に変換されてから処理されます。

拡張モードでの算術演算の計算には、次のような特性があります。

- 固定小数点の中間結果の場合は、最大 31 桁が使用されます。
- 浮動小数点組み込み関数は、拡張精度 (128 ビットの) 浮動小数点結果を戻します。
- 浮動小数点オペランド、分数指数、または浮動小数点組み込み関数を含む式は、浮動小数点でないすべてのオペランドが拡張精度浮動小数点に変換され、式の計算に浮動小数点演算が使用されるかのように評価されます。
- 浮動小数点リテラルおよび外部浮動小数点データ項目は、拡張精度浮動小数点に変換されてから処理されます。

関連概念

39 ページの『数値データの形式』
53 ページの『固定小数点演算と浮動小数点演算の対比』

関連参照

533 ページの『固定小数点データと中間結果』
538 ページの『浮動小数点データと中間結果』
539 ページの『非算術ステートメントの算術式』
253 ページの『ARITH』

© Copyright IBM Corp. 2021

中間結果用の用語

中間結果に関するこの情報を理解するには、以下の用語を理解する必要があります。

i
中間結果用に保持された整数の桁数。 (ROUNDED 句を使用している場合は、正確さを得るために必要なら、整数がもう 1 桁余分に保持します。)

d
中間結果用に保持された小数部の桁数。 (ROUNDED 句を使用している場合は、正確さを得るために必要なら、小数部がもう 1 桁余分に保持します。)

dmax
特定のステートメントで、次の項目のうち最も大きいもの。

- 最終結果フィールド (1 つ以上) に必要な小数部の桁数。
- 除数または指数を除き、オペランドに対して定義された小数部の最大桁数。
- 関数オペランドの *outer-dmax*。

inner-dmax
関数への参照で、次の項目のうち最も大きいもの。

- その基本引数に対して定義された小数部の桁数。
- いずれかの算術式の引数の *dmax*。
- そのいずれかの組み込み関数の *outer-dmax*。

outer-dmax
関数結果がそれ自体の計算の外で演算に寄与する小数部の桁数 (例えば、その関数が算術式中のオペランドであるか、または別の関数への引数である場合)。

op1
生成される算術ステートメントの第 1 オペランド (除算では除数)。

op2
生成される算術ステートメントの第 2 オペランド (除算では被除数)。

i1 , i2
それぞれ *op1* と *op2* 内の整数の数値。

d1 , d2
それぞれ、*op1* および *op2* 内の小数部の桁数。

ir
生成された算術ステートメントまたは演算が実行されたときの中間結果。 (中間結果は、レジスターまたは保管場所のいずれかで生成されます。)

ir1 , ir2
連続する中間結果。 (連続する中間結果は同じ保管場所にすることができます。)

関連参照
ROUNDED 句 (*COBOL for Linux on x86* 言語解説書)

例: 中間結果の計算

次の例は、コンパイラーが算術ステートメントを一連の操作として実行し、必要に応じて中間結果を保管する方法を示しています。

```
COMPUTE Y = A + B * C - D / E + F ** G
```

結果は、以下の順序で計算されます。

1. F を G 乗すると *ir1* が得られる。
2. B に C を掛けると *ir2* が得られる。

3. E を D で割ると *ir3* が得られる。
4. A を ir2 に加えると *ir4* が得られる。
5. *ir3* を *ir4* から引くと *ir5* が得られる。
6. *ir5* を *ir1* に加えると Y が得られる。

関連タスク
49 ページの『算術式の使用』

関連参照
532 ページの『中間結果用の用語』

固定小数点データと中間結果

コンパイラーは、中間結果における整数および小数点以下の桁数を、決定します。

加算、減算、乗算、および除算

次の表は、加算、減算、乗算、または除算の結果として理論上可能な精度を示しています。

演算	整数桁	小数桁
+ または -	(*i1* または *i2*) + 1、どちらか大きい方	*d1* または *d2*、どちらか大きい方
*	*i1* + *i2*	*d1* + *d2*
/	*i2* + *d1*	(*d2* - *d1*) または *dmax*、どちらか大きい方

算術ステートメントのオペランドを十分な小数桁で定義して、最終結果の正確度が希望どおりになるようにしなければなりません。

次の表は、互換モード (つまり、デフォルトのコンパイラー・オプション ARITH(COMPAT) が有効である場合) で加算、減算、乗算、または除算を伴う算術演算の固定小数点中間結果においてコンパイラーが 保持する桁数を示しています。

***i* + *d* の値**	***d* の値**	***i* + *dmax* の値**	***ir* 用に保持される桁数**
<30 または =30	任意の値	任意の値	*i* 整数桁数および *d* 小数桁数
>30	<*dmax* または =*dmax*	任意の値	30-*d* 整数桁数および *d* 小数桁数
	>*dmax*	<30 または =30	*i* 整数桁数および 30-*i* 小数桁数
		>30	30-*dmax* 整数桁数および *dmax* 小数桁数

次の表は、拡張モード (つまり、コンパイラー・オプション ARITH(EXTEND) が有効である場合) で加算、減算、乗算、または除算を伴う算術演算の固定小数点中間結果においてコンパイラーが 保持する桁数を示しています。

***i* + *d* の値**	***d* の値**	***i* + *dmax* の値**	***ir* 用に保持される桁数**
<31 または =31	任意の値	任意の値	*i* 整数桁数および *d* 小数桁数

i + *d* の値	*d* の値	*i* + *dmax* の値	*ir* 用に保持される桁数
>31	<*dmax* または =*dmax*	任意の値	31-*d* 整数桁数および *d* 小数桁数
	>*dmax*	<31 または =31	*i* 整数桁数および 31-*i* 小数桁数
		>31	31-*dmax* 整数桁数および *dmax* 小数桁数

指数

指数は、式 *op1* ** *op2* で表されます。 *op2* の特性に基づいて、コンパイラーは固定小数点数のべき乗計算を次の 3 つのいずれかの方法で処理します。

- *op2* が小数部で表されている場合は、浮動小数点命令が使用されます。
- *op2* が整数リテラルまたは定数である場合、値 *d* は次のように計算されます。

```
d = d1 * |op2|
```

また、値 *i* は、*op1* の特性に基づいて、次のように計算されます。

– *op1* がデータ名または変数である場合は、次のように計算されます。

```
i = i1 * |op2|
```

– *op1* がリテラルまたは定数である場合、*i* は、*op1* ** |*op2*| の値の整数の桁数に等しく設定されます。

互換モード (ARITH(COMPAT) を使用してコンパイルする場合) では、コンパイラーは *i* および *d* を計算し終えると、次の表に示すアクションを取り、べき乗計算の中間結果 *ir* を処理します。

i + *d* の値	その他の条件	取られるアクション
<30	任意	*ir* において、*i* の整数桁数および *d* の小数桁数が保持される。
=30	*op1* が奇数の桁数を持つ。	*ir* において、*i* の整数桁数および *d* の小数桁数が保持される。
	op1 が偶数の桁数を持つ。	*op2* が整数データ名または変数である場合と同じアクション (以下を参照)。 例外: リテラル 1 の累乗に計算される 30 桁の整数の場合は、*ir* において、*i* の整数桁数および *d* の小数桁数が保持される。
>30	任意	*op2* が整数データ名または変数である場合と同じアクション (以下を参照)

拡張モード (ARITH(EXTEND) を使用してコンパイルする場合) では、コンパイラーは *i* および *d* を計算し終えると、次の表に示すアクションを取り、指数計算の中間結果 *ir* を処理します。

i + *d* の値	その他の条件	取られるアクション
<31	任意	*ir* において、*i* の整数桁数および *d* の小数桁数が保持される。
=31 または >31	任意	*op2* が整数データ名または変数である場合と同じアクション (以下を参照)。 例外: リテラル 1 の累乗に計算される 31 桁の整数の場合は、*ir* において、*i* の整数桁数および *d* の小数桁数が保持される。

op2 が負である場合は、1 という値が予備計算によって作成された結果で除算されます。 使用される *i* と *d* の値は、上記に示された固定小数点データの除算規則に従って計算されます。

- *op2* が整数データ名または変数である場合は、*dmax* の小数桁数と、30-*dmax* (互換モード) または 31-*dmax* (拡張モード) 整数桁数が使用されます。 *op1* はそれ自体によって (|*op2*| - 1) 回、乗算されます (*op2* がゼロ以外の場合)。

  op2 が 0 であると、結果は 1 になります。0 による除算とべき乗計算の SIZE ERROR 条件が適用されます。

9 桁を超える有効数字を持つ固定小数点の指数部は、常に 9 桁に切り捨てられます。 指数がリテラルまたは定数である場合、E レベルのコンパイラー診断メッセージが発行されます。それ以外の場合は、実行時に通知メッセージが発行されます。

535 ページの『例: 固定小数点の算術での指数』

関連参照
532 ページの『中間結果用の用語』
535 ページの『中間結果での切り捨て』
536 ページの『バイナリー・データと中間結果』
538 ページの『浮動小数点データと中間結果』
536 ページの『固定小数点算術で評価される組み込み関数』
253 ページの『ARITH』
SIZE ERROR 句 (*COBOL for Linux on x86* 言語解説書)

例: 固定小数点の算術での指数

次の例は、コンパイラーが、必要に応じて中間結果を保管しながら、ゼロ以外の整数累乗へのべき乗計算を一連の乗算として実行する方法を示しています。

```
COMPUTE Y = A ** B
```

B が 4 であれば、結果は次のように計算されます。 使用される *i* と *d* の値は、固定小数点データおよび中間結果に関する乗算規則に従って計算されます (以下を参照してください)。

1. A に A を掛けると *ir1* が得られる。
2. *ir1* に A を掛けると *ir2* が得られる。
3. *ir2* に A を掛けると *ir3* が得られる。
4. *ir3* を *ir4* に移動する。

   ir4 は *dmax* の小数桁数を持っています。 B は正であるので、*ir4* は Y に移動されます。しかし B が -4 であった場合は、さらに 5 番目のステップが実行されます。
5. 1 を *ir4* で割ると *ir5* が得られる。

ir5 は *dmax* の小数桁数を持っており、Y に移動されます。

関連参照
532 ページの『中間結果用の用語』
533 ページの『固定小数点データと中間結果』

中間結果での切り捨て

中間結果において桁数が互換モードの 30、または拡張モードの 31 を超えるときは常に、コンパイラーは、30 (互換モード) または 31 (拡張モード) 桁までに 切り捨て、警告を表示します。 切り捨てが実行時に起こった場合は、メッセージが出され、プログラムは実行を続けます。

固定小数点計算で発生する可能性のある中間結果の切り捨てを回避したい場合には、代わりに浮動小数点オペランド (COMP-1 または COMP-2) を使用してください。

関連概念
39 ページの『数値データの形式』

関連参照
533 ページの『固定小数点データと中間結果』
253 ページの『ARITH』

バイナリー・データと中間結果

2 進法オペランドを伴う操作で 18 桁より多い中間結果が必要となる場合、コンパイラーはオペランドを内部 10 進数に変換してから操作を実行します。 結果フィールドが 2 進数である場合、コンパイラーは、結果を内部 10 進数から 2 進数に変換します。

2 進数オペランドが最も効率的に使用されるのは、中間結果が 9 桁を超えない場合です。

関連参照
533 ページの『固定小数点データと中間結果』
253 ページの『ARITH』

固定小数点算術で評価される組み込み関数

コンパイラーは、組み込み関数の *inner-dmax* 値および *outer-dmax* 値を、関数の特性から決定します。

整数関数

整数組み込み関数は整数を戻します。したがって、この関数の *outer-dmax* は常に 0 です。 引数がすべて整数でなければならない整数関数の場合は、*inner-dmax* も常に 0 になります。

次の表に、*inner-dmax* および関数結果の精度を要約します。

関数	*Inner-dmax*	関数結果の桁精度
DATE-OF-INTEGER	0	8
DATE-TO-YYYYMMDD	0	8
DAY-OF-INTEGER	0	7
DAY-TO-YYYYDDD	0	7
FACTORIAL	0	30 (互換モード)、31 (拡張モード)
INTEGER-OF-DATE	0	7
INTEGER-OF-DAY	0	7
LENGTH	n/a	9
MOD	0	min(*i1 i2*)
ORD	n/a	3
ORD-MAX		9
ORD-MIN		9
YEAR-TO-YYYY	0	4
INTEGER		固定小数点引数の場合: 引数より 1 桁大きくなります。 浮動小数点引数の場合: 30 (互換モード)、31 (拡張モード)
INTEGER-PART		固定小数点引数の場合: 引数と同じ桁数になります。 浮動小数点引数の場合: 30 (互換モード)、31 (拡張モード)

混合関数

混合 組み込み関数とは、結果の型がその引数の型に依存する関数です。 引数がすべて数値で、どの引数も浮動小数点でない場合、その混合関数は固定小数点です。 (混合関数のいずれかの引数が浮動小数点である場合、その関数は浮動小数点命令によって評価され、浮動小数点結果を戻します。) 混合関数が固定小数点算術演算で評価されたときは、引数がすべて整数であれば結果は整数になり、それ以外の場合は結果は固定小数点になります。

混合関数 MAX、MIN、RANGE、REM、および SUM の場合、*outer-dmax* は常に *inner-dmax* に等しくなります (したがって、引数がすべて整数であれば、両方とも 0 になります)。 これらの関数について戻される結果の精度を判別するためには、固定小数点算術演算および中間結果に関する規則 (以下を参照) を、アルゴリズムの各ステップに適用してください。

MAX

1. 最初の引数を関数結果に割り当てる。
2. 残りの引数ごとに、以下のステップを実行する。
   a. 関数結果の代数値を引数と比較する。
   b. 2 つの値のうちの大きな方を関数結果に割り当てる。

MIN

1. 最初の引数を関数結果に割り当てる。
2. 残りの引数ごとに、以下のステップを実行する。
   a. 関数結果の代数値を引数と比較する。
   b. 2 つの値のうちの小さな方を関数結果に割り当てる。

RANGE

1. MAX 用のステップを使用して、最大の引数を選択する。
2. MIN 用のステップを使用して、最小の引数を選択する。
3. 最大の引数から最小の引数を引く。
4. その差を関数結果に割り当てる。

REM

1. 引数 1 を引数 2 で割る。
2. ステップ 1 の結果からすべての非整数桁を除去する。
3. ステップ 2 の結果に引数 2 を掛ける。
4. ステップ 3 の結果を引数 1 から引く。
5. その差を関数結果に割り当てる。

SUM

1. 値 0 を関数結果に割り当てる。
2. 引数ごとに、以下のステップを実行する。
   a. その引数を関数結果に足す。
   b. その和を関数結果に割り当てる。

関連参照

532 ページの『中間結果用の用語』
533 ページの『固定小数点データと中間結果』
538 ページの『浮動小数点データと中間結果』
253 ページの『ARITH』

浮動小数点データと中間結果

算術式の演算が浮動小数点で計算される場合は、すべてのオペランドが浮動小数点に変換され、しかも浮動小数点命令を使用して演算が行われたかのように、式全体が計算されます。

算術式に関して以下の条件のいずれかが真である場合、浮動小数点命令を使用して算術式が計算されます。

- 受け取り側またはオペランドが COMP-1、COMP-2、外部浮動小数点、または浮動小数点リテラルである。
- 指数に小数位が含まれている。
- 指数が、べき乗計算または除算演算子を含む式であり、かつ *dmax* が 0 より大きい。
- 組み込み関数が浮動小数点関数である。

互換モードでは、式が浮動小数点算術演算で計算される場合、算術演算を評価するために使用される精度は、次のように決まります。

- 受け取り側およびオペランドがすべて COMP-1 データ項目で、式に乗算またはべき乗演算が含まれていない場合には、単精度が使用されます。
- これ以外の場合には、長精度が使用されます。

長精度浮動小数点が算術式の 1 つの演算で使用された場合は、その式のすべての演算は、長精度浮動小数点命令が使用されたかのように計算されます。

拡張モードでは、式が浮動小数点算術演算で計算される場合、算術演算を評価するために使用される精度は、次のように決まります。

- 受け取り側およびオペランドがすべて COMP-1 データ項目で、式に乗算またはべき乗演算が含まれていない場合には、単精度が使用されます。
- 受け取り側およびオペランドがすべて COMP-1 または COMP-2 データ項目で、受け取り側またはオペランドの少なくとも 1 つが COMP-2 データ項目であり、かつ式に乗算またはべき乗演算が含まれていない場合には、長精度が使用されます。
- これ以外の場合には、拡張精度が使用されます。

拡張精度浮動小数点が算術式の 1 つの演算で使用された場合は、その式のすべての演算は、拡張精度浮動小数点命令が使用されたかのように計算されます。

注意: 浮動小数点演算で、指数オーバーフローが発生した中間結果フィールドがあると、ジョブは異常終了します。

浮動小数点演算で評価される指数

互換モードでは、浮動小数点の指数は、常に長浮動小数点演算を使用して評価されます。 拡張モードでは、浮動小数点べき乗計算は常に、拡張精度浮動小数点算術演算を使用して評価されます。

COBOL では、負の数を小数で累乗した値は定義されていません。 例えば、(-2) ** 3 は -8 ですが、(-2) ** (3.000001) は定義されていません。 べき乗計算が浮動小数点で行われ、結果が未定義である可能性がある場合には、実行時に指数の値が評価され、それが実際に整数値を持つかどうか判別されます。 整数値を持っていない場合には、診断メッセージが出されます。

浮動小数点演算で評価される組み込み関数

互換モードでは、浮動小数点組み込み関数は常に長精度 (64 ビット) 浮動小数点値を戻します。 拡張モードでは、浮動小数点組み込み関数は常に、拡張精度 (128 ビット) の浮動小数点値を戻します。

少なくとも 1 つの浮動小数点引数を持つ混合関数は、浮動小数点算術演算を使用して評価されます。

関連参照

532 ページの『中間結果用の用語』
253 ページの『ARITH』

非算術ステートメントの算術式

算術式は、算術ステートメント以外のコンテキストでも使用できます。 例えば、IF または EVALUATE ステートメントを持つ算術式を使用することができます。

このようなステートメントでは、固定小数点データを持つ中間結果および浮動小数点データを持つ中間結果に関する規則が適用されます。ただし、次のような変更があります。

- 省略された IF ステートメントは、省略されていないかのように処理されます。
- 被比較数の少なくとも 1 つが算術式であるような明示比較条件では、*dmax* は、いずれかの被比較数の任意のオペランド (除数と指数を除く) 用に定義された小数部の最大小数桁数になります。 以下のいずれかの条件が真である場合は、浮動小数点算術演算の規則が適用されます。
  - いずれかの被比較数のオペランドが COMP-1、COMP-2、外部浮動小数点、または浮動小数点リテラルである。
  - 指数に小数位が含まれている。
  - 指数が、べき乗計算または除算演算子を含む式であり、かつ *dmax* が 0 より大きい。

  次に例を示します。

```
IF operand-1 = expression-1 THEN . . .
```

  operand-1 が COMP-2 と定義されるデータ名である場合は、*expression-1* には、たとえ固定小数点オペランドしか含まれていなくても、浮動小数点算術演算の規則が適用されます。というのは、これは固定小数点オペランドと比較されるためです。
- ある算術式と別のデータ項目または算術式との間の比較で、関係演算子が使用されない場合 (すなわち、明示的な比較条件がない場合) は、その算術式は、被比較数の属性を考慮に入れずに評価されます。 以下に例を示します。

```
EVALUATE expression-1
  WHEN expression-2 THRU expression-3
  WHEN expression-4
  . . .
END-EVALUATE
```

  上記のステートメントでは、それぞれの算術式は、その特性に応じて、固定小数点または浮動小数点算術で評価されます。

関連概念
53 ページの『固定小数点演算と浮動小数点演算の対比』

関連参照
532 ページの『中間結果用の用語』
533 ページの『固定小数点データと中間結果』
538 ページの『浮動小数点データと中間結果』
IF ステートメント (*COBOL for Linux on x86* 言語解説書)
EVALUATE ステートメント (*COBOL for Linux on x86* 言語解説書)
条件式 (*COBOL for Linux on x86* 言語解説書)

付録 D 日時呼び出し可能サービス

日時の呼び出し可能サービスを使用すると、現行の現地時間および日付をいくつかの形式で入手し、日時の変換を行うことができます。

使用可能な日時呼び出し可能サービスを以下に示します。 2 つのサービス CEEQCEN と CEESCEN は、2 桁の年号 (例えば、1991 を表す 91、2010 を表す 10 など) を扱う予測可能な方法を提供します。

表 *58.* **日時呼び出し可能サービス**

呼び出し可能サービス	**説明**
542 ページの『CEECBLDY: 日付から COBOL 整数形式への変換』	文字日付値を COBOL 整数日付形式に変換します。 1 日目は 1601 年 1 月 1 日で、その後 1 日ごとに値が 1 ずつ増えます。
546 ページの『CEEDATE: リリアン日付から文字形式への変換』	リリアン形式の日付を文字値に変換します。
549 ページの『CEEDATM: 秒から文字タイム・スタンプへの変換』	秒数を文字タイム・スタンプに変換します。
553 ページの『CEEDAYS: 日付からリリアン形式への変換』	文字日付値をリリアン形式に変換します。 1 日目は 1582 年 10 月 15 日で、その後 1 日ごとに値が 1 ずつ増えます。
556 ページの『CEEDYWK: リリアン日付からの曜日の計算』	曜日計算を行います。
558 ページの『CEEGMT: 現在のグリニッジ標準時の取得』	現在のグリニッジ標準時 (日時) を取得します。
560 ページの『CEEGMTO: グリニッジ標準時から現地時間までのオフセットの取得』	グリニッジ標準時と現地時間の時差を取得します。
562 ページの『CEEISEC: 整数から秒への変換』	2 進数の年、月、日、時、分、秒、ミリ秒を、1582 年 10 月 15 日の 00:00:00 から数えた秒数を表す数値に変換します。
564 ページの『CEELOCT: 現在の現地日時の取得』	現在の日時を取得します。
566 ページの『CEEQCEN: 世紀ウィンドウの照会』	呼び出し可能サービスの世紀ウィンドウを照会します。
567 ページの『CEESCEN: 世紀ウィンドウの設定』	呼び出し可能サービスの世紀ウィンドウを設定します。
568 ページの『CEESECI: 秒から整数への変換』	1582 年 10 月 15 日の 00:00:00 から数えた秒数を表す数値を、年、月、日、時、分、秒、ミリ秒を表す 7 つの 2 進整数に変換します。
571 ページの『CEESECS: タイム・スタンプから秒への変換』	文字タイム・スタンプ (日時) を、1582 年 10 月 15 日の 00:00:00 から数えた秒数に変換します。
575 ページの『CEEUTC: 協定世界時の取得』	CEEGMT と同じ。
575 ページの『IGZEDT4: 現在日付の取得』	4 桁年号を使用した現在日付を YYYMMDD 形式で戻します。

これらすべての日時呼び出し可能サービスで、Enterprise COBOL for z/OS との ソース・コード互換性があります。 ただし、条件の処理方法には大きな違いがあります。

日時の呼び出し可能サービスは、以下に表す日付/時刻組み込み関数への追加です。

© Copyright IBM Corp. 2021

表 59. **日時組み込み関数**	
組み込み関数	**説明**
CURRENT-DATE	現在の日時とグリニッジ標準時からの時間差
DATE-OF-INTEGER[1]	整数で表された日付に相当する標準フォーマットの日付 (YYYYMMDD)
DATE-TO-YYYYMMDD[1]	指定された 100 年間隔に従ったウィンドウ化西暦年を使用した、整数表現の日付に相当する標準フォーマットの日付 (YYYYMMDD)
DATEVAL[1]	整数または英数字で表された日付に相当する日付フィールド
DAY-OF-INTEGER[1]	整数で表された日付に相当する年間通算日フォーマットの日付 (YYYYDDD)
DAY-TO-YYYYDDD[1]	指定された 100 年間隔に従ったウィンドウ化西暦年を使用した、整数表現の日付に相当するユリウス日付 (YYYYMMDD)
INTEGER-OF-DATE	標準フォーマットの日付 (YYYYMMDD) に相当する整数で表された日付
INTEGER-OF-DAY	年間通算日 (YYYYDDD) に相当する整数で表された日付
UNDATE[1]	整数または英数字で表された日付フィールドに相当する非日付
YEAR-TO-YYYY[1]	指定された 100 年間隔に従ったウィンドウ化西暦年に相当する拡張西暦年 (YYYY)
YEARWINDOW[1]	YEARWINDOW コンパイラー・オプションで指定された世紀ウィンドウの開始年
1. DATEPROC コンパイラー・オプションの設定によって動作が異なります。	

512 ページの『例: 出力用の日付形式』

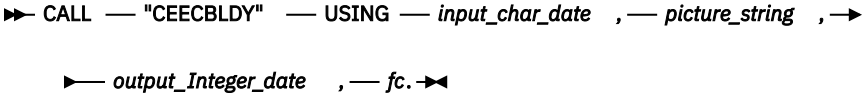
関連参照
513 ページの『フィードバック・トークン』
CALL ステートメント (*COBOL for Linux on x86* 言語解説書)
関数定義 (*COBOL for Linux on x86* 言語解説書)

CEECBLDY: 日付から COBOL 整数形式への変換

CEECBLDY は、日付を表すストリングを 1600 年 12 月 31 日から数えた日数に変換します。 日時の呼び出し可能サービスの世紀ウィンドウにアクセスする場合や、COBOL 組み込み関数を使用して日付計算を実行する場合には、CEECBLDY を使用します。

このサービスは CEEDAYS と似ていますが、COBOL 組み込み関数と互換性のある COBOL 整数形式でストリングを戻す点が異なります。

CALL CEECBLDY の構文



***input_char_date* (入力)**
picture_string の指定に準拠した形式で日付またはタイム・スタンプを表す、ハーフワード長の接頭部の付いた文字ストリング。

文字ストリングに含められる文字数は 5 から 255 文字です。 *input_char_date* には、先行または末尾ブランクを含めることができます。 日付の構文解析は、最初の非ブランク文字から始まります (ピクチャー・ストリング自体に先行ブランクが含まれる場合は、CEECBLDY がその位置を正確にスキップした後、構文解析が始まります)。

CEECBLDY は、*picture_string* で指定された日付形式によって判別される有効な日付を解析したら、残りの文字をすべて無視します。 有効な日付範囲は、1601 年 1 月 1 日から 9999 年 12 月 31 日です。

***picture_string* (入力)**
input_char_date で指定された日付の形式を示す、ハーフワード長の接頭部の付いた文字ストリング。

picture_string 内の各文字は、*input_char_date* 内の文字に対応します。 例えば、MMDDYY を *picture_string* として指定すると、CEECBLDY は *input_char_date* の値 060288 を 1988 年 6 月 2 日として読み取ります。

スラッシュ (/) などの区切り文字がピクチャー・ストリング内にある場合は、先行ゼロを省略することができます。 例えば、次の CEECBLDY の呼び出しは、同じ値 141502 (1988 年 6 月 2 日) をそれぞれ COBINTDTE に割り当てます。

```
MOVE '6/2/88' TO DATEVAL-STRING.
MOVE 6 TO DATEVAL-LENGTH.
MOVE 'MM/DD/YY' TO PICSTR-STRING.
MOVE 8 TO PICSTR-LENGTH.
CALL CEECBLDY USING DATEVAL, PICSTR, COBINTDTE, FC.
```

```
MOVE '06/02/88' TO DATEVAL-STRING.
MOVE 8 TO DATEVAL-LENGTH.
MOVE 'MM/DD/YY' TO PICSTR-STRING.
MOVE 8 TO PICSTR-LENGTH.
CALL CEECBLDY USING DATEVAL, PICSTR, COBINTDTE, FC.
```

```
MOVE '060288' TO DATEVAL-STRING.
MOVE 6 TO DATEVAL-LENGTH.
MOVE 'MMDDYY' TO PICSTR-STRING.
MOVE 6 TO PICSTR-LENGTH.
CALL CEECBLDY USING DATEVAL, PICSTR, COBINTDTE, FC.
```

```
MOVE '88154' TO DATEVAL-STRING.
MOVE 5 TO DATEVAL-LENGTH.
MOVE 'YYDDD' TO PICSTR-STRING.
MOVE 5 TO PICSTR-LENGTH.
CALL CEECBLDY USING DATEVAL, PICSTR, COBINTDTE, FC.
```

picture_string にコロンやスラッシュなどの文字 (例: HH:MI:SS YY/MM/DD) が含まれる場合はプレースホルダーと見なされますが、それ以外の場合は無視されます。

picture_string に日本元号のシンボル <JJJJ> が含まれる場合は、*input_char_date* の YY の位置に、日本元号での年号が入ります。 例えば、1988 年は日本の昭和 63 年に相当します。

***output_Integer_date* (出力)**
COBOL 整数日付 (1600 年 12 月 31 日から数えた日数) を表す 32 ビットの 2 進整数。 例えば、1988 年 5 月 16 日は、日数 141485 に相当します。

input_char_date に有効な日付が含まれていない場合は、*output_Integer_date* が 0 に設定され、CEECBLDY が終了して非 CEE000 シンボリック・フィードバック・コードが戻されます。

output_Integer_date は整数なので、*output_Integer_date* では日付計算を容易に行うことができます。うるう年や年末偏差は計算に影響しません。

***fc* (出力)**

このサービスの結果を示す 12 バイトの フィードバック・コード (オプション)。

表 *60.* ***CEECBLDY* のシンボリック条件**

シンボリック・フィードバック・コード	重大度	メッセージ番号	メッセージ・テキスト
CEE000	0	--	サービスが正しく完了しました。
CEE2EB	3	2507	CEEDAYS または CEESECS に渡されたデータが不十分です。リリアン日付の値は計算されませんでした。
CEE2EC	3	2508	CEEDAYS または CEESECS に渡された日付値が無効です。
CEE2ED	3	2509	CEEDAYS または CEESECS に渡された元号が認識されませんでした。
CEE2EH	3	2513	CEEISEC、CEEDAYS、CEESECS のいずれかの呼び出しで渡された入力日付が、対応範囲内にありませんでした。
CEE2EL	3	2517	CEEISEC 呼び出し内の月の値が認識されませんでした。
CEE2EM	3	2518	日時サービスへの呼び出しに無効なピクチャー・ストリングが指定されました。
CEE2EO	3	2520	CEEDAYS が数値フィールド内に非数値データを検出したか、あるいは日付ストリングとピクチャー・ストリングが一致しませんでした。
CEE2EP	3	2521	CEEDAYS または CEESECS に渡された <JJJJ>、<CCCC>、または <CCCCCCCC> の元号年数値がゼロでした。

使用上の注意

- CEECBLDY の呼び出しは必ず、戻り値を COBOL 組み込み関数への入力として使用する COBOL プログラムから行ってください。 CEEDAYS とは異なり、CEECBLDY の逆関数はありません。CEECBLDY は、COBOL ユーザーが日時の世紀ウィンドウ・サービスを COBOL 組み込み関数とともに使用して、日付計算を行う場合にのみ使用されます。 CEECBLDY の逆は、DATE-OF-INTEGER および DAY-OF-INTEGER 組み込み関数によって実現されます。
- 1601 年 1 月 1 日よりも前の日付に対して計算を実行するには、各日付の年号に 4000 を加算し、その日付を COBOL 整数形式に変換してから計算します。 計算結果が日数ではなく日付になる場合は、計算結果を日付ストリングに変換し、年号から 4000 を減算します。
- デフォルトでは、2 桁の年号は、システム日付より 80 年前から始まる 100 年間にあります。 したがって、2010 年の場合、2 桁の年号はすべて 1930 から 2029 年の範囲内の日付を表します。 このデフォルトの範囲を変更するには、CEESCEN 呼び出し可能サービスを使用します。

例

```
      ************************************************
      **                                            **
      ** Function: Invoke CEECBLDY callable service  **
      ** to convert date to COBOL integer format.    **
      ** This service is used when using the         **
      ** Century Window feature of the date and time **
      ** callable services mixed with COBOL          **
      ** intrinsic functions.                        **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLDY.
      *
       DATA DIVISION.
       WORKING-STORAGE SECTION.
```

```
       01  CHRDATE.
           02  Vstring-length      PIC S9(4) BINARY.
           02  Vstring-text.
               03  Vstring-char    PIC X
                           OCCURS 0 TO 256 TIMES
                           DEPENDING ON Vstring-length
                               of CHRDATE.
       01  PICSTR.
           02  Vstring-length      PIC S9(4) BINARY.
           02  Vstring-text.
               03  Vstring-char    PIC X
                           OCCURS 0 TO 256 TIMES
                           DEPENDING ON Vstring-length
                               of  PICSTR.
       01  INTEGER                 PIC S9(9) BINARY.
       01  NEWDATE                 PIC 9(8).
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) COMP.
                   04  Msg-No      PIC S9(4) COMP.
               03  Case-2-Condition-ID
                       REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) COMP.
                   04  Cause-Code  PIC S9(4) COMP.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) COMP.
      *
       PROCEDURE DIVISION.
       PARA-CBLDAYS.
      ************************************************
      ** Specify input date and length               **
      ************************************************
           MOVE 25 TO Vstring-length of CHRDATE.
           MOVE '1 January 00'
               to Vstring-text of CHRDATE.
      ************************************************
      ** Specify a picture string that describes     **
      ** input date, and set the string's length.    **
      ************************************************
           MOVE 23 TO Vstring-length of PICSTR.
           MOVE 'ZD Mmmmmmmmmmmmmmz YY'
                     TO Vstring-text of PICSTR.
      ************************************************
      ** Call CEECBLDY to convert input date to a    **
      ** COBOL integer date                          **
      ************************************************
           CALL 'CEECBLDY' USING CHRDATE, PICSTR,
                                 INTEGER, FC.
      ************************************************
      ** If CEECBLDY runs successfully, then compute **
      **     the date of the 90th day after the      **
      **     input date using Intrinsic Functions    **
      ************************************************
           IF CEE000 of FC  THEN
               COMPUTE INTEGER = INTEGER + 90
               COMPUTE NEWDATE = FUNCTION
                   DATE-OF-INTEGER (INTEGER)
               DISPLAY NEWDATE
                   ' is Lilian day: ' INTEGER
           ELSE
               DISPLAY 'CEEBLDY failed with msg '
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.
      *
           GOBACK.
```

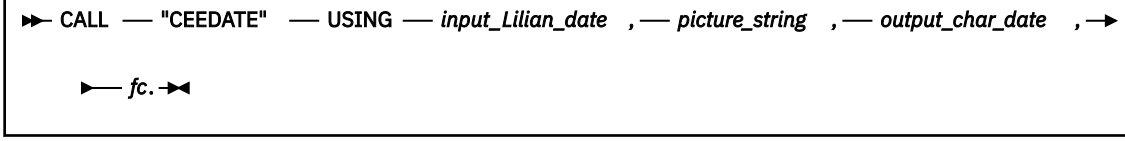
関連参照

514 ページの『ピクチャー文字項およびストリング』

CEEDATE: リリアン日付から文字形式への変換

CEEDATE は、リリアン日付を表す数値を文字形式の日付に変換します。 出力は、2010/04/23 などの文字ストリングになります。

CALL CEEDATE の構文



***input_Lilian_date* (入力)**
リリアン日付を表す 32 ビットの整数。 このリリアン日付は、1582 年 10 月 14 日から数えた日数です。 例えば、1988 年 5 月 16 日は、リリアン日数 148138 に相当します。 有効なリリアン日付の範囲は 1 から 3,074,324 (1582 年 10 月 15 日から 9999 年 12 月 31 日) です。

***picture_string* (入力)**
output_char_date の必須形式 (例: MM/DD/YY) を表す ハーフワード長の接頭部付きの文字ストリング。 *picture_string* 内の各文字は、*output_char_date* 内の文字を表します。 スラッシュ (/) などの区切り文字がピクチャー・ストリング内にある場合は、現状のまま *output_char_date* にコピーされます。

picture_string に日本元号のシンボル <JJJJ> が含まれる場合は、*output_char_date* の YY の位置に、日本元号での年号が入ります。 例えば、1988 年は日本の昭和 63 年に相当します。

***output_char_date* (出力)**
input_Lilian_date を *picture_string* で指定された形式に変換した結果として生成される、固定長の 80 文字のストリング。 *input_Lilian_date* が無効な場合は、*output_char_date* がすべてブランクに設定され、CEEDATE が終了して非 CEE000 シンボリック・フィードバック・コードが戻されます。

***fc* (出力)**
このサービスの結果を示す 12 バイトの フィードバック・コード (オプション)。

表 *61.* ***CEEDATE* のシンボリック条件**

シンボリック・フィードバック・コード	重大度	メッセージ番号	メッセージ・テキスト
CEE000	0	--	サービスが正しく完了しました。
CEE2EG	3	2512	CEEDATE または CEEDYWK への呼び出しで渡されたリリアン日付値が、対応範囲内にありませんでした。
CEE2EM	3	2518	日時サービスへの呼び出しに無効なピクチャー・ストリングが指定されました。
CEE2EQ	3	2522	CEEDATE に渡されたピクチャー・ストリング内に元号 (<JJJJ>、<CCCC>、または <CCCCCCCC>) が使用されていましたが、リリアン日付値が対応範囲内にありませんでした。元号を判別できませんでした。
CEE2EU	2	2526	CEEDATE によって戻された日付ストリングが切り捨てられました。
CEE2F6	1	2534	CEEDATE または CEEDATM への呼び出しで、月または曜日名に対して指定されたフィールド幅が不十分です。 出力はブランクに設定されました。

使用上の注意: CEEDATE の逆は CEEDAYS です。CEEDAYS は文字日付をリリアン形式に変換します。

例

```
*************************************************
**                                             **
** Function: CEEDATE - convert Lilian date to **
**                     character format        **
**                                             **
** In this example, a call is made to CEEDATE **
** to convert a Lilian date (the number of    **
** days since 14 October 1582) to a character **
** format (such as 6/22/98). The result is    **
** displayed.  The Lilian date is obtained    **
** via a call to CEEDAYS.                      **
**                                             **
*************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID. CBLDATE.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  LILIAN                  PIC S9(9) BINARY.
 01  CHRDATE                 PIC X(80).
 01  IN-DATE.
     02  Vstring-length      PIC S9(4) BINARY.
     02  Vstring-text.
         03  Vstring-char    PIC X
                     OCCURS 0 TO 256 TIMES
                     DEPENDING ON Vstring-length
                         of IN-DATE.
 01  PICSTR.
     02  Vstring-length      PIC S9(4) BINARY.
     02  Vstring-text.
         03  Vstring-char    PIC X
                     OCCURS 0 TO 256 TIMES
                     DEPENDING ON Vstring-length
                        of PICSTR.
 01  FC.
     02  Condition-Token-Value.
     COPY  CEEIGZCT.
         03  Case-1-Condition-ID.
             04  Severity    PIC S9(4) COMP.
             04  Msg-No      PIC S9(4) COMP.
         03  Case-2-Condition-ID
                   REDEFINES Case-1-Condition-ID.
             04  Class-Code  PIC S9(4) COMP.
             04  Cause-Code  PIC S9(4) COMP.
         03  Case-Sev-Ctl    PIC X.
         03  Facility-ID     PIC XXX.
     02  I-S-Info            PIC S9(9) COMP.
*
 PROCEDURE DIVISION.
 PARA-CBLDAYS.
************************************************
** Call CEEDAYS to convert date of 6/2/98 to   **
**     Lilian representation                    **
************************************************
     MOVE 6 TO Vstring-length of IN-DATE.
     MOVE '6/2/98' TO Vstring-text of IN-DATE(1:6).
     MOVE 8 TO Vstring-length of PICSTR.
     MOVE 'MM/DD/YY' TO Vstring-text of PICSTR(1:8).
     CALL 'CEEDAYS' USING IN-DATE, PICSTR,
                          LILIAN, FC.

************************************************
** If CEEDAYS runs successfully, display result**
************************************************
     IF  CEE000 of FC  THEN
         DISPLAY Vstring-text of IN-DATE
             ' is Lilian day: ' LILIAN
     ELSE
         DISPLAY 'CEEDAYS failed with msg '
             Msg-No of FC UPON CONSOLE
         STOP RUN
     END-IF.

************************************************
** Specify picture string that describes the   **
**  required format of the output from CEEDATE, **
**  and the picture string's length.           **
************************************************
     MOVE 23 TO Vstring-length OF PICSTR.
```

```
           MOVE 'ZD Mmmmmmmmmmmmmmz YYYY' TO
                        Vstring-text OF PICSTR(1:23).

      ************************************************
      ** Call CEEDATE to convert the Lilian date     **
      **     to  a picture string.                    **
      ************************************************
           CALL 'CEEDATE' USING LILIAN, PICSTR,
                                CHRDATE, FC.

      ************************************************
      ** If CEEDATE runs successfully, display result**
      ************************************************
           IF CEE000 of FC  THEN
               DISPLAY 'Input Lilian date of ' LILIAN
                   ' corresponds to:  ' CHRDATE
           ELSE
               DISPLAY 'CEEDATE failed with msg '
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.

           GOBACK.
```

次の表に、CEEDATE からの出力例を示します。

input_Lilian_date	picture_string	output_char_date
148138	YY	98
	YYMM	9805
	YY-MM	98-05
	YYMMDD	980516
	YYYYMMDD	19980516
	YYYY-MM-DD	1998-05-16
	YYYY-ZM-ZD	1998-5-16
	<JJJJ> YY.MM.DD	昭和 63.05.16 (DBCS ストリング)
148139	MM	05
	MMDD	0517
	MM/DD	05/17
	MMDDYY	051798
	MM/DD/YYYY	05/17/1998
	ZM/DD/YYYY	5/17/1998
148140	DD	18
	DDMM	1805
	DDMMYY	180598
	DD.MM.YY	18.05.98
	DD.MM.YYYY	18.05.1998
	DD Mmm YYYY	18 May 1998

input_Lilian_date	picture_string	output_char_date
148141	DDD	140
	YYDDD	98140
	YY.DDD	98.140
	YYYY.DDD	1998.140
148142	YY/MM/DD HH:MI:SS.99	98/05/20 00:00:00.00
	YYYY/ZM/ZD ZH:MI AP	1998/5/20 0:00 AM
148143	WWW., MMM DD, YYYY	SAT., MAY 21, 1998
	Www., Mmm DD, YYYY	Sat., May 21, 1998
	Wwwwwwwwww, Mmmmmmmmm DD, YYYY	Saturday, May 21, 1998
	Wwwwwwwwwz, Mmmmmmmmmz DD, YYYY	Saturday, May 21, 1998

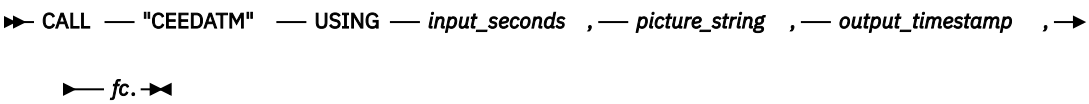
516 ページの『例: 日時のピクチャー・ストリング』

関連参照
514 ページの『ピクチャー文字項およびストリング』

CEEDATM: 秒から文字タイム・スタンプへの変換

CEEDATM は、1582 年 10 月 14 日の 00:00:00 から数えた秒数を表す数値を、文字ストリングに変換します。 出力は、1988/07/26 20:37:00 などの文字ストリングのタイム・スタンプになります。

CALL CEEDATM の構文



***input_seconds* (入力)**
1582 年 10 月 14 日の 00:00:00 から数えた (うるう秒は数えない) 秒数を表す、64 ビット長の浮動小数点数。

例えば、1582 年 10 月 15 日の 00:00:01 は秒数 86,401 (24*60*60 + 01) に相当します。 *input_seconds* の有効範囲は 86,400 から 265,621,679,999.999 (9999 年 12 月 31 日の 23:59:59.999) です。

***picture_string* (入力)**
output_timestamp の必須形式 (例: MM/DD/YY HH:MI AP) を表す ハーフワード長の接頭部付きの文字ストリング。

picture_string 内の各文字は、*output_timestamp* 内の文字を表します。 スラッシュ (/) などの区切り文字がピクチャー・ストリング内で使用されている場合は、現状のまま *output_timestamp* にコピーされます。

picture_string に日本元号のシンボル <JJJJ> が含まれる場合は、*output_timestamp* の YY の位置に、日本元号での年号が入ります。

***output_timestamp* (出力)**
input_seconds を *picture_string* で指定された形式に変換した結果として生成される、固定長の 80 文字のストリング。

必要に応じて、出力が *output_timestamp* の長さまで切り詰められます。

input_seconds が無効な場合は、*output_timestamp* がすべてブランクに設定され、CEEDATM が終了して非 CEE000 シンボリック・フィードバック・コードが戻されます。

***fc* (出力)**
このサービスの結果を示す 12 バイトの フィードバック・コード (オプション)。

表 *62.* ***CEEDATM* のシンボリック条件**

シンボリック・フィードバック・コード	重大度	メッセージ番号	メッセージ・テキスト
CEE000	0	--	サービスが正しく完了しました。
CEE2E9	3	2505	CEEDATM または CEESECI への呼び出し内の input_seconds 値が、対応範囲内にありませんでした。
CEE2EA	3	2506	CEEDATM に渡されたピクチャー・ストリング内に元号 (<JJJJ>、<CCCC>、または <CCCCCCCC>) が 使用されていましたが、入力された秒数値が対応範囲内にありませんでした。元号を判別できませんでした。
CEE2EM	3	2518	日時サービスへの呼び出しに無効なピクチャー・ストリングが指定されました。
CEE2EV	2	2527	CEEDATM によって戻されたタイム・スタンプ・ストリングが切り捨てられました。
CEE2F6	1	2534	CEEDATE または CEEDATM への呼び出しで、月または曜日名に対して指定されたフィールド幅が不十分です。 出力はブランクに設定されました。

使用上の注意: CEEDATM の逆は CEESECS です。CEESECS は、タイム・スタンプを秒数に変換します。

例

```
      ************************************************
      **                                            **
      ** Function: CEEDATM - convert seconds to     **
      **                     character time stamp   **
      **                                            **
      ** In this example, a call is made to CEEDATM **
      ** to convert a date represented in Lilian    **
      ** seconds (the number of seconds since       **
      ** 00:00:00 14 October 1582) to a character   **
      ** format (such as 06/02/88 10:23:45). The    **
      ** result is displayed.                       **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLDATM.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  DEST          PIC S9(9) BINARY VALUE 2.
       01  SECONDS                 COMP-2.
       01  IN-DATE.
           02  Vstring-length      PIC S9(4) BINARY.
           02  Vstring-text.
               03  Vstring-char    PIC X
                           OCCURS 0 TO 256 TIMES
                           DEPENDING ON Vstring-length
                               of IN-DATE.
       01  PICSTR.
           02  Vstring-length      PIC S9(4) BINARY.
           02  Vstring-text.
               03  Vstring-char    PIC X
                           OCCURS 0 TO 256 TIMES
                           DEPENDING ON Vstring-length
                              of PICSTR.
       01  TIMESTP                 PIC X(80).
```

```
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) COMP.
                   04  Msg-No      PIC S9(4) COMP.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) COMP.
                   04  Cause-Code  PIC S9(4) COMP.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) COMP.
      *
       PROCEDURE DIVISION.
       PARA-CBLDATM.
      ************************************************
      ** Call CEESECS to convert time stamp of 6/2/88**
      **     at 10:23:45 AM to Lilian representation **
      ************************************************
           MOVE 20 TO Vstring-length of IN-DATE.
           MOVE '06/02/88 10:23:45 AM'
               TO Vstring-text of IN-DATE.
           MOVE 20 TO Vstring-length of PICSTR.
           MOVE 'MM/DD/YY HH:MI:SS AP'
               TO Vstring-text of PICSTR.
           CALL 'CEESECS' USING IN-DATE, PICSTR,
                                SECONDS, FC.

      ************************************************
      ** If CEESECS runs successfully, display result**
      ************************************************
           IF  CEE000 of FC  THEN
               DISPLAY Vstring-text of IN-DATE
                   ' is Lilian second:  ' SECONDS
           ELSE
               DISPLAY 'CEESECS failed with msg '
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.

      ************************************************
      ** Specify required format of the output.       **
      ************************************************
           MOVE 35 TO Vstring-length OF PICSTR.
           MOVE 'ZD Mmmmmmmmmmmmmmz YYYY at HH:MI:SS'
                   TO Vstring-text OF PICSTR.

      ************************************************
      ** Call CEEDATM to convert Lilian seconds to   **
      **     a character time stamp                   **
      ************************************************
           CALL 'CEEDATM' USING SECONDS, PICSTR,
                                TIMESTP, FC.

      ************************************************
      ** If CEEDATM runs successfully, display result**
      ************************************************
           IF CEE000 of FC  THEN
               DISPLAY 'Input seconds of ' SECONDS
                   ' corresponds to: ' TIMESTP
           ELSE
               DISPLAY 'CEEDATM failed with msg '
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.

           GOBACK.
```

次の表に、CEEDATM からの出力例を示します。

input_seconds	picture_string	output_timestamp
12,799,191,601.000	YYMMDD	880516
	HH:MI:SS	19:00:01
	YY-MM-DD	88-05-16
	YYMMDDHHMISS	880516190001
	YY-MM-DD HH:MI:SS	88-05-16 19:00:01
	YYYY-MM-DD HH:MI:SS AP	1988-05-16 07:00:01 PM
12,799,191,661.986	DD Mmm YY	16 May 88
	DD MMM YY HH:MM	16 MAY 88 19:01
	WWW, MMM DD, YYYY	MON, MAY 16, 1988
	ZH:MI AP	7:01 PM
	Wwwwwwwwwz, ZM/ZD/YY	Monday, 5/16/88
	HH:MI:SS.99	19:01:01.98
12,799,191,662.009	YYYY	1988
	YY	88
	Y	8
	MM	05
	ZM	5
	RRRR	V

input_seconds	picture_string	output_timestamp
12,799,191,662.009	MMM	MAY
	Mmm	May
	Mmmmmmmmmm	May
	Mmmmmmmmmz	May
	DD	16
	ZD	16
	DDD	137
	HH	19
	ZH	19
	MI	01
	SS	02
	99	00
	999	009
	AP	PM
	WWW	MON
	Www	Mon
	Wwwwwwwwww	Monday
	Wwwwwwwwwz	Monday

516 ページの『例: 日時のピクチャー・ストリング』

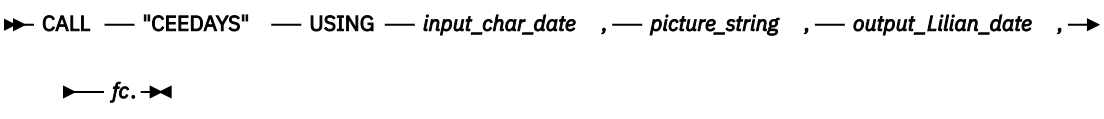
関連参照
514 ページの『ピクチャー文字項およびストリング』

CEEDAYS: 日付からリリアン形式への変換

CEEDAYS は、日付を表すストリングをリリアン形式に変換します。リリアン形式では、グレゴリオ暦の開始日 (1582 年 10 月 14 日 金曜日) から数えた日数として日付を表します。

CEEDAYS を COBOL 組み込み関数と併用しないでください。 CEECBLDY は、組み込み関数を使用するプログラムに使用します。

CALL CEEDAYS の構文



***input_char_date* (入力)**
picture_string の指定に準拠した形式で日付またはタイム・スタンプを表す、ハーフワード長の接頭部の付いた文字ストリング。

文字ストリングに含められる文字数は 5 から 255 文字です。 *input_char_date* には、先行または末尾ブランクを含めることができます。 日付の構文解析は、最初の非ブランク文字から始まります (ピクチャー・ストリング自体に先行ブランクが含まれる場合は、CEEDAYS がその位置を正確にスキップした後、構文解析が始まります)。

CEEDAYS は、*picture_string* で指定された日付形式によって判別される有効な日付を解析したら、残りの文字をすべて無視します。 有効な日付範囲は、1582 年 10 月 15 日から 9999 年 12 月 31 日です。

***picture_string* (入力)**
input_char_date で指定された日付の形式を示す、ハーフワード長の接頭部の付いた文字ストリング。

picture_string 内の各文字は、*input_char_date* 内の文字に対応します。 例えば、MMDDYY を *picture_string* として指定すると、CEEDAYS は *input_char_date* の値 060288 を 1988 年 6 月 2 日として読み取ります。

スラッシュ (/) などの区切り文字がピクチャー・ストリング内にある場合は、先行ゼロを省略することができます。 例えば、次の CEEDAYS の呼び出しは、同じ値 148155 (1988 年 6 月 2 日) をそれぞれ *lildate* に割り当てます。

```
CALL CEEDAYS USING '6/2/88'   , 'MM/DD/YY', lildate, fc.
CALL CEEDAYS USING '06/02/88', 'MM/DD/YY', lildate, fc.
CALL CEEDAYS USING '060288'   , 'MMDDYY'   , lildate, fc.
CALL CEEDAYS USING '88154'    , 'YYDDD'    , lildate, fc.
```

picture_string にコロンやスラッシュなどの文字 (例: HH:MI:SS YY/MM/DD) が含まれる場合はプレースホルダーと見なされますが、それ以外の場合は無視されます。

picture_string に日本元号のシンボル <JJJJ> が含まれる場合は、*input_char_date* の YY の位置に、日本元号での年号が入ります。 例えば、1988 年は日本の昭和 63 年に相当します。

***output_Lilian_date* (出力)**
リリアン日付 (1582 年 10 月 14 日から数えた日数) を表す 32 ビットの 2 進整数。 例えば、1988 年 5 月 16 日は、日数 148138 に相当します。

input_char_date に有効な日付が含まれていない場合は、*output_Lilian_date* が 0 に設定され、CEEDAYS が終了して非 CEE000 シンボリック・フィードバック・コードが戻されます。

output_Lilian_date は整数なので、日付計算を容易に行うことができます。 うるう年や年末偏差は計算に影響しません。

***fc* (出力)**
このサービスの結果を示す 12 バイトの フィードバック・コード (オプション)。

表 *63.* ***CEEDAYS* のシンボリック条件**

シンボリック・フィードバック・コード	重大度	メッセージ番号	メッセージ・テキスト
CEE000	0	--	サービスが正しく完了しました。
CEE2EB	3	2507	CEEDAYS または CEESECS に渡されたデータが不十分です。リリアン日付の値は計算されませんでした。
CEE2EC	3	2508	CEEDAYS または CEESECS に渡された日付値が無効です。
CEE2ED	3	2509	CEEDAYS または CEESECS に渡された元号が認識されませんでした。
CEE2EH	3	2513	CEEISEC、CEEDAYS、CEESECS のいずれかの呼び出しで渡された入力日付が、対応範囲内にありませんでした。
CEE2EL	3	2517	CEEISEC 呼び出し内の月の値が認識されませんでした。
CEE2EM	3	2518	日時サービスへの呼び出しに無効なピクチャー・ストリングが指定されました。
CEE2EO	3	2520	CEEDAYS が数値フィールド内に非数値データを検出したか、あるいは日付ストリングとピクチャー・ストリングが一致しませんでした。

表 63. ***CEEDAYS* のシンボリック条件** *(続き)*			
シンボリック・フィードバック・コード	**重大度**	**メッセージ番号**	**メッセージ・テキスト**
CEE2EP	3	2521	CEEDAYS または CEESECS に渡された <JJJJ>、<CCCC>、または <CCCCCCCC> の元号年数値がゼロでした。

使用上の注意

- CEEDAYS の逆は CEEDATE です。CEEDATE は、*output_Lilian_date* をリリアン形式から文字形式に変換します。
- 1582 年 10 月 15 日よりも前の日付に対して計算を実行するには、各日付の年号に 4000 を加算し、その日付をリリアン形式に変換してから計算します。 計算結果が日数ではなく日付になる場合は、計算結果を日付ストリングに変換し、年号から 4000 を減算します。
- デフォルトでは、2 桁の年号は、システム日付より 80 年前から始まる 100 年間にあります。 したがって、2010 年の場合、2 桁の年号はすべて 1930 から 2029 年の範囲内の日付を表します。 このデフォルトの範囲を変更するには、CEESCEN 呼び出し可能サービスを使用します。
- *output_Lilian_date* は整数なので、日付計算を容易に行うことができます。 うるう年や年末偏差は回避されます。

例

```
      ******************************************
      **                                      **
      ** Function: CEEDAYS - convert date to  **
      **                     Lilian format    **
      **                                      **
      ******************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLDAYS.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  CHRDATE.
           02  Vstring-length      PIC S9(4) BINARY.
           02  Vstring-text.
               03  Vstring-char    PIC X
                           OCCURS 0 TO 256 TIMES
                           DEPENDING ON Vstring-length
                               of CHRDATE.
       01  PICSTR.
           02  Vstring-length      PIC S9(4) BINARY.
           02  Vstring-text.
               03  Vstring-char    PIC X
                           OCCURS 0 TO 256 TIMES
                           DEPENDING ON Vstring-length
                               of PICSTR.
       01  LILIAN                  PIC S9(9) BINARY.
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) COMP.
                   04  Msg-No      PIC S9(4) COMP.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) COMP.
                   04  Cause-Code  PIC S9(4) COMP.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) COMP.
      *
       PROCEDURE DIVISION.
       PARA-CBLDAYS.
      ***********************************************
      ** Specify input date and length             **
      ***********************************************
           MOVE 16 TO Vstring-length of CHRDATE.
           MOVE '1 January 2005'
```

```
           TO Vstring-text of CHRDATE.

      ************************************************
      ** Specify a picture string that describes     **
      ** input date, and the picture string's length.**
      ************************************************
           MOVE 25 TO Vstring-length of PICSTR.
           MOVE 'ZD Mmmmmmmmmmmmmmz YYYY'
                   TO Vstring-text of PICSTR.

      ************************************************
      ** Call CEEDAYS to convert input date to a     **
      ** Lilian date                                 **
      ************************************************
           CALL 'CEEDAYS' USING CHRDATE, PICSTR,
                                LILIAN, FC.

      ************************************************
      ** If CEEDAYS runs successfully, display result**
      ************************************************
           IF  CEE000 of FC  THEN
               DISPLAY Vstring-text of CHRDATE
                   ' is Lilian day: ' LILIAN
           ELSE
               DISPLAY 'CEEDAYS failed with msg '
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.

           GOBACK.
```

516 ページの『例: 日時のピクチャー・ストリング』

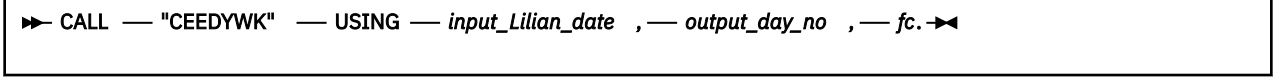
関連参照
514 ページの『ピクチャー文字項およびストリング』

CEEDYWK: リリアン日付からの曜日の計算

CEEDYWK は、リリアン日付の曜日を 1 から 7 までの数字として計算します。

CEEDYWK から戻される数値から、曜日を計算することができます。

CALL CEEDYWK の構文



***input_Lilian_date* (入力)**
リリアン日付 (1582 年 10 月 14 日から数えた日数) を表す 32 ビットの 2 進整数。

例えば、1988 年 5 月 16 日は、日数 148138 に相当します。 *input_Lilian_date* の有効範囲は 1 から 3,074,324 (1582 年 10 月 15 日から 9999 年 12 月 31 日) です。

***output_day_no* (出力)**
input_Lilian_date の曜日を表す 32 ビットの 2 進整数 (1 = 日曜日、2 = 月曜日、. . . 7 = 土曜日)。

input_Lilian_date が無効な場合は、*output_day_no* が 0 に設定され、CEEDYWK が終了して非 CEE000 シンボリック・フィードバック・コードが戻されます。

***fc* (出力)**
このサービスの結果を示す 12 バイトの フィードバック・コード (オプション)。

表 *64.* ***CEEDYWK* のシンボリック条件**

シンボリック・フィードバック・コード	重大度	メッセージ番号	メッセージ・テキスト
CEE000	0	--	サービスが正しく完了しました。

表 64. ***CEEDYWK* のシンボリック条件** *(続き)*			
シンボリック・フィードバック・コード	**重大度**	**メッセージ番号**	**メッセージ・テキスト**
CEE2EG	3	2512	CEEDATE または CEEDYWK への呼び出しで渡されたリリアン日付値が、対応範囲内にありませんでした。

例

```
      ************************************************
      **                                            **
      ** Function: Call CEEDYWK to calculate the    **
      **           day of the week from Lilian date **
      **                                            **
      ** In this example, a call is made to CEEDYWK **
      ** to return the day of the week on which a   **
      ** Lilian date falls. (A Lilian date is the   **
      ** number of days since 14 October 1582)      **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLDYWK.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  LILIAN                  PIC S9(9) BINARY.
       01  DAYNUM                  PIC S9(9) BINARY.
       01  IN-DATE.
           02  Vstring-length      PIC S9(4) BINARY.
           02  Vstring-text.
               03  Vstring-char        PIC X,
                           OCCURS 0 TO 256 TIMES
                           DEPENDING ON Vstring-length
                               of IN-DATE.
       01  PICSTR.
           02  Vstring-length      PIC S9(4) BINARY.
           02  Vstring-text.
               03  Vstring-char        PIC X,
                           OCCURS 0 TO 256 TIMES
                           DEPENDING ON Vstring-length
                               of PICSTR.
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) COMP.
                   04  Msg-No      PIC S9(4) COMP.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) COMP.
                   04  Cause-Code  PIC S9(4) COMP.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) COMP.

       PROCEDURE DIVISION.
       PARA-CBLDAYS.
      ** Call CEEDAYS to convert date of 6/2/88 to
      **     Lilian representation
           MOVE 6 TO Vstring-length of IN-DATE.
           MOVE '6/2/88' TO Vstring-text of IN-DATE(1:6).
           MOVE 8 TO Vstring-length of PICSTR.
           MOVE 'MM/DD/YY' TO Vstring-text of PICSTR(1:8).
           CALL 'CEEDAYS' USING IN-DATE, PICSTR,
               LILIAN, FC.

      ** If CEEDAYS runs successfully, display result.
           IF  CEE000 of FC  THEN
               DISPLAY Vstring-text of IN-DATE
                   ' is Lilian day: ' LILIAN
           ELSE
               DISPLAY 'CEEDAYS failed with msg '
                   Msg-No of FC UPON CONSOLE
               STOP RUN
```

```
           END-IF.

       PARA-CBLDYWK.

      ** Call CEEDYWK to return the day of the week on
      ** which the Lilian date falls
           CALL 'CEEDYWK' USING LILIAN , DAYNUM , FC.

      ** If CEEDYWK runs successfully, print results
           IF CEE000 of FC  THEN
               DISPLAY 'Lilian day ' LILIAN
                   ' falls on day ' DAYNUM
                   ' of the week, which is a:'
      ** Select DAYNUM to display the name of the day
      **     of the week.
               EVALUATE DAYNUM
                 WHEN 1
                   DISPLAY 'Sunday.'
                 WHEN 2
                   DISPLAY 'Monday.'
                 WHEN 3
                   DISPLAY 'Tuesday'
                 WHEN 4
                   DISPLAY 'Wednesday.'
                 WHEN 5
                   DISPLAY 'Thursday.'
                 WHEN 6
                   DISPLAY 'Friday.'
                 WHEN 7
                   DISPLAY 'Saturday.'
               END-EVALUATE
           ELSE
               DISPLAY 'CEEDYWK failed with msg '
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.

           GOBACK.
```

CEEGMT: 現在のグリニッジ標準時の取得

CEEGMT は、現在のグリニッジ標準時 (GMT) をリリアン日、および 1582 年 10 月 14 日 00:00:00 以降の秒数として戻します。 戻り値は、他の日時の呼び出し可能サービスによって生成および使用される値と互換性があります。

ALL CEEGMT の構文

***output_GMT_Lilian* (出力)**
グリニッジ (イングランド) の現在の日付をリリアン形式 (1582 年 10 月 14 日から数えた日数) で表す 32 ビットの 2 進整数。

例えば、1988 年 5 月 16 日は、日数 148138 に相当します。 システムから GMT を使用できない場合は、*output_GMT_Lilian* が 0 に設定され、CEEGMT が終了して非 CEE000 シンボリック・フィードバック・コードが戻されます。

***output_GMT_seconds* (出力)**
グリニッジ (イングランド) の現在の日時を 1582 年 10 月 14 日の 00:00:00 から数えた (うるう秒は数えない) 秒数で表す、64 ビット長の浮動小数点数。

例えば、1582 年 10 月 15 日の 00:00:01 は秒数 86,401 (24*60*60 + 01) に相当します。 1988 年 5 月 16 日の 19:00:01.078 は、秒数 12,799,191,601.078 に相当します。 システムから GMT を使用できない場合は、*output_GMT_seconds* が 0 に設定され、CEEGMT が終了して非 CEE000 シンボリック・フィードバック・コードが戻されます。

***fc* (出力)**
このサービスの結果を示す 12 バイトの フィードバック・コード (オプション)。

表 65. ***CEEGMT* のシンボリック条件**			
シンボリック・フィードバック・コード	**重大度**	**メッセージ番号**	**メッセージ・テキスト**
CEE000	0	--	サービスが正しく完了しました。
CEE2E6	3	2502	システムから UTC/GMT を使用できませんでした。

使用上の注意

- CEEDATE は *output_GMT_Lilian* を文字日付に変換し、CEEDATM は *output_GMT_seconds* を文字タイム・スタンプに変換します。
- このサービスから意味のある結果を得るには、システムのクロックを現地時間に設定し、環境変数 TZ を正しく設定する必要があります。
- CEEGMT から戻される値から、経過時間を計算することができます。 例えば、CEEGMT への呼び出しが行われてから、次に同じ呼び出しが行われるまでの経過時間を計算するには、2 つの戻り値の差を計算します。
- CEEUTC はこのサービスと同じです。

例

```
      ************************************************
      **                                            **
      ** Function: Call CEEGMT to get current       **
      **           Greenwich Mean Time              **
      **                                            **
      ** In this example, a call is made to CEEGMT  **
      ** to return the current GMT as a Lilian date **
      ** and as Lilian seconds. The results are     **
      ** displayed.                                 **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. IGZTGMT.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  LILIAN                  PIC S9(9) BINARY.
       01  SECS                    COMP-2.
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) COMP.
                   04  Msg-No      PIC S9(4) COMP.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) COMP.
                   04  Cause-Code  PIC S9(4) COMP.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) COMP.
       PROCEDURE DIVISION.
       PARA-CBLGMT.
           CALL 'CEEGMT' USING LILIAN , SECS , FC.

           IF CEE000 of FC  THEN
               DISPLAY 'The current GMT is also '
                   'known as Lilian day: ' LILIAN
               DISPLAY 'The current GMT in Lilian '
                   'seconds is: ' SECS
           ELSE
               DISPLAY 'CEEGMT failed with msg '
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.

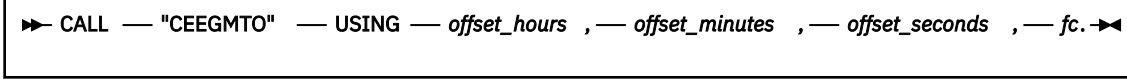
           GOBACK.
```

関連タスク
215 ページの『環境変数の設定』

CEEGMTO: グリニッジ標準時から現地時間までのオフセットの取得

CEEGMTO は、ローカル・システムの時刻とグリニッジ標準時 (GMT) の差を表す値を呼び出しルーチンに戻します。

CALL CEEGMTO の構文



***offset_hours* (出力)**
GMT から現地時間までのオフセットを時間単位で表す 32 ビットの 2 進整数。

例えば太平洋標準時の場合、*offset_hours* は -8 に相当します。

offset_hours の範囲は、-12 から +13 (+13 = +12 の時間帯における夏時間調整) です。

現地時間のオフセットを使用できない場合は、*offset_hours* が 0 になり、CEEGMTO が終了して非 CEE000 シンボリック・フィードバック・コードが戻されます。

***offset_minutes* (出力)**
現地時間が GMT よりも何分進んでいるか、または何分遅れているかを表す、32 ビットの 2 進整数。

offset_minutes の範囲は 0 から 59 です。

現地時間のオフセットを使用できない場合は、*offset_minutes* が 0 になり、CEEGMTO が終了して非 CEE000 シンボリック・フィードバック・コードが戻されます。

***offset_seconds* (出力)**
GMT から現地時間までのオフセットを秒単位で表す 64 ビット長の浮動小数点数。

例えば、太平洋標準時は GMT よりも 8 時間遅れています。 現地時間が標準時で太平洋標準時間帯に属する場合、CEEGMTO は -28,800 (-8 * 60 * 60) を戻します。 *offset_seconds* の範囲は -43,200 から +46,800 です。 *offset_seconds* を CEEGMT で使用すると、現地日時を計算することができます。

システムから現地時間のオフセットを使用できない場合は、*offset_seconds* が 0 に設定され、CEEGMTO が終了して非 CEE000 シンボリック・フィードバック・コードが戻されます。

***fc* (出力)**
このサービスの結果を示す 12 バイトの フィードバック・コード (オプション)。

表 66. ***CEEGMTO* のシンボリック条件**

シンボリック・フィードバック・コード	重大度	メッセージ番号	メッセージ・テキスト
CEE000	0	--	サービスが正しく完了しました。
CEE2E7	3	2503	UTC/GMT から現地時間までのオフセットをシステムから使用できませんでした。

使用上の注意

- CEEDATM は *offset_seconds* を文字タイム・スタンプに変換します。
- このサービスから意味のある結果を得るには、システムのクロックを現地時間に設定し、環境変数 TZ を正しく設定する必要があります。

例

```
      ************************************************
      **                                            **
      ** Function:  Call CEEGMTO to get offset from  **
      **            Greenwich Mean Time to local     **
      **            time                             **
      **                                            **
      ** In this example, a call is made to CEEGMTO  **
      ** to return the offset from GMT to local time **
      ** as separate binary integers representing    **
      ** offset hours, minutes, and seconds. The     **
      ** results are displayed.                      **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. IGZTGMTO.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  HOURS                   PIC S9(9) BINARY.
       01  MINUTES                 PIC S9(9) BINARY.
       01  SECONDS COMP-2.
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) COMP.
                   04  Msg-No      PIC S9(4) COMP.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) COMP.
                   04  Cause-Code  PIC S9(4) COMP.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) COMP.
       PROCEDURE DIVISION.
       PARA-CBLGMTO.
           CALL 'CEEGMTO' USING HOURS , MINUTES ,
               SECONDS , FC.

           IF CEE000 of FC  THEN
               DISPLAY 'Local time differs from GMT '
                   'by: ' HOURS ' hours, '
                   MINUTES ' minutes, OR  '
                   SECONDS ' seconds. '
           ELSE
               DISPLAY 'CEEGMTO failed with msg '
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.

           GOBACK.
```

関連タスク

215 ページの『環境変数の設定』

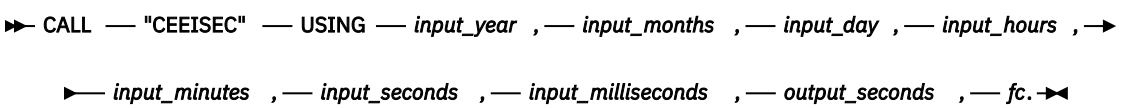
関連参照

216 ページの『コンパイラー環境変数とランタイム環境変数』
558 ページの『CEEGMT: 現在のグリニッジ標準時の取得』

CEEISEC: 整数から秒への変換

CEEISEC は、年、月、日、時、分、秒、ミリ秒を表す 2 進整数を、1582 年 10 月 14 日の 00:00:00 から数えた秒数を表す数値に変換します。

CALL CEEISEC の構文



***input_year* (入力)**
年を表す 32 ビットの 2 進整数。

input_year の有効な値範囲は 1582 から 9999 です。

***input_month* (入力)**
月を表す 32 ビットの 2 進整数。

input_month の有効な値範囲は 1 から 12 です。

***input_day* (入力)**
日を表す 32 ビットの 2 進整数。

input_day の有効な値範囲は 1 から 31 です。

***input_hours* (入力)**
時を表す 32 ビットの 2 進整数。

input_hours の有効な値範囲は 0 から 23 です。

***input_minutes* (入力)**
分を表す 32 ビットの 2 進整数。

input_minutes の有効な値範囲は 0 から 59 です。

***input_seconds* (入力)**
秒を表す 32 ビットの 2 進整数。

input_seconds の有効な値範囲は 0 から 59 です。

***input_milliseconds* (入力)**
ミリ秒を表す 32 ビットの 2 進整数。

input_milliseconds の有効な値範囲は 0 から 999 です。

***output_seconds* (出力)**
1582 年 10 月 14 日の 00:00:00 から数えた (うるう秒は数えない) 秒数を表す、64 ビット長の浮動小数点数。

例えば、1582 年 10 月 15 日の 00:00:01 は秒数 86,401 (24*60*60 + 01) に相当します。*output_seconds* の有効範囲は 86,400 から 265,621,679,999.999 (9999 年 12 月 31 日の 23:59:59.999) です。

入力値が無効な場合は、*output_seconds* が 0 に設定されます。

output_seconds をリリアン日数に変換するには、 divide *output_seconds* を 86,400 (1 日分の秒数) で除算します。

***fc* (出力)**
このサービスの結果を示す 12 バイトの フィードバック・コード (オプション)。

表 67. ***CEEISEC* のシンボリック条件**			
シンボリック・フィードバック・コード	**重大度**	**メッセージ番号**	**メッセージ・テキスト**
CEE000	0	--	サービスが正しく完了しました。
CEE2EE	3	2510	CEEISEC または CEESECS への呼び出しで時間の値が認識されませんでした。
CEE2EF	3	2511	CEEISEC の呼び出しで渡された日のパラメーターが、指定された年および月に対して無効です。
CEE2EH	3	2513	CEEISEC、CEEDAYS、CEESECS のいずれかの呼び出しで渡された入力日付が、対応範囲内にありませんでした。
CEE2EI	3	2514	CEEISEC の呼び出しで渡された年の値が、対応範囲内にありませんでした。
CEE2EJ	3	2515	CEEISEC 呼び出し内のミリ秒の値が認識されませんでした。
CEE2EK	3	2516	CEEISEC 呼び出し内の分の値が認識されませんでした。
CEE2EL	3	2517	CEEISEC 呼び出し内の月の値が認識されませんでした。
CEE2EN	3	2519	CEEISEC 呼び出し内の秒の値が認識されませんでした。

使用上の注意: CEEISEC の逆は CEESECI です。CEESECI は、秒数を整数の年、月、日、時、分、秒、ミリ秒に変換します。

例

```
      ************************************************
      **                                            **
      ** Function: Call CEEISEC to convert integers  **
      **           to seconds                       **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLISEC.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  YEAR                    PIC S9(9) BINARY.
       01  MONTH                   PIC S9(9) BINARY.
       01  DAYS                    PIC S9(9) BINARY.
       01  HOURS                   PIC S9(9) BINARY.
       01  MINUTES                 PIC S9(9) BINARY.
       01  SECONDS                 PIC S9(9) BINARY.
       01  MILLSEC                 PIC S9(9) BINARY.
       01  OUTSECS                 COMP-2.
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) COMP.
                   04  Msg-No      PIC S9(4) COMP.
               03  Case-2-Condition-ID
                       REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) COMP.
                   04  Cause-Code  PIC S9(4) COMP.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) COMP.
       PROCEDURE DIVISION.
       PARA-CBLISEC.
      ************************************************
      ** Specify seven binary integers representing  **
      ** the date and time as input to be converted  **
      ** to Lilian seconds                          **
      ************************************************
```

```
           MOVE 2000 TO YEAR.
           MOVE 1 TO MONTH.
           MOVE 1 TO DAYS.
           MOVE 0 TO HOURS.
           MOVE 0 TO MINUTES.
           MOVE 0 TO SECONDS.
           MOVE 0 TO MILLSEC.
      ************************************************
      ** Call CEEISEC to convert the integers        **
      ** to seconds                                  **
      ************************************************
           CALL 'CEEISEC' USING YEAR, MONTH, DAYS,
                                HOURS, MINUTES, SECONDS,
                                MILLSEC, OUTSECS , FC.
      ************************************************
      ** If CEEISEC runs successfully, display result**
      ************************************************
           IF CEE000 of FC  THEN
               DISPLAY MONTH '/' DAYS '/' YEAR
                 ' AT ' HOURS ':' MINUTES ':' SECONDS
                 ' is equivalent to ' OUTSECS ' seconds'
           ELSE
               DISPLAY 'CEEISEC failed with msg '
                 Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.

           GOBACK.
```

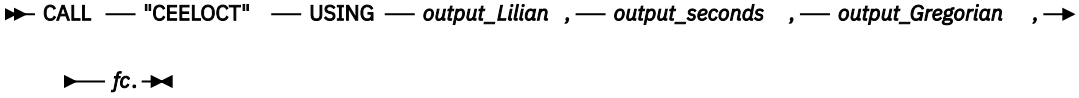
CEELOCT: 現在の現地日時の取得

CEELOCT は、現在の現地日時をリリアン日付 (1582 年 10 月 14 日から数えた日数)、リリアン秒数 (1582 年 10 月 14 日の 00:00:00 から数えた秒数)、グレゴリオ文字ストリング (YYYYMMDDHHMISS999) としてそれぞれ返します。

これらの値は、他の日時の呼び出し可能サービスや、既存の組み込み関数と互換性があります。

CEELOCT は、CEEGMT、CEEGMTO、および CEEDATM の各サービスを個別に呼び出すのと同じ機能を実行します。 ただし、CEELOCT の呼び出しの方が、はるかに高速です。

CALL CEELOCT の構文



***output_Lilian* (出力)**
現在の現地日付をリリアン形式 (1 日目は 1582 年 10 月 15 日、148,887 日目は 1990 年 6 月 4 日) で表す、32 ビットの 2 進整数。

システムから現地時間を使用できない場合は、*output_Lilian* が 0 に設定され、CEELOCT が終了して非 CEE000 シンボリック・フィードバック・コードが戻されます。

***output_seconds* (出力)**
現在の現地日時を 1582 年 10 月 14 日の 00:00:00 から数えた (うるう秒は数えない) 秒数で表す、64 ビット長の浮動小数点数。 例えば、1582 年 10 月 15 日の 00:00:01 は秒数 86,401 (24*60*60 + 01) に相当します。 1990 年 6 月 4 日の 19:00:01.078 は、秒数 12,863,905,201.078 に相当します。

システムから現地時間を使用できない場合は、*output_seconds* が 0 に設定され、CEELOCT が終了して非 CEE000 シンボリック・フィードバック・コードが戻されます。

***output_Gregorian* (出力)**
現地の年、月、日、時、分、秒、ミリ秒を表す、YYYYMMDDHHMISS999 形式の 17 バイト固定長文字ストリング。

output_Gregorian の形式が必要な形式と合わない場合は、CEEDATM 呼び出し可能サービスを使用して、*output_seconds* を別の形式に変換することができます。

***fc* (出力)**

このサービスの結果を示す 12 バイトの フィードバック・コード (オプション)。

表 *68.* ***CEELOCT* のシンボリック条件**			
シンボリック・フィードバック・コード	**重大度**	**メッセージ番号**	**メッセージ・テキスト**
CEE000	0	--	サービスが正しく完了しました。
CEE2F3	3	2531	システムから現地時間を使用できませんでした。

使用上の注意

- CEEGMT 呼び出し可能サービスを使用して、グリニッジ標準時 (GMT) を判断することができます。
- CEEGMTO 呼び出し可能サービスを使用して、GMT から現地時間までのオフセットを取得することができます。
- CEELOCT により戻される文字値は、既存の組み込み関数から生成される文字値と一致するようになっています。 戻される数値を使用して、日付計算を単純化することができます。

例

```
      ************************************************
      **                                            **
      ** Function: Call CEELOCT to get current      **
      **           local time                       **
      **                                            **
      ** In this example, a call is made to CEELOCT **
      ** to return the current local time in Lilian **
      ** days (the number of days since 14 October  **
      ** 1582), Lilian seconds (the number of       **
      ** seconds since 00:00:00 14 October 1582),   **
      ** and a Gregorian string (in the form        **
      ** YYYMMDDMISS999). The Gregorian character   **
      ** string is then displayed.                  **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLLOCT.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  LILIAN                  PIC S9(9) BINARY.
       01  SECONDS                 COMP-2.
       01  GREGORN                 PIC X(17).
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) COMP.
                   04  Msg-No      PIC S9(4) COMP.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) COMP.
                   04  Cause-Code  PIC S9(4) COMP.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) COMP.
       PROCEDURE DIVISION.
       PARA-CBLLOCT.
           CALL 'CEELOCT' USING LILIAN, SECONDS,
                                GREGORN, FC.
      ************************************************
      ** If CEELOCT runs successfully, display      **
      **     Gregorian character string             **
      ************************************************
           IF CEE000 of FC  THEN
               DISPLAY 'Local Time is ' GREGORN
           ELSE
               DISPLAY 'CEELOCT failed with msg '
                   Msg-No of FC UPON CONSOLE
```

```
            STOP RUN
        END-IF.

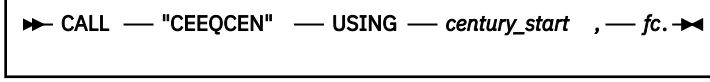
        GOBACK.
```

CEEQCEN: 世紀ウィンドウの照会

CEEQCEN は、2 桁の年号値の世紀ウィンドウを照会します。

世紀ウィンドウを変更する場合は、CEEQCEN を使用して設定を取得した後、CEESCEN を使用して現行の設定を保存して復元します。

CALL CEEQCEN の構文



***century_start* (出力)**
世紀ウィンドウの基になる年を表す、0 から 100 の整数。

例えば、日時の呼び出し可能サービスのデフォルトが有効な場合、2 桁の年号はすべて、システム日付より 80 年前から始まる 100 年間に属します。 この後、CEEQCEN が値 80 を戻します。 例えば、2010 年の場合、80 は、すべての 2 桁年号が 100 年間 (1930 年から 2029 年まで) にあることを示します。

***fc* (出力)**
このサービスの結果を示す 12 バイトの フィードバック・コード (オプション)。

表 *69.* ***CEEQCEN* のシンボリック条件**

シンボリック・フィードバック・コード	重大度	メッセージ番号	メッセージ・テキスト
CEE000	0	--	サービスが正しく完了しました。

例

```
      ************************************************
      **                                            **
      ** Function: Call CEEQCEN to query the        **
      **           date and time callable services  **
      **           century window                   **
      **                                            **
      ** In this example, CEEQCEN is called to query **
      ** the date at which the century window starts **
      ** The century window is the 100-year window   **
      ** within which the date and time callable     **
      ** services assume all two-digit years lie.    **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLQCEN.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  STARTCW                 PIC S9(9) BINARY.
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) COMP.
                   04  Msg-No      PIC S9(4) COMP.
               03  Case-2-Condition-ID
                         REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) COMP.
                   04  Cause-Code  PIC S9(4) COMP.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) COMP.
```

```
       PROCEDURE DIVISION.

       PARA-CBLQCEN.
      ************************************************
      ** Call CEEQCEN to return the start of the     **
      **     century window                           **
      ************************************************

           CALL 'CEEQCEN' USING STARTCW, FC.
      ************************************************
      ** CEEQCEN has no nonzero feedback codes to    **
      **     check, so just display result.           **
      ************************************************
           IF CEE000 of FC  THEN
               DISPLAY 'The start of the century '
                   'window is: ' STARTCW
           ELSE
               DISPLAY 'CEEQCEN failed with msg '
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.

           GOBACK.
```

CEESCEN: 世紀ウィンドウの設定

CEESCEN は、世紀ウィンドウを、他の日時の呼び出し可能サービスが使用できる 2 桁年号値に設定します。

次のような場合は、CEEDAYS または CEESECS と組み合わせて CEESCEN を使用します。

- 2 桁の年号を含む日付値を処理する場合 (YYMMDD 形式など)
- デフォルトの世紀間隔が特定のアプリケーションの要件に合わない場合

世紀ウィンドウを照会するには、CEEQCEN を使用します。

CALL CEESCEN の構文

century_start
世紀ウィンドウを設定する、0 から 100 の整数。

例えば、値 80 の場合は、すべての 2 桁年桁が、システム日付より 80 年前から始まる 100 年間に属します。 したがって、2010 年の場合、2 桁の年号はすべて 1930 から 2029 年の範囲内の日付を表すものと想定されます。

***fc* (出力)**
このサービスの結果を示す 12 バイトの フィードバック・コード (オプション)。

表 *70.* ***CEESCEN* のシンボリック条件**

シンボリック・フィードバック・コード	重大度	メッセージ番号	メッセージ・テキスト
CEE000	0	--	サービスが正しく完了しました。
CEE2E6	3	2502	システムから UTC/GMT を使用できませんでした。
CEE2F5	3	2533	CEESCEN に渡された値が 0 から 100 の範囲内にありませんでした。

例

```
      ************************************************
```

```
      **                                               **
      ** Function: Call CEESCEN to set the            **
      **           date and time callable services    **
      **           century window                     **
      **                                               **
      ** In this example, CEESCEN is called to change **
      ** the start of the century window to 30 years  **
      ** before the system date. CEEQCEN is then      **
      ** called to query that the change made.  A     **
      ** message that this has been done is then      **
      ** displayed.                                   **
      **                                               **
      ***************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLSCEN.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  STARTCW                 PIC S9(9) BINARY.
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) COMP.
                   04  Msg-No      PIC S9(4) COMP.
               03  Case-2-Condition-ID
                       REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) COMP.
                   04  Cause-Code  PIC S9(4) COMP.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) COMP.
       PROCEDURE DIVISION.
       PARA-CBLSCEN.
      ***************************************************
      ** Specify 30 as century start, and two-digit
      **     years will be assumed to lie in the
      **     100-year window starting 30 years before
      **     the system date.
      ***************************************************
           MOVE 30 TO STARTCW.

      ***************************************************
      ** Call CEESCEN to change the start of the century
      **     window.
      ***************************************************
           CALL 'CEESCEN' USING STARTCW, FC.
           IF NOT CEE000 of FC  THEN
               DISPLAY 'CEESCEN failed with msg '
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.

       PARA-CBLQCEN.
      ***************************************************
      ** Call CEEQCEN to return the start of the century
      **     window
      ***************************************************
           CALL 'CEEQCEN' USING STARTCW, FC.

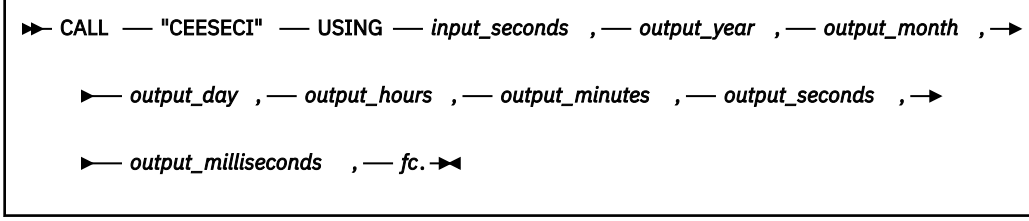
      ***************************************************
      ** CEEQCEN has no nonzero feedback codes to
      **     check, so just display result.
      ***************************************************
               DISPLAY 'The start of the century '
                   'window is: ' STARTCW
           GOBACK.
```

CEESECI: 秒から整数への変換

CEESECI は、1582 年 10 月 14 日の 00:00:00 から数えた秒数を表す数値を、年、月、日、時、分、秒、ミリ秒を表す 2 進整数に変換します。

文字形式ではなく数値形式の出力が必要な場合は、 CEEDATM ではなく CEESECI を使用します。

CALL CEESECI の構文



input_seconds
1582 年 10 月 14 日の 00:00:00 から数えた (うるう秒は数えない) 秒数を表す、64 ビット長の浮動小数点数。

例えば、1582 年 10 月 15 日の 00:00:01 は秒数 86,401 (24*60*60 + 01) に相当します。
input_seconds の有効な値範囲は 86,400 から 265,621,679,999.999 (9999 年 12 月 31 日の 23:59:59.999) です。

input_seconds が無効な場合は、フィードバック・コードを除くすべての出力パラメーターが 0 に設定されます。

***output_year* (出力)**
年を表す 32 ビットの 2 進整数。

output_year の有効な値範囲は 1582 から 9999 です。

***output_month* (出力)**
月を表す 32 ビットの 2 進整数。

output_month の有効な値範囲は 1 から 12 です。

***output_day* (出力)**
日を表す 32 ビットの 2 進整数。

output_day の有効な値範囲は 1 から 31 です。

***output_hours* (出力)**
時を表す 32 ビットの 2 進整数。

output_hours の有効な値範囲は 0 から 23 です。

***output_minutes* (出力)**
分を表す 32 ビットの 2 進整数。

output_minutes の有効な値範囲は 0 から 59 です。

***output_seconds* (出力)**
秒を表す 32 ビットの 2 進整数。

output_seconds の有効な値範囲は 0 から 59 です。

***output_milliseconds* (出力)**
ミリ秒を表す 32 ビットの 2 進整数。

output_milliseconds の有効な値範囲は 0 から 999 です。

***fc* (出力)**
このサービスの結果を示す 12 バイトの フィードバック・コード (オプション)。

表 *71.* ***CEESECI* のシンボリック条件**

シンボリック・フィードバック・コード	重大度	メッセージ番号	メッセージ・テキスト
CEE000	0	--	サービスが正しく完了しました。

表 71. ***CEESECI* のシンボリック条件** (続き)			
シンボリック・フィードバック・コード	**重大度**	**メッセージ番号**	**メッセージ・テキスト**
CEE2E9	3	2505	CEEDATM または CEESECI への呼び出し内の input_seconds 値が、対応範囲内にありませんでした。

使用上の注意

- CEESECI の逆は CEEISEC です。CEEISEC は、年、月、日、時、分、秒、ミリ秒を表す個々の 2 進整数を秒数に変換します。
- 入力値が秒ではなくリリアン日付の場合は、リリアン日付に 86,400 (1 日分の秒数) を乗算してから、新しい値を CEESECI に渡します。

例

```
      ************************************************
      **                                            **
      ** Function: Call CEESECI to convert seconds  **
      **           to integers                      **
      **                                            **
      ** In this example a call is made to CEESECI  **
      ** to convert a number representing the number **
      ** of seconds since 00:00:00 14 October 1582  **
      ** to seven binary integers representing year, **
      ** month, day, hour, minute, second, and      **
      ** millisecond.  The results are displayed in **
      ** this example.                              **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLSECI.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  INSECS                  COMP-2.
       01  YEAR                    PIC S9(9) BINARY.
       01  MONTH                   PIC S9(9) BINARY.
       01  DAYS                    PIC S9(9) BINARY.
       01  HOURS                   PIC S9(9) BINARY.
       01  MINUTES                 PIC S9(9) BINARY.
       01  SECONDS                 PIC S9(9) BINARY.
       01  MILLSEC                 PIC S9(9) BINARY.
       01  IN-DATE.
           02  Vstring-length      PIC S9(4) BINARY.
           02  Vstring-text.
               03  Vstring-char        PIC X,
                       OCCURS 0 TO 256 TIMES
                       DEPENDING ON Vstring-length
                           of IN-DATE.
       01  PICSTR.
           02  Vstring-length      PIC S9(4) BINARY.
           02  Vstring-text.
               03  Vstring-char        PIC X,
                       OCCURS 0 TO 256 TIMES
                       DEPENDING ON Vstring-length
                          of PICSTR.
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) COMP.
                   04  Msg-No      PIC S9(4) COMP.
               03  Case-2-Condition-ID
                       REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) COMP.
                   04  Cause-Code  PIC S9(4) COMP.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) COMP.
       PROCEDURE DIVISION.
```

```
 PARA-CBLSECS.
*************************************************
** Call CEESECS to convert time stamp of 6/2/88
**     at 10:23:45 AM to Lilian representation
*************************************************
     MOVE 20 TO Vstring-length of IN-DATE.
     MOVE '06/02/88 10:23:45 AM'
             TO Vstring-text of IN-DATE.
     MOVE 20 TO Vstring-length of PICSTR.
     MOVE 'MM/DD/YY HH:MI:SS AP'
             TO Vstring-text of PICSTR.
     CALL 'CEESECS' USING IN-DATE, PICSTR,
                          INSECS, FC.
     IF NOT CEE000 of FC  THEN
         DISPLAY 'CEESECS failed with msg '
             Msg-No of FC UPON CONSOLE
         STOP RUN
     END-IF.

 PARA-CBLSECI.
*************************************************
** Call CEESECI to convert seconds to integers
*************************************************
     CALL 'CEESECI' USING INSECS, YEAR, MONTH,
                          DAYS, HOURS,  MINUTES,
                          SECONDS, MILLSEC, FC.
*************************************************
** If CEESECI runs successfully, display results
*************************************************
     IF CEE000 of FC  THEN
         DISPLAY 'Input seconds of ' INSECS
             ' represents:'
         DISPLAY '   Year......... ' YEAR
         DISPLAY '   Month........ ' MONTH
         DISPLAY '   Day.......... ' DAYS
         DISPLAY '   Hour......... ' HOURS
         DISPLAY '   Minute....... ' MINUTES
         DISPLAY '   Second....... ' SECONDS
         DISPLAY '   Millisecond.. ' MILLSEC
     ELSE
         DISPLAY 'CEESECI failed with msg '
             Msg-No of FC UPON CONSOLE
         STOP RUN
     END-IF.

     GOBACK.
```

CEESECS: タイム・スタンプから秒への変換

CEESECS は、タイム・スタンプを表すストリングをリリアン秒 (1582 年 10 月 14 日の 00:00:00 から数えた秒数) に変換します。 このサービスを使用すると、2 つのタイム・スタンプ間の経過時間を計算するなどの時間演算が容易になります。

CALL CEESECS の構文

***input_timestamp* (入力)**

picture_string の指定と一致した形式で日付またはタイム・スタンプを表す、ハーフワード長の接頭部の付いた文字ストリング。

文字ストリングに含められる文字数は 5 から 80 ピクチャー文字です。 *input_timestamp* には、先行または末尾ブランクを含めることができます。 構文解析は、最初の非ブランク文字から始まります (ピクチャー・ストリング自体に先行ブランクが含まれる場合は、CEESECS がその位置を正確にスキップした後、構文解析が始まります)。

picture_string で指定された日付形式によって判別される有効な日付を解析したら、CEESECS は残りの文字をすべて無視します。 有効な日付範囲は、1582 年 10 月 15 日から 9999 年 12 月 31 日です。 完全な日付を指定する必要があります。 有効な時刻範囲は 00:00:00.000 から 23:59:59.999 です。

時刻値の一部または全部を省略すると、残りの値には 0 が代入されます。 次に例を示します。

```
1992-05-17-19:02 is equivalent to 1992-05-17-19:02:00
1992-05-17       is equivalent to 1992-05-17-00:00:00
```

***picture_string* (入力)**

input_timestamp で指定された日付またはタイム・スタンプ値の形式を示す、ハーフワード長の接頭部の付いた文字ストリング。

picture_string 内の各文字は、*input_timestamp* 内の文字を表します。 例えば、MMDDYY HH.MI.SS を *picture_string* として指定すると、CEESECS は *input_char_date* の値 060288 15.35.02 を 1988 年 6 月 2 日の 3:35:02 PM として読み取ります。 スラッシュ (/) などの区切り文字がピクチャー・ストリング内にある場合は、先行ゼロを省略することができます。 例えば、次に示す CEESECS への呼び出しはすべて、同じ値をデータ項目 *secs* に割り当てます。

```
CALL CEESECS USING '92/06/03 15.35.03',
                   'YY/MM/DD HH.MI.SS', secs, fc.
CALL CEESECS USING '92/6/3 15.35.03',
                   'YY/MM/DD HH.MI.SS', secs, fc.
CALL CEESECS USING '92/6/3 3.35.03 PM',
                   'YY/MM/DD HH.MI.SS AP', secs, fc.
CALL CEESECS USING '92.155 3.35.03 pm',
                   'YY.DDD   HH.MI.SS AP', secs, fc.
```

picture_string に日本元号のシンボル <JJJJ> が含まれる場合は、*input_timestamp* の YY の位置に、日本元号での年号が入ります。 例えば、1988 年は日本の昭和 63 年に相当します。

***output_seconds* (出力)**

1582 年 10 月 14 日の 00:00:00 から数えた (うるう秒は数えない) 秒数を表す、64 ビット長の浮動小数点数。 例えば、1582 年 10 月 15 日の 00:00:01 は、リリアン形式の秒数 86,401 (24*60*60 + 01) に相当します。 1988 年 5 月 16 日の 19:00:01.12 は、秒数 12,799,191,601.12 に相当します。

表現される最大値は 9999 年 12 月 31 日の 23:59:59.999 です。これは、リリアン形式では秒数 265,621,679,999.999 に相当します。

64 ビット長の浮動小数点値は、精度を失うことなく約 16 桁の有効小数桁数を正確に表現することができます。 このため、最も近いミリ秒 (15 桁の小数桁数) を正確に使用することができます。

input_timestamp に有効な日付またはタイム・スタンプが含まれていない場合は、*output_seconds* が 0 に設定され、CEESECS が終了して非 CEE000 シンボリック・フィードバック・コードが戻されます。

output_seconds は経過時間を表すので、経過時間の計算を容易に行うことができます。 うるう年や年末偏差は計算に影響しません。

***fc* (出力)**

このサービスの結果を示す 12 バイトの フィードバック・コード (オプション)。

表 *72.* ***CEESECS* のシンボリック条件**

シンボリック・フィードバック・コード	重大度	メッセージ番号	メッセージ・テキスト
CEE000	0	--	サービスが正しく完了しました。
CEE2EB	3	2507	CEEDAYS または CEESECS に渡されたデータが不十分です。リリアン日付の値は計算されませんでした。
CEE2EC	3	2508	CEEDAYS または CEESECS に渡された日付値が無効です。
CEE2ED	3	2509	CEEDAYS または CEESECS に渡された元号が認識されませんでした。

表 72. ***CEESECS* のシンボリック条件** *(続き)*			
シンボリック・フィードバック・コード	**重大度**	**メッセージ番号**	**メッセージ・テキスト**
CEE2EE	3	2510	CEEISEC または CEESECS への呼び出しで時間の値が認識されませんでした。
CEE2EH	3	2513	CEEISEC、CEEDAYS、CEESECS のいずれかの呼び出しで渡された入力日付が、対応範囲内にありませんでした。
CEE2EK	3	2516	CEEISEC 呼び出し内の分の値が認識されませんでした。
CEE2EL	3	2517	CEEISEC 呼び出し内の月の値が認識されませんでした。
CEE2EM	3	2518	日時サービスへの呼び出しに無効なピクチャー・ストリングが指定されました。
CEE2EN	3	2519	CEEISEC 呼び出し内の秒の値が認識されませんでした。
CEE2EP	3	2521	CEEDAYS または CEESECS に渡された <JJJJ>、<CCCC>、または <CCCCCCCC> の元号年数値がゼロでした。
CEE2ET	3	2525	CEESECS が数値フィールド内に非数値データを検出したか、あるいはタイム・スタンプ・ストリングとピクチャー・ストリングが一致しませんでした。

使用上の注意

- CEESECS の逆は CEEDATM です。CEEDATM は、*output_seconds* を文字形式に変換します。
- デフォルトでは、2 桁の年号は、システム日付より 80 年前から始まる 100 年間にあります。 したがって、2010 年の場合、2 桁の年号はすべて 1930 から 2029 年の範囲内の日付を表します。 この範囲を変更するには、CEESCEN 呼び出し可能サービスを使用します。

例

```
      ************************************************
      **                                            **
      ** Function: Call CEESECS to convert          **
      **           time stamp to number of seconds  **
      **                                            **
      ** In this example, calls are made to CEESECS **
      ** to convert two time stamps to the number   **
      ** of seconds since 00:00:00 14 October 1582. **
      ** The Lilian seconds for the earlier         **
      ** time stamp are then subtracted from the    **
      ** Lilian seconds for the later time stamp    **
      ** to determine the number of between the     **
      ** two.  This result is displayed.            **
      **                                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLSECS.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  SECOND1                 COMP-2.
       01  SECOND2                 COMP-2.
       01  TIMESTP.
           02  Vstring-length      PIC S9(4) BINARY.
           02  Vstring-text.
               03  Vstring-char        PIC X,
                           OCCURS 0 TO 256 TIMES
                           DEPENDING ON Vstring-length
                              of TIMESTP.
       01  TIMESTP2.
           02  Vstring-length      PIC S9(4) BINARY.
           02  Vstring-text.
```

```
           03  Vstring-char        PIC X,
                       OCCURS 0 TO 256 TIMES
                       DEPENDING ON Vstring-length
                          of TIMESTP2.
       01  PICSTR.
           02  Vstring-length      PIC S9(4) BINARY.
           02  Vstring-text.
               03  Vstring-char        PIC X,
                       OCCURS 0 TO 256 TIMES
                       DEPENDING ON Vstring-length
                          of PICSTR.
       01  FC.
           02  Condition-Token-Value.
           COPY  CEEIGZCT.
               03  Case-1-Condition-ID.
                   04  Severity    PIC S9(4) COMP.
                   04  Msg-No      PIC S9(4) COMP.
               03  Case-2-Condition-ID
                        REDEFINES Case-1-Condition-ID.
                   04  Class-Code  PIC S9(4) COMP.
                   04  Cause-Code  PIC S9(4) COMP.
               03  Case-Sev-Ctl    PIC X.
               03  Facility-ID     PIC XXX.
           02  I-S-Info            PIC S9(9) COMP.
       PROCEDURE DIVISION.

       PARA-SECS1.
      ***********************************************
      ** Specify first time stamp and a picture string
      **     describing the format of the time stamp
      **     as input to CEESECS
      ***********************************************
           MOVE 25 TO Vstring-length of TIMESTP.
           MOVE '1969-05-07 12:01:00.000'
                   TO Vstring-text of TIMESTP.
           MOVE 25 TO Vstring-length of PICSTR.
           MOVE 'YYYY-MM-DD HH:MI:SS.999'
                   TO Vstring-text of PICSTR.

      ***********************************************
      ** Call CEESECS to convert the first time stamp
      ** to Lilian seconds
      ***********************************************
           CALL 'CEESECS' USING TIMESTP, PICSTR,
                                SECOND1, FC.
           IF NOT CEE000 of FC  THEN
               DISPLAY 'CEESECS failed with msg '
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.

       PARA-SECS2.
      ***********************************************
      ** Specify second time stamp and a picture string
      **     describing the format of the time stamp as
      **     input to CEESECS.
      ***********************************************
           MOVE 25 TO Vstring-length of TIMESTP2.
           MOVE '2004-01-01 00:00:01.000'
                   TO Vstring-text of TIMESTP2.
           MOVE 25 TO Vstring-length of PICSTR.
           MOVE 'YYYY-MM-DD HH:MI:SS.999'
                   TO Vstring-text of PICSTR.

      ***********************************************
      ** Call CEESECS to convert the second time stamp
      **     to Lilian seconds
      ***********************************************
           CALL 'CEESECS' USING TIMESTP2, PICSTR,
                                SECOND2, FC.
           IF NOT CEE000 of FC  THEN
               DISPLAY 'CEESECS failed with msg '
                   Msg-No of FC UPON CONSOLE
               STOP RUN
           END-IF.

       PARA-SECS2.
      ***********************************************
      ** Subtract SECOND2 from SECOND1 to determine the
      **     number of seconds between the two time stamps
      ***********************************************
           SUBTRACT SECOND1 FROM SECOND2.
```

```
           DISPLAY 'The number of seconds between '
               Vstring-text OF TIMESTP ' and '
               Vstring-text OF TIMESTP2 ' is: ' SECOND2.

           GOBACK.
```

516 ページの『例: 日時のピクチャー・ストリング』

関連参照
514 ページの『ピクチャー文字項およびストリング』

CEEUTC: 協定世界時の取得

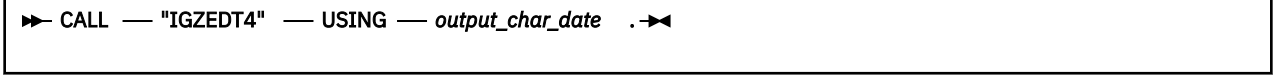
CEEUTC は CEEGMT と同じです。

関連参照
558 ページの『CEEGMT: 現在のグリニッジ標準時の取得』

IGZEDT4: 現在日付の取得

IGZEDT4 は、4 桁年号を使用した現在日付を YYYYMMDD 形式で戻します。

CALL IGZEDT4 の構文



***output_char_date* (出力)**
現在の年、月、日を表す、YYYYMMDD 形式の 8 バイト固定長文字ストリング。

使用上の注意: IGZEDT4 は、CICS ではサポートされません。

例

```
      ************************************************
      ** Function: IGZEDT4 - get current date in the  **
      **                     format YYYYMMDD.         **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CBLEDT4.
       . . .
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  CHRDATE                 PIC S9(8) USAGE DISPLAY.
       . . .
       PROCEDURE DIVISION.
       PARA-CBLEDT4.
      ************************************************
      ** Call IGZEDT4.
      ************************************************
           CALL 'IGZEDT4' USING BY REFERENCE CHRDATE.
      ************************************************
      ** IGZEDT4 has no nonzero return code to
      **     check, so just display result.
      ************************************************
           DISPLAY 'The current date is: '
               CHRDATE
           GOBACK.
```

付録 E XML 参照資料

ここでは、XML 構文解析中または XML 生成中に返される XML 例外コードについて説明します。 また、パーサーが検査する、*XML specification* の整形式性制約条件についても記載します。

関連参照

577 ページの『XML PARSE 例外』
585 ページの『XML 準拠』
587 ページの『XML GENERATE 例外』
XML specification

XML PARSE 例外

例外イベントが発生すると、XML パーサーは、特殊レジスター XML-CODE に、例外を識別する値を設定します。 以下で参照されている情報に詳述されているように、XML-CODE の値に応じて、例外の後にパーサーが処理を続行できる場合と、続行できない場合があります。

関連参照
577 ページの『継続を許可する XML PARSE 例外』
582 ページの『継続を許可しない XML PARSE 例外』

継続を許可する XML PARSE 例外

例外イベントの後に XML パーサーが処理を続行できるかどうかは、例外コードの値によって決まります。

特殊レジスター XML-CODE 内の例外コードが次のいずれかの範囲にあれば、パーサーは処理を続行することができます。

- 1 から 99
- 100,001 から 165,535
- 200,001 - 265,535

次の表には、それぞれの例外の説明と、例外発生後の続行要求時にパーサーが実行するアクションを示しています。 記述の中には、以下の用語を使用しているものがあります。

- 実際の文書エンコード
- 文書エンコード宣言
- 外部 *ASCII* コード・ページ
- 外部 *EBCDIC* コード・ページ

用語の定義については、XML 入力文書エンコードに関する関連概念を参照してください。

表 *73.* **続行可能な *XML PARSE* 例外**

例外コード (10 進数)	**説明**	**続行に関するパーサーのアクション**
1	パーサーで、エレメントの内容に含まれない空白文字を走査中に、無効文字が見つかりました。 空白文字の詳細については、XML 入力文書エンコードに関する概念を参照してください。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。 パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。

© Copyright IBM Corp. 2021

表 73. **続行可能な *XML PARSE* 例外** (続き)

例外コード (10 進数)	説明	続行に関するパーサーのアクション
2	パーサーで、エレメント内容に含まれない、処理命令、エレメント、コメント、または文書タイプ宣言の無効な開始が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。 パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
3	パーサーで、重複する属性名が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。 パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
4	パーサーで、属性値にマークアップ文字 '<' が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。 パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
5	エレメントの開始および終了タグ名が一致しません。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。 パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
6	パーサーで、エレメント内容に無効文字が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。 パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
7	パーサーで、エレメント内容に、エレメント、コメント、処理命令、または CDATA セクションの無効な開始が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。 パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
8	パーサーで、エレメント内容に、一致する開始文字シーケンス '<![CDATA[' のない、CDATA 終了文字シーケンス ']]>' が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。 パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
9	パーサーで、コメント内に無効文字が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。 パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
10	パーサーで、コメント内に、後にパーサーで、コメント内に、後に '>'が付いていない 文字シーケンス '--' (2 つのハイフン) が 見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。 パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。

表 73. **続行可能な *XML PARSE* 例外** (続き)

例外コード (10 進数)	説明	続行に関するパーサーのアクション
11	パーサーで、処理命令データ・セグメント内に無効文字が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。 パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
12	XML 宣言が文書の先頭にありませんでした。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。 パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
13	パーサーで、16 進文字参照 (形式 &#x*dddd*; の) 内に無効な数字が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。 パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
14	パーサーで、10 進数文字参照 (形式 &#*dddd*; の) 内に無効な数字が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。 パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
15	XML 宣言内のエンコード宣言値が小文字または大文字の A から Z で始まっていませんでした。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。 パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
16	文字参照が適切な XML 文字を参照しませんでした。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。 パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
17	パーサーで、エンティティー参照名に無効文字が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。 パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
18	パーサーで、属性値に無効文字が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。 パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
70	実際の文書エンコードは EBCDIC であり、 外部 EBCDIC コード・ページがサポートされていますが、文書エンコード宣言ではサポートされる EBCDIC コード・ページが指定されていませんでした。	パーサーは、外部 EBCDIC コード・ページで指定されたエンコードを使用します。

表 73. **続行可能な *XML PARSE* 例外** (続き)

例外コード (10 進数)	説明	続行に関するパーサーのアクション
71	実際の文書エンコードは EBCDIC であり、文書エンコード宣言ではサポートされる EBCDIC エンコードが指定されていましたが、外部 EBCDIC コード・ページがサポートされていません。	パーサーは、文書エンコード宣言で指定されたエンコードを使用します。
72	実際の文書エンコードは EBCDIC ですが、 外部 EBCDIC コード・ページがサポートされておらず、文書はエンコード宣言を含んでいませんでした。	パーサーは、EBCDIC コード・ページ 1140 (USA、カナダ、. . . ユーロ国別拡張コード・ページ) を使用します。
73	実際の文書エンコードは EBCDIC ですが、 外部 EBCDIC コード・ページおよび文書エンコード宣言のどちらにもサポートされる EBCDIC コード・ページが指定されていませんでした。	パーサーは、EBCDIC コード・ページ 1140 (USA、カナダ、. . . ユーロ国別拡張コード・ページ) を使用します。
80	実際の文書エンコードは ASCII であり、外部 ASCII コード・ページがサポートされていますが、文書エンコード宣言ではサポートされる ASCII コード・ページが指定されていませんでした。	パーサーは、外部 ASCII コード・ページで指定されたエンコードを使用します。
81	実際の文書エンコードは ASCII であり、文書エンコード宣言ではサポートされる ASCII エンコードが指定されていましたが、外部 ASCII コード・ページがサポートされていません。	パーサーは、文書エンコード宣言で指定されたエンコードを使用します。
82	実際の文書エンコードは ASCII でしたが、外部 ASCII コード・ページがサポートされておらず、文書にエンコード宣言が含まれていませんでした。	パーサーは、ASCII コード・ページ 819 (ISO-8859-1 Latin 1/オープン・システム) を使用します。
83	実際の文書エンコードは ASCII でしたが、外部 ASCII コード・ページと文書エンコード宣言の両方がサポートされる ASCII コード・ページを指定していませんでした。	パーサーは、ASCII コード・ページ 819 (ISO-8859-1 Latin 1/オープン・システム) を使用します。
84	実際の文書エンコードは ASCII でしたが無効な UTF-8 であり、外部コード・ページには UTF-8 が指定されていて、 文書にエンコード宣言が含まれていませんでした。	パーサーは UTF-8 を使用します。

表 73. **続行可能な *XML PARSE* 例外** (続き)		
例外コード (10 進数)	**説明**	**続行に関するパーサーのアクション**
85	実際の文書エンコードは ASCII でしたが無効な UTF-8 であり、外部コード・ページには UTF-8 が指定されていて、 文書エンコード宣言にはサポート対象の ASCII コード・ページも UTF-8 も指定されていませんでした。	パーサーは UTF-8 を使用します。
86	実際の文書エンコードは ASCII でしたが無効な UTF-8 であり、外部コード・ページにはサポート対象の ASCII コード・ページが指定されていて、 文書エンコード宣言には UTF-8 が指定されていました。	パーサーは UTF-8 を使用します。
87	実際の文書エンコードは ASCII でしたが無効な UTF-8 であり、外部コード・ページおよび文書エンコード宣言の両方に UTF-8 が指定されていました。	パーサーは UTF-8 を使用します。
88	実際の文書エンコードは ASCII でしたが無効な UTF-8 であり、外部コード・ページにはサポート対象の ASCII コード・ページも UTF-8 も指定されておらず、 文書エンコード宣言には UTF-8 が指定されていました。	パーサーは UTF-8 を使用します。
89	実際の文書エンコードは ASCII でしたが無効な UTF-8 であり、外部コード・ページには UTF-8 が指定されていて、 文書エンコード宣言にはサポート対象の ASCII コード・ページが指定されていました。	パーサーは UTF-8 を使用します。
92	文書データ項目は英数字でしたが、実際の文書エンコード は Unicode UTF-16 でした。	パーサーはコード・ページ 1200 (Unicode UTF-16) を使用します。
100,001 から 165,535	外部 EBCDIC コード・ページおよび文書エンコード宣言に指定された、サポートされる EBCDIC コード・ページが それぞれ異なっていました。 XML-CODE には、エンコード宣言に 100,000 をプラスするためのコード・ページ CCSID が含まれています。	EXCEPTION イベントから戻る前に、XML-CODE をゼロに設定した場合、パーサーでは、外部 EBCDIC コード・ページによって指定したエンコードが使用されます。 文書エンコード宣言に対して (100,000 を減算して) XML-CODE を CCSID に設定した場合、パーサーではこのエンコードが使用されます。
200,001 - 265,535	外部 ASCII コード・ページおよび 文書エンコード宣言に指定された、サポートされる ASCII コード・ページが それぞれ異なっていました。 XML-CODE には、エンコード宣言に 200,000 をプラスするための CCSID が含まれています。	EXCEPTION イベントから戻る前に、XML-CODE をゼロに設定した場合、パーサーでは、外部 ASCII コード・ページによって指定したエンコードが使用されます。 文書エンコード宣言に対して (200,000 を減算して) XML-CODE を CCSID に設定した場合、パーサーではこのエンコードが使用されます。

関連概念
395 ページの『XML-CODE』
398 ページの『XML 入力文書エンコード』

関連タスク
401 ページの『XML PARSE の例外処理』

継続を許可しない XML PARSE 例外

以下に記述した例外が発生すると、XML パーサーは処理を続行できません。

処理プロシージャーがパーサーに制御を戻す前に XML-CODE をゼロに設定していても、 これらの例外について、パーサーからこれ以上のイベントは返されません。 ON EXCEPTION 句が指定されている場合、パーサーは、この句のステートメントに制御を渡します。 指定されていない場合は、XML PARSE ステートメントの最後に制御を渡します。

表 *74.* **継続を許可しない *XML PARSE* 例外**

例外コード (10 進数)	**説明**
100	パーサーで、XML 宣言の開始を走査中に、文書の末尾に達しました。
101	パーサーで、XML 宣言の末尾を走査中に、文書の末尾に達しました。
102	パーサーで、ルート・エレメントを走査中に、文書の末尾に達しました。
103	パーサーで、XML 宣言内のバージョン情報を走査中に、文書の末尾に達しました。
104	パーサーで、XML 宣言内のバージョン情報値を走査中に、文書の末尾に達しました。
106	パーサーで、XML 宣言内のエンコード宣言値を走査中に、文書の末尾に達しました。
108	パーサーで、XML 宣言内の standalone 宣言値を走査中に、文書の末尾に達しました。
109	パーサーで、属性名を走査中に、文書の末尾に達しました。
110	パーサーで、属性値を走査中に、文書の末尾に達しました。
111	パーサーで、属性値内の文字参照またはエンティティー参照を走査中に、文書の末尾に達しました。
112	パーサーで、空のエレメント・タグを走査中に、文書の末尾に達しました。
113	パーサーで、ルート・エレメント名を走査中に、文書の末尾に達しました。
114	パーサーで、エレメント名を走査中に、文書の末尾に達しました。
115	パーサーで、エレメント内容の文字データを走査中に、文書の末尾に達しました。
116	パーサーで、エレメント内容の処理命令を走査中に、文書の末尾に達しました。
117	パーサーで、エレメント内容のコメントまたは CDATA セクションを走査中に 文書の末尾に達しました。
118	パーサーで、エレメント内容のコメントを走査中に文書の末尾に達しました。
119	パーサーで、エレメント内容の CDATA セクションを走査中に文書の末尾に達しました。
120	パーサーで、エレメント内容の文字参照またはエンティティー参照を走査中に 文書の末尾に達しました。
121	パーサーで、ルート・エレメントの末尾を走査中に、文書の末尾に達しました。
122	パーサーで、文書タイプ宣言の無効の可能性のある開始が見つかりました。
123	パーサーで、2 つ目の文書タイプ宣言が見つかりました。

表 74. **継続を許可しない *XML PARSE* 例外** (続き)	
例外コード (10 進数)	**説明**
124	ルート・エレメントの先頭文字が、文字、'_'、または ':' ではありませんでした。
125	エレメントの先頭の属性名の先頭文字が、文字、'_'、または ':' ではありませんでした。
126	パーサーで、エレメント名内に、またはエレメント名の後のいずれかに無効文字が 見つかりました。
127	パーサーで、属性名の後に '=' 以外の文字が見つかりました。
128	パーサーで、無効な属性値区切り文字が見つかりました。
130	属性名の先頭文字が、文字、'_'、または ':' ではありませんでした。
131	パーサーで、属性名内に、または属性名の後のいずれかに無効文字が 見つかりました。
132	空のエレメント・タグが、'/' の後に続く '>' で終了しませんでした。
133	エレメント終了タグ名の先頭文字が、文字、'_'、または ':' ではありませんでした。
134	エレメント終了タグ名が '>' で終了しませんでした。
135	エレメント名の先頭文字が、文字、'_'、または ':' ではありませんでした。
136	パーサーで、エレメント内容に、コメントまたは CDATA セクションの無効な開始が見つかりました。
137	パーサーで、コメントの無効な開始が見つかりました。
138	処理命令ターゲット名の先頭文字が、文字、'_'、または ':' ではありませんでした。
139	パーサーで、処理命令ターゲット名内に、または処理命令ターゲット名の後のいずれかに無効文字が見つかりました。
140	処理命令が終了文字シーケンス '?>' で終了しませんでした。
141	パーサーで、文字参照またはエンティティー参照内の '&' の後に 無効文字が見つかりました。
142	バージョン情報が XML 宣言にありませんでした。
143	XML 宣言内の 'version' の後に '=' がありませんでした。
144	XML 宣言内のバージョン宣言値が欠落しているか、または不適切に区切られています。
145	XML 宣言内のバージョン情報値が不適切な文字を指定したか、または 開始と終了の区切り文字が一致しませんでした。
146	パーサーで、XML 宣言内のバージョン情報値の終了区切り文字の後に 無効文字が見つかりました。
147	パーサーで、XML 宣言にオプションのエンコード宣言ではない、無効な属性が見つかりました。
148	XML 宣言内の 'encoding' の後に '=' がありませんでした。
149	XML 宣言内のエンコード宣言値が欠落しているか、または不適切に区切られています。
150	XML 宣言内のエンコード宣言値が不適切な文字を指定したか、または 開始と終了の区切り文字が一致しませんでした。
151	パーサーで、XML 宣言内のエンコード宣言値の終了区切り文字の後に 無効文字が見つかりました。

表 74. **継続を許可しない *XML PARSE* 例外** (続き)	
例外コード (10 進数)	**説明**
152	パーサーで、XML 宣言にオプションの standalone 宣言ではない、無効な属性が見つかりました。
153	XML 宣言内の standalone の後に = がありませんでした。
154	XML 宣言内の standalone 宣言値が欠落しているか、または不適切に区切られています。
155	standalone 宣言値が 'yes' または 'no' 以外の値になっていました。
156	XML 宣言内の standalone 宣言値が不適切な文字を指定したか、または 開始と終了の区切り文字が一致しませんでした。
157	パーサーで、XML 宣言内の standalone 宣言値の終了区切り文字の後に 無効文字が見つかりました。
158	XML 宣言が正しい文字シーケンス '?>' で終了しなかったか、無効属性が含まれていました。
159	パーサーで、ルート・エレメントの末尾の後に 文書タイプ宣言の開始が見つかりました。
160	パーサーで、ルート・エレメントの末尾の後に エレメントの開始が見つかりました。
161	パーサーは、無効な UTF-8 バイト・シーケンスを検出しました。
162	パーサーは、x'FFFF' より大きな Unicode スカラー値を持つ UTF-8 文字を検出しました。
315	実際の文書エンコードは、UTF-16 リトル・エンディアンですが、パーサーはこのプラットフォームではリトル・エンディアンをサポートしません。
316	実際の文書エンコードは UCS4 ですが、パーサーは UCS4 をサポートしません。
317	パーサーで、文書エンコードを判別できません。 文書は破損している可能性があります。
318	実際の文書エンコードは UTF-8 ですが、パーサーは UTF-8 をサポートしません。
320	文書データ項目は国別でしたが、実際の文書エンコード は EBCDIC でした。
321	文書データ項目は国別でしたが、実際の文書エンコードは ASCII でした。
322	文書データ項目はネイティブ英数字データ項目でしたが、実際の文書エンコードは EBCDIC でした。
323	文書データ項目はホスト英数字データ項目でしたが、実際の文書エンコードは ASCII でした。
324	文書データ項目は国別でしたが、実際の文書エンコード は UTF-8 でした。
325	文書データ項目はホスト英数字データ項目でしたが、実際の文書エンコードは UTF-8 でした。
500 - 599	内部エラー。エラーをサービス担当者に報告してください。

関連概念
395 ページの『XML-CODE』

関連タスク
401 ページの『XML PARSE の例外処理』

XML 準拠

COBOL for Linux に 含まれる組み込み COBOL XML パーサーは、*XML specification* の 定義によると、XML 準拠のプロセッサーではありません。 パーサーは、構文解析する XML 文書の妥当性検査を 行いません。 XML パーサーは、各種の整形式性エラーの検査は行いますが、妥当性検証をしない XML プロセッサーに必要とされるアクションのすべてを行うわけではありません。

特に、XML パーサーは、内部文書タイプ定義 (DTD 内部サブセット) を処理しません。 したがって、XML パーサーは、デフォルト属性値の提供、属性値の正規化、および事前定義エンティティーを除く、内部エンティティーの置換テキストの組み込みを行いません。 ただし、XML パーサーは、文書タイプ宣言全体を DOCUMENT-TYPE-DECLARATION XML イベントの XML-TEXT または XML-NTEXT の内容として 渡します。 したがって、アプリケーションは、必要に応じて、これらのアクションを実行することができます。

オプションとして、パーサーを使用すると、プログラムはエラー後に XML 文書の処理を 続行することができます。 処理の続行を可能にする目的は、XML 文書および処理プロシージャーのデバッグを容易にすることです。

XML specification での定義を要約すると、以下のような、テキスト・オブジェクトは整形式の XML 文書です。

- 概して言えば、XML 文書の文法に準拠している。
- *XML specification* にリストされた明示的な整形式性制約をすべて満たす。
- 文書内で直接的または間接的に参照する、構文解析したエンティティー (テキストのセグメント) がそれぞれ、整形式である。

COBOL XML パーサーでは、文書タイプ宣言を除き、文書が XML 文法に準拠しているかを検査します。 文書タイプ宣言は、 チェックされない状態でそのままアプリケーションに渡されます。

次の情報は、*XML specification* からの注釈です。 オリジナル URL (www.w3.org/TR/REC-xml) に 存在しない内容については、W3C は責任を負いません。 注釈はすべて、仕様には含まれておらず、イタリック体で記述されています。

Copyright © 1994-2001 W3C® (マサチューセッツ工科大学、フランス国立情報学自動制御研究所、慶応義塾大学)、All Rights Reserved. W3C の責任、商標、文書使用、およびソフトウェア・ライセンスの規則が適用されます。 (www.w3.org/Consortium/Legal/ipr-notice-20000612)

また、*XML specification* には、12 個の明示的な整形式性制約が 記述されています。 COBOL XML パーサーが部分的または全面的に検査する制約事項は**太字**で 記述されています。

1. 内部サブセット内のパラメーター・エンティティー (PE): 「内部 DTD サブセットの中では、パラメーター・エンティティー参照は、マークアップ宣言の内部ではなく、マークアップ宣言が発生できる場所でしか発生できない。 (この制約は、外部パラメーター・エンティティーの中で発生する参照や 外部サブセットには適用されない。)」

   パーサーは内部 *DTD* サブセットを処理しないので、この制約を強制しません。
2. 外部サブセット:「外部サブセットが存在している場合は、そのプロダクションを extSubset に 一致させなくてはならない。」

   パーサーは外部サブセットを処理しないので、この制約を強制しません。
3. 宣言と宣言の間のパラメーター・エンティティー:「DeclSep のパラメーター・エンティティー参照のテキストの置換はそのプロダクションを extSubsetDecl に一致させなくてはならない。」

   パーサーは内部 *DTD* サブセットを処理しないので、この制約を強制しません。
4. **エレメント・タイプの突き合わせ:** 「エレメントの終了タグの名前は、開始タグのエレメント・タイプと一致させなければならない。」

   パーサーはこの制約を強制します。
5. **属性指定の一意性:** 「同じ開始タグまたは空エレメント・タグの中に 1 回を超えて現れてよい属性名はない。」

パーサーで、一意性について、指定したエレメント内の最大 *10* までの属性名が検査され、この制約は部分的にサポートされます。 アプリケーションは、この限度を超える属性名を検査できます。

6. 外部エンティティー参照の禁止:「属性値は、外部エンティティーへの直接的または間接的エンティティー参照を含むことができない。」

   パーサーはこの制約を強制しません。

7. 属性値内の '<' の禁止:「属性値の中で直接的または間接的に参照するエンティティーの置換テキストは、'<' を含んではならない。」

   パーサーはこの制約を強制しません。

8. **正しい文字:**「文字参照の使用に言及する文字は、そのプロダクションを Char に一致させなくてはならない。」

   パーサーはこの制約を強制します。

9. エンティティーの宣言:「DTD のない文書、パラメーター・エンティティー参照が含まれない内部 DTD サブセットだけしかない文書、または standalone='yes' の文書においては、外部サブセットまたはパラメーター・エンティティー内で発生しないエンティティー参照に おいて与えられている Name が、外部サブセット、またはパラメーター・エンティティー内で発生しないエンティティー宣言の中の ものと一致しなければならない。 ただし、整形式文書は、次のエンティティー amp、lt、gt、apos、quot を宣言する必要はない。 一般的エンティティーの宣言は、属性リスト宣言内のデフォルト値に現れる、それに対するどの参照よりも先行しなければならない。」

   エンティティーが外部サブセットまたは外部パラメーター・エンティティーで宣言されている場合、妥当性検査をしないプロセッサーは、それらの宣言を読み取って処理するよう強制されないことに注意してください。そのような文書の場合、エンティティーを宣言しなければならないという規則は、standalone='yes' の場合にのみ 整形式性制約です。

   パーサーはこの制約を強制しません。

10. 解析済みエンティティー:「エンティティー参照は、解析対象外エンティティーの名前を含んではならない。 解析対象外エンティティーを参照してもよいのは、ENTITY 型または ENTITIES 型として宣言した属性値の中だけである。」

    パーサーはこの制約を強制しません。

11. 再帰の禁止:「解析されるエンティティーは、直接的または間接的を問わず、それ自身への再帰的参照を含めてはならない。」

    パーサーはこの制約を強制しません。

12. DTD 内:「パラメーター・エンティティー参照が出現してよいのは、DTD の中だけである。」

    このエラーは起こらないので、パーサーはこの制約を強制しません。

上記の情報は、*XML specification* からの注釈です。 オリジナル URL (www.w3.org/TR/REC-xml) に 存在しない内容については、W3C は責任を負いません。上記の注釈はすべて、仕様には含まれていません。 本文書は、W3C メンバーや他の関係者による校閲を受け、ディレクターによって W3C 推奨文書として 承認されています。 本文書は継続的に提供される文書であり、参照資料として使用されたり、別文書で仕様に含まれる資料として引用される可能性があります。 仕様の正規版は英語バージョンで、W3C のサイトで確認することができます。翻訳文書については、翻訳上の誤りが含まれる可能性があります。

関連概念
391 ページの『COBOL での XML パーサー』

関連参照

Extensible Markup Language (XML)
XML specification (Prolog and document type declaration)

XML GENERATE 例外

XML の生成時に、いずれかの例外コードが XML-CODE 特殊レジスターで戻されることがあります。 このような例外が発生すると、ON EXCEPTION 句で指定されたステートメント、または、ON EXCEPTION 句をコーディングしていない場合には、XML GENERATE ステートメントの末尾に制御が渡されます。

表 75. ***XML GENERATE* 例外**

例外コード (10 進数)	**説明**
400®	受信は小さすぎて、生成された XML 文書を入れられませんでした。 指定されていれば、COUNT IN データ項目に、実際に生成された文字位置のカウントが格納されています。
401	マルチバイト・データ名は、Unicode への変換時に XML エレメントまたは属性名では無効な文字を含んでいました。
402	Unicode への変換時に、マルチバイト・データ名の先頭文字は、XML エレメントまたは属性名の先頭文字としては無効なものでした。
403	OCCURS DEPENDING ON 変数の値が 16,777,215 を超えました。
410	EBCDIC_CODEPAGE 環境変数で指定された CCSID ページは、 Unicode への変換に関してはサポートされません。
411	EBCDIC_CODEPAGE 環境変数で指定された CCSID は、サポート対象の 1 バイト EBCDIC コード・ページではありません。
412	受け取り側はネイティブ英数字でしたが、文書に対して指定された エンコードは UTF-8 でもサポート対象の 1 バイト ASCII コード・ページでもありませんでした。
413	受け取り側は英数字でしたが、ランタイム・ロケールがコンパイル時ロケールと一致していませんでした。
414	XML 文書に指定されたエンコードが無効だったか、またはサポートされるコード・ページではありませんでした。
415	受け取り側は国別でしたが、文書に対して指定されたエンコードは UTF-16 ではありませんでした。
416	XML 名前空間 ID に無効な XML 文字が含まれていました。
417	エレメント文字内容または属性値に XML コンテンツでは正しくない文字が含まれていました。 文書内の「hex」が接頭部のエレメント・タグ名または属性名および元のデータ値を 16 進表記して、XML の生成を続行しました。
418	置換文字がエンコード変換で生成されました。
419	XML 名前空間接頭部が無効でした。
420	ソース・データ項目にはマルチバイトの名前または内容が含まれていて、受け取り側はネイティブ英数字でしたが、 文書に対して指定されたエンコードは UTF-8 ではありませんでした。
600-699	内部エラー。エラーをサービス担当者に報告してください。

関連タスク

414 ページの『XML GENERATE 例外の処理』

関連参照

付録 F EXIT コンパイラー・オプション

EXIT コンパイラー・オプションを使用して、各種コンパイラー関数に代わるユーザー提供モジュールを指定できます。 各出口モジュールの処理、出口モジュールのエラー処理、または CICS ステートメントおよび SQL ステートメントでの EXIT オプションの使用方法について詳しくは、以下のトピックを参照してください。

関連参照

589 ページの『ユーザー出口作業域と作業域拡張 (work area extension)』
589 ページの『出口モジュールのパラメーター・リスト』
591 ページの『INEXIT の処理』
592 ページの『LIBEXIT の処理』
592 ページの『PRTEXIT の処理』

593 ページの『MSGEXIT の処理』
600 ページの『出口モジュールでのエラー処理』
265 ページの『EXIT』

ユーザー出口作業域と作業域拡張 (work area extension)

ユーザー出口を使用するとき、コンパイラーは作業域と作業域拡張 (work area extension)を提供します。 これらには、出口モジュールが取得したストレージのアドレスを保存できます。 こうした作業域により、モジュールを再入可能にすることができます。

ユーザー出口作業域 (INEXIT、LIBEXIT、および PRTEXIT によって使用) は、フルワード境界に位置する 4 つのフルワードから構成されます。 ユーザー出口作業域拡張 (MSGEXIT によって使用) も、フルワード境界上の 8 つのフルワードから構成されます。 これらのフルワードは、最初の出口ルーチンが呼び出される前に、2 進ゼロに初期化され、パラメーター・リストで出口モジュールに渡されます。 初期化後、コンパイラーは作業域に参照を行いません。

関連参照
591 ページの『INEXIT の処理』
592 ページの『LIBEXIT の処理』
592 ページの『PRTEXIT の処理』

593 ページの『MSGEXIT の処理』

出口モジュールのパラメーター・リスト

コンパイラーは、参照で渡される構造を使用して、出口モジュールと連絡します。

表 76. **出口モジュールのパラメーター・リスト**

パラメーター・オフセット	含まれる項目	項目の説明
0	ユーザー出口タイプ	操作を実行するユーザー出口を識別するハーフワード: • 1=INEXIT • 2=LIBEXIT • 3=PRTEXIT • 5=予約済み • 6=MSGEXIT

© Copyright IBM Corp. 2021

表 76. **出口モジュールのパラメーター・リスト** (続き)

パラメーター・オフセット	含まれる項目	項目の説明
2	命令コード	操作のタイプを示すハーフワード。 • 0=OPEN • 1=CLOSE • 2=GET • 3=PUT • 4=FIND • 5=MSGSEV: メッセージ重大度をカスタマイズします
4	戻りコード	出口モジュールによって設定されるフルワードで、要求された操作の成功を示します。 命令コード 0 から 4 の場合: • 0=成功 • 4= データの終わり • 12=失敗 命令コード 5 の場合: • 0= メッセージはカスタマイズされませんでした • 4= メッセージが検出され、カスタマイズされました • 12= 操作は失敗しました
8	レコード長	出口モジュールが設定するフルワードで、GET 操作によって戻されるレコード、または PUT 操作によって与えられるレコードの長さを示します。
12	レコードのアドレスまたは *str2*	出口モジュールにより、GET 操作の際、ユーザー所有のバッファー内のレコードのアドレスに設定されるフルワード、または、コンパイラーにより PUT 操作のレコードのアドレスに設定されるフルワード。 *str2* は OPEN にのみ適用されます。 最初のハーフワード (ハーフワード境界上の) にストリングの長さが入り、その後にストリングが続きます。
16	ユーザー出口作業域	ユーザー出口モジュールで使用できるように、コンパイラーが提供する 4 フルワードの作業域: • 最初のワード: INEXIT が使用 • 2 番目のワード: LIBEXIT が使用 • 3 番目のワード: PRTEXIT が使用
32	*Text-name*	完全修飾 *text-name* を含むヌル終了ストリングのアドレスが入るフルワード。 FIND にのみ適用されます。 (LIBEXIT の専用)

表 76. **出口モジュールのパラメーター・リスト** (続き)		
パラメーター・オフセット	**含まれる項目**	**項目の説明**
36	ユーザー出口パラメーター・ストリング	6 エレメント配列のアドレスが入るフルワード。各エレメントは、2 バイト長のフィールドの後に、出口パラメーター・ストリングを含む 64 文字のストリングが続く構造になっています。 6 番目のエレメントは、MSGEXIT ストリングです。
40	ソース・コード行のタイプ	ハーフワード (INEXIT の専用)
42	ステートメント標識	ハーフワード (INEXIT の専用)
44	ステートメント桁番号	ハーフワード (INEXIT の専用)
46	予約済み	ハーフワード
48	ユーザー出口作業域拡張	ユーザー出口モジュールで使用できるように、コンパイラーが提供する 8 フルワードの作業域: • 最初のワード: 予約済み • 2 番目のワード: MSGEXIT
80	メッセージ出口データ	コンパイラーが提供する 3 ハーフワードの領域。 • 最初のハーフワード: カスタマイズするメッセージのメッセージ番号 • 2 番目のハーフワード: 診断メッセージの場合、デフォルトの重大度。 FIPS メッセージの場合、数字コードによる FIPS カテゴリー • 3 番目のハーフワード: ユーザーが要求したメッセージ重大度 (-1 は抑制を示します)

関連参照
589 ページの『ユーザー出口作業域と作業域拡張 (work area extension)』

INEXIT の処理

INEXIT を指定すると、コンパイラーは初期化時に出口モジュール (*mod1*) をロードし、OPEN 命令コードを使用してモジュールを呼び出します。 次に、モジュールは、処理を行うためのソースを準備し、OPEN 要求の状況をコンパイラーに戻すことができます。 その後は、コンパイラーがソース・ステートメントを要求するたびに、GET 命令コードによって出口モジュールが呼び出されます。

出口モジュールは、次のステートメントのアドレスと長さ、またはソース・ステートメントがそれ以上存在しない場合は、データ終了標識のいずれかを戻します。 データ終了が発生すると、コンパイラーは CLOSE 命令コードを使用して出口モジュールを呼び出し、モジュールがその入力に関係するリソースをすべて解放できるようにします。

コンパイラーは、参照で渡される構造を使用して、出口モジュールと連絡します。

関連参照
589 ページの『出口モジュールのパラメーター・リスト』

LIBEXIT の処理

LIBEXIT を指定すると、コンパイラーは初期化時に出口モジュール (*mod2*) をロードします。 コンパイラーは、COPY または BASIS ステートメントが検出されるたびに、このモジュールを呼び出してコピーブックを入手します。

最初の呼び出しでは、OPEN 命令コードを使用してモジュールが呼び出されます。 次に、モジュールは、指定された *library-name* を処理のために準備します。 新規の *library-name* が初めて指定された場合にも、OPEN 命令コードが発行されます。 出口モジュールは、OPEN 要求の状況を戻りコードによってコンパイラーに渡します。

OPEN 命令コードによって呼び出された出口が戻されると、コンパイラーは GET 命令コードを使用して出口モジュールを呼び出し、出口モジュールはアクティブ・コピーブックからコピーされるレコードの長さとアドレスをコンパイラーに渡します。GET 操作は、データ終了標識がコンパイラーに渡されるまで繰り返されます。

データ終了が発生すると、コンパイラーは CLOSE 要求を発行して、出口モジュールがその入力に関係するリソースをすべて解放できるようにします。

ネスト済み COPY ステートメント: アクティブ・コピーブックからの レコードに、COPY ステートメントを含めることができます。 (ただし、ネストされた COPY ステートメントに REPLACING 句を含めたり、REPLACING 句を持つ COPY ステートメントに、ネストされた COPY ステートメントを含めたりすることはできません。) 有効な、ネストされた COPY ステートメントが検出されると、コンパイラーは OPEN を発行してから、LIBEXIT から EOD を受け取るまで一連の GET を発行します。

text-name の再帰呼び出しは行えません。 つまり、コピーブックは、そのコピーブックのデータの終わりに達するまでの間、ネストされた一連の COPY ステートメントの中で一度しか指定できません。

出口モジュールは、OPEN 要求を受け取ると、アクティブ・コピーブックに関する制御情報をスタックにプッシュしてから、OPEN により要求された操作を完了します。新しく要求された *text-name* (または *basis-name*) がアクティブ・コピーブックになります。

データ終了標識がコンパイラーに渡されるまで、一連の GET 要求を使用した通常の方法で処理が続けられます。

注 : コンパイラーが GET を発行すると、そのコンパイラーが CLOSE コマンドを発行するまで、ユーザー出口は継続してデータ終了標識を渡す必要があります。

ネストされたアクティブ・コピーブックのデータ終了が検出され、コンパイラーにより CLOSE 操作が発行されると、出口モジュールはその制御情報をスタックからポップします。 コンパイラーからの次の要求は GET です。コンパイラーによる GET の発行が終わったところから、ユーザー出口は処理を継続する必要があります。

コンパイラーが GET 要求を使用して出口モジュールを呼び出したら、出口モジュールはこのコピーブックから前に渡された同一レコードを渡す必要があります。 同一レコードが渡されたことをコンパイラーが検証すると、データ終了標識が渡されるまで、GET 要求を使用して処理が続けられます。

コンパイラーは、参照で渡される構造を使用して、出口モジュールと連絡します。

関連参照
589 ページの『出口モジュールのパラメーター・リスト』

PRTEXIT の処理

PRTEXIT を指定すると、コンパイラーは初期化時に出口モジュール (*mod3*) をロードします。 SYSPRINT ファイルの代わりに出口モジュールを使用します。

コンパイラーは、OPEN 命令コードを使用してこのモジュールを呼び出します。 次に、モジュールは、処理を行うための出力先を準備し、OPEN 要求の状況をコンパイラーに戻すことができます。 その後は、コンパイラーが行を印刷しようとするたびに、PUT 命令コードによって出口モジュールが呼び出されます。コンパイラーが印刷対象レコードのアドレスと長さを渡すと、出口モジュールは戻りコードによって PUT

要求の状況をコンパイラーに渡します。 印刷されるレコードの最初のバイトには、ANSI プリンター制御文字が入ります。

コンパイルが完了する前に、コンパイラーは CLOSE 命令コードを使用して出口モジュールを呼び出し、モジュールがその出力先に関係するリソースをすべて解放できるようにします。

コンパイラーは、参照で渡される構造を使用して、出口モジュールと連絡します。

関連参照
589 ページの『出口モジュールのパラメーター・リスト』

MSGEXIT の処理

MSGEXIT モジュールは、コンパイラー診断メッセージおよび FIPS メッセージのカスタマイズに使用します。 このモジュールは、メッセージ重大度を変更するか、またはメッセージを抑制することによって、メッセージをカスタマイズします。

MSGEXIT モジュールが FIPS メッセージに重大度を割り当てると、メッセージは診断メッセージに変換されます。 (メッセージは、リスト内の診断メッセージの要約に示されます。)

コンパイラー・リストの終わりにある MSGEXIT の要約に、重大度が変更されたメッセージの数と抑制されたメッセージの数が示されます。

表 77. ***MSGEXIT* の処理**

コンパイラーのアクション	**出口モジュールのアクション**
初期化時に出口モジュール (*mod5*) をロードします	
OPEN 命令コード (op コード) を使用してこの出口モジュールを呼び出します	オプションで *str5* を処理し、 OPEN 要求の状況をコンパイラーに渡します
コンパイラーが診断メッセージまたは FIPS メッセージを出そうとするときに、MSGSEV 命令コード (op コード) を使用して出口モジュールを呼び出します	次のいずれかのアクション。 • (戻りコードに 0 を設定することによって) メッセージをカスタマイズしないことを示します。 • メッセージの新しい重大度 (または抑制) を指定し、戻りコードに 4 を設定します • (戻りコードに 12 を設定することによって) 操作が失敗したことを示します
CLOSE 命令コードを使用してこの出口モジュールを呼び出します	オプションでストレージを解放し、 CLOSE 要求の状況をコンパイラーに渡します
コンパイラー終了時に出口モジュール (*mod5*) を削除します	

596 ページの『例: MSGEXIT ユーザー出口』

関連タスク
594 ページの『コンパイラー・メッセージの重大度のカスタマイズ』

関連参照

589 ページの『出口モジュールのパラメーター・リスト』

コンパイラー・メッセージの重大度のカスタマイズ

コンパイラー・メッセージの重大度を変更したり、コンパイラー・メッセージ (FIPS メッセージを含む) を抑制したりするには、 以下のステップを行います。

1. ERRMSG という COBOL プログラムをコーディングしてコンパイルします。 関連タスクに説明されているように、プログラムに必要なのは、PROGRAM-ID 段落のみです。
2. コンパイラー・メッセージとそのメッセージ番号、重大度、およびメッセージ・テキストの全リストが入っている ERRMSG のリストを確認します。
3. カスタマイズするメッセージを決定します。

   可能なカスタマイズについては、カスタマイズ可能なコンパイラー・メッセージ重大度に関する関連参照を参照してください。
4. カスタマイズを実装する MSGEXIT モジュールをコーディングします。
   a. 命令コードのパラメーターがメッセージ重大度のカスタマイズを指示することを確認します。
   b. メッセージ出口データ・パラメーターで、メッセージ番号と、 診断メッセージの場合にはデフォルト重大度、または FIPS メッセージの場合には FIPS カテゴリーの 2 つの入力値を検査します。

      FIPS カテゴリーは、数字コードで表されます。 詳細については、カスタマイズ可能なコンパイラー・メッセージ重大度に関する関連参照を参照してください。
   c. カスタマイズするメッセージについて、メッセージ出口データ・パラメーターのユーザー要求の重大度が、以下のいずれかを示すように設定します。
      - 新しいメッセージ重大度。重大度 0、4、8、または 12 をコーディングします。
      - メッセージの抑制。重大度 -1 をコーディングします。
   d. 戻りコードを次のいずれかの値に設定します。
      - 0: メッセージがカスタマイズされなかったことを示します。
      - 4: メッセージが見つかり、カスタマイズされたことを示します。
      - 12: 操作が失敗し、コンパイルが終了することを示します。
5. MSGEXIT モジュールをコンパイルし、リンクします。

   モジュールが共用ライブラリーとしてリンクされるようにします。 次に例を示します。

   ```
   cob2 -o IGYMGXT -q32 IGYMSGXT.cbl -e IGYMSGXT
   ```
6. LD_LIBRARY_PATH を設定して、MSGEXIT モジュールがコンパイラーで使用可能になるようにします。

   例えば、共有オブジェクトが /u1/cobdev/exits にある場合は、次のコマンドを使用します。

   ```
   export LD_LIBRARY_PATH=/u1/cobdev/exits:$LD_LIBRARY_PATH
   ```
7. コンパイラー・オプション EXIT(MSGEXIT(*msgmod*)) を使用してプログラム ERRMSG を再コンパイルします。ここで、*msgmod* は MSGEXIT モジュールの名前です。
8. リストを確認し、以下を検査します。
   - 更新されたメッセージ重大度
   - 抑制されたメッセージ (重大度の代わりに XX で示されます)
   - サポートされていない重大度変更、またはサポートされていないメッセージ抑制 (重大度 U の診断メッセージと、戻りコード 16 のコンパイラー終了で示されます)

関連タスク
231 ページの『コンパイラー・メッセージのリストの生成』

関連参照
220 ページの『ランタイム環境変数』
230 ページの『コンパイラー診断メッセージの重大度コード』
595 ページの『カスタマイズ可能なコンパイラー・メッセージ重大度』

596 ページの『メッセージ・カスタマイズがコンパイル戻りコードに及ぼす影響』
600 ページの『出口モジュールでのエラー処理』

カスタマイズ可能なコンパイラー・メッセージ重大度

コンパイラー・メッセージの重大度をカスタマイズするには、 コンパイラー診断メッセージに可能な重大度、FIPS メッセージのレベルまたはカテゴリー、およびメッセージ重大度に関して可能なカスタマイズについて理解する必要があります。

コンパイラー診断メッセージに可能な重大度コードについては、重大度コードに関する関連参照に説明があります。

次の表に、FIPS (FLAGSTD) メッセージの 8 つのカテゴリーを示します。 FIPS メッセージのカテゴリーは、数字コードで MSGEXIT モジュールに渡されます。 2 番目の列にその数字コードが示されています。

表 *78.* ***FIPS (FLAGSTD)* メッセージのカテゴリー**

FIPS レベルまたはカテゴリー	**数字コード**	**説明**
D	81	デバッグ・モジュール・レベル 1
E	82	拡張 (IBM)
H	83	上位レベル
I	84	中間レベル
N	85	分割モジュール・レベル 1
O	86	廃止されるエレメント
Q	87	上位レベルの廃止されるエレメント
S	88	分割モジュール・レベル 2

FIPS メッセージの暗黙の重大度はゼロ (重大度 I) です。

メッセージ重大度の可能なカスタマイズ:

コンパイラー・メッセージの重大度は、次のように変更できます。

- 重大度 I および重大度 W のコンパイラー診断メッセージ、および FIPS メッセージは、I から S までの重大度に変更できます。

  FIPS メッセージに重大度を割り当てると、FIPS メッセージは、割り当てられた重大度の診断メッセージに変換されます。

  例として、以下のことを行うことができます。

  - 最適化プログラムの警告を重大度 I に下げます。
  - メッセージ 1154 の重大度を上げることによって、小さい項目のより大きい項目による REDEFINING を禁止します。
  - FIPS メッセージ 8235 をカテゴリー E の FIPS メッセージから重大度 S のコンパイラー診断メッセージに変更することによって、 複雑な OCCURS DEPENDING ON を禁止します。
- 重大度 E のメッセージは、プログラムでエラー条件が発生しているため、重大度 S に上げることはできますが、重大度 I または W に下げることはできません。
- 重大度 S および重大度 U のメッセージは、別の重大度に変更できません。

コンパイラー・メッセージの抑制は、次のように要求できます。

- I、W、および FIPS のメッセージは抑制できます。
- E および S のメッセージは抑制できません。

関連参照
230 ページの『コンパイラー診断メッセージの重大度コード』
268 ページの『FLAGSTD』
596 ページの『メッセージ・カスタマイズがコンパイル戻りコードに及ぼす影響』

メッセージ・カスタマイズがコンパイル戻りコードに及ぼす影響

MSGEXIT モジュールを使用した場合、プログラム・コンパイルの最終戻りコードが以下のような影響を受けることがあります。

メッセージの重大度を変更すると、コンパイルの戻りコードも変わる場合があります。 例えば、コンパイルで 1 件の診断メッセージが生成され、それが重大度 E のメッセージである場合、 コンパイルの戻りコードは通常 8 になります。 しかし、MSGEXIT モジュールでそのメッセージの重大度が重大度 S に変更されると、コンパイルの戻りコードは 12 になります。

メッセージを抑制した場合、コンパイルの戻りコードは、そのメッセージの重大度に影響されなくなります。 例えば、コンパイルで 1 件の診断メッセージが生成され、それが重大度 W のメッセージである場合、コンパイルの戻りコードは通常 4 になります。 しかし、MSGEXIT モジュールでそのメッセージが抑制されると、コンパイルの戻りコードは 0 になります。

関連タスク
594 ページの『コンパイラー・メッセージの重大度のカスタマイズ』

関連参照
230 ページの『コンパイラー診断メッセージの重大度コード』

例: MSGEXIT ユーザー出口

次の例は、メッセージ重大度を変更し、メッセージを抑制する MSGEXIT ユーザー出口モジュールを示しています。

また、上記の例の完全なソース・コードが COBOL インストール・ディレクトリーの samples サブディレクトリーに入っています (通常は /opt/ibm/cobol/1.1.0/samples/msgexit)。

メッセージ出口モジュールの使用に役立つヒントについては、コード内のコメントを参照してください。

```
*****************************************************************
*  IGYMSGXT - Sample COBOL program for MSGEXIT                  *
*****************************************************************
*  Function:  This is a SAMPLE user exit for the MSGEXIT        *
*             suboption of the EXIT compiler option.  This exit *
*             can be used to customize the severity of or       *
*             suppress compiler diagnostic messages and FIPS    *
*             messages.  This example program includes several  *
*             sample customizations to show how customizations  *
*             are done.  Feel free to change the customizations *
*             as appropriate to meet your requirements.         *
*                                                               *
*---------------------------------------------------------------*
*  COMPILE NOTE: To prepare a compiler user exit in COBOL,      *
*                it should be a shared library module:          *
*   cob2 -o IGYMSGXT -q32 IGYMSGXT.cbl -e IGYMSGXT              *
*                                                               *
*  USAGE NOTE: The compiler needs to have access to IGYMSGXT at *
*              compile time, so set LD_LIBRARY_PATH accordingly:*
*                                                               *
*   EX:       export LD_LIBRARY_PATH=/u1/cobdev/exits:          *
*             $LD_LIBRARY_PATH                                  *
*                                                               *
*               (This assumes the shared object is in           *
*                 /u1/cobdev/exits )                            *
*                                                               *
*****************************************************************
*****************************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  IGYMSGXT.
 DATA DIVISION.

   WORKING-STORAGE SECTION.
```

```
*****************************************************************
*                                                               *
*   Local variables.                                            *
*                                                               *
*****************************************************************

    77 EXIT-TYPEN            PIC 9(4).
    77 EXIT-DEFAULT-SEV-FIPS PIC X.

*****************************************************************
*                                                               *
*   Definition of the User-Exit Parameter List, which is        *
*   passed from the COBOL compiler to the user-exit module      *
*                                                               *
*****************************************************************

   LINKAGE SECTION.
    01 UXPARM.
       02 EXIT-TYPE        PIC 9(4)   COMP.
       02 EXIT-OPERATION   PIC 9(4)   COMP.
       02 EXIT-RETURNCODE  PIC 9(9)   COMP.
       02 EXIT-DATALENGTH  PIC 9(9)   COMP.
       02 EXIT-DATA        POINTER.
       02 EXIT-WORK-AREA.
          03 EXIT-WORK-AREA-PTR  OCCURS 4  POINTER.
       02 EXIT-TEXT-NAME   POINTER.
       02 EXIT-PARMS       POINTER.
       02 EXIT-LINFO       PIC X(8).
       02 EXIT-X-WORK-AREA PIC X(4) OCCURS 8.
       02 EXIT-MESSAGE-PARMS.
          03 EXIT-MESSAGE-NUM PIC 9(4)   COMP.
          03 EXIT-DEFAULT-SEV PIC 9(4)   COMP.
          03 EXIT-USER-SEV    PIC S9(4)  COMP.

    01 EXIT-STRINGS.
       02 EXIT-STRING OCCURS 6.
          03 EXIT-STR-LEN PIC 9(4)   COMP.
          03 EXIT-STR-TXT PIC X(64).

*****************************************************************
*                                                               *
*  Begin PROCEDURE DIVISION                                     *
*                                                               *
*  Invoke the section to handle the exit.                       *
*                                                               *
*****************************************************************

 Procedure Division Using UXPARM.

     Set Address of EXIT-STRINGS to EXIT-PARMS

     COMPUTE EXIT-RETURNCODE = 0

     Evaluate
TRUE

*****************************************************************
* Handle a bad invocation of this exit by the compiler.         *
* This could happen if this routine was used for one of the     *
* other EXITs, such as INEXIT, PRTEXIT or LIBEXIT.              *
*****************************************************************
       When EXIT-TYPE Not = 6
       Move EXIT-TYPE   to  EXIT-TYPEN
         Display '**** Invalid exit routine identifier'
         Display '**** EXIT TYPE =  '  EXIT-TYPE
         Compute EXIT-RETURNCODE = 16

*****************************************************************
* Handle the OPEN call to this exit by the compiler             *
*        Display the exit string (labeled 'str5' in the syntax  *
*        diagram in the COBOL for Linux Programming Guide) from   *
*        the EXIT(MSGEXIT('str5',mod5)) option specification.   *
*        (Note that str5 is placed in element 6 of the array of *
*        user exit parameter strings.)                          *
*****************************************************************
       When EXIT-OPERATION = 0
*        Display 'Opening MSGEXIT'
*        If EXIT-STR-LEN(6) Not Zero Then
*          Display ' str5 len = ' EXIT-STR-LEN(6)
*          Display ' str5 = ' EXIT-STR-TXT(6)(1:EXIT-STR-LEN(6))
*        End-If
         Continue
```

```
*****************************************************************
* Handle the CLOSE call to this exit by the compiler            *
*        NOTE: Unlike the z/OS MSGEXIT, you must not use        *
*             STOP RUN here.  On Linux, use GOBACK.               *
*****************************************************************
       When EXIT-OPERATION = 1
*        Display 'Closing MSGEXIT'
         Goback

*****************************************************************
* Handle the customize message severity call to this exit       *
*        Display information about every customized severity.   *
*****************************************************************
       When EXIT-OPERATION = 5
*        Display 'MSGEXIT called with MSGSEV'
         If EXIT-MESSAGE-NUM < 8000 Then
           Perform Error-Messages-Severity
         Else
           Perform FIPS-Messages-Severity
         End-If

*        If EXIT-RETURNCODE = 4 Then
*          Display '>>>> Customizing message ' EXIT-MESSAGE-NUM
*                  ' with new severity ' EXIT-USER-SEV '  <<<<'
*          If EXIT-MESSAGE-NUM > 8000 Then
*            Display 'FIPS sev =' EXIT-DEFAULT-SEV-FIPS '<<<<'
*          End-If
*        End-If

*****************************************************************
* Handle a bad invocation of this exit by the compiler          *
* The compiler should not invoke this exit with EXIT-TYPE = 6   *
* and an opcode other than 0, 1, or 5.  This should not happen  *
* and IBM service should be contacted if it does.               *
*****************************************************************
       When Other
         Display '**** Invalid MSGEXIT routine operation '
         Display '**** EXIT OPCODE =  '  EXIT-OPERATION
         Compute EXIT-RETURNCODE = 16

     End-Evaluate


Goback.


*****************************************************************
*    ERROR MESSAGE   PROCESSOR                                  *
*****************************************************************
 Error-Messages-Severity.

*    Assume message severity will be customized...
     COMPUTE EXIT-RETURNCODE = 4

     Evaluate EXIT-MESSAGE-NUM

*****************************************************************
*      Change severity of message 1154(W) to 12 ("S")
*      This is the case of redefining a large item
*      with a smaller item, IBM Req # MR0904063236
*****************************************************************
       When(1154)
         COMPUTE EXIT-USER-SEV = 12

*****************************************************************
*      Message severity Not customized
*****************************************************************
       When Other
         COMPUTE EXIT-RETURNCODE = 0

     End-Evaluate
     .
*****************************************************************
*    FIPS MESSAGE   PROCESSOR                                   *
*****************************************************************
 Fips-Messages-Severity.

*    Assume message severity will be customized...
     COMPUTE EXIT-RETURNCODE = 4

*    Convert numeric 'category' to character
```

```
     EVALUATE EXIT-DEFAULT-SEV
       When 81
         MOVE 'D' To EXIT-DEFAULT-SEV-FIPS
       When 82
         MOVE 'E' To EXIT-DEFAULT-SEV-FIPS
       When 83
         MOVE 'H' To EXIT-DEFAULT-SEV-FIPS
       When 84
         MOVE 'I' To EXIT-DEFAULT-SEV-FIPS
       When 85
         MOVE 'N' To EXIT-DEFAULT-SEV-FIPS
       When 86
         MOVE 'O' To EXIT-DEFAULT-SEV-FIPS
       When 87
         MOVE 'Q' To EXIT-DEFAULT-SEV-FIPS
       When 88
         MOVE 'S' To EXIT-DEFAULT-SEV-FIPS
       When Other
         Continue
     End-Evaluate

*****************************************************************
*  Examples of using FIPS category to force coding
*  restrictions.  These are not recommendations!
*****************************************************************
*      Change severity of all OBSOLETE item FIPS
*       messages to 'S'
*****************************************************************
*    If EXIT-DEFAULT-SEV-FIPS = 'O' Then
*      DISPLAY ">>>> Default customizing FIPS category "
*        EXIT-DEFAULT-SEV-FIPS " msg " EXIT-MESSAGE-NUM "<<<<"
*      COMPUTE EXIT-USER-SEV = 12
*    End-If

     Evaluate EXIT-MESSAGE-NUM
*****************************************************************
*      Change severity of message 8062(O) to 8 ("E")
*        8062 = GO TO without proc name
*****************************************************************
       When(8062)
*        DISPLAY ">>>> Customizing message 8062 with 8 <<<<"
*        DISPLAY 'FIPS sev =' EXIT-DEFAULT-SEV-FIPS '='
         COMPUTE EXIT-USER-SEV = 8

*****************************************************************
*      Change severity of message 8193(E) to 0("I")
*        8193 = GOBACK
*****************************************************************
       When(8193)
*        DISPLAY ">>>> Customizing message 8193 with 0 <<<<"
*        DISPLAY 'FIPS sev =' EXIT-DEFAULT-SEV-FIPS '='
         COMPUTE EXIT-USER-SEV = 0

*****************************************************************
*      Change severity of message 8235(E) to 8 (Error)
*      to disalllow Complex Occurs Depending On
*        8235 = Complex Occurs Depending On
*****************************************************************
       When(8235)
*        DISPLAY ">>>> Customizing message 8235 with 8 <<<<"
*        DISPLAY 'FIPS sev =' EXIT-DEFAULT-SEV-FIPS '='
         COMPUTE EXIT-USER-SEV = 08

*****************************************************************
*      Change severity of message 8270(O) to -1 (Suppress)
*        8270 = SERVICE LABEL
*****************************************************************
       When(8270)
*        DISPLAY ">>>> Customizing message 8270 with -1 <<<<"
*        DISPLAY 'FIPS sev =' EXIT-DEFAULT-SEV-FIPS '='
         COMPUTE EXIT-USER-SEV = -1

*****************************************************************
*      Message severity Not customized
*****************************************************************
       When Other
*        For the default set 'O' to 'S' case...
*        If EXIT-USER-SEV = 12 Then
*          COMPUTE EXIT-RETURNCODE = 4
*        Else
           COMPUTE EXIT-RETURNCODE = 0
*        End-If
```

```
    End-Evaluate
.
END PROGRAM IGYMSGXT.
```

出口モジュールでのエラー処理

ユーザー出口の処理中に、以下に記述した条件が発生することがあります。

出口ロードの失敗:

ユーザー出口の LOAD 要求が失敗すると、メッセージ IGYSI5207-U がオペレーター宛てに書き込まれます。

```
ユーザー出口 exit-name をロードしようとしたときにエラーが発生しました。
```

出口オープンの失敗:

ユーザー出口の OPEN 要求が失敗すると、メッセージ IGYSI5208-U がオペレーター宛てに書き込まれます。

```
ユーザー出口 exit-name をオープンしようとしたときにエラーが発生しました。
```

PRTEXIT PUT の失敗:

- メッセージ IGYSI5203-U がリストに書き込まれます。

```
PRTEXIT ユーザー出口への PUT 要求が失敗し、戻りコード nn が戻されました。
(A PUT request to the PRTEXIT user exit failed with return code nn.)
```

- メッセージ IGYSI5217-U がオペレーター宛てに書き込まれます。

```
PRTEXIT ユーザー出口 exit-name でエラーが発生しました。 コンパイラーは終了しました。
```

SYSIN GET の失敗:

以下のメッセージがリストに書き込まれることがあります。

- IGYSI5204-U:

```
レコード・アドレスが exit-name ユーザー出口によって設定されていません。
```

- IGYSI5205-U:

```
INEXIT ユーザー出口からの GET 要求が失敗し、戻りコード nn が戻されました。
```

- IGYSI5206-U:

```
レコード長が exit-name ユーザー出口によって設定されていません。
```

MSGEXIT の失敗:

カスタマイズの失敗: サポートされていない重大度変更、またはサポートされていないメッセージ抑制が試行されると、 メッセージ IGY*PP*5293-U がリストに書き込まれます。

```
MSGEXIT ユーザー出口 exit-name で、サポー
トされていないメッセージ重大度のカスタマイズが指定されました。
メッセージ番号、デフォルトの重大度、およびユーザーが指定した重大度は、mm、ds、us です。
MSGEXIT ユーザー出口 exit-name を変更してこのエラーを訂正してください。
(MSGEXIT user exit exit-name specified a message severity customization that is
```

```
not supported. The message number, default severity, and user-specified severity
were: mm, ds, us. Change MSGEXIT user exit exit-name to correct this error.)
```

一般障害: MSGEXIT モジュールが 4 以外の非ゼロ値を戻りコードに設定した場合、 メッセージ IGY*PP*5064-U がリストに書き込まれます。

```
MSGEXIT ユーザー出口ルーチン exit-name の呼び出しが失敗し、戻りコード nn が戻されました。
(A call to the MSGEXIT user exit routine exit-name failed with return code nn.)
```

MSGEXIT メッセージで、*PP* という 2 文字は、 MSGEXIT モジュールを呼び出すことになったメッセージを出したコンパイラー・フェーズを示します。

関連タスク
594 ページの『コンパイラー・メッセージの重大度のカスタマイズ』

付録 G ランタイム・メッセージ

COBOL for Linux のメッセージには、メッセージ接頭語、メッセージ番号、重大度コード、および説明テキストが含まれています。

メッセージ接頭語は、常に IWZ です。 重大度コードは、I (通知)、W (警告)、S (重大)、C (クリティカル) のいずれかになります。 メッセージ・テキストは、当該条件に関する簡単な説明です。

```
IWZ2519S  The seconds value in a CEEISEC call was not recognized.
```

上のメッセージ例を以下に説明します。

- メッセージ接頭語は IWZ です。
- メッセージ番号は 2519 です。
- 重大度コードは S です。
- メッセージ・テキストは「The seconds value in a CEEISEC call was not recognized.」です。

日時 の呼び出し可能サービスからのメッセージには、シンボリック・フィードバック・コードも含まれています。このコードは、12 バイトの条件トークンのうち、最初の 8 バイトを表しています。 シンボリック・フィードバック・コードは、条件のニックネームと考えることができます。 呼び出し可能サービスのメッセージには、4 桁のメッセージ番号が含まれています。

コマンド行からアプリケーションを実行する場合に、ランタイム・メッセージを取り込むには、stdout および stderr をファイルに転送します。 次に例を示します。

program-name program-arguments >combined-output-file 2>&1

次の例は、出力内容を別のファイルに書き込む方法を示しています。

program-name program-arguments >output-file 2>error-file

表 *79.* **ランタイム・メッセージ**

メッセージ番号	メッセージ・テキスト
610 ページの『IWZ006S』	行 *line-number* で、verb 番号 *verb-number* によるテーブル *table-name* への参照が、テーブル領域外の領域をアドレス指定しました。
610 ページの『IWZ007S』	行 *line-number* で、verb 番号 *verb-number* による可変長グループ *group-name* への参照が、グループの最大定義長を超える領域をアドレス指定しました。
611 ページの『IWZ012I』	ソートまたはマージの実行中に、無効な実行単位終了が発生しました。
611 ページの『IWZ013S』	ソートまたはマージが要求されましたが、別のスレッドでソートまたはマージを実行中です。
611 ページの『IWZ026W』	SORT-RETURN 特殊レジスターは参照されませんでしたが、現在の内容は、行番号 *line-number* でプログラム *program-name* のソートまたはマージ操作が失敗したことを示しています。 ソートまたはマージの戻りコードは *return code* です。
611 ページの『IWZ029S』	プログラム *program-name* の行 *line-number* にある関数 *function-name* の Argument-1 が 0 未満でした。
612 ページの『IWZ030S』	プログラム *program-name* の行 *line-number* にある関数 *function-name* の Argument-2 が正の整数ではありません。
612 ページの『IWZ036W』	プログラム *program-name* の行番号 *line-number* で、高位桁位置の切り捨てが発生しました。

© Copyright IBM Corp. 2021

表 79. **ランタイム・メッセージ** (続き)	
612 ページの『IWZ037I』	プログラム *program-name* の制御フローが、プログラムの最終行を超えました。 制御権は、プログラム *program-name* の呼び出し元に戻りました。
612 ページの『IWZ038S』	行 *line-number* にある *reference-modification-value* の参照変更長の値が 1 ではないことが、データ項目 *data-item* への参照内で検出されました。
613 ページの『IWZ039S』	無効なオーバーパンチ符号が検出されました。
613 ページの『IWZ040S』	無効な分離符号が検出されました。
613 ページの『IWZ048W』	指数式で、負の基数が小数で累乗されました。 基数の絶対値が使用されました。
613 ページの『IWZ049W』	指数式で、0 の基数が 0 で累乗されました。 結果は 1 に設定されました。
614 ページの『IWZ050S』	指数式で、0 の基数が負数で累乗されました。
614 ページの『IWZ051S』	オペランドまたは受信側で指定された小数点位が多すぎるため、プログラム *program-name* の固定小数点指数演算に有効桁が残っていません。
614 ページの『IWZ053S』	浮動小数点への変換時にオーバーフローが発生しました。
614 ページの『IWZ054S』	浮動小数点例外が発生しました。
615 ページの『IWZ055W』	浮動小数点への変換時にアンダーフローが発生しました。 結果は 0 に設定されました。
615 ページの『IWZ058S』	指数オーバーフローが発生しました。
615 ページの『IWZ059W』	9 桁を超える指数が切り捨てられました。
615 ページの『IWZ060W』	高位桁位置の切り捨てが発生しました。
615 ページの『IWZ061S』	0 による除算が発生しました。
616 ページの『IWZ063S』	*program-name* の行番号 *line-number* にある数値編集送信フィールドで、無効な符号が検出されました。
616 ページの『IWZ064S』	コンパイル単位 *compilation-unit* でのアクティブ・プログラム *program-name* に対する再帰呼び出しが試行されました。
616 ページの『IWZ065I』	コンパイル単位 *compilation-unit* で、アクティブ・プログラム *program-name* のキャンセルが試行されました。
616 ページの『IWZ066S』	プログラム *program-name* の長さが、既存のレコード長と一致しませんでした。
616 ページの『IWZ071S』	行 *line-number* で、verb 番号 *verb-number* によるテーブル *table-name* への ALL 添え字付きテーブル参照に、OCCURS DEPENDING ON 次元に対して指定された ALL 添え字があり、オブジェクトの値は 0 以下でした。
617 ページの『IWZ072S』	行 *line-number* で、*reference-modification-value* の参照変更開始位置の値が、データ項目 *data-item* の領域外の領域を参照しました。

表 79. **ランタイム・メッセージ** (続き)	
617 ページの『IWZ073S』	行 *line-number* にある *reference-modification-value* の参照変更長の値が正数ではないことが、データ項目 *data-item* の参照内で検出されました。
617 ページの『IWZ074S』	行 *line-number* にある *reference-modification-value* の参照変更開始位置の値と *length* の長さ値により、データ項目 *data-item* の右端文字を超える参照が行われました。
617 ページの『IWZ075S』	プログラム *program-name* の EXTERNAL ファイル *file-name* で矛盾が検出されました。 *attribute-1 attribute-2 attribute-3 attribute-4 attribute-5 attribute-6 attribute-7* の各ファイル属性が、設定済み外部ファイルのファイル属性と一致しませんでした。
618 ページの『IWZ076W』	INSPECT REPLACING CHARACTERS BY データ名の文字数が 1 ではありません。 先頭文字が使用されました。
618 ページの『IWZ077W』	INSPECT データ項目の長さが等しくありません。 短い方の長さが使用されました。
618 ページの『IWZ078S』	行 *line-number* で、verb 番号 *verb-number* によるテーブル *table-name* への ALL 添え字付き参照が、テーブルの上限を超えています。
618 ページの『IWZ096C』	メッセージには、次のような種類があります。 • プログラム *program-name* の動的呼び出しが失敗しました。 モジュール *module-name* のロードが失敗しました。エラー・コードは *error-code* です。 • プログラム *program-name* の動的呼び出しが失敗しました。 モジュール *module-name* のロードが失敗しました。戻りコードは *return-code* です。 • プログラム *program-name* の動的呼び出しが失敗しました。 リソースが不十分です。 • プログラム *program-name* の動的呼び出しが失敗しました。 環境内で COBPATH が検出されませんでした。 • プログラム *program-name* の動的呼び出しが失敗しました。 入力項目 *entry-name* が検出されませんでした。 • 動的呼び出しが失敗しました。 ターゲット・プログラムの名前に有効な文字が含まれていません。 • プログラム *program-name* の動的呼び出しが失敗しました。 ロード・モジュール *load-module* が、COBPATH 環境変数で識別されたディレクトリー内にありませんでした。
619 ページの『IWZ097S』	関数 *function-name* の Argument-1 に桁がありません。
619 ページの『IWZ100S』	関数 *function-name* の Argument-1 の値が -1 以下です。
619 ページの『IWZ103S』	関数 *function-name* の Argument-1 の値が 0 未満か、または 99 を超えています。
619 ページの『IWZ104S』	関数 *function-name* の Argument-1 の値が 0 未満か、または 99999 を超えています。
620 ページの『IWZ105S』	関数 *function-name* の Argument-1 の値が 0 未満か、または 999999 を超えています。
620 ページの『IWZ151S』	関数 *function-name* の Argument-1 が 18 桁を超えています。
620 ページの『IWZ152S』	関数 *function-name* の argument-1 にある列 *column-number* で、無効な文字 *character* が検出されました。

表 79. **ランタイム・メッセージ** (続き)	
620 ページの『IWZ155S』	関数 *function-name* の argument-2 にある列 *column-number* で、無効な文字 *character* が検出されました。
620 ページの『IWZ156S』	関数 *function-name* の Argument-1 の値が 0 未満か、または 28 を超えています。
621 ページの『IWZ157S』	関数 *function-name* の Argument-1 の長さが 1 ではありません。
621 ページの『IWZ158S』	関数 *function-name* の Argument-1 の値が 0 未満か、または 29 を超えています。
621 ページの『IWZ159S』	関数 *function-name* の Argument-1 の値が 1 未満か、または 3067671 を超えています。
621 ページの『IWZ160S』	関数 *function-name* の Argument-1 の値が 16010101 未満か、または 99991231 を超えています。
621 ページの『IWZ161S』	関数 *function-name* の Argument-1 の値が 1601001 未満か、または 9999365 を超えています。
621 ページの『IWZ162S』	関数 *function-name* の Argument-1 の値が 1 未満か、またはプログラムの照合シーケンスの桁数を超えています。
622 ページの『IWZ163S』	関数 *function-name* の Argument-1 の値が 0 未満です。
622 ページの『IWZ165S』	行 *line-number* で、*start-position-value* の参照変更開始位置の値が、*function-result* の関数結果領域外の領域を参照しました。
622 ページの『IWZ166S』	行 *line-number* にある *length* の参照変更長の値が正数ではないことが、*function-result* の関数結果への参照内で検出されました。
622 ページの『IWZ167S』	行 *line-number* にある参照変更の開始位置の値 *start-position* と長さの値 *length* が原因で、関数結果 *function-result* の右端の文字を超えて参照が行われました。
623 ページの『IWZ168W』	SYSPUNCH/SYSPCH は、システムの論理出力装置をデフォルトに取ります。対応する環境変数が設定されていません。
623 ページの『IWZ169S』	DISPLAY ステートメントの装置タイプが不明です。
623 ページの『IWZ170S』	DISPLAY オペランドのデータ型が正しくありません。
623 ページの『IWZ171I』	*string-name* は有効なランタイム・オプションではありません。
623 ページの『IWZ172I』	ストリング *string-name* は、ランタイム・オプション *option-name* の有効なサブオプションではありません。
624 ページの『IWZ173I』	ランタイム・オプション *option-name* のサブオプション・ストリング *string-name* の文字長は *number* でなければなりません。 デフォルトが使用されます。
624 ページの『IWZ174I』	ランタイム・オプション *option-name* のサブオプション・ストリング *string-name* に無効な文字が 1 つ以上含まれています。 デフォルトが使用されます。
624 ページの『IWZ175S』	このシステムでは、ルーチン *routine-name* がサポートされていません。
624 ページの『IWZ176S』	関数 *function-name* の Argument-1 が *decimal-value* より大きい値でした。

表 79. **ランタイム・メッセージ** (続き)	
624 ページの『IWZ177S』	関数 *function-name* の Argument-2 が *decimal-value* と等しい値でした。
625 ページの『IWZ178S』	関数 *function-name* の Argument-1 の値が *decimal-value* 以下です。
625 ページの『IWZ179S』	関数 *function-name* の Argument-1 が *decimal-value* より小さい値でした。
625 ページの『IWZ180S』	関数 *function-name* の Argument-1 の値が整数ではありません。
625 ページの『IWZ181I』	ランタイム・オプション *option-name* の数値ストリング *string* で、無効な文字が検出されました。 デフォルトが使用されます。
625 ページの『IWZ182I』	ランタイム・オプション *option-name* の数値 *number* が、*min-range* から *max-range* の範囲を超えています。 デフォルトが使用されます。
625 ページの『IWZ183S』	_IWZCOBOLInit 内の関数名が戻りを行いました。
626 ページの『IWZ200S』	ファイル *file-name* に対する *I/O operation* の実行中に、エラーが検出されました。 ファイル状況は *file-status* です。
626 ページの『IWZ200S』	入出力エラー *error-code* により、STOP または ACCEPT が失敗しました。 実行単位は終了します。
626 ページの『IWZ201C』	
627 ページの『IWZ203S』	有効なコード・ページが DBCS コード・ページではありません。
627 ページの『IWZ204S』	ASCII DBCS から EBCDIC DBCS への変換中にエラーが発生しました。
627 ページの『IWZ221S』	コード・ページ *codepage value* の ICU コンバーターをオープンできません。 エラー・コードは *error code value* です。
627 ページの『IWZ222S』	エラー・コード *error code value* により、ICU を使用したデータ変換が失敗しました。
628 ページの『IWZ223W』	エラー・コード *error code value* により、ICU コンバーターのクローズが失敗しました。
628 ページの『IWZ224S』	ロケール値 *locale value* の ICU コレーターをオープンできません。 エラー・コードは *error code value* です。
628 ページの『IWZ225S』	ICU を使用した Unicode ケース・マッピング関数がエラー・コード *error code value* で失敗しました。有効なロケールは *locale value* です。
628 ページの『IWZ230W』	現行のコード・ページ *ASCII codeset-id* から EBCDIC コード・ページ *EBCDIC codeset-id* への変換テーブルを使用できません。デフォルトの ASCII/EBCDIC 変換テーブルが使用されます。
628 ページの『IWZ230W』	指定された EBCDIC コード・ページ *EBCDIC codepage* がロケール *locale* と一貫していませんが、要求どおりに使用されます。
629 ページの『IWZ230W』	指定された EBCDIC コード・ページ *EBCDIC codepage* はサポートされていません。 デフォルトの EBCDIC コード・ページ *EBCDIC codepage* が使用されます。
629 ページの『IWZ230S』	EBCDIC 変換テーブルをオープンできません。

表 79. **ランタイム・メッセージ** (続き)	
629 ページの『IWZ230S』	EBCDIC 変換テーブルを構築できません。
629 ページの『IWZ230S』	メインプログラムは -host フラグと CHAR(NATIVE) オプションの両方でコンパイルされましたが、これらには互換性がありません。
630 ページの『IWZ231S』	現行のロケール設定の照会が失敗しました。
630 ページの『IWZ232W』	メッセージには、次のような種類があります。 • データ項目 *data-name* を EBCDIC に変換するときに、プログラム *program-name* の行番号 *decimal-value* でエラーが発生しました。 • データ項目 *data-name* を ASCII に変換するときに、プログラム *program-name* の行番号 *decimal-value* でエラーが発生しました。 • データ項目 *data-name* について EBCDIC に変換するときに、プログラム *program-name* の行番号 *decimal-value* でエラーが発生しました。 • データ項目 *data-name* について ASCII に変換するときに、プログラム *program-name* の行番号 *decimal-value* でエラーが発生しました。 • プログラム *program-name* の行番号 *decimal-value* で、ASCII から EBCDIC への変換中にエラーが発生しました。 • プログラム *program-name* の行番号 *decimal-value* で、EBCDIC から ASCII への変換中にエラーが発生しました。
630 ページの『IWZ240S』	プログラム *program-name* の基本年が 1900 から 1999 の有効範囲内にありません。 スライディング・ウィンドウ値 *window-value* は、*base-year* の基本年になります。
631 ページの『IWZ241S』	現行年が、プログラム *program-name* に使用されている *year-start* から *year-end* の 100 年間隔内にありません。
631 ページの『IWZ242S』	XML PARSE ステートメントを開始しようとしましたが、この操作は無効です。
631 ページの『IWZ243S』	XML PARSE ステートメントを終了しようとしましたが、この操作は無効です。
631 ページの『IWZ813S』	使用可能なストレージが不十分なため、ストレージ取得要求を満たすことができません。
632 ページの『IWZ901S』	メッセージには、次のような種類があります。 • 重大エラーまたはクリティカル・エラーが発生したため、プログラムが終了します。 • プログラムの終了: ERRCOUNT を超えるエラー数が発生しました。
632 ページの『IWZ902S』	システムが 10 進数除算例外を検出しました。
632 ページの『IWZ903S』	システムがデータ例外を検出しました。
632 ページの『IWZ907S』	メッセージには、次のような種類があります。 • ストレージが不十分です。 • ストレージが不十分です。 *storage* 用に *number-bytes* バイトのスペースを取得できません。

表 79. **ランタイム・メッセージ** (続き)	
633 ページの『IWZ993W』	ストレージが不十分です。 メッセージ *message-number* 用のスペースを検出できません。
633 ページの『IWZ994W』	メッセージ・カタログ 内でメッセージ *message-number* を検出できません。
633 ページの『IWZ995C』	メッセージには、次のような種類があります。 • オフセット 0x*offset-value* でルーチン *routine-name* の実行中に、*System exception* シグナルを受信しました。 • 0x*offset-value* の場所でコードの実行中に、*System exception* シグナルを受信しました。 • *System exception* シグナルを受信しました。 場所を判別できませんでした。
633 ページの『IWZ2502S』	システムから UTC/GMT を使用できませんでした。
634 ページの『IWZ2503S』	UTC/GMT から現地時間までのオフセットをシステムから使用できませんでした。
634 ページの『IWZ2505S』	CEEDATM または CEESECI への呼び出し内の input_seconds 値が、対応範囲内にありませんでした。
634 ページの『IWZ2506S』	CEEDATM に渡されたピクチャー・ストリング内に元号 (<JJJJ>、<CCCC>、または <CCCCCCCC>) が 使用されていましたが、入力された秒数値が対応範囲内にありませんでした。 元号を判別できませんでした。
634 ページの『IWZ2507S』	CEEDAYS または CEESECS に渡されたデータが不十分です。 リリアン日付の値は計算されませんでした。
635 ページの『IWZ2508S』	CEEDAYS または CEESECS に渡された日付値が無効です。
635 ページの『IWZ2509S』	CEEDAYS または CEESECS に渡された元号が認識されませんでした。
635 ページの『IWZ2510S』	CEEISEC または CEESECS への呼び出しで時間の値が認識されませんでした。
635 ページの『IWZ2511S』	CEEISEC の呼び出しで渡された日のパラメーターが、指定された年および月に対して無効です。
636 ページの『IWZ2512S』	CEEDATE または CEEDYWK への呼び出しで渡されたリリアン日付値が、対応範囲内にありませんでした。
636 ページの『IWZ2513S』	CEEISEC、CEEDAYS、CEESECS のいずれかの呼び出しで渡された入力日付が、対応範囲内にありませんでした。
636 ページの『IWZ2514S』	CEEISEC の呼び出しで渡された年の値が、対応範囲内にありませんでした。
636 ページの『IWZ2515S』	CEEISEC 呼び出し内のミリ秒の値が認識されませんでした。
637 ページの『IWZ2516S』	CEEISEC 呼び出し内の分の値が認識されませんでした。
637 ページの『IWZ2517S』	CEEISEC 呼び出し内の月の値が認識されませんでした。
637 ページの『IWZ2518S』	日時サービスへの呼び出しに無効なピクチャー・ストリングが指定されました。

表 79. **ランタイム・メッセージ** (続き)	
638 ページの『IWZ2519S』	CEEISEC 呼び出し内の秒の値が認識されませんでした。
638 ページの『IWZ2520S』	CEEDAYS が数値フィールド内に非数値データを検出したか、あるいは日付ストリングとピクチャー・ストリングが一致しませんでした。
638 ページの『IWZ2521S』	CEEDAYS または CEESECS に渡された <JJJJ>、<CCCC>、または <CCCCCCCC> の元号年数値がゼロでした。
638 ページの『IWZ2522S』	CEEDATE に渡されたピクチャー・ストリング内に元号 (<JJJJ>、<CCCC>、または <CCCCCCCC>) が使用されていましたが、リリアン日付値が対応範囲内にありませんでした。 元号を判別できませんでした。
639 ページの『IWZ2525S』	CEESECS が数値フィールド内に非数値データを検出したか、あるいはタイム・スタンプ・ストリングとピクチャー・ストリングが一致しませんでした。
639 ページの『IWZ2526S』	CEEDATE によって戻された日付ストリングが切り捨てられました。
639 ページの『IWZ2527S』	CEEDATM によって戻されたタイム・スタンプ・ストリングが切り捨てられました。
639 ページの『IWZ2531S』	システムから現地時間を使用できませんでした。
640 ページの『IWZ2533S』	CEESCEN に渡された値が 0 から 100 の範囲内にありませんでした。
640 ページの『IWZ2534W』	CEEDATE または CEEDATM への呼び出しで、月または曜日名に対して指定されたフィールド幅が不十分です。 出力はブランクに設定されました。

IWZ006S

行 *line-number* で、verb 番号 *verb-number* によるテーブル *table-name* への参照が、テーブル領域外の領域をアドレス指定しました。

説明: SSRANGE オプションが有効なときに、このメッセージが出された場合は、固定長テーブルに添え字があるためにテーブルの定義サイズを超えているか、または可変長テーブルの場合はテーブルの最大サイズを超えていることを示します。

添え字の複合に対して範囲検査が実行された結果、テーブルの領域外のアドレスが戻されました。 可変長テーブルで、すべての OCCURS DEPENDING ON オブジェクトが最大値になっている場合、このアドレスは定義されたテーブルの領域外にあります。ODO オブジェクトの現行値は考慮されません。 個々の添え字に対しては、検査が行われていません。

プログラマー応答: 実行時に評価されるリテラル添え字や変数添え字の値が、失敗したステートメント内の添え字付きデータに対する添え字付き次元を超えないようにしてください。

システム処置: アプリケーションは終了します。

IWZ007S

行 *line-number* で、verb 番号 *verb-number* による可変長グループ *group-name* への参照が、グループの最大定義長を超える領域をアドレス指定しました。

説明: SSRANGE オプションが有効なときに、このメッセージが出された場合は、OCCURS DEPENDING ON によって生成された可変長グループの長さが 0 未満か、または OCCURS DEPENDING ON 文節で定義された限度を超えていることを示します。

範囲検査は、個々の OCCURS DEPENDING ON オブジェクトに対してではなく、グループの複合長に対して実行されています。

プログラマー応答: 実行時に評価される OCCURS DEPENDING ON オブジェクトが、参照先グループ項目内のテーブルに関する次元の最大オカレンス数を超えないようにしてください。

システム処置: アプリケーションは終了します。

IWZ012I

ソートまたはマージの実行中に、無効な実行単位終了が発生しました。

説明: COBOL プログラムによって開始されたソートまたはマージの進行中に、次のいずれかが試行されました。

1. STOP RUN が出された。
2. ソートまたはマージを開始した COBOL プログラムの入力プロシージャーまたは出力プロシージャーで、GOBACK または EXIT PROGRAM が出された。 GOBACK および EXIT PROGRAM ステートメントは、入力プロシージャーまたは出力プロシージャーによって呼び出されたプログラム内で使用できることに注意してください。

プログラマー応答: 上記のメソッドを使用せずにソートまたはマージを終了するようにアプリケーションを変更してください。

システム処置: アプリケーションは終了します。

IWZ013S

ソートまたはマージが要求されましたが、別のスレッドでソートまたはマージを実行中です。

説明: 2 つ以上のスレッドでソートまたはマージを同時に実行することはできません。

プログラマー応答: ソートまたはマージは、必ず同じスレッドで実行してください。 あるいは、ソートまたはマージの各呼び出しの前に、ソートまたはマージを別のスレッドで実行しているかどうかを判断するコードを組み込むことができます。 ソートまたはマージが別のスレッドで実行中の場合は、そのスレッドが完了するのを待ちます。 そうでない場合は、ソートまたはマージを実行することを示すフラグを設定してから、ソートまたはマージを呼び出します。

システム処置: スレッドは終了します。

IWZ026W

SORT-RETURN 特殊レジスターは参照されませんでしたが、現在の内容は、行番号 *line-number* でプログラム *program-name* のソートまたはマージ操作が失敗したことを示しています。 ソートまたはマージの戻りコードは *return code* です。

説明: COBOL ソースには、SORT-RETURN レジスターへの参照が含まれていません。 コンパイラーは、ソートまたはマージの各 verb の後でテストを生成します。 ソートまたはマージによって、ゼロ以外の戻りコードがプログラムに戻されています。

プログラマー応答: ソートまたはマージが失敗した理由を判断し、問題を修正してください。 可能な戻りコードのリストについては 161 ページの『ソートおよびマージ・エラー番号』を参照してください。

システム処置: システムのアクションはとられません。

IWZ029S

プログラム *program-name* の行 *line-number* にある関数 *function-name* の Argument-1 が 0 未満です。

説明: argument-1 に対して無効な値が使用されました。

プログラマー応答: argument-1 の値が 0 以上であることを確認してください。

システム処置: アプリケーションは終了します。

IWZ030S

プログラム *program-name* の行 *line-number* にある関数 *function-name* の Argument-2 が正の整数ではありません。

説明: argument-1 に対して無効な値が使用されました。

プログラマー応答: argument-2 の値が正の整数であることを確認してください。

システム処置: アプリケーションは終了します。

IWZ036W

プログラム *program-name* の行番号 *line-number* で、高位桁位置の切り捨てが発生しました。

説明: 生成されたコードによって、中間結果 (算術演算中に使用される一時記憶域) が 30 桁に切り詰められましたが、切り捨てられた桁の中に、値が 0 でないものがあります。

プログラマー応答: 中間結果については、このセクションの末尾にある関連概念を参照してください。

システム処置: システムのアクションはとられません。

IWZ037I

プログラム *program-name* の制御フローが、プログラムの最終行を超えました。 制御権は、プログラム *program-name* の呼び出し元に戻りました。

説明: プログラムに終止符 (STOP、GOBACK、または EXIT) がないため、制御権が最後の命令をフォールスルーしました。

プログラマー応答: プログラムのロジックを検査してください。 次のいずれかの論理エラーがあると、このエラーが発生する場合があります。

- プログラム内の最後の段落は PERFORM ステートメントの結果として制御権を受け取るものと想定されていたが、論理エラーがあるために、GO TO ステートメントによって分岐された。
- プログラム内の最後の段落が「フォールスルー」パスの結果として実行されたが、段落の末尾にプログラムを終了するためのステートメントがない。

システム処置: システムのアクションはとられません。

IWZ038S

行 *line-number* にある *reference-modification-value* の参照変更長の値が 1 ではないことが、データ項目 *data-item* への参照内で検出されました。

説明: 参照変更で指定された長さ値が 1 ではありません。この長さ値は 1 でなければなりません。

プログラマー応答: プログラム内の指定された行番号を検査して、参照変更された長さ値がすべて 1 になっていることを確認してください (そうでない場合は、値を修正してください)。

システム処置: アプリケーションは終了します。

IWZ039S

無効なオーバーパンチ符号が検出されました。

説明: 符号位置の値が無効です。

X'*sd*' (*s* は符号表現であり、*d* は数字を表す) の場合、外部 10 進数 (SIGN IS SEPARATE 文節を指定しない USAGE DISPLAY) についての有効な符号表現は、以下のとおりです。

正:	0、1、2、3、8、9、A、および B
負:	4、5、6、7、C、D、E、および F

内部生成される符号は、正および符号なしの場合は 3、負の場合は 7 になります。

X'*ds*' (*d* は数字、*s* は符号表現を表す) の場合、内部 10 進数 (USAGE PACKED-DECIMAL) COBOL データの有効な符号表現は次のようになります。

正:	A、C、E、および F
負:	B および D

内部生成される符号は、正および符号なしの場合は C、負の場合は D になります。

プログラマー応答: REDEFINES 文節に符号位置が含まれている場合、符号位置を含むグループを移動した場合、または位置が初期設定されていない場合には、このエラーが発生する場合があります。 このようなことがないかどうかを検査してください。

システム処置: アプリケーションは終了します。

IWZ040S

無効な分離符号が検出されました。

説明: 分離記号を使用して定義されたデータを操作しようとしました。 符号位置の値はプラス (+) またはマイナス (-) ではありませんでした。

プログラマー応答: REDEFINES 文節に符号位置が含まれている場合、符号位置を含むグループを移動した場合、または位置が初期設定されていない場合には、このエラーが発生する場合があります。 このようなことがないかどうかを検査してください。

システム処置: アプリケーションは終了します。

IWZ048W

指数式で、負の基数が小数で累乗されました。 基数の絶対値が使用されました。

説明: ライブラリー・ルーチン内で、負数が小数で累乗されました。

COBOL では、負の数を小数で累乗した値は定義されていません。 当該ステートメントに SIZE ERROR 文節があれば、SIZE ERROR 命令が使用されますが、 実際には SIZE ERROR 文節は存在しないため、基数の絶対値が指数に使用されました。

プログラマー応答: 失敗したステートメント内のプログラム変数が正しく設定されていることを確認してください。

システム処置: システムのアクションはとられません。

IWZ049W

指数式で、0 の基数が 0 で累乗されました。 結果は 1 に設定されました。

説明: ライブラリー・ルーチン内で、0 の値が 0 で累乗されました。

COBOL では、0 の値を 0 で累乗する演算は定義されていません。 当該ステートメントに SIZE ERROR 文節があれば、SIZE ERROR 命令が使用されますが、 実際には SIZE ERROR 文節は存在しないため、戻り値は 1 になりました。

プログラマー応答: 失敗したステートメント内のプログラム変数が正しく設定されていることを確認してください。

システム処置: システムのアクションはとられません。

IWZ050S

指数式で、0 の基数が負数で累乗されました。

説明: ライブラリー・ルーチン内で、0 の値が負数で累乗されました。

0 の値を負数で累乗する演算は定義されていません。 当該ステートメントに SIZE ERROR 文節があれば、SIZE ERROR 命令が使用されますが、 実際には SIZE ERROR 文節は存在しません。

プログラマー応答: 失敗したステートメント内のプログラム変数が正しく設定されていることを確認してください。

システム処置: アプリケーションは終了します。

IWZ051S

オペランドまたは受信側で指定された小数点位が多すぎるため、プログラム *program-name* の固定小数点指数演算に有効桁が残っていません。

説明: オペランドまたは受信側で指定されている小数点位が多すぎるため、固定小数点計算で生成された結果に有効数字が含まれていません。

プログラマー応答: 必要に応じて、オペランドまたは受け取り側の数値項目の PICTURE 文節を変更して、整数位を増やし、小数点位を減らしてください。

システム処置: アプリケーションは終了します。

IWZ053S

浮動小数点への変換時にオーバーフローが発生しました。

説明: プログラム内で生成された数値が大きすぎて、浮動小数点で表現できません。

プログラマー応答: プログラムを正しく修正して、オーバーフローを回避する必要があります。

システム処置: アプリケーションは終了します。

IWZ054S

浮動小数点例外が発生しました。

説明: 浮動小数点計算で生成された結果が正しくありません。 浮動小数点計算は IEEE 浮動小数点数演算を使用して実行されますが、その際に NaN (非数値) と呼ばれる結果が出る場合があります。 例えば、0 を 0 で除算すると、結果は NaN になります。

プログラマー応答: NaN が生成されないように、この演算に対する引数をテストするようにプログラムを修正してください。

システム処置: アプリケーションは終了します。

IWZ055W

浮動小数点への変換時にアンダーフローが発生しました。 結果は 0 に設定されました。

説明: 浮動小数点への変換時に、負の指数がハードウェアの限度を超えました。 浮動小数点値は 0 に設定されました。

プログラマー応答: 必須の処置は特にありませんが、必要に応じて、アンダーフローを回避するようにプログラムを修正してください。

システム処置: システムのアクションはとられません。

IWZ058S

指数オーバーフローが発生しました。

説明: 浮動小数点の指数オーバーフローが、ライブラリー・ルーチンで発生しました。

プログラマー応答: 失敗したステートメント内のプログラム変数が正しく設定されていることを確認してください。

システム処置: アプリケーションは終了します。

IWZ059W

9 桁を超える指数が切り捨てられました。

説明: 固定小数点の累乗法では、指数は 9 桁以下でなければなりません。 指数が 9 桁に切り詰められましたが、切り捨てられた桁の中に、値が 0 でないものがあります。

プログラマー応答: 必須の処置は特にありませんが、必要に応じて、失敗したステートメント内の指数を調整してください。

システム処置: システムのアクションはとられません。

IWZ060W

高位桁位置の切り捨てが発生しました。

説明: ライブラリー・ルーチン内のコードによって、中間結果 (算術演算中に使用される一時記憶域) が 30 桁に切り詰められましたが、切り捨てられた桁の中に、値が 0 でないものがあります。

プログラマー応答: 中間結果については、このセクションの末尾にある関連概念を参照してください。

システム処置: システムのアクションはとられません。

IWZ061S

0 による除算が発生しました。

説明: ライブラリー・ルーチン内で、0 による除算が行われました。 0 による除算は定義されていません。 当該ステートメントに SIZE ERROR 文節があれば、SIZE ERROR 命令が使用されますが、 実際には SIZE ERROR 文節は存在しません。

プログラマー応答: 失敗したステートメント内のプログラム変数が正しく設定されていることを確認してください。

システム処置: アプリケーションは終了します。

IWZ063S

program-name の行番号 _line-number_ にある数値編集送信フィールドで、無効な符号が検出されました。

説明: MOVE ステートメントで、符号付き数値編集フィールドを符号付き数値または数値編集受信フィールドに移動しようとしましたが、送信フィールド内の符号位置に含まれている文字が、対応する PICTURE に有効な符号文字ではありませんでした。

プログラマー応答: 失敗したステートメント内のプログラム変数が正しく設定されていることを確認してください。

システム処置: アプリケーションは終了します。

IWZ064S

コンパイル単位 _compilation-unit_ で、アクティブ・プログラム _program-name_ への再帰呼び出しが試行されました。

説明: COBOL では、内部プログラムの再帰呼び出しを行うことはできません。プログラムは実行を開始しましたが、まだ終了していません。 例えば、内部プログラム A と B が収容プログラムの兄弟で、A が B を呼び出し、B が A を呼び出すと、このメッセージが出されます。

プログラマー応答: プログラムを調べて、アクティブな内部プログラムへの呼び出しを除去してください。

システム処置: アプリケーションは終了します。

IWZ065I

コンパイル単位 _compilation-unit_ で、アクティブ・プログラム _program-name_ のキャンセルが試行されました。

説明: アクティブな内部プログラムをキャンセルしようとしました。 例えば、内部プログラム A と B が収容プログラム内の兄弟で、A が B を呼び出し、B が A をキャンセルすると、このメッセージが出されます。

プログラマー応答: プログラムを調べて、アクティブな内部プログラムのキャンセルを除去してください。

システム処置: アプリケーションは終了します。

IWZ066S

プログラム _program-name_ の外部データ・レコード _data-record_ の長さが、既存のレコード長と一致しませんでした。

説明: プログラムの初期化で外部データ・レコードを処理しているときに、実行単位内の別のプログラムで前に外部データ・レコードが定義されており、現行のプログラムで指定されているレコードの長さが、前に定義された長さと異なることが判別されました。

プログラマー応答: 現在のファイルを調べて、外部データ・レコードが正しく指定されていることを確認してください。

システム処置: アプリケーションは終了します。

IWZ071S

行 *line-number* で、verb 番号 *verb-number* によるテーブル *table-name* への ALL 添え字付きテーブル参照に、OCCURS DEPENDING ON 次元に対して指定された ALL 添え字があり、オブジェクトの値は 0 以下でした。

説明: SSRANGE オプションが有効なときに、このメッセージが出された場合は、ALL による添え字付きの次元のオカレンス数が 0 であることを示します。

OCCURS DEPENDING ON オブジェクトの現行値に対して検査が実行されます。

プログラマー応答: 当該ステートメントにあるすべての添え字付き項目について、ALL 添え字付き次元の ODO オブジェクトが正数であることを確認してください。

システム処置: アプリケーションは終了します。

IWZ072S

行 *line-number* で、*reference-modification-value* の参照変更開始位置の値が、データ項目 *data-item* の領域外の領域を参照しました。

説明: 参照変更指定の開始位置の値が 1 未満か、または参照変更されていたデータ項目の現行長を超えています。 開始位置の値は、参照変更されるデータ項目の文字数以下の、正の整数でなければなりません。

プログラマー応答: 参照変更指定の開始位置の値を検査してください。

システム処置: アプリケーションは終了します。

IWZ073S

行 *line-number* にある *reference-modification-value* の参照変更長の値が正数ではないことが、データ項目 *data-item* への参照内で検出されました。

説明: 参照変更で指定された長さ値が 0 以下です。この長さ値は正の整数でなければなりません。

プログラマー応答: プログラム内の指定された行番号を検査して、参照変更された長さ値がすべて正の整数になっていることを確認してください (そうでない場合は、値を修正してください)。

システム処置: アプリケーションは終了します。

IWZ074S

行 *line-number* にある *reference-modification-value* の参照変更開始位置の値と *length* の長さ値により、データ項目 *data-item* の右端文字を超える参照が行われました。

説明: 参照変更指定の開始位置と長さ値を組み合わせてアドレス指定された領域が、参照変更されるデータ項目の終わりを超えています。 開始位置と長さ値の合計から 1 を引いた値が、参照変更されるデータ項目の文字数以下でなければなりません。

プログラマー応答: プログラム内の指定された行番号を検査して、参照がデータ項目の右端文字を超えないように、参照変更される開始値と長さ値が設定されていることを確認してください。

システム処置: アプリケーションは終了します。

IWZ075S

プログラム *program-name* の EXTERNAL ファイル *file-name* で矛盾が検出されました。 *attribute-1 attribute-2 attribute-3 attribute-4 attribute-5 attribute-6 attribute-7* の各ファイル属性が、設定済み外部ファイルのファイル属性と一致しませんでした。

説明: 外部ファイルの 1 つ以上の属性が、その外部ファイルを定義した 2 つのプログラム間で一致していません。

プログラマー応答: 外部ファイルを訂正してください。 同じ外部ファイルの定義間で一致させる必要があるファイル属性の要約については、「*COBOL for Linux on x86* 言語解説書」を参照してください。

システム処置: アプリケーションは終了します。

IWZ076W

INSPECT REPLACING CHARACTERS BY データ名の文字数が 1 ではありません。 先頭文字が使用されました。

説明: INSPECT ステートメントの REPLACING 句の中の CHARACTERS 句にあるデータ項目の長さは、1 文字として定義する必要があります。 このデータ項目の参照変更指定により、結果として生じる長さ値が 1 になりませんでした。 長さ値は 1 でなければなりません。

プログラマー応答: 必要に応じて、失敗した INSPECT ステートメントの参照変更指定を訂正して、参照変更の長さが 1 になるようにしてください。プログラマーによる処置は必須ではありません。

システム処置: システムのアクションはとられません。

IWZ077W

INSPECT データ項目の長さが等しくありません。 短い方の長さが使用されました。

説明: INSPECT ステートメントの REPLACING または CONVERTING 句にある 2 つのデータ項目は、2 番目の項目が表意定数の場合を除き、同じ長さでなければなりません。 このようなデータ項目の一方または両方に対して参照変更が行われたため、結果的に長さ値が同じでなくなりました。 短い方の長さが両方の項目に適用され、実行が続けられます。

プログラマー応答: 必要に応じて、失敗した INSPECT ステートメント内で長さの等しくないオペランドを調整してください。プログラマーによる処置は必須ではありません。

システム処置: システムのアクションはとられません。

IWZ078S

行 *line-number* で、verb 番号 *verb-number* によるテーブル *table-name* への ALL 添え字付き参照が、テーブルの上限を超えています。

説明: SSRANGE オプションが有効なときに、このメッセージが出された場合は、多次元テーブルで ALL が 1 つ以上の添え字として指定されているために、参照がテーブルの上限を超えてしまうことを示します。

範囲検査は、添え字の複合と、ALL 添え字付き次元の最大オカレンスに対して実行されています。 可変長テーブルで、すべての OCCURS DEPENDING ON オブジェクトが最大値になっている場合、このアドレスは定義されたテーブルの領域外にあります。ODO オブジェクトの現行値は考慮されません。 個々の添え字に対しては、検査が行われていません。

プログラマー応答: 実行時に評価される OCCURS DEPENDING ON オブジェクトが、失敗したステートメント内で参照されるテーブル項目に関する次元の最大オカレンス数を超えないようにしてください。

システム処置: アプリケーションは終了します。

IWZ096C

メッセージには、次のような種類があります。

- **プログラム *program-name* の動的呼び出しが失敗しました。 モジュール *module-name* のロードが失敗しました。エラー・コードは *error-code* です。**
- **プログラム *program-name* の動的呼び出しが失敗しました。 モジュール *module-name* のロードが失敗しました。戻りコードは *return-code* です。**
- **プログラム *program-name* の動的呼び出しが失敗しました。 リソースが不十分です。**
- **プログラム *program-name* の動的呼び出しが失敗しました。 環境内で COBPATH が検出されませんでした。**
- **プログラム *program-name* の動的呼び出しが失敗しました。 入力項目 *entry-name* が検出されませんでした。**
- **動的呼び出しが失敗しました。 ターゲット・プログラムの名前に有効な文字が含まれていません。**
- **プログラム *program-name* の動的呼び出しが失敗しました。 ロード・モジュール *load-module* が、COBPATH 環境変数で識別されたディレクトリー内にありませんでした。**

説明: 上記の各種メッセージに示されたいずれかの理由で、動的呼び出しが失敗しました。 上記の *error-code* の 値は、load によって設定された エラー番号です。

プログラマー応答: COBPATH が定義されていることを確認してください。 モジュールが存在していることを確認してください。ロードされるモジュールの名前と、呼び出される入力項目の名前が一致していることを確認してください。 適切な cob2 オプション。

システム処置: アプリケーションは終了します。

IWZ097S

関数 *function-name* の Argument-1 に桁がありません。

説明: 指定された関数の Argument-1 には、少なくとも 1 桁が含まれていなければなりません。

プログラマー応答: 失敗したステートメントで Argument-1 の桁数を調整してください。

システム処置: アプリケーションは終了します。

IWZ100S

関数 *function* の Argument-1 の値が -1 以下です。

説明: Argument-1 に使用された値が正しくありません。

プログラマー応答: argument-1 が -1 より大きいことを確認してください。

システム処置: アプリケーションは終了します。

IWZ103S

関数 *function-name* の Argument-1 の値が 0 未満か、または 99 を超えています。

説明: Argument-1 に使用された値が正しくありません。

プログラマー応答: 関数の引数が有効範囲内にあることを確認してください。

システム処置: アプリケーションは終了します。

IWZ104S

関数 *function-name* の Argument-1 の値が 0 未満か、または 99999 を超えています。

説明: Argument-1 に使用された値が正しくありません。

プログラマー応答: 関数の引数が有効範囲内にあることを確認してください。

システム処置: アプリケーションは終了します。

IWZ105S

関数 *function-name* の Argument-1 の値が 0 未満か、または 999999 を超えています。

説明: Argument-1 に使用された値が正しくありません。

プログラマー応答: 関数の引数が有効範囲内にあることを確認してください。

システム処置: アプリケーションは終了します。

IWZ151S

関数 *function-name* の Argument-1 が 18 桁を超えています。

説明: 指定された関数の Argument-1 の合計桁数が 18 桁を超えています。

プログラマー応答: 失敗したステートメントで Argument-1 の桁数を調整してください。

システム処置: アプリケーションは終了します。

IWZ152S

関数 *function-name* の argument-1 にある列 *column-number* で、無効な文字 *character* が検出されました。

説明: NUMVAL/NUMVAL-C 関数の argument-1 で、小数点、コンマ、スペース、または符号 (+、-、CR、DB) 以外の非数字文字が検出されました。

プログラマー応答: 指定されたステートメントで NUMVAL または NUMVAL-C の argument-1 を調整してください。

システム処置: アプリケーションは終了します。

IWZ155S

関数 *function-name* の argument-2 にある列 *column-number* で、無効な文字 *character* が検出されました。

説明: NUMVAL-C 関数の argument-2 で、無効な文字が検出されました。

プログラマー応答: 関数の引数が構文規則に従っていることを確認してください。

システム処置: アプリケーションは終了します。

IWZ156S

関数 *function-name* の Argument-1 の値が 0 未満か、または 28 を超えています。

説明: 関数 FACTORIAL に対する入力引数が 28 を超えているか、または 0 未満です。

プログラマー応答: 関数の引数が有効範囲内にあることを確認してください。

システム処置: アプリケーションは終了します。

IWZ157S

関数 *function-name* の Argument-1 の長さが 1 ではありません。

説明: ORD 関数に対する入力引数の長さが 1 ではありません。

プログラマー応答: 関数の引数が 1 バイト長であることを確認してください。

システム処置: アプリケーションは終了します。

IWZ158S

関数 *function-name* の Argument-1 の値が 0 未満か、または 29 を超えています。

説明: 関数 FACTORIAL に対する入力引数が 29 を超えているか、または 0 未満です。

プログラマー応答: 関数の引数が有効範囲内にあることを確認してください。

システム処置: アプリケーションは終了します。

IWZ159S

関数 *function-name* の Argument-1 の値が 1 未満か、または 3067671 を超えています。

説明: DATE-OF-INTEGER または DAY-OF-INTEGER 関数に対する入力引数が 1 未満か、または 3067671 を超えています。

プログラマー応答: 関数の引数が有効範囲内にあることを確認してください。

システム処置: アプリケーションは終了します。

IWZ160S

関数 *function-name* の Argument-1 の値が 16010101 未満か、または 99991231 を超えています。

説明: INTEGER-OF-DATE 関数に対する入力引数が 16010101 未満か、または 99991231 を超えています。

プログラマー応答: 関数の引数が有効範囲内にあることを確認してください。

システム処置: アプリケーションは終了します。

IWZ161S

関数 *function-name* の Argument-1 の値が 1601001 未満か、または 9999365 を超えています。

説明: INTEGER-OF-DAY 関数に対する入力引数が 1601001 未満か、または 9999365 を超えています。

プログラマー応答: 関数の引数が有効範囲内にあることを確認してください。

システム処置: アプリケーションは終了します。

IWZ162S

関数 *function-name* の Argument-1 の値が 1 未満か、またはプログラムの照合シーケンスの桁数を超えています。

説明: CHAR 関数に対する入力引数が 1 未満か、またはプログラムの照合シーケンスの最高位桁を超えています。

プログラマー応答: 関数の引数が有効範囲内にあることを確認してください。

システム処置: アプリケーションは終了します。

IWZ163S

関数 *function-name* の Argument-1 の値が 0 未満です。

説明: RANDOM 関数に対する入力引数が 0 未満です。

プログラマー応答: 失敗したステートメントで RANDOM 関数の引数を訂正してください。

システム処置: アプリケーションは終了します。

IWZ165S

行 *line-number* で、*start-position-value* の参照変更開始位置の値が、*function-result* の関数結果領域外の領域を参照しました。

説明: 参照変更指定の開始位置の値が 1 未満か、または参照変更されていた関数結果の現行長を超えています。 開始位置の値は、参照変更される関数結果の文字数以下の、正の整数でなければなりません。

プログラマー応答: 参照変更指定の開始位置の値と、実際の関数結果の長さを検査してください。

システム処置: アプリケーションは終了します。

IWZ166S

行 *line-number* にある *length* の参照変更長の値が正数ではないことが、*function-result* の関数結果への参照内で検出されました。

説明: 関数結果の参照変更で指定された長さ値が 0 以下です。この長さ値は正の整数でなければなりません。

プログラマー応答: 長さ値を確認し、適切な修正を行ってください。

システム処置: アプリケーションは終了します。

IWZ167S

行 *line-number* にある *start-position* の参照変更開始位置の値と *length* の長さ値により、*function-result* の関数結果の右端文字を超える参照が行われました。

説明: 参照変更指定の開始位置と長さ値を組み合わせてアドレス指定された領域が、参照変更される関数結果の終わりを超えています。 開始位置と長さ値の合計から 1 を引いた値が、参照変更される関数結果の文字数以下でなければなりません。

プログラマー応答: 参照変更指定の長さと実際の関数結果の長さを照合検査し、適切な修正を行ってください。

システム処置: アプリケーションは終了します。

IWZ168W

SYSPUNCH/SYSPCH は、システムの論理出力装置をデフォルトに取ります。 対応する環境変数が設定されていません。

説明: COBOL 環境名 (SYSPUNCH/SYSPCH など) は、ACCEPT および DISPLAY ステートメントで使用される簡略名に対応する環境変数名として使用されます。 これらは、既存のディレクトリー名ではなくファイルと等しくなるように設定します。 環境変数は、 `export` コマンドを使用して設定してください。

環境変数は、永続的に設定することも一時的に設定することも可能です。

プログラマー応答: SYSPUNCH/SYSPCH が表示画面をデフォルトに取らないようにするには、対応する環境変数を設定してください。

システム処置: システムのアクションはとられません。

IWZ169S

DISPLAY ステートメントの装置タイプが不明です。

説明: DISPLAY ステートメントの *environment-name-1* か、*mnemonic-name-1* に関連付けられている環境名に、不明な装置タイプが指定されました。

プログラマー応答: 有効な装置タイプを指定してください。有効なタイプについては、SPECIAL-NAMES 段落を参照してください。

システム処置: アプリケーションは終了します。

IWZ170S

DISPLAY オペランドのデータ型が正しくありません。

説明: DISPLAY ステートメントのターゲットとして、無効なデータ型が指定されました。

プログラマー応答: 有効なデータ型を指定してください。 次のデータ型は**無効**です。

- `USAGE IS FUNCTION-POINTER` で定義されたデータ項目
- `USAGE IS PROCEDURE-POINTER` で定義されたデータ項目
- `USAGE IS INDEX` で定義されたデータ項目または索引名

システム処置: アプリケーションは終了します。

IWZ171I

***string-name* は有効なランタイム・オプションではありません。**

説明: *string-name* は有効なオプションではありません。

プログラマー応答: `CHECK`、`DEBUG`、`ERRCOUNT`、`FILESYS`、`TRAP`、および `UPSI` が有効なランタイム・オプションです。

システム処置: *string-name* が無視されます。

IWZ172I

ストリング *string-name* は、ランタイム・オプション *option-name* の有効なサブオプションではありません。

説明: *string-name* は、認識済みの値セットの中に含まれていませんでした。

プログラマー応答: 無効なサブオプション *string* を、ランタイム・オプション *option-name* から削除してください。

システム処置: 無効なサブオプションが無視されます。

IWZ173I

ランタイム・オプション *option-name* のサブオプション・ストリング *string-name* の文字長は *number of* でなければなりません。 デフォルトが使用されます。

説明: ランタイム・オプション *option-name* のサブオプション・ストリング *string-name* の文字数が無効です。

プログラマー応答: デフォルトを使用したくない場合は、有効な文字長を指定してください。

システム処置: デフォルト値が使用されます。

IWZ174I

ランタイム・オプション *option-name* のサブオプション・ストリング *string-name* に無効な文字が 1 つ以上含まれています。 デフォルトが使用されます。

説明: 指定されたサブオプション内で、無効文字が 1 つ以上検出されました。

プログラマー応答: デフォルトを使用したくない場合は、有効な文字を指定してください。

システム処置: デフォルト値が使用されます。

IWZ175S

このシステムでは、ルーチン *routine-name* がサポートされていません。

説明: *routine-name* はサポートされていません。

プログラマー応答:

システム処置: アプリケーションは終了します。

IWZ176S

関数 *function-name* の Argument-1 の値が *decimal-value* を超えています。

説明: argument-1 に対して無効な値が使用されました。

プログラマー応答: argument-1 の値が *decimal-value* 以下であることを確認してください。

システム処置: アプリケーションは終了します。

IWZ177S

関数 *function-name* の Argument-2 の値が *decimal-value* と等しくなっています。

説明: argument-2 に対して無効な値が使用されました。

プログラマー応答: argument-1 の値が *decimal-value* と等しくないことを確認してください。

システム処置: アプリケーションは終了します。

IWZ178S

関数 *function-name* の Argument-1 の値が *decimal-value* 以下です。

説明: Argument-1 に対して無効な値が使用されました。

プログラマー応答: Argument-1 が *decimal-value* より大きいことを確認してください。

システム処置: アプリケーションは終了します。

IWZ179S

関数 *function-name* の Argument-1 の値が *decimal-value* 未満です。

説明: Argument-1 に対して無効な値が使用されました。

プログラマー応答: Argument-1 が *decimal-value* 以上であることを確認してください。

システム処置: アプリケーションは終了します。

IWZ180S

関数 *function-name* の Argument-1 の値が整数ではありません。

説明: Argument-1 に対して無効な値が使用されました。

プログラマー応答: Argument-1 が整数であることを確認してください。

システム処置: アプリケーションは終了します。

IWZ181I

ランタイム・オプション *option-name* の数値ストリング *string* で、無効な文字が検出されました。 デフォルトが使用されます。

説明: *string* にすべての 10 進数字が含まれていませんでした。

プログラマー応答: デフォルト値を使用したくない場合は、ランタイム・オプションのストリングにすべての数字が含まれるように修正してください。

システム処置: デフォルトが使用されます。

IWZ182I

ランタイム・オプション *option-name* の数値 *number* が、*min-range* から *max-range* の範囲を超えています。 デフォルトが使用されます。

説明: *number* が *min-range* から *max-range* の範囲を超えました。

プログラマー応答: ランタイム・オプションのストリングが有効範囲内になるように修正してください。

システム処置: デフォルトが使用されます。

IWZ183S

_iwzCOBOLInit 内の関数名が戻りを行いました。

説明: 実行単位の終了出口ルーチンが、そのルーチンを呼び出した関数 (function_code で指定された関数) に戻りました。

プログラマー応答: 実行単位の終了出口ルーチンが、関数に戻るのではなく、longjump または exit を実行するように関数を書き直してください。

システム処置: アプリケーションは終了します。

IWZ200S

ファイル *file-name* に対する *I/O operation* の実行中に、エラーが検出されました。 ファイル状況は *file-status* です。

説明: ファイル入出力操作中に、エラーが検出されました。 当該ファイルに対して、ファイル状況が指定されておらず、また該当するエラー宣言も有効になっていません。

プログラマー応答: このメッセージで説明されている状態を修正してください。 エラーを検出して、ソース・プログラムで適切な処置をとりたい場合は、当該ファイルに FILE STATUS 文節を指定してください。

システム処置: アプリケーションは終了します。

IWZ200S

入出力エラー *error-code* により、STOP または ACCEPT が失敗しました。 実行単位は終了します。

説明: STOP または ACCEPT ステートメントが失敗しました。

プログラマー応答: STOP または ACCEPT が正当なファイルまたは端末を参照していることを確認してください。

システム処置: アプリケーションは終了します。

IWZ201C

メッセージには、次のような種類があります。

```
Access Intent List Error.
Concurrent Opens Exceeds Maximum.
Cursor Not Selecting a Record Position.
Data Stream Syntax Error.
Duplicate Key Different Index.
Duplicate Key Same Index.
Duplicate Record Number.
File Temporarily Not Available.
File system cannot be found.
File Space Not Available.
File Closed with Damage.
Invalid Key Definition.
Invalid Base File Name.
Key Update Not Allowed by Different Index.
Key Update Not Allowed by Same Index.
No Update Intent on Record.
Not Authorized to Use Access Method.
Not Authorized to Directory.
Not Authorized to Function.
Not authorized to File.
Parameter Value Not Supported.
Parameter Not Supported.
Record Number Out of Bounds.
Record Length Mismatch.
Resource Limits Reached in Target System.
Resource Limits Reached in Source System.
```

```
Address Error.
Command Check.
Duplicate File Name.
End of File Condition.
Existing Condition.
File Handle Not Found.
Field Length Error.
File Not Found.
File Damaged.
File is Full.
File In Use.
Function Not Supported.
Invalid Access Method.
Invalid Data Record.
Invalid Key Length.
Invalid File Name.
Invalid Request.
Invalid Flag.
Object Not Supported.
Record Not Available.
Record Not Found.
Record Inactive.
Record Damaged.
Record In Use.
Update Cursor Error.
```

説明: STL ファイルに対する入出力操作中に、エラーが検出されました。当該ファイルに対して、ファイル状況が指定されておらず、また該当するエラー宣言も有効になっていません。

プログラマー応答: このメッセージで説明されている状態を修正してください。

システム処置: アプリケーションは終了します。

IWZ203S

有効なコード・ページが DBCS コード・ページではありません。

説明: 非 DBCS コード・ページが有効な状態で、DBCS データを指す参照が行われました。

プログラマー応答: DBCS データの場合は、有効な DBCS コード・ページを指定してください。 有効な DBCS コード・ページを次に示します。

国または地域	コード・ページ
日本	IBM-932
韓国	IBM-1363
中華人民共和国 (簡体字)	IBM-1386
台湾 (繁体字)	

注: 上記のコード・ページは、プラットフォームの特定のバージョンまたはリリースではサポートされない場合があります。

システム処置: アプリケーションは終了します。

IWZ204S

ASCII DBCS から EBCDIC DBCS への変換中にエラーが発生しました。

説明: ASCII 文字ストリングから EBCDIC ストリングへの変換中にエラーが検出されたため、漢字または DBCS クラス・テストが失敗しました。

プログラマー応答: 有効なロケールと、テスト対象の ASCII 文字ストリングが一貫していることを検証してください。 ロケール設定が正しい場合は、特に処置は必要ありません。 クラス・テストは、当該ストリングが漢字または DBCS ではないことを正しく示していると思われます。

システム処置: アプリケーションは終了します。

IWZ221S

コード・ページ *codepage value* の ICU コンバーターをオープンできません。 エラー・コードは *error code value* です。

説明: コード・ページと UTF-16 間の変換を行う ICU コンバーターをオープンできません。

プログラマー応答: ICU 変換ライブラリーでサポートされる基本コード・ページ名またはコード・ページ別名をコード・ページ値が識別することを確認してください (*International Components for Unicode: Converter Explorer* を参照)。コード・ページ値が有効な場合は、IBM 担当員に連絡してください。

システム処置: アプリケーションは終了します。

IWZ222S

エラー・コード *error code value* により、ICU を使用したデータ変換が失敗しました。

説明: ICU を使用したデータ変換が失敗しました。

プログラマー応答: IBM 担当員に連絡してください。

システム処置: アプリケーションは終了します。

IWZ223W

エラー・コード *error code value* により、ICU コンバーターのクローズが失敗しました。

説明: ICU コンバーターのクローズが失敗しました。

プログラマー応答: IBM 担当員に連絡してください。

システム処置: システムのアクションはとられません。

IWZ224S

ロケール値 *locale value* の ICU コレーターをオープンできません。 エラー・コードは *error code value* です。

説明: 当該ロケールの ICU コレーターをオープンできません。

プログラマー応答: IBM 担当員に連絡してください。

システム処置: アプリケーションは終了します。

IWZ225S

ICU を使用した Unicode ケース・マッピング関数がエラー・コード *error code value* で失敗しました。有効なロケールは *locale value* です。

説明: ICU のケース・マッピング関数が失敗しました。

プログラマー応答: IBM 担当員に連絡してください。

システム処置: アプリケーションは終了します。

IWZ230W

現行のコード・ページ *ASCII codeset-id* から EBCDIC コード・ページ *EBCDIC codeset-id* への変換テーブルを使用できません。 デフォルトの ASCII/EBCDIC 変換テーブルが使用されます。

説明: アプリケーションに、CHAR(EBCDIC) コンパイラー・オプションを使用してコンパイルされたモジュールがあります。 実行時には、現行の ASCII コード・ページから、EBCDIC_CODEPAGE 環境変数で指定された EBCDIC コード・ページへの変換を処理するための変換テーブルが構築されます。 指定されたコード・ページに対して使用可能な変換テーブルがないか、または EBCDIC_CODE ページの指定が無効なため、このエラーが発生しました。 実行は、ASCII コード・ページ IBM-1252 または同等のページおよび EBCDIC コード・ページ IBM-037 または同等のページに基づくデフォルトの変換テーブルを使用して続けられます。

プログラマー応答: EBCDIC_CODEPAGE 環境変数に有効な値が指定されていることを確認してください。

EBCDIC_CODEPAGE を設定しない場合は、デフォルト値の IBM-037 が使用されます。 これは、Enterprise COBOL for z/OS によって使用されるデフォルトのコード・ページです。

システム処置: システムのアクションはとられません。

IWZ230W

指定された EBCDIC コード・ページ *EBCDIC codepage* がロケール *locale* と一貫していませんが、要求どおりに使用されます。

説明: アプリケーションに、CHAR(EBCDIC) コンパイラー・オプションを使用してコンパイルされたモジュールがあります。 指定されたコード・ページが現行ロケールと同じ言語でないため、このエラーが発生しました。

プログラマー応答: EBCDIC_CODEPAGE 環境変数がこのロケールに対して有効なことを確認してください。

システム処置: システムのアクションはとられません。

IWZ230W

指定された EBCDIC コード・ページ *EBCDIC codepage* はサポートされていません。 デフォルトの EBCDIC コード・ページ *EBCDIC codepage* が使用されます。

説明: アプリケーションに、CHAR(EBCDIC) コンパイラー・オプションを使用してコンパイルされたモジュールがあります。 EBCDIC_CODEPAGE 環境変数の指定が無効なため、このエラーが発生しました。 実行は、現行ロケールに対応するデフォルトのホスト・コード・ページを使用して続けられます。

プログラマー応答: EBCDIC_CODEPAGE 環境変数に有効な値が指定されていることを確認してください。

システム処置: システムのアクションはとられません。

IWZ230S

EBCDIC 変換テーブルをオープンできません。

説明: 現行システムのインストールに、デフォルトの ASCII および EBCDIC コード・ページ用の変換テーブルが組み込まれていません。

プログラマー応答: コンパイラーおよびランタイムを再インストールしてください。 それでも問題が続く場合は、IBM 担当員に連絡してください。

システム処置: アプリケーションは終了します。

IWZ230S

EBCDIC 変換テーブルを構築できません。

説明: ASCII から EBCDIC への変換テーブルはオープンされましたが、変換が失敗しました。

プログラマー応答: 新規ウィンドウから実行を再試行してください。

システム処置: アプリケーションは終了します。

IWZ230S

メインプログラムは -host フラグと CHAR(NATIVE) オプションの両方でコンパイルされましたが、これらには互換性がありません。

説明: -host フラグと CHAR(NATIVE) オプションの両方でのコンパイルはサポートされていません。

プログラマー応答: -host フラグまたは CHAR(NATIVE) オプションのいずれかを除去してください。 -host フラグは CHAR(EBCDIC) を設定します。

システム処置: アプリケーションは終了します。

IWZ231S

現行のロケール設定の照会が失敗しました。

説明: 実行環境の照会で、有効なロケール設定を識別できませんでした。 適切なメッセージ・ファイルにアクセスして照合順序を設定するには、現行ロケールを設定する必要があります。 この設定は、日時サービスや EBCDIC 文字のサポートでも使用されます。

プログラマー応答: 次の環境変数の設定を確認してください。

LANG
これは、ご使用のマシンにインストールされているロケールに設定してください。 有効な値のリストを取得するには、`locale -a` と入力してください。 デフォルトは en_US です。

システム処置: アプリケーションは終了します。

IWZ232W

メッセージには、次のような種類があります。

- **プログラム *program-name* の行番号 *decimal-value* で、データ項目 *data-name* から EBCDIC への変換中にエラーが発生しました。**
- **プログラム *program-name* の行番号 *decimal-value* で、データ項目 *data-name* から ASCII への変換中にエラーが発生しました。**
- **プログラム *program-name* の行番号 *decimal-value* で、データ項目 *data-name* の EBCDIC への変換中にエラーが発生しました。**
- **プログラム *program-name* の行番号 *decimal-value* で、データ項目 *data-name* の ASCII への変換中にエラーが発生しました。**
- **プログラム *program-name* の行番号 *decimal-value* で、ASCII から EBCDIC への変換中にエラーが発生しました。**
- **プログラム *program-name* の行番号 *decimal-value* で、EBCDIC から ASCII への変換中にエラーが発生しました。**

説明: CHAR(EBCDIC) コンパイラー・オプションで要求されたとおりに、ID 内のデータを ASCII 形式と EBCDIC 形式間で変換できませんでした。

プログラマー応答: 適切な ASCII および EBCDIC ロケールがインストール済みで、選択されていることを確認してください。 ID 内のデータが有効で、ASCII と EBCDIC の両方の形式で表現できることを確認してください。

システム処置: システムのアクションはとられません。 データは未変換形式のままになります。

IWZ240S

プログラム *program-name* の基本年が 1900 から 1999 の有効範囲内にありません。 スライディング・ウィンドウ値 *window-value* は、*base-year* の基本年になります。

説明: 現行年と、YEARWINDOW コンパイラー・オプションで指定されたスライディング・ウィンドウ値を使用して 100 年間の範囲を計算しましたが、100 年間の基本となる年が 1900 から 1999 の有効範囲内にありませんでした。

プログラマー応答: アプリケーション設計を調べて、YEARWINDOW オプション値を変更できるかどうかを判別してください。 YEARWINDOW オプション値を変更してアプリケーションを実行できる場合は、適切な YEARWINDOW オプション値を使用してプログラムをコンパイルしてください。 YEARWINDOW オプション値を変更してアプリケーションを実行できない場合は、すべての日付フィールドを拡張日付に変換してから、NODATEPROC を使用してプログラムをコンパイルしてください。

システム処置: アプリケーションは終了します。

IWZ241S

現行年が、プログラム *program-name* に使用されている *year-start* から *year-end* の 100 年間内にありません。

説明: 現行年が、YEARWINDOW コンパイラー・オプション値で指定された 100 年間の固定範囲内にありませんでした。

例えば、YEARWINDOW(1920) を使用して COBOL プログラムをコンパイルした場合、そのプログラムに使用される 100 年間の範囲は 1920 から 2019 となります。 このプログラムを 2020 年に実行すると、現行年がこの 100 年間の範囲内にないため、エラー・メッセージが出されます。

プログラマー応答: アプリケーション設計を調べて、YEARWINDOW オプション値を変更できるかどうかを判別してください。 YEARWINDOW オプション値を変更してアプリケーションを実行できる場合は、適切な YEARWINDOW オプション値を使用してプログラムをコンパイルしてください。 YEARWINDOW オプション値を変更してアプリケーションを実行できない場合は、すべての日付フィールドを拡張日付に変換してから、NODATEPROC を使用してプログラムをコンパイルしてください。

システム処置: アプリケーションは終了します。

IWZ242S

XML PARSE ステートメントを開始しようとしましたが、この操作は無効です。

説明: COBOL プログラムによって開始された XML PARSE ステートメントがすでに進行中のときに、同じ COBOL プログラムが別の XML PARSE ステートメントを実行しようとしました。 1 回の COBOL プログラム呼び出しでアクティブにできる XML PARSE ステートメントは 1 つだけです。

プログラマー応答: 同じ COBOL プログラム内から別の XML PARSE ステートメントを開始しないようにアプリケーションを変更してください。

システム処置: アプリケーションは終了します。

IWZ243S

XML PARSE ステートメントを終了しようとしましたが、この操作は無効です。

説明: COBOL プログラムによって開始された XML PARSE ステートメントの進行中に、次のいずれかが試行されました。

- XML PARSE ステートメントを開始した COBOL プログラム内で、GOBACK または EXIT PROGRAM ステートメントが発行された。
- XML PARSE ステートメントを開始したプログラムと関連付けられたユーザー・ハンドラーが、条件ハンドラー再開カーソルを移動した後で、アプリケーションを再開した。

プログラマー応答: 上記のメソッドを使用せずに XML PARSE ステートメントを終了するようにアプリケーションを変更してください。

システム処置: アプリケーションは終了します。

IWZ813S

使用可能なストレージが不十分なため、ストレージ取得要求を満たすことができません。

説明: 使用可能なフリー・ストレージが不十分なため、ストレージ取得要求または再割り振り要求を満たすことができません。 このメッセージは、ストレージ管理でオペレーティング・システムから十分なストレージを取得できなかったことを意味します。

プログラマー応答: アプリケーションを実行できるだけの十分なストレージを確保してください。

システム処置: ストレージは割り振られません。

シンボリック・フィードバック・コード: CEE0PD

IWZ901S

メッセージには、次のような種類があります。

- **重大エラーまたはクリティカル・エラーが発生したため、プログラムが終了します。**
- **プログラムの終了: ERRCOUNT を超えるエラー数が発生しました。**

説明: それぞれの重大メッセージまたはクリティカル・メッセージの後には、IWZ901 メッセージが続きます。 ERRCOUNT ランタイム・オプションを使用した場合に、警告メッセージの数が ERRCOUNT を超えると、IWZ901 メッセージも出されます。

プログラマー応答: 重大メッセージまたはクリティカル・メッセージを参照して、ERRCOUNT の値を増やしてください。

システム処置: アプリケーションは終了します。

IWZ902S

システムが 10 進数除算例外を検出しました。

説明: ある数値を 0 で除算しようとしたことが検出されました。

プログラマー応答: プログラムを修正してください。 例えば、フラグ付きのステートメントに ON SIZE ERROR を追加します。

システム処置: アプリケーションは終了します。

IWZ903S

システムがデータ例外を検出しました。

説明: データに無効値が含まれているため、パック 10 進数データまたはゾーン 10 進数データに対する操作が失敗しました。

プログラマー応答: データが有効なパック 10 進数データまたはゾーン 10 進数データであることを確認してください。

システム処置: アプリケーションは終了します。

IWZ907S

メッセージには、次のような種類があります。

- **ストレージが不十分です。**
- **ストレージが不十分です。 *storage* 用に *number-bytes* バイトのスペースを取得できません。**

説明: ランタイム・ライブラリーが仮想メモリー・スペースを要求しましたが、オペレーティング・システムがこの要求を拒否しました。

プログラマー応答: プログラムが大量の仮想メモリーを使用するため、スペースが不足しています。 一般に、この問題の原因は特定のステートメントではなく、プログラム全体に関連しています。 OCCURS 文節の使用状況を調べて、テーブルのサイズを減らしてください。

システム処置: アプリケーションは終了します。

IWZ993W

ストレージが不十分です。 メッセージ *message-number* 用のスペースを検出できません。

説明: ランタイム・ライブラリーが仮想メモリー・スペースを要求しましたが、オペレーティング・システムがこの要求を拒否しました。

プログラマー応答: プログラムが大量の仮想メモリーを使用するため、スペースが不足しています。 一般に、この問題の原因は特定のステートメントではなく、プログラム全体に関連しています。 OCCURS 文節の使用状況を調べて、テーブルのサイズを減らしてください。

システム処置: システムのアクションはとられません。

IWZ994W

メッセージ・カタログ 内でメッセージ *message-number* を検出できません。

説明: ランタイム・ライブラリーが、メッセージ・カタログ自体またはメッセージ・カタログ内の特定のメッセージを検出できません。

プログラマー応答: COBOL ライブラリーおよびメッセージが正しくインストールされていることと、LANG および NLSPATH が正しく指定されていることを確認してください。

システム処置: システムのアクションはとられません。

IWZ995C

メッセージには、次のような種類があります。

- **オフセット 0x*offset-value* でルーチン *routine-name* の実行中に、*system exception* シグナルを受信しました。**
- **0x*offset-value* の場所でコードの実行中に、*system exception* シグナルを受信しました。**
- ***system exception* シグナルを受信しました。 場所を判別できませんでした。**

説明: オペレーティング・システムが、無許可のアクション (保護記憶域にデータを格納しようとしたなど) を検出したか、または割り込みキー (一般には Control + C キーだが、再構成はできない) が押されたことを検出しました。

プログラマー応答: 無許可のアクションがシグナルの原因の場合は、デバッガーでプログラムを実行すると、エラーの発生場所に関する詳細情報を得ることができます。 このタイプのエラーの例としては、無効な値を持つポインターなどが挙げられます。

システム処置: アプリケーションは終了します。

IWZ2502S

システムから UTC/GMT を使用できませんでした。

説明: システム・クロックが無効な状態になっていたため、CEEUTC または CEEGMT の呼び出しが失敗しました。 現在時刻を判別できませんでした。

プログラマー応答: システム・クロックが無効な状態になっていることをシステム・サポート担当者に連絡してください。

システム処置: 出力値がすべて 0 に設定されます。

シンボリック・フィードバック・コード: CEE2E6

IWZ2503S

UTC/GMT から現地時間までのオフセットをシステムから使用できませんでした。

説明: (1) 現行のオペレーティング・システムを判別できなかったか、(2) オペレーティング・システムの制御ブロックにある時間帯フィールドに無効なデータが含まれている可能性があるため、CEEGMTO の呼び出しが失敗しました。

プログラマー応答: オペレーティング・システムに格納されている現地時間のオフセットに無効なデータが含まれている可能性があることをシステム・サポート担当者に連絡してください。

システム処置: 出力値がすべて 0 に設定されます。

シンボリック・フィードバック・コード: CEE2E7

IWZ2505S

CEEDATM または CEESECI への呼び出し内の input_seconds 値が、対応範囲内にありませんでした。

説明: CEEDATM または CEESECI の呼び出しで渡された input_seconds 値が、86,400.0 から 265,621,679,999.999 の範囲内の浮動小数点数ではありませんでした。入力パラメーターは、1582 年 10 月 14 日の 00:00:00 から数えた秒数で表す必要があります。ここでサポートされる最初の日時は 1582 年 10 月 15 日の 00:00:00.000、最後の日時は 9999 年 12 月 31 日の 23:59:59.999 です。

プログラマー応答: 入力パラメーターに含まれている浮動小数点値が 86,400.0 から 265,621,679,999.999 の範囲内にあることを確認してください。

システム処置: CEEDATM の場合は、出力値がブランクに設定されます。 CEESECI の場合は、出力パラメーターがすべて 0 に設定されます。

シンボリック・フィードバック・コード: CEE2E9

IWZ2506S

CEEDATM に渡されたピクチャー・ストリング内に元号 (<JJJJ>、<CCCC>、 <CCCCCCCC>) が使用されていましたが、入力された秒数値が対応範囲内にありませんでした。 元号を判別できませんでした。

説明: CEEDATM の呼び出しでは、ピクチャー・ストリングは入力値が元号に変換されることを示しますが、指定された入力値は、サポートされる元号の範囲内にありません。

プログラマー応答: サポートされる元号の範囲内にある有効な秒数値が入力値に含まれていることを確認してください。

システム処置: 出力値がブランクに設定されます。

IWZ2507S

CEEDAYS または CEESECS に渡されたデータが不十分です。 リリアン日付の値は計算されませんでした。

説明: CEEDAYS または CEESECS の呼び出しで渡されたピクチャー・ストリングに、十分な情報が含まれていませんでした。 例えば、CEEDAYS または CEESECS の呼び出しでピクチャー・ストリング 'MM/DD' (月

と日のみ) を使用すると、年の値がないためエラーとなります。 リリアン日付の値を計算するために最低限必要な情報は、(1) 月、日、年、または (2) 年、ユリウス日のいずれかです。

プログラマー応答: CEEDAYS または CEESECS の呼び出しで指定されたピクチャー・ストリングに少なくとも、(1) 月、日、年、または (2) 年、ユリウス日のいずれかの入力ストリング内の場所が指定されていることを確認してください。

システム処置: 出力値が 0 に設定されます。

シンボリック・フィードバック・コード: CEE2EB

IWZ2508S

CEEDAYS または CEESECS に渡された日付値が無効です。

説明: CEEDAYS または CEESECS の呼び出しで、DD または DDD フィールドの値が当該年/月に対して無効です。 例えば、1990 年はうるう年ではないため、'MM/DD/YY' の値が '02/29/90' の場合や 'YYYY.DDD' の値が '1990.366' の場合は無効となります。 また、6 月 31 日や 1 月 0 日など、存在しない日付値を指定した場合にも、このコードが戻されることがあります。

プログラマー応答: 入力データの形式がピクチャー・ストリング指定と一致していることと、入力データに有効な日付が含まれていることを確認してください。

システム処置: 出力値が 0 に設定されます。

シンボリック・フィードバック・コード: CEE2EC

IWZ2509S

CEEDAYS または CEESECS に渡された元号が認識されませんでした。

説明: CEEDAYS または CEESECS の呼び出しで渡された <JJJJ>、<CCCC>、または <CCCCCCCC> フィールドの値に、サポートされる元号名が含まれていません。

プログラマー応答: 入力データの形式がピクチャー・ストリング指定と一致していることと、元号名のスペルが正しいことを確認してください。 元号名は、正しい DBCS ストリングでなければならないことに注意してください。'<' の位置には、元号名の先頭バイトが含まれていなければなりません。

システム処置: 出力値が 0 に設定されます。

IWZ2510S

CEEISEC または CEESECS への呼び出しで時間の値が認識されませんでした。

説明: (1) CEEISEC の呼び出しで、時のパラメーターに 0 から 23 までの範囲内の数値が含まれていませんでした。あるいは、(2) CEESECS の呼び出しで、HH (時) フィールドの値に 0 から 23 までの範囲内の数値が含まれていないか、または「AP」(午前/午後) フィールドが指定されていて、HH フィールドに 1 から 12 までの範囲内の数値が含まれていません。

プログラマー応答: CEEISEC の場合は、時のパラメーターに 0 から 23 までの範囲内の整数が含まれていることを確認してください。 CEESECS の場合は、入力データの形式がピクチャー・ストリング指定と一致することと、時のフィールドに 0 から 23 (「AP」フィールドを使用している場合は 1 から 12) の範囲内の値が含まれていることを確認してください。

システム処置: 出力値が 0 に設定されます。

シンボリック・フィードバック・コード: CEE2EE

IWZ2511S

CEEISEC の呼び出しで渡された日のパラメーターが、指定された年および月に対して無効です。

説明: CEEISEC の呼び出しで渡された日のパラメーターに、有効な日数値が含まれていませんでした。 年、月、日の組み合わせが、無効な日付値になっています。 例えば、1990 年 2 月 29 日、6 月 31 日、0 日などの日付は無効です。

プログラマー応答: 日のパラメーターに 1 から 31 の範囲内の整数が含まれていることと、年、月、日の組み合わせが有効な日付を表していることを確認してください。

システム処置: 出力値が 0 に設定されます。

シンボリック・フィードバック・コード: CEE2EF

IWZ2512S

CEEDATE または CEEDYWK への呼び出しで渡されたリリアン日付値が、対応範囲内にありませんでした。

説明: CEEDATE または CEEDYWK の呼び出しで渡されたリリアン日付値が、1 から 3,074,324 の範囲内の数値ではありませんでした。

プログラマー応答: 入力パラメーターに 1 から 3,074,324 の範囲内の整数が含まれていることを確認してください。

システム処置: 出力値がブランクに設定されます。

シンボリック・フィードバック・コード: CEE2EG

IWZ2513S

CEEISEC、CEEDAYS、CEESECS のいずれかの呼び出しで渡された入力日付が、対応範囲内にありませんでした。

説明: CEEISEC、CEEDAYS、CEESECS のいずれかの呼び出しで渡された入力日付が、1582 年 10 月 15 日より前か、9999 年 12 月 31 日より後に設定されていました。

プログラマー応答: CEEISEC の場合は、年、月、日のパラメーターが、1582 年 10 月 15 日以降の日付になっていることを確認してください。 CEEDAYS および CEESECS の場合は、入力される日付の形式がピクチャー・ストリング指定と一致することと、入力される日付が対応範囲内にあることを確認してください。

システム処置: 出力値が 0 に設定されます。

シンボリック・フィードバック・コード: CEE2EH

IWZ2514S

CEEISEC の呼び出しで渡された年の値が、対応範囲内にありませんでした。

説明: CEEISEC の呼び出しで渡された年のパラメーターに、1582 から 9999 の範囲内の数値が含まれていませんでした。

プログラマー応答: 年のパラメーターに有効なデータが含まれていることと、年のパラメーターに世紀が含まれている (例えば、90 年ではなく 1990 年と指定している) ことを確認してください。

システム処置: 出力値が 0 に設定されます。

シンボリック・フィードバック・コード: CEE2EI

IWZ2515S

CEEISEC 呼び出し内のミリ秒の値が認識されませんでした。

説明: CEEISEC の呼び出しで、ミリ秒のパラメーター (*input_milliseconds*) に 0 から 999 の範囲内の数値が含まれていませんでした。

プログラマー応答: ミリ秒のパラメーターに 0 から 999 の範囲内の整数が含まれていることを確認してください。

システム処置: 出力値が 0 に設定されます。

シンボリック・フィードバック・コード: CEE2EJ

IWZ2516S

CEEISEC 呼び出し内の分の値が認識されませんでした。

説明: (1) CEEISEC の呼び出しの場合は、分のパラメーター (*input_minutes*) に 0 から 59 の範囲内の数値が含まれていませんでした。(2) CEESECS の呼び出しの場合は、MI (分) フィールドの値に 0 から 59 の範囲内の数値が含まれていませんでした。

プログラマー応答: CEEISEC の場合は、分のパラメーターに 0 から 59 の範囲内の整数が含まれていることを確認してください。 CEESECS の場合は、入力データの形式がピクチャー・ストリング指定と一致することと、分のフィールドに 0 から 59 の範囲内の値が含まれていることを確認してください。

システム処置: 出力値が 0 に設定されます。

シンボリック・フィードバック・コード: CEE2EK

IWZ2517S

CEEISEC 呼び出し内の月の値が認識されませんでした。

説明: (1) CEEISEC の呼び出しの場合は、月のパラメーター (*input_month*) に 1 から 12 の範囲内の数値が含まれていませんでした。(2) CEEDAYS または CEESECS の呼び出しの場合は、MM フィールドの値に 1 から 12 の範囲内の数値が含まれていないか、あるいは MMM や MMMM などのフィールドの値に、現在アクティブな各国語で正しいスペルの月の名前または省略後が含まれていませんでした。

プログラマー応答: CEEISEC の場合は、月のパラメーターに 1 から 12 の範囲内の整数が含まれていることを確認してください。 CEEDAYS および CEESECS の場合は、入力データの形式がピクチャー・ストリング指定と一致することを確認してください。 MM フィールドの場合は、入力値が 1 から 12 の範囲内にあることを確認してください。 月名 (MMM、MMMM など) を指定する場合は、その月名のスペルまたは省略語が、現在アクティブな各国語で正しく指定されていることを確認してください。

システム処置: 出力値が 0 に設定されます。

シンボリック・フィードバック・コード: CEE2EL

IWZ2518S

日時サービスへの呼び出しに無効なピクチャー・ストリングが指定されました。

説明: いずれかの日時サービスの呼び出しで指定されたピクチャー・ストリングが無効です。 指定できるのは 1 つの元号文字ストリングだけです。

プログラマー応答: ピクチャー・ストリングに有効なデータが含まれていることを確認してください。 ピクチャー・ストリングに複数の元号記述子 (<JJJJ> と <CCCC> の両方など) が含まれている場合は、一方の元号だけを使用するようにピクチャー・ストリングを変更してください。

システム処置: 出力値が 0 に設定されます。

シンボリック・フィードバック・コード: CEE2EM

IWZ2519S

CEEISEC 呼び出し内の秒の値が認識されませんでした。

説明: (1) CEEISEC の呼び出しの場合は、秒のパラメーター (*input_seconds*) に 0 から 59 の範囲内の数値が含まれていませんでした。(2) CEESECS の呼び出しの場合は、SS (秒) フィールドの値に 0 から 59 の範囲内の数値が含まれていませんでした。

プログラマー応答: CEEISEC の場合は、秒のパラメーターに 0 から 59 の範囲内の整数が含まれていることを確認してください。 CEESECS の場合は、入力データの形式がピクチャー・ストリング指定と一致することと、秒のフィールドに 0 から 59 の範囲内の値が含まれていることを確認してください。

システム処置: 出力値が 0 に設定されます。

シンボリック・フィードバック・コード: CEE2EN

IWZ2520S

CEEDAYS が数値フィールド内に非数値データを検出したか、あるいは日付ストリングとピクチャー・ストリングが一致しませんでした。

説明: CEEDAYS の呼び出しで渡された入力値が、ピクチャー指定で記述された形式ではありませんでした (例えば、数字のみが期待される場所に非数字があるなど)。

プログラマー応答: 入力データの形式がピクチャー・ストリング指定と一致していることと、数値フィールドに数値データしか含まれていないことを確認してください。

システム処置: 出力値が 0 に設定されます。

シンボリック・フィードバック・コード: CEE2EO

IWZ2521S

CEEDAYS または CEESECS に渡された <JJJJ>、<CCCC>、または <CCCCCCCC> の元号年数値がゼロでした。

説明: CEEDAYS または CEESECS の呼び出しで、YY または ZYY ピクチャー・トークンが指定されている場合で、なおかつピクチャー・ストリングにいずれかの元号トークン (<CCCC> や <JJJJ> など) が含まれている場合は、年の値が 1 以上で、その元号に対して有効な値でなければなりません。 この場合、YY または ZYY フィールドは当該元号における年を意味します。

プログラマー応答: 入力データの形式がピクチャー・ストリング指定と一致していることと、入力データが有効なことを確認してください。

システム処置: 出力値が 0 に設定されます。

IWZ2522S

CEEDATE に渡されたピクチャー・ストリング内に元号 (<JJJJ>、<CCCC>、 <CCCCCCCC>) が使用されていましたが、リリアン日付値が対応範囲内にありませんでした。 元号を判別できませんでした。

説明: CEEDATE の呼び出しでは、ピクチャー・ストリングはリリアン日付が元号に変換されることを示しますが、リリアン日付がサポートされる元号の範囲内にありません。

プログラマー応答: サポートされる元号の範囲内にある有効なリリアン日付値が入力値に含まれていることを確認してください。

システム処置: 出力値がブランクに設定されます。

IWZ2525S

CEESECS が数値フィールド内に非数値データを検出したか、あるいはタイム・スタンプ・ストリングとピクチャー・ストリングが一致しませんでした。

説明: CEESECS の呼び出しで渡された入力値が、ピクチャー指定で記述された形式ではありませんでした。例えば、数字のみが期待される場所に非数字がある、a.m./p.m. フィールド (AP、A.P. など) に 'AM' または 'PM' のストリングが含まれていないなどが考えられます。

プログラマー応答: 入力データの形式がピクチャー・ストリング指定と一致していることと、数値フィールドに数値データしか含まれていないことを確認してください。

システム処置: 出力値が 0 に設定されます。

シンボリック・フィードバック・コード: CEE2ET

IWZ2526S

CEEDATE によって戻された日付ストリングが切り捨てられました。

説明: CEEDATE の呼び出しで、出力ストリングのサイズが足りないため、フォーマットした日付値を格納できませんでした。

プログラマー応答: 出力ストリングのデータ項目が、フォーマットされた日付全体を格納できるだけの十分なサイズになっていることを確認してください。 出力パラメーターが、少なくともピクチャー・ストリング・パラメーターと同じ長さになっていることを確認してください。

システム処置: 出力値が出力パラメーターの長さまで切り詰められます。

シンボリック・フィードバック・コード: CEE2EU

IWZ2527S

CEEDATM によって戻されたタイム・スタンプ・ストリングが切り捨てられました。

説明: CEEDATM の呼び出しで、出力ストリングのサイズが足りないため、フォーマットしたタイム・スタンプ値を格納できませんでした。

プログラマー応答: 出力ストリングのデータ項目が、フォーマットされたタイム・スタンプ全体を格納できるだけの十分なサイズになっていることを確認してください。 出力パラメーターが、少なくともピクチャー・ストリング・パラメーターと同じ長さになっていることを確認してください。

システム処置: 出力値が出力パラメーターの長さまで切り詰められます。

シンボリック・フィードバック・コード: CEE2EV

IWZ2531S

システムから現地時間を使用できませんでした。

説明: システム・クロックが無効な状態になっていたため、CEELOCT の呼び出しが失敗しました。 現在時刻を判別できません。

プログラマー応答: システム・クロックが無効な状態になっていることをシステム・サポート担当者に連絡してください。

システム処置: 出力値がすべて 0 に設定されます。

シンボリック・フィードバック・コード: CEE2F3

IWZ2533S

CEESCEN に渡された値が 0 から 100 の範囲内にありませんでした。

説明: CEESCEN の呼び出しで渡された *century_start* の値が、0 から 100 の範囲内にありませんでした。

プログラマー応答: 入力パラメーターが有効範囲内にあることを確認してください。

システム処置: システムのアクションはとられません。すべての 2 桁年号に対して想定される 100 年間の範囲は変わりません。

シンボリック・フィードバック・コード: CEE2F5

IWZ2534W

CEEDATE または CEEDATM への呼び出しで、月または曜日名に対して指定されたフィールド幅が不十分です。 出力はブランクに設定されました。

説明: CEEDATE または CEEDATM 呼び出し可能サービスで、ピクチャー・ストリングに正しいスペルの月名や曜日名が要求される MMM、MMMMMZ、WWW、 Wwww などが含まれているにもかかわらず、現在フォーマットされている月名に含まれている文字数が指定フィールド内に収まらない場合は、このメッセージが出されます。

プログラマー応答: フォーマットされる最長の月名または曜日名を格納できるだけの十分な数の M または W を指定して、フィールド幅を増やしてください。

システム処置: 幅が不十分な月名および曜日名フィールドは、ブランクに設定されます。 残りの出力ストリングは影響を受けません。 処理を続行します。

シンボリック・フィードバック・コード: CEE2F6

関連概念
531 ページの『付録 C 中間結果および算術精度』

関連タスク

215 ページの『環境変数の設定』
231 ページの『コンパイラー・メッセージのリストの生成』

特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。 日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。 本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。 これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。 ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。 本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。 実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒 103-8510
東京都中央区日本橋箱崎町 19 番 21 号
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

以下の保証は、国または地域の法律に沿わない場合は、適用されません。IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。 国または地域によっては、法律の強行規定により、保証責任の制限が 禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。 本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
U.S.A.

本プログラムに関する上記の情報は、適切な使用条件の下で使用すること ができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で 決定されたものです。 そのため、他の操作環境で得られた結果は、異なる可能性があります。 一部の測定が、開発レベルのシステムで行われた可能性がありますが、 その測定値が、一般に利用可能なシステムのものと同じである保証はありません。 さらに、一部の測定値が、推定値である可能性があります。 実際の結果は、異なる可能性があります。 お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

© Copyright IBM Corp. 2021

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。 より具体性を与えるために、それらの例には、個人、企業、ブランド、 あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、 名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。 お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。 このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。 従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。 これらのサンプル・プログラムは特定物として現存するままの状態で提供されるものであり、いかなる保証も提供されません。 IBM は、お客様の当該サンプル・プログラムの使用から生ずるいかなる損害に対しても一切の責任を負いません。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生的創作物にも、次のように、著作権表示を入れていただく必要があります。

© (お客様の会社名) (西暦年). このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。© Copyright IBM Corp. 1995, 2019.

プライバシー・ポリシーに関する考慮事項:

サービス・ソリューションとしてのソフトウェアも含めた IBM ソフトウェア製品 (「ソフトウェア・オファリング」) では、製品の使用に関する情報の収集、エンド・ユーザーの使用感の向上、エンド・ユーザーとの対話またはその他の目的のために、Cookie はじめさまざまなテクノロジーを使用することがあります。多くの場合、ソフトウェア・オファリングにより個人情報が収集されることはありません。 IBM の「ソフトウェア・オファリング」の一部には、個人情報を収集できる機能を持つものがあります。 ご使用の「ソフトウェア・オファリング」が、これらの Cookie およびそれに類するテクノロジーを通じてお客様による個人情報の収集を可能にする場合、以下の具体的事項をご確認ください。

この「ソフトウェア・オファリング」は、Cookie もしくはその他のテクノロジーを使用して個人情報を収集することはありません。

この「ソフトウェア・オファリング」が Cookie およびさまざまなテクノロジーを使用してエンド・ユーザーから個人を特定できる情報を収集する機能を提供する場合、お客様は、このような情報を収集するにあたって適用される法律、ガイドライン等を遵守する必要があります。 これには、エンド・ユーザーへの通知や同意の要求も含まれますがそれらには限られません。

このような目的での Cookie を含む様々なテクノロジーの使用の詳細については、『IBM オンラインでのプライバシー・ステートメント』(http://www.ibm.com/privacy/details/jp/ja/) の『クッキー、ウェブ・ビーコン、その他のテクノロジー』および「IBM Software Products and Software-as-a-Service Privacy Statement」(http://www.ibm.com/software/info/product-privacy) を参照してください。

商標

IBM、IBM ロゴおよび ibm.com® は、世界の多くの国で登録された International Business Machines Corporation の商標です。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。 現時点での IBM の商標リストについては、www.ibm.com/legal/copytrade.shtml をご覧ください。

Intel は、Intel Corporation または子会社の米国およびその他の国における商標または登録商標です。

Java およびすべての Java 関連の商標およびロゴは Oracle やその関連会社の米国およびその他の国における商標または登録商標です。

Linux は、Linus Torvalds の米国およびその他の国における商標です。

Microsoft および Windows は、Microsoft Corporation の米国およびその他の国における商標です。

UNIX は The Open Group の米国およびその他の国における登録商標です。

用語集

この用語集に記載されている用語は、COBOL における意味に従って定義されています。 これらの用語は、他の言語では同じ意味を持つことも、持たないこともあります。

glossary.html

この用語集には、以下の資料からの用語および定義が記載されています。

- 「*ANSI INCITS 23-1985, Programming languages - COBOL*」(「*ANSI INCITS 23a-1989, Programming Languages - COBOL - Intrinsic Function Module for COBOL*」および「*ANSI INCITS 23b-1993, Programming Languages - Correction Amendment for COBOL*」で改訂)
- 「*ISO 1989:1985, Programming languages - COBOL*」は「*ISO/IEC 1989/AMD1:1992, Programming languages - COBOL: Intrinsic function module*」に改訂されました。
- 「*ANSI X3.172-2002, American National Standard Dictionary for Information Systems*」
- *INCITS/ISO/IEC 1989-2002, Information technology - Programming languages - COBOL*
- *INCITS/ISO/IEC 1989:2014, Information technology - Programming languages, their environments and system software interfaces - Programming language COBOL*

米国標準規格 (ANS) の定義の前にはアスタリスク (*) を付けています。

A

簡略複合比較条件 (* abbreviated combined relation condition)
連続した一連の比較条件において、共通サブジェクトの明示的な省略、または共通サブジェクトと共通関係演算子の明示的な省略によって生じる複合条件。

異常終了 (abend)
プログラムの異常終了。

アクセス・モード (* access mode)
ファイル内でレコードが操作される方式。

実際の小数点 (* actual decimal point)
10 進小数点文字のピリオド (.) またはコンマ (,) によるデータ項目における小数点位置の物理表現。

実際の文書エンコード (actual document encoding)
XML 文書のエンコード・カテゴリーで、以下のいずれかとなる。XML パーサーは文書の最初の数バイトを調べて判別する。

- ASCII
- EBCDIC
- UTF-8
- UTF-16 (ビッグ・エンディアンまたはリトル・エンディアンのいずれか)
- これ以外のサポートされないエンコード
- 認識不能なエンコード

Linux ネイティブ・ファイル・システム (Linux native file system)
エンコード・ファイルまたはバイナリー・ストリーム・ファイルを直接サポートする任意のローカルまたはネットワーク・ファイル・システム。

Linux ネイティブ・ファイル・システムは、行順次ファイルを直接サポートし、他のすべての COBOL ファイル・タイプのファイル・ストアとして使用される。

英字名 (* alphabet-name)
ENVIRONMENT DIVISION の SPECIAL-NAMES 段落のユーザー定義語であり、特定の文字セットまたは照合シーケンス (あるいはその両方) に名前を割り当てるもの。

英字 (* alphabetic character)
文字または空白文字。

© Copyright IBM Corp. 2021

英字データ項目 (alphabetic data item)
記号 A のみを含む PICTURE 文字ストリングが記述されたデータ項目。英字データ項目は USAGE DISPLAY を持ちます。

英数字 (* alphanumeric character)
コンピューターの 1 バイト文字セットの任意の文字。

英数字位置 (alphanumeric character position)
「文字位置 *(character position)*」を参照。

英数字データ項目 (alphanumeric data item)
暗黙的または明示的に USAGE DISPLAY として記述された、カテゴリー英数字、英数字編集、または数字編集を持つデータ項目を 指す一般的な呼び方。

英数字編集データ項目 (alphanumeric-edited data item)
少なくとも 1 つの記号 A または X のインスタンスおよび少なくとも 1 つの単純挿入記号 B、0、または / を含んでいる、PICTURE 文字ストリングで記述されたデータ項目。 英数字編集データ項目は USAGE DISPLAY を持ちます。

英数字関数 (* alphanumeric function)
コンピューターの英数字セットからの 1 つ以上の文字のストリングで値が構成されている関数。

英数字グループ項目 (alphanumeric group item)
GROUP-USAGE NATIONAL 節なしで定義されたグループ項目。 INSPECT、STRING、および UNSTRING などの操作の場合、英数字グループ項目は、実際のグループの内容にかかわらず、その内容すべてが USAGE DISPLAY として記述されているかのように処理されます。 グループ内の基本項目を処理する必要のある操作 (MOVE CORRESPONDING、ADD CORRESPONDING、または INITIALIZE など) の場合、英数字グループ項目はグループ・セマンティクスを使用して処理されます。

英数字リテラル (alphanumeric literal)
次のセットからの開始区切り文字を有するリテラル。'、"、X'、X"、Z'、または Z"。 この文字ストリングには、コンピューターの有する文字セットの任意の文字を含めることができる。

代替レコード・キー (* alternate record key)
基本レコード・キー以外のキーであり、その内容が索引付きファイル内のレコードを識別する。

米国規格協会 (American National Standards Institute: ANSI)
米国で認定された組織が自発的工業規格を作成して維持する手順を設定する組織であり、製造業者、消費者、および一般の利害関係者で構成される。

引数 (argument)
(1) ID、リテラル、算術式、または関数 ID で、これにより関数の評価に使用する値を指定する。 (2) CALL ステートメント の USING 句のオペランドであり、呼び出されたプログラムに値を渡すのに使用されます。

算術演算 (* arithmetic operation)
ある算術ステートメントが実行されることにより、またはある算術式が計算されることにより生じるプロセスで、そこで与えられている引数に対して数学的に正しい解が求められる。

算術演算子 (* arithmetic operator)
次に示す集合に属する 1 文字、または 2 文字で構成された固定した組み合わせ。

文字	意味
+	加算
-	減算
*	乗算
/	除算
**	指数

算術ステートメント (* arithmetic statement)
算術演算を実行させるステートメント。 算術ステートメントには、ADD、COMPUTE、DIVIDE、MULTIPLY、および SUBTRACT の各ステートメントがある。

配列 (array)
データ・オブジェクトで構成される集合体。それぞれのオブジェクトは添え字付けによって一意的に参照できる。 配列は、COBOL ではテーブルに類似する。

昇順キー (* ascending key)
データ項目を比較する際の規則に一致するように、最低のキー値から始めて最高のキー値へとデータを順序付けている値に即したキー。

ASCII
情報交換用米国標準コード。 7 ビットのコード化文字をベースとする 1 つのコード化文字セット (パリティー・チェックを含む 8 ビット) を 使用する標準コードであり、データ処理システム、データ通信システム、および関連装置の間での情報交換に使用される。 ASCII セットは、制御文字と図形文字から構成されている。

IBM は、ASCII に対する拡張 (文字 128 から 255) を定義している。

ASCII ベース・マルチバイト・コード・ページ (ASCII-based multibyte code page)
UTF-8、EUC、または ASCII DBCS コード・ページ。 各 ASCII ベース・マルチバイト・コード・ページは、1 バイト文字とマルチバイト文字の両方を含む。 1 バイト文字のエンコードは ASCII エンコードである。

ASCII DBCS
「*2* バイト *ASCII (double-byte ASCII)*」を参照。

割り当て名 (assignment-name)
COBOL ファイルの編成を識別する名前で、システムがこれを認識する際に使用する。

想定小数点 (* assumed decimal point)
データ項目の中に実際には小数点のための文字が入っていない小数点位置。 想定小数点には、論理的な意味があり、物理的には表現されない。

AT END 条件 (AT END condition)
次のような特定の条件のもとで、READ、RETURN、または SEARCH ステートメントを実行した場合に引き起こされる条件。

- 順次アクセス・ファイルに対して READ ステートメントを実行中に、そのファイル内に次の論理レコードが存在しない場合、または相対レコード番号中の有効数字の桁数が相対キー・データ項目のサイズより大きい場合、またはオプションの入力ファイルが使用可能でない場合。
- RETURN ステートメントの実行中に、関連するソート・ファイルまたはマージ・ファイルについての次の論理レコードが存在しない場合。
- SEARCH ステートメントの実行中に、関連する WHEN 句のいずれかで指定された条件を満足することなく、検索操作が終了した場合。

B

基本文字セット (basic character set)
言語のワード、文字ストリング、および区切り文字の作成時に使用される基本的な文字セット。 基本文字セットは 1 バイトの文字でインプリメントされる。 拡張文字セットには DBCS、UTF-8、または EUC 文字が含まれる。これは、コメント、リテラル、およびユーザー定義語で使用できる。

85 COBOL 標準 における「*COBOL* 文字セット *(COBOL character set)*」と同義。

ビッグ・エンディアン (big-endian)
メインフレームおよび Linux ワークステーションがバイナリー・データおよび UTF-16 文字を保存するときに使用するデフォルト形式。 この形式では、バイナリー・データ項目の最下位バイトが最高位アドレス にあり UTF-16 文字の最下位バイトが最高位アドレスにあります。リトル・エンディアン *(little-endian)* と比較。

バイナリー項目 (binary item)
2 進表記 (基数 2 の数体系) で表される数値データ項目。 等価の 10 進数は、10 進数字 0 から 9 に演算符号を加えたもので構成される。 項目の左端ビットは演算符号。

二分探索 (binary search)
二分探索では、探索の各ステップで、一連のデータ・エレメントの集合が 2 つに分割される。 エレメントの数が奇数の場合には、何らかの適切なアクションが取られる。

ブロック (* block)
通常は 1 つ以上の論理レコードで構成される物理的データ単位。 大容量記憶ファイルの場合、ある論理レコードの一部がブロックに入ることがある。 ブロックのサイズは、そのブロックが含まれているファイルのサイズと直接関係はなく、 そのブロックに含まれているか、そのブロックにオーバーラップしている論理レコード のサイズとも直接関係はない。 「物理レコード *(physical record)*」と同義。

ブール条件 (boolean condition)
ブール条件は、ブール・リテラルが true であるか false であるかを決定する。 ブール条件は定数条件式でのみ使用できる。

ブール・リテラル (boolean literal)
true 値を示す B'1'、または false 値を示す B'0' のどちらか。 ブール・リテラルは定数条件式でのみ使用できる。

停止点 (breakpoint)
通常は命令によって指定されるコンピューター・プログラムの場所であり、プログラムの実行は外部からの介入またはモニター・プログラムによって割り込まれる場合がある。

バッファー (buffer)
入力データまたは出力データを一時的に保持するために使用されるストレージの一部分。

組み込み関数 (built-in function)
組み込み関数 *(intrinsic function)* を参照。

バイト (byte)
特定の数のビット (通常 8 ビット) から成るストリングであり、1 つの単位として処理され、1 つの文字または制御機能を表す。

byte order mark (BOM)
UTF-16 または UTF-32 テキストの先頭に使用して、後続テキストのバイト・オーダーを示す Unicode 文字。バイト・オーダーには、「ビッグ・エンディアン (big-endian)」または「リトル・エンディアン (little-endian)」がある。

バイトコード (bytecode)
Java コンパイラーによって生成され、Java インタープリターによって実行される、マシンから独立したコード。(Oracle)

C

呼び出し先プログラム (called program)
CALL ステートメントの対象となるプログラム。 呼び出し先プログラムと呼び出し側プログラムが実行時に結合されて、1 つの実行単位が作成される。

呼び出し側プログラム (* calling program)
別のプログラムへの CALL を実行するプログラム。

ケース構造 (case structure)
結果として生じた多数のアクションの中から選択を行うために、一連の条件をテストするプログラム処理ロジック。

CCSID
コード化文字セット *ID (coded character set identifier)* を参照。

世紀ウィンドウ (century window)
2 桁年号が固有に決まる 100 年間のこと。 COBOL プログラマーが使用できる世紀ウィンドウには、いくつかのタイプがある。

- ウィンドウ表示日付フィールドについては、YEARWINDOW コンパイラー・オプションを使用する。
- ウィンドウ操作組み込み関数 DATE-TO-YYYYMMDD、DAY-TO-YYYYDDD、および YEAR-TO-YYYY については、引数*-2 (argument-2)* によって世紀ウィンドウを指定する。

文字 (* character)
言語のそれ以上分割できない基本単位。

文字エンコード・ユニット (character encoding unit)
コード化文字セット内の 1 つのコード・ポイントに相当するデータの単位。 1 つ以上の文字エンコード・ユニットを使用して、コード化文字セットの文字が表現される。 エンコード・ユニット とも呼ばれる。

USAGE NATIONAL の場合、文字エンコード・ユニットは、 UTF-16 の 2 バイト・コード・ポイントに対応している。

USAGE DISPLAY の場合、文字エンコード・ユニットは、1 つのバイトに対応している。

USAGE DISPLAY-1 の場合、文字エンコード・ユニットは、 DBCS 文字セットの 2 バイト・コード・ポイントに対応している。

文字位置 (character position)
1 文字を保持または表示するために必要な物理ストレージまたは表示スペースの量。 この用語はどのような文字のクラスにも適用される。 文字の特定のクラスについては、以下の用語が適用される。

- 英数字文字位置。USAGE DISPLAY を使用して表される DBCS 文字。
- *DBCS* 文字位置。USAGE DISPLAY-1 を使用して表される DBCS 文字。
- 国別文字位置。USAGE NATIONAL を使用して表される文字。UTF-16 の 文字エンコード・ユニットと同義。

文字セット (character set)
テキスト情報を表すために使用されるエレメントの集合。ただし、コード化表現は想定されていない。コード化文字セット *(coded character set)* も参照。

文字ストリング (character string)
COBOL ワード、リテラル、PICTURE 文字ストリング、またはコメント記入項目を形成する一連の連続した文字。 文字ストリングは区切り文字で区切らなければならない。

チェックポイント (checkpoint)
ジョブ・ステップを後で再始動することができるように、 ジョブとシステムの状況に関する情報を記録しておくことができるポイント。

クラス (* class)
ゼロ、1 つ、または複数のオブジェクトの共通の動作およびインプリメンテーシ ョンを定義するエンティティー。 同じ具体化を共用するオブジェクトは、同じクラスのオブジェクトとみなされる。 クラスは階層として定義でき、あるクラスを別のクラスから継承することができる。

クラス条件 (* class condition)
項目の内容がすべて英字であるか、すべて数字であるか、 すべて DBCS であるか、すべて漢字であるか、あるいはクラス名の定義においてリストされた文字だけで構成されるかという命題で、それに関して真の値を判別することができる。

クラス名 (データの) (* class-name (of data))
ENVIRONMENT DIVISION の SPECIAL-NAMES 段落で定義されるユーザー定義語であり、真理値を定義できる命題に名前を割り当てる。データ項目の内容は、クラス名の定義にリストされている文字だけで構成される。

節 (* clause)
記入項目の属性を指定するという目的で順番に並べられた連続する COBOL 文字ストリング。

COBOL 文字セット (COBOL character set)
COBOL 構文を作成する際に使用される文字セット。 完全な COBOL 文字セットは、以下の文字で構成される。

文字	意味
0,1, . . . ,9	数字
A,B, . . . ,Z	英大文字
a,b, . . . ,z	英小文字
	スペース
+	正符号
-	負符号 (-) (ハイフン)
*	アスタリスク

文字	意味
/	斜線 (スラッシュ)
=	等号
$	通貨記号
,	コンマ
;	セミコロン
.	ピリオド (小数点、終止符)
"	引用符
'	アポストロフィ
(	左括弧
)	右括弧
>	より大きい
<	より小さい
:	コロン
_	下線

COBOL ワード (* COBOL word)
ワード *(word)* を参照。

コード・ページ (code page)
すべてのコード・ポイントに図形文字および制御機能の意味を割り当てるもの。 例えば、あるコード・ページでは、8 ビット・コードに対して 256 コード・ポイントに文字と意味を割り当て、別のコード・ページでは、7 ビット・コードに対して 128 コード・ポイントに文字と意味を割り当てることができる。 ワークステーション上の英語の IBM コード・ページは IBM-1252 で、ホストは IBM-1047 である。

コード・ポイント (code point)
コード化文字セット (コード・ページ) に定義する固有のビット・パターン。 コード・ポイントには、グラフィック・シンボルおよび制御文字が割り当てられる。

コード化文字セット (coded character set)
文字セットを設定し、その文字セットの文字とコード化表現との間の関係を設定する明確な規則の集まり。 コード化文字セットの例として、ASCII もしくは EBCDIC コード・ページで、または Unicode 対応の UTF-16 エンコード・スキームで表す文字セットがある。

コード化文字セット ID (coded character set identifier (CCSID))
特定のコード・ページを識別する 1 から 65,535 までの IBM 定義番号。

照合シーケンス (* collating sequence)
コンピューターに受け入れられる文字がソート、マージ、比較を行うため、また索引付きファイルを順次処理するために順序付けられているシーケンス。

列 (* column)
印刷行または参照形式行におけるバイト位置。 列は、行の左端の位置から始めて行の右端の位置まで、1 から 1 ずつ増やして番号が付けられる。 列は 1 つの 1 バイト文字を保持する。

複合条件 (* combined condition)
2 つ以上の条件を AND または OR 論理演算子で結合した結果生じる条件。 条件 *(condition)* および 複合否定条件 *(negated combined condition)* も参照。

コメント記入項目 (* comment-entry)
ドキュメンテーションに使用される IDENTIFICATION DIVISION 内の項目で、実行に影響しない。

コメント行 (comment line)
行の標識区域のアスタリスク (*) と、または、プログラム・テキスト区域 (区域 A および区域 B) の最初の文字ストリングとしてのアスタリスクと大なり記号 (*>) と、 その行の区域 A および区域 B に続くコ

ンピューターの文字セットの任意の文字によって表されるソース・プログラム行。 コメント行は、文書化にのみ役立つ。 行の標識区域では斜線 (/)、そしてその行の区域 A および B ではコンピューター文字セットの任意の文字で表される特殊形式のコメント行があると、コメントの印刷前に改ページが行われる。

共通プログラム (* common program)
別のプログラムに直接的に含まれているにもかかわらず、その別のプログラムに直接的または間接的に含まれている任意のプログラムから呼び出すことができるプログラム。

互換性のある日付フィールド (compatible date field)
互換 という用語の意味は、日付フィールドに適用される場合、それが COBOL のどの部で使用されるかによって異なる。

- DATA DIVISION: 2 つの日付フィールドが同一の USAGE を持ち、以下の条件の少なくとも 1 つを満たしている場合、それらの日付フィールドは互換性があります。
  - 同じ日付形式を持つ。
  - ともにウィンドウ表示日付フィールドであり、一方がウィンドウ表示西暦年 DATE FORMAT YY だけで構成される。
  - ともに拡張日付フィールドであり、一方が拡張西暦年 DATE FORMAT YYYY だけで構成される。
  - 一方が DATE FORMAT YYXXXX で、他方が YYXX の形式である。
  - 一方が DATE FORMAT YYYYXXXX で、他方が YYYYXX の形式である。

  ウィンドウ表示日付フィールドは、拡張日付グループであるデータ項目に従属することができる。 2 つの日付フィールドに互換性があると言われるのは、従属日付フィールドが USAGE DISPLAY を持ち、グループ拡張日付フィールドの開始より 2 バイト後で始まっており、2 つのフィールドが以下の少なくとも 1 つの条件を満たしている場合である。
  - 従属日付フィールドの DATE FORMAT パターンが、グループ日付フィールドの DATE FORMAT パターンと同じ数の X を持つ。
  - 従属日付フィールドが DATE FORMAT YY を持つ。
  - グループ日付フィールドが DATE FORMAT YYYYXXXX を持ち、従属日付フィールドが DATE FORMAT YYXX を持つ。
- PROCEDURE DIVISION: 2 つの日付フィールドが、ウィンドウ表示または拡張できる年部分を除いて、同じ日付形式を持っている場合、それらのフィールドは互換性があります。 例えば、DATE FORMAT YYXXX という形式のウィンドウ表示日付フィールドは、以下のものと互換性がある。
  - DATE FORMAT YYXXX という形式の別のウィンドウ表示日付フィールド。
  - DATE FORMAT YYYYXXX という形式の拡張日付フィールド。

コンパイル (* compile)
(1) 高水準言語で表現されたプログラムを、中間言語、アセンブリー言語、または コンピューター言語で表現されたプログラムに変換すること。 (2) あるプログラミング言語で書かれたコンピューター・プログラムから、プログラムの全体的なロジック構造を利用することによって、または 1 つの記号ステートメントから複数のコンピューター命令を作り出すことによって、またはアセンブラーの機能のようにこれら両方を使用することによって、マシン言語プログラムを生成すること。

コンパイル変数 (compilation variable)
ある特定のリテラル値のシンボル名、あるいは DEFINE ディレクティブまたは DEFINE コンパイラー・オプションによって指定されたコンパイル時演算式のシンボル名。

コンパイル時間 (* compile time)
COBOL コンパイラーによって、COBOL ソース・コードが COBOL オブジェクト・プログラムに変換される時間。

コンパイル時演算式 (compile-time arithmetic expression)
DEFINE ディレクティブおよび EVALUATE ディレクティブに、または定数条件式に指定されている算術式のサブセット。 コンパイル時演算式における、正規演算式との違い:

- 指数演算子を指定することはできません。

- オペランドはすべて、整数リテラルか、すべてのオペランドが整数リテラルである演算式でなければなりません。
- 式はゼロによる除算が発生しないように指定する必要があります。

コンパイラー (compiler)
高水準言語で記述されたソース・コードをマシン言語のオブジェクト・コードに変換するプログラム。

コンパイラー指示ステートメント (compiler-directing statement)
コンパイル時にコンパイラーに特定の処置を行わせるステートメント。 標準コンパイラー指示ステートメントには、 COPY、REPLACE、および USE がある。

コンパイラー指示 (compiler directive)
コンパイル時にコンパイラーに特定の処置を行わせる指示。 COBOL for Linux は、CALLINTERFACE コンパイラー指示、および条件付きコンパイルのコンパイラー指示 (DEFINE、EVALUATE、および IF) をサポートします。

複合条件 (* complex condition)
1 つ以上の論理演算子が 1 つ以上の条件に基づいて作動する条件。 条件 *(condition)*、単純否定条件 *(negated simple condition)*、および 複合否定条件 *(negated combined condition)* も参照。

複合 ODO (complex ODO)
次のような OCCURS DEPENDING ON 節の特定の形式。

- 可変位置項目またはグループ: DEPENDING ON オプションを指定した OCCURS 節によって記述されたデータ項目の後に、非従属データ項目またはグループが続く。 グループは英数字グループでも国別グループでも構いません。
- 可変位置テーブル: DEPENDING ON オプションを指定した OCCURS 節によって記述されたデータ項目の後に、OCCURS 節に よって記述された非従属データ項目が続く。
- 可変長エレメントを持つテーブル: OCCURS 節によって記述されたデータ項目に、DEPENDING ON オプションを指定した OCCURS 節によって記述された従属データ項目が含まれている。
- 可変長エレメントを持つテーブルの指標名。
- 可変長エレメントを持つテーブルのエレメント。

コンポーネント (component)
(1) 関連ファイルからなる機能グループ化。 (2) オブジェクト指向プログラミングでは、特定の機能を実行し、他のコンポーネントやアプリケーションと連携するように設計されている、再使用可能なオブジェクトまたはプログラム。 JavaBeans は、コンポーネントを作成するための、Oracle が提供するアーキテクチャーである。

コンピューター名 (* computer-name)
プログラムがコンパイルまたは実行されるコンピューターを識別するシステム名。

条件 (例外) (condition (exception))
アプリケーションの通常のプログラミングされたフローを変えるもの。 条件は、ハードウェアまたはオペレーティング・システムによって検出され、その結果、割り込みが起こる。 このほかにも、条件は言語特定の生成コードまたは言語ライブラリー・コードによっても検出できる。

条件 (式) (condition (expression))
真理値を判別できる、実行時のデータの状況。 条件という用語が本書で 一般形式の「条件」(条件-*1*、条件-*2*、...) の中、またはこれに関連して使用された場合は、 . .) 次のいずれかである。オプションとして括弧で囲まれた単純条件からなる条件式、あるいは、単純条件、論理演算子、および括弧の構文的に正しい組み合わせ (真理値を判別できる) からなる複合条件。 単純条件 *(simple condition)*、複合条件 *(complex condition)*、単純否定条件 *(negated simple condition)*、複合条件 *(combined condition)*、および 複合否定条件 *(negated combined condition)* も参照。

条件式 (* conditional expression)
EVALUATE、IF、PERFORM、または SEARCH ステートメントの中で指定される単純条件または複合条件。 単純条件 *(simple condition)* および 複合条件 *(complex condition)* も参照。

条件句 (* conditional phrase)
ある条件ステートメントが実行された結果得られる条件の真理値の判別に基づいてとられるべき処置を指定する句。

条件ステートメント (* conditional statement)
条件の真理値を判別することと、オブジェクト・プログラムの次の処理がこの真理値によって決まることを指定するステートメント。

条件変数 (* conditional variable)
1 つ以上の値を持つデータ項目であり、これらの値が、 そのデータ項目に割り当てられた条件名を持つ。

条件名 (* condition-name)
条件変数が想定できる値のサブセットに名前を割り当てるユーザー定義語。または、インプリメントする人が定義したスイッチまたは装置の状況に割り当てられるユーザー定義語。

条件名条件 (* condition-name condition)
真理値を判別できる命題で、かつ、条件変数の値が、その条件変数と関連する条件名に属する一連の値のメンバーである命題。

*** CONFIGURATION SECTION**
ENVIRONMENT DIVISION のセクションであり、ソース・プログラムとオブジェクト・プログラムの全体的な仕様を記述する。

CONSOLE
オペレーター・コンソールと関連する COBOL 環境名。

定数条件式 (constant conditional expression)
IF ディレクティブで、または EVALUATE ディレクティブの WHEN 句で使用される可能性がある条件式のサブセット。

定数条件式は、以下の項目のいずれかでなければなりません。

- 両方のオペランドがリテラルであるか、リテラル項のみを含む算術式である比較条件。条件は比較条件の規則に従う必要があり、以下の追加事項があります。
  - オペランドは同じカテゴリーでなければなりません。算術式は数値カテゴリーです。
  - リテラルが指定され、それらが数値リテラルではない場合、関係演算子は “IS EQUAL TO”、“IS NOT EQUAL TO”、“IS =”、“IS NOT =”、または “IS <>” でなければなりません。

  比較条件 *(relation condition)* も参照。
- 定義済み条件。定義済み条件 *(defined condition)* も参照。
- ブール条件。 ブール条件 *(boolean condition)* も参照。
- 前述の単純条件の形式を、AND、OR、および NOT を使用して複合条件に結合することによって形成した複合条件。 簡略複合比較条件を指定することはできません。複合条件 *(complex condition)* も参照。

含まれているプログラム (contained program)
別の COBOL プログラムにネストされている COBOL プログラム。

連続項目 (* contiguous items)
DATA DIVISION 内の連続する記入項目によって記述され、相互に一定の階層関係を持っている項目。

コピーブック (copybook)
一連のコードが含まれたファイルまたはライブラリー・メンバーであり、コンパイル時に COPY ステートメントを使用してソース・プログラムに組み込まれる。 ファイルはユーザーが作成する場合、COBOL によって提供される場合、 または他の製品によって供給される場合とがある。 「コピー・ファイル *(copy file)*」と同義。

カウンター (* counter)
他の数字を使ってその数字分だけ増減したり、 あるいは 0 または任意の正もしくは負の値に変更またはリセットしたりできるようにした、 数または数表現を収めるために使用されるデータ項目。

相互参照リスト (cross-reference listing)
コンパイラー・リストの一部であり、プログラム内においてファイル、フィールド、および標識が定義、参照、および変更される場所に関する情報が入る。

通貨記号値 (currency-sign value)
数字編集項目に保管される通貨単位を識別する文字ストリング。 典型的な例としては、$、USD、EUR などがある。 通貨記号値は、CURRENCY コンパイラー・オプションで定義するか、ENVIRONMENT DIVISION の SPECIAL-NAMES 段落内の CURRENCY SIGN 節によって定義することができる。 CURRENCY SIGN 節が指定されない場合、NOCURRENCY コンパイラー・オプションが有効であれば、ドル記号 ($) がデフォルトの通貨記号値として使用される。 通貨記号 *(currency symbol)* も参照。

通貨記号 (currency symbol)
数字編集項目内の通貨記号値の部分を示すために、PICTURE 節で使用される文字。 通貨記号は、CURRENCY コンパイラー・オプションで定義するか、ENVIRONMENT DIVISION の SPECIAL-NAMES 段落内の CURRENCY SIGN 節によって定義することができる。 CURRENCY SIGN 節が指定されない場合、NOCURRENCY コンパイラー・オプションが有効であれば、ドル記号 ($) がデフォルトの通貨記号値および通貨記号として使用される。 通貨記号と通貨符号値は複数定義可能。 通貨記号値 *(currency sign value)* も参照。

現行レコード (* current record)
ファイル処理では、ファイルに関連したレコード域に使用できるレコード。

現行ボリューム・ポインター (* current volume pointer)
順次ファイルの現行のボリュームを指している概念上のエンティティー。

D

データ節 (* data clause)
COBOL プログラムの DATA DIVISION のデータ記述記入項目に現れる節で、データ項目の特定の属性を記述する情報を提供する。

データ記述記入項目 (* data description entry)
COBOL プログラムの DATA DIVISION 内の記入項目であり、レベル番号の後に必要に応じてデータ名が続き、その後に必要に応じて一連のデータ節で構成されるもの。

DATA DIVISION
COBOL プログラムの 1 つの部 (division)。 使用するファイルとファイルに含まれるレコード、必要となる内部 WORKING-STORAGE レコード、COBOL 実行単位内の複数のプログラムで使用可能なデータなど、プログラムで処理するデータを記述する。

データ項目 (* data item)
COBOL プログラムによってまたは関数演算の規則によって定義されるデータ単位 (リテラルを除く)。

データ名 (* data-name)
データ記述記入項目で記述されたデータ項目に名前を割り当てるユーザー定義語。 一般形式で使用された場合、データ名は、その形式の規則で特に許可されていない限り、参照変更、添え字付け、または修飾してはならないワードを表す。

日付フィールド (date field)
次の項目のいずれか。

- データ記述記入項目に DATE FORMAT 節が含まれているデータ項目。
- 次の組み込み関数の 1 つで戻される値。

```
DATE-OF-INTEGER
DATE-TO-YYYYMMDD
DATEVAL
DAY-OF-INTEGER
DAY-TO-YYYYDDD
YEAR-TO-YYYY
YEARWINDOW
```

- ACCEPT ステートメントの概念上のデータ項目 DATE、DATE YYYYMMDD、DAY、および DAY YYYYDDD。
- ある種の算術演算の結果。 詳細については、『日付フィールドを使用する算術計算』(*COBOL for Linux on x86* 言語解説書) を参照してください。

「日付フィールド *(date field)*」という用語は、「拡張日付フィールド *(expanded date field)*」と「ウィンドウ表示日付フィールド *(windowed date field)*」の両方を指す。 非日付 *(nondate)* も参照。

日付形式 (date format)
次のいずれかの方法で指定される 日付フィールドの日付パターン。

- DATE FORMAT 節または DATEVAL 組み込み関数 argument-2 によって明示的に。
- 日付フィールドを戻すステートメントまたは組み込み関数によって暗黙的に。 詳細については、日付フィールド (*COBOL for Linux on x86* 言語解説書) を参照してください。

Db2 ファイル・システム (Db2 file system)
Db2 ファイル・システムは、順次ファイル、索引付きファイル、および相対ファイルをサポートします。Db2 ファイル・システムは CICS との拡張相互協調処理を提供しており、Db2 に保管されている CICS ESDS、KSDS、および RRDS ファイルにバッチ COBOL プログラムがアクセスできるようにします。

DBCS
2 バイト文字セット *(double-byte character set (DBCS))* を参照

DBCS 文字 (DBCS character)
IBM 2 バイト文字セット (DBCS) に定義された任意の文字。

DBCS 文字位置 (DBCS character position)
文字位置 *(character position)* を参照。

DBCS データ項目 (DBCS data item)
少なくとも 1 つの記号 G または少なくとも 1 つの記号 N (NSYMBOL(DBCS) コンパイラー・オプションが有効なとき) を含んでいる PICTURE 文字ストリングで記述されたデータ項目。DBCS データ項目は USAGE DISPLAY-1 を持っています。

デバッグ行 (* debugging line)
行の標識区域に文字 D がある行のこと。

デバッグ・セクション (* debugging section)
USE FOR DEBUGGING ステートメントが含まれているセクション。

宣言文 (* declarative sentence)
区切り記号のピリオドによって終了する 1 つの USE ステートメントから構成されるコンパイラー指示文。

宣言部分 (* declaratives)
PROCEDURE DIVISION の先頭に書き込まれた 1 つ以上の特殊目的セクションの集合であり、その先頭にはキーワード DECLARATIVE が付き、その最後にはキーワード END DECLARATIVES が続いている。宣言部分は、セクション・ヘッダー、USE コンパイラー指示文、および 0 個、1 個、または複数個の関連する段落で構成される。

編集解除 (* de-edit)
項目の編集解除された数値を判別するために、数字編集データ項目からすべての編集文字を論理的に除去すること。

定義済み条件 (defined condition)
コンパイル変数が定義されているかどうかをテストするコンパイル時条件。 定義済み条件は、IF ディレクティブで、または EVALUATE ディレクティブの WHEN 句で指定される。

範囲区切りステートメント (* delimited scope statement)
明示的範囲終了符号を含んでいるステートメント。

区切り文字 (* delimiter)
1 つの文字、または一連の連続する文字であり、文字ストリングの終わりを識別し、その文字ストリングを後続の文字ストリングから区切る。 区切り文字は、これを使用して区切られる文字ストリングの一部ではない。

降順キー (* descending key)
値に基づくキーであり、そのデータが、キーの最高値からキーの最低値まで、データ項目比較規則に従って順序付けられている。

数字 (digit)
0 から 9 までの任意の数字。COBOL では、この用語を用いて他の記号を参照することはない。

桁位置 (* digit position)
1 つの桁を保管するために必要な物理ストレージの大きさ。 この大きさは、データ項目を定義するデータ記述記入項目に 指定された用途によって異なる。

直接アクセス (* direct access)
プロセスが、以前にアクセスされたデータへの参照ではなく、そのデータの位置にのみ依存する方法で、ストレージ・デバイスからデータを入手したり、ストレージ・デバイスにデータを入力したりする機能。

表示浮動小数点データ項目 (display floating-point data item)
暗黙的または明示的に USAGE DISPLAY として記述されており、外部浮動小数点データ項目を記述する PICTURE 文字ストリングを持っている、データ項目。

部 (* division)
部の本体と呼ばれる、0 個、1 個、または複数個のセクションまたは段落の集合であり、特定の規則に従って形成および結合されたもの。 それぞれの部は、部のヘッダーおよび関連した部の本体で構成される。 COBOL プログラムには、見出し部、環境部、データ部、および手続き部の 4 つの部がある。

部の見出し (* division header)
ワードとその後に続く、部の先頭を示す分離文字ピリオドの組み合わせ。 部のヘッダーは次のとおり。

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.
```

Do 構造 (do construct)
構造化プログラミングでは、DO ステートメントを使えば、プロシージャー内の複数のステートメントをグループ化できる。 COBOL では、インライン PERFORM ステートメントが同様に機能する。

do-until
構造化プログラミングにおいて、do-until ループは、少なくとも 1 回は実行され、所定の条件が真になるまで実行される。 COBOL では、TEST AFTER 句を PERFORM ステートメントで使用すれば、同様に機能する。

do-while
構造化プログラミングにおいて、do-while ループは、所定の条件が真である場合、およ び真である間に実行される。 COBOL では、TEST BEFORE 句を PERFORM ステートメントで使用すれば、同様に機能する。

文書タイプ宣言 (document type declaration)
あるクラスの文書に対する文法を規定するマークアップ宣言を含む、または 指示する XML エレメント。この文法は、文書タイプ定義または DTD とも呼ばれる。

文書タイプ定義 (document type definition (DTD))
XML 文書のクラスの文法。 文書タイプ宣言を参照。

2 バイト ASCII (double-byte ASCII)
DBCS 文字および 1 バイト ASCII 文字を含む IBM の文字セット (「ASCII DBCS」とも呼ばれる)。

2 バイト EBCDIC (double-byte EBCDIC)
DBCS 文字および 1 バイト EBCDIC 文字を含む IBM の文字セット (「EBCDIC DBCS」とも呼ばれる)。

2 バイト文字セット (double-byte character set (DBCS))
それぞれの文字が 2 バイトで表現される 1 組の文字。 256 個のコード・ポイントで表現される記号より多くの記号を含んでいる 言語 (日本語、中国語、および韓国語など) は、2 バイト文字セットを必要とする。 各文字に 2 バイトが必要なため、DBCS 文字の入力、表示、および印刷には、DBCS を受け入れ可能なハードウェアおよびサポートされるソフトウェアが必要。

DWARF
DWARF は UNIX International Programming Languages Special Interest Group (SIG) で開発された。言語に依存しないデバッグ情報を提供することにより、さまざまな言語の、統一された方法でのシンボリックなソース・レベル・デバッグのニーズを満たすように設計されている。 DWARF ファイルには、

さまざまなエレメントに編成されたデバッグ・データが含まれている。 詳しくは、「*DWARF/ELF エクステンション ライブラリー・リファレンス*」の『*DWARF program information*』を参照。

動的アクセス (* dynamic access)
1 つの OPEN ステートメントの実行範囲内において、特定の論理レコードを、大容量記憶ファイルからは順次アクセス以外の方法で取り出したりそのファイルに入れたりでき、またファイルからは順次アクセスの方法で取り出せるアクセス・モード。

動的 CALL (dynamic CALL)
DYNAM オプションを使用してコンパイルされたプログラム内の CALL *literal* ステートメント、またはプログラム内の CALL *identifier* ステートメント。

E

EBCDIC (拡張 2 進化 10 進コード) (* EBCDIC (Extended Binary-Coded Decimal Interchange Code))
8 ビット・コード化文字をベースとするコード化文字セット。

EBCDIC 文字 (EBCDIC character)
EBCDIC (拡張 2 進化 10 進コード) セットに含まれているいずれかの記号。

EBCDIC DBCS
「*2 バイト EBCDIC (double-byte EBCDIC)*」を参照。

編集済みデータ項目 (edited data item)
0 の抑止または編集文字の挿入、あるいはその両方を行うことによって変更されたデータ項目。

編集用文字 (* editing character)
次に示す集合に属する 1 文字、または 2 文字で構成される固定した組み合わせ。

文字	意味
	スペース
0	ゼロ
+	正符号
-	負符号
CR	貸方
DB	借方
Z	ゼロの抑止
*	チェック・プロテクト
$	通貨記号
,	コンマ (小数点)
.	ピリオド (小数点)
/	斜線 (スラッシュ)

エレメント (テキスト・エレメント) (element (text element))
1 つのデータ項目または動詞の記述などのようなテキスト・ストリングの 1 つの論理単位で、その前にエレメント・タイプを識別する固有のコードが付けられたもの。

基本項目 (* elementary item)
それ以上論理的に分割されないものとして記述されるデータ項目。

CICS SFS ファイル・システム (CICS SFS file system)
SFS ファイル・システム (SFS file system) を参照。

エンコード・ユニット (encoding unit)
文字エンコード・ユニット *(character encoding unit)* を参照。

PROCEDURE DIVISION の終わり (* end of PROCEDURE DIVISION)
COBOL ソース・プログラムにおいて、それ以後にはプロシージャーが存在しない物理的な位置。

プログラム終了マーカー (* end program marker)
語の組み合わせに分離文字ピリオドが続いたもので、COBOL ソース・プログラムの終わりを示す。 プログラム終了マーカーは、次のように記述する。

```
END PROGRAM program-name.
```

項目 (* entry)
分離文字ピリオドで終了させられる連続する節の記述セットであり、COBOL プログラムの IDENTIFICATION DIVISION、ENVIRONMENT DIVISION、または DATA DIVISION に書き込まれる。

環境節 (* environment clause)
ENVIRONMENT DIVISION 記入項目の一部として現れる節。

ENVIRONMENT DIVISION
COBOL プログラムの 4 つの主コンポーネントの 1 つ。 ENVIRONMENT DIVISION では、ソース・プログラムがコンパイルされるコンピューターと、オブジェクト・プログラムが実行されるコンピューターを記述する。 この部では、ファイルの論理概念とそのレコードの間のリンケージ、およびファイルが保管される装置の物理的局面を提供する。

環境名 (environment-name)
IBM が指定する名前であり、システム論理装置、プリンターおよびカード穿孔装置の制御文字、報告書コード、またはプログラム・スイッチ、あるいはそれらの組み合わせを識別する。環境名が ENVIRONMENT DIVISION の簡略名と関連付けられている場合は、その簡略名を、置換が有効な任意の形式で置き換えることができる。

環境変数 (environment variable)
コンピューター環境の一部の局面を定義する多数の変数のいずれかであり、その環境で動作するプログラムからアクセス可能。 環境変数は、動作環境に依存するプログラムの動作に影響を与える。

実行時 (execution time)
実行時 *(run time)* を参照。

実行時環境 (execution-time environment)
ランタイム環境 *(runtime environment)* を参照。

拡張日付フィールド (expanded date field)
拡張 (4 桁) 年を含む日付フィールド。 日付フィールド *(date field)* および 拡張西暦年 *(expanded year)* も参照。

拡張西暦年 (expanded year)
4 桁の年だけから構成される日付フィールド。 その値には世紀が含まれる (例えば、1998)。 ウィンドウ表示西暦年 *(windowed year)* と比較。

明示的範囲終了符号 (* explicit scope terminator)
特定の PROCEDURE DIVISION ステートメントの有効範囲を終わらせる予約語。

指数 (exponent)
別の数 (底) をべき乗する指数を示す数。 正の指数は乗算を示し、負の指数は除算を示し、小数の指数は数量の根を示す。 COBOL では、指数式は記号 ** の後に指数を付けて表す。

式 (* expression)
算術式または条件式。

拡張モード (* extend mode)
ファイルに対する EXTEND 句の指定のある OPEN ステートメントが実行されてから、そのファイルに対する REEL または UNIT 句の指定のない CLOSE ステートメントが実行される前までの、ファイルの状態。

Extensible Markup Language
XML を参照。

拡張 (extensions)
85 COBOL 標準 に記述されているものの他に、IBM コンパイラーでサポートされる COBOL 構文およびセマンティクス。

外部コード・ページ (external code page)
ASCII または UTF-8 XML 文書の場合は、現在のランタイム・ロケールが指示するコード・ページ。EBCDIC XML 文書の場合は、次のいずれか。

- EBCDIC_CODEPAGE 環境変数で指定されたコード・ページ
- EBCDIC_CODEPAGE 環境変数が設定されていない場合に現在のランタイム・ロケールとして選択されたデフォルトの EBCDIC コード・ページ。

外部データ (* external data)
プログラムの中で外部データ項目および外部ファイル結合子として記述されるデータ。

外部データ項目 (* external data item)
実行単位の 1 つ以上のプログラムにおいて外部レコードの一部として記述されるデータ項目であり、その項目が記述されている任意のプログラムから参照することができる。

外部データ・レコード (* external data record)
実行単位の 1 つ以上のプログラムにおいて記述される論理レコードであり、そのデータ項目は、それらが記述されている任意のプログラムから参照できる。

外部 10 進数データ項目 (external decimal data item)
ゾーン *10* 進数データ項目 *(zoned decimal data item)* および 国別 *10* 進数データ項目 *(national decimal data item)* を参照。

外部ファイル結合子 (* external file connector)
実行単位の 1 つ以上のオブジェクト・プログラムにアクセス可能なファイル結合子。

外部浮動小数点データ項目 (external floating-point data item)
表示浮動小数点データ項目 *(display floating-point data item)* および 国別浮動小数点データ項目 *(national floating-point data item)* を参照。

外部プログラム (external program)
最外部プログラム。 ネストされていないプログラム。

外部スイッチ (* external switch)
インプリメントする人によって定義され指名されたハードウェアまたはソフトウェア装置であり、2 つの代替状態のいずれかが存在していることを示す。

F

形象定数 (* figurative constant)
ある予約語を使用して参照されるコンパイラー生成の値。

ファイル (* file)
論理レコードの集合。

ファイル属性対立条件 (* file attribute conflict condition)
ファイルに入出力操作の実行を試みて失敗した場合に、プログラムの中でそのファイルに対して指定されたファイル属性が、そのファイルの固定属性と一致しないこと。

ファイル節 (* file clause)
DATA DIVISION の記入項目であるファイル記述項目 (FD 記入項目) およびソート・マージ・ファイル記述項目 (SD 記入項目) のいずれかの一部として現れる節。

ファイル結合子 (* file connector)
ファイルに関する情報が入っており、ファイル名と物理ファイルの間のリンケージとして、さらにファイル名とその関連レコード域の間のリンケージとして使用されるストレージ域。

ファイル制御記入項目 (* file control entry)
SELECT 節と、ファイルの関連物理属性を宣言するすべての従属節。

FILE-CONTROL 段落 (FILE-CONTROL paragraph)
ENVIRONMENT DIVISION 内の段落であり、この中では、特定のソース単位で使用されるデータ・ファイルが宣言される。

ファイル記述記入項目 (* file description entry)
DATA DIVISION の FILE SECTION の中にある記入項目。レベル標識 FD と、それに続くファイル名、および、必要に応じて、次に続く一連のファイル節から構成される。

ファイル名 (* file-name)
DATA DIVISION の FILE SECTION の中のファイル記述項目またはソート・マージ・ファイル記述項目で記述されるファイル結合子に名前を付けるユーザー定義語。

ファイル編成 (* file organization)
ファイルの作成時に確立される永続論理ファイル構造。

ファイル位置標識 (file position indicator)
概念的エンティティーであり、索引付きファイルの場合は参照キー内の現行キーの値、順次ファイルの場合は現行レコードのレコード番号、相対ファイルの場合は現行レコードの相対レコード番号が入っている。あるいは、次の論理レコードが存在しないことを示すか、オプションの入力ファイルが使用可能でないことを示すか、AT END 条件が既に存在していることを示すか、もしくは有効な次のレコードが設定されていないことを示す。

*** FILE SECTION**
DATA DIVISION のセクションであり、ファイル記述項目、ソート・マージ・ファイル記述項目、および関連するレコード記述が入っている。

ファイル・システム (file system)
データ・レコードおよびファイル記述プロトコルの特定のセットに準拠するファイルの集合、およびこれらのファイルを管理する一連のプログラム。

固定ファイル属性 (* fixed file attributes)
ファイルに関する情報であり、ファイルの作成時に設定され、それ以降はファイルが存在する限り変更できない。 これらの属性には、ファイル (順次、相対、または索引付き) の編成、基本レコード・キー、代替レコード・キー、コード・セット、最小および最大レコード・サイズ、レコード・タイプ (固定または可変)、索引付きファイルのキーの照合シーケンス、ブロック化因数、埋め込み文字、およびレコード区切り文字がある。

固定長レコード (* fixed-length record)
ファイル記述項目またはソート・マージ記述記入項目が、 すべてのレコードのバイトの個数が同じであるように要求しているファイルに関連付けられたレコード。

固定小数点項目 (fixed-point item)
PICTURE 節で定義される数値データ項目であり、オプションの符号の位置、その中に含まれる桁数、およびオプションの小数点の位置を指定するもの。 2 進数、パック 10 進数、または外部 10 進数のいずれかのフォーマットをとることができる。

浮動コメント標識 (*>) (floating comment indicators (*>))
浮動コメント標識がプログラムのテキスト域 (領域 A プラス領域 B) 内の最初の文字ストリングである場合は、この行がコメント行であることを示します。 また、浮動コメント標識がプログラムのテキスト領域内の 1 つ以上の文字ストリングの後にある場合は、インライン・コメントを示します。

浮動小数点 (floating point)
実数を 1 対の数表示で表す、数を表記するための形式。 浮動小数点表記では、固定小数点部分 (最初の数表示) と、暗黙浮動小数点の底を指数で表される数だけ累乗して得られる値 (2 番目の数表示) との積が、実数になります。 例えば、数値 0.0001234 の浮動小数点表記は 0.1234 -3 です (ここで、0.1234 は 小数部であり、-3 は指数です)。

浮動小数点データ項目 (floating-point data item)
小数部と指数が入っている数値データ項目。 その値は、小数部に、指数で指定されただけ累乗された数字データ項目の底を乗算することによって得られる。

フォーマット (* format)
一連のデータの特定の配置。

機能 (* function)
ステートメントの実行中に参照された時点で決定される値を持つ、 一時的なデータ項目。

関数 ID (* function-identifier)
関数を参照する文字ストリングと区切り文字の構文的に正しい組み合わせ。 関数で表現されるデータ項目は、関数名と引数 (ある場合) によって一意的に識 別される。 関数 ID は、参照修飾子を含むことができる。 英数字関数を参照する関数 ID は、 一定の制限に従いつつ ID が指定できる一般フォーマットの中ならばどこにでも指定できる。 整数関数または数字関数を参照する関数 ID は、算術式が指定できる一般フォーマットの中ならばどこにおいても指定できる。

関数名 (function-name)
必要な引数を指定した呼び出しによって、関数の値が決定されるメカニズムを指名するワード。

関数ポインター・データ項目 (function-pointer data item)
入り口点を指すポインターを保管できるデータ項目。 USAGE IS FUNCTION-POINTER 節で定義されるデータ項目に、関数入り口点のアドレスが含められる。 一般的に、C および Java プログラムと通信するために使用される。

G

ガーベッジ・コレクション (garbage collection)
参照されなくなったオブジェクト用のメモリーが Java ランタイム・システムによって自動的に解放されること。

GDG
世代別データ・グループ *(GDG) (generation data group (GDG))* を参照。

GDS
世代別データ・セット *(GDS) (generation data set (GDS))* を参照。

世代別データ・グループ (GDG) (generation data group (GDG))
発生順に関連するファイルの集合。このような各ファイルは、世代別データ・セット *(GDS)* または世代と呼ばれる。

世代別データ・セット (GDS) (generation data set (GDS))
世代別データ・グループ *(GDG)* 内のファイルのうちの 1 つ。このような各ファイルは、グループ内の他のファイルと発生順に関連している。

グローバル名 (* global name)
1 つのプログラムにおいてのみ宣言されるが、そのプログラム、またはそのプログラム内に含まれている任意のプログラムから参照できる名前。 条件名、データ名、ファイル名、レコード名、報告書名、およびいくつかの特殊レジスターが、グローバル名となり得る。

グループ項目 (group item)
(1) 複数の従属データ項目で構成されるデータ項目。 英数字グループ項目 *(alphanumeric group item)* および 国別グループ項目 *(national group item)* を参照。 (2) 国別グループまたは英数字グループとして明示的に (またはコンテキスト で) 限定されていない場合、この用語は一般のグループを指します。

グループ区切り文字 (grouping separator)
読みやすさのために数値を何桁かまとめて区切るのに使用される文字。 デフォルトはコンマである。

H

ヘッダー・ラベル (header label)
(1) 記録メディア・ユニットのデータ・レコードの前にある、ラベル。 (2)「ファイル開始ラベル *(beginning-of-file label)*」の同義語。

高位終了 (* high-order end)
文字ストリングの左端の文字。

ホスト英数字データ項目 (host alphanumeric data item)
(XML 文書の場合) NATIVE 句が含まれないデータ記述記入項目を持つ カテゴリー英数字データ項目のうち、CHAR(EBCDIC) オプションを有効にしてコンパイルされたもの。 データ項目のエンコードは、有効な EBCDIC コード・ページである。 このコード・ページは、EBCDIC_CODEPAGE 環境変数が設定されている場合はこの環境変数から決定され、 設定されていない場合は、ランタイム・ロケールに関連付けられたデフォルトのコード・ページから決定される。

I

IBM COBOL 拡張部分 (IBM COBOL extension)
85 COBOL 標準 に記述されているものの他に、IBM コンパイラーでサポートされる COBOL 構文およびセマンティクス。

ICU
International Components for Unicode (ICU) を参照。

IDENTIFICATION DIVISION
COBOL プログラムの 4 つの主コンポーネントの 1 つ。 IDENTIFICATION DIVISION では、プログラム、クラスを識別する。 IDENTIFICATION DIVISION には、作成者名、インストール、または日付を含めることができる。

ID (* identifier)
データ項目に名前を付けるための文字ストリングと区切り文字の構文的に正しい組み合わせ。 関数ではないデータ項目を参照するときは、ID は、データ名と、修飾子、添え字、または 参照修飾子 (一意的に参照するために必要な場合) から構成される。 関数であるデータ項目を参照する際には、関数 ID が使われる。

命令ステートメント (* imperative statement)
命令の動詞で開始して、行うべき無条件の処置を指定するステートメント。または明示範囲終了符号によって区切られた条件ステートメント (範囲区切りステートメント)。 1 つの命令ステートメントは、一連の命令ステートメントから構成することができる。

暗黙の範囲終了符号 (* implicit scope terminator)
終了していないステートメントが前にある場合、その範囲を区切る分離文字ピリオド。 または、前にある句の中に含まれるステートメントがある場合、 そのステートメントの範囲の終わりをそれが現れることによって示すステートメントの句。

指標 (* index)
その内容が、テーブル内の特定エレメントの識別を表す、コンピューターのストレージ域またはレジスター。

指標データ項目 (* index data item)
索引名および関連する値をインストール先指定の形式で保管できるデータ項目。

索引付きデータ名 (indexed data-name)
データ名とそれに続く 1 つ以上の (括弧で囲まれた) 索引名で構成される ID。

索引付きファイル (* indexed file)
索引編成のファイル。

指標付き編成 (* indexed organization)
各レコードが、そのレコード内の 1 つ以上のキーの値で識別される、永続論理ファイル構造。

指標付け (indexing)
指標名を使用しての添え字付け と同義。

索引名 (* index-name)
特定のテーブルに関係付けられた指標を指名するユーザー定義語。

初期設定プログラム (* initial program)
プログラムが実行単位で呼び出されるたびに初期状態に設定されるプログラム。

初期状態 (* initial state)
実行単位で最初に呼び出されるときのプログラムの状態。

インライン (inline)
プログラムでは、ルーチン、サブルーチン、または他のプログラムに分岐することなく、順次に実行される命令。

入力ファイル (* input file)
入力モードでオープンされるファイル。

入力モード (* input mode)
ファイルに対する INPUT 句の指定のある OPEN ステートメントが実行されてから、そのファイルに対する REEL または UNIT 句の指定のない CLOSE ステートメントが実行される前までの、ファイルの状態。

入出力ファイル (* input-output file)
I-O モードでオープンされるファイル。

*** INPUT-OUTPUT SECTION**
ENVIRONMENT DIVISION のセクションであり、オブジェクト・プログラムに必要なファイルおよび外部メディアに名前を付け、実行時にデータの伝送および処理に必要な情報を提供する。

入出力ステートメント (* input-output statement)
個々のレコードに対して操作を行うことにより、またはファイルを 1 つの単位として操作することにより、ファイルの処理を行うステートメント。 入出力ステートメントには、ACCEPT (ID 句付き)、CLOSE、DELETE、DISPLAY、OPEN、READ、REWRITE、SET (TO ON または TO OFF 句付き)、START、および WRITE がある。

入力プロシージャー (* input procedure)
ソートすべき特定のレコードの解放を制御する目的で、SORT ステートメントの実行時に制御が渡されるステートメントの集合。

整数 (* integer)
(1) 小数点の右側に桁位置がない数値リテラル。 (2) DATA DIVISION に定義される数値データ項目であり、小数点の右側に桁位置を含まないもの。 (3) 関数の起こりうるすべての評価の戻り値で、小数点の右側の桁がすべてゼロであることが定義されて いる数字関数。

整数関数 (integer function)
カテゴリーが数字であり、小数点の右側の桁位置が定義に入っていない関数。

言語間通信 (ILC) (interlanguage communication (ILC))
異なるプログラム言語で書かれた複数のルーチンが通信できること。 ILC サポートにより、各種言語で書かれたコンポーネント・ルーチンからアプリケーションを簡単に構築することができる。

中間結果 (intermediate result)
連続して行われる算術演算の結果を収める中間フィールド。

内部データ (* internal data)
プログラムの中で記述されるデータで、すべての外部データ項目および外部ファイル結合子を除いたもの。 プログラムの LINKAGE SECTION で記述された項目は、内部データとして扱われる。

内部データ項目 (* internal data item)
実行単位内の 1 つのプログラムの中で記述されるデータ項目。 内部データ項目は、グローバル名を持つことができる。

内部 10 進数データ項目 (internal decimal data item)
USAGE PACKED-DECIMAL または USAGE COMP-3 として記述されており、項目を数値として定義する PICTURE 文字ストリング (記号 9、S、P、または V の有効な組み合わせ) を持っている、データ項目。 「パック *10* 進数データ項目 *(packed-decimal data item)*」と同義。

内部ファイル結合子 (* internal file connector)
実行単位内にあるただ 1 つのオブジェクト・プログラムのみがアクセスできるファイル結合子。

内部浮動小数点データ項目 (internal floating-point data item)
USAGE COMP-1 または USAGE COMP-2 として記述されているデータ項目。 COMP-1 は、単精度浮動小数点データ項目を定義します。 COMP-2 は、倍精度浮動小数点データ項目を定義します。 内部浮動小数点データ項目に関連した PICTURE 節はありません。

International Components for Unicode (ICU)
IBM が後援、サポート、および使用するオープン・ソース開発プロジェクト。ICU ライブラリーは、AIX および Linux をはじめする、さまざまなプラットフォーム上で、強力かつフル機能の Unicode サービスを提供する。

レコード内データ構造 (* intrarecord data structure)
連続したデータ記述記入項目のサブセットによって定義される、1 つの論理レコードから得られるグループ・データ項目および基本データ項目の集合全体。 これらのデータ記述記入項目には、レコード内データ構造を記述している最初のデータ記述記入項目のレベル番号より大きいレベル番号を持つすべての記入項目が含まれる。

組み込み関数 (intrinsic function)
よく使用される算術関数のような事前定義関数で、組み込み関数参照によって呼び出される。

無効キー条件 (* invalid key condition)
索引付きファイルまたは相対ファイルに関連するキーの特定値が無効であると判別された場合に生じる、実行時の条件。

*** I-O-CONTROL**
ENVIRONMENT DIVISION 段落の名前。この段落では、再実行開始点についてのオブジェクト・プログラム要件、複数データ・ファイルによる同じ区域の共用、および単一入出力装置上の複数のファイル・ストレージが指定される。

I-O-CONTROL 記入項目 (* I-O-CONTROL entry)
ENVIRONMENT DIVISION の I-O-CONTROL 段落内の記入項目であり、プログラム実行中に指定のファイルへのデータの伝送と処理を行うために必要な情報を提供する節が入っている。

入出力モード (* I-O mode)
ファイルに対する I-O 句の指定のある OPEN ステートメントが実行されてから、そのファイルに対する REEL または UNIT 句の指定のない CLOSE ステートメントが実行される前までの、ファイルの状態。

入出力状況 (* I-O status)
入出力操作の結果としての状況を示す 2 文字の値を収める概念上のエンティティー。 この値は、そのファイルについてのファイル制御記入項目で FILE STATUS 節を使用することによって、プログラムに使用可能にされる。

反復構造 (iteration structure)
ある条件が真である間、あるいはある条件が真になるまで、一連のステートメントが繰り返して実行されるプログラムの処理ロジック。

J

J2EE
Java 2 Platform, Enterprise Edition (J2EE) を参照。

Java 2 Platform, Enterprise Edition (J2EE)
エンタープライズ・アプリケーションの開発とデプロイメントのために、Oracle によって定義された環境。 J2EE プラットフォームは、多層の Web ベース・アプリケーションを開発するための機能を提供するサービス、アプリケーション・プログラミング・インターフェース (API)、およびプロトコルで構成されている。 (Oracle)

Java Native Interface (JNI)
Java 仮想マシン (JVM) 内で実行される Java コードが、他のプログラム言語で記述されたアプリケーションおよびライブラリーと連携できるようにするプログラミング・インターフェース。

Java 仮想マシン (JVM) (Java virtual machine (JVM))
コンパイル済みの Java プログラムを実行する中央演算処理装置のソフトウェア・インプリメンテーション。

JSON
JSON (JavaScript Object Notation) とは、単純なデータ交換フォーマットである。

JVM
Java 仮想マシン (JVM) (Java virtual machine (JVM)) を参照。

K

K
記憶容量に関連して使われるときは、2 の 10 乗。10 進表記では 1024。

キー (* key)
レコードの位置を識別するデータ項目、またはデータの順序付けを識別するための一連のデータ項目。

参照キー (* key of reference)
索引付きファイルの中のレコードをアクセスするために現在使用されている基本キーまたは代替キー。

キーワード (* keyword)
コンテキスト・センシティブ語または予約語。その語の表示フォーマットがソース単位で使用されるときは、その語は必須である。

キロバイト (KB) (kilobyte (KB))
1 キロバイトは 1024 バイトに相当する。

L

言語名 (* language-name)
特定のプログラミング言語を指定するシステム名。

最後に使われた状態 (last-used state)
内部値がプログラム終了時と同じままで、初期値にリセットされない、プログラムの状態を言う。

文字 (* letter)
以下の 2 つのセットのいずれかに属する文字。

1. 英大文字: A、B、C、D、E、F、G、H、I、J、 K、L、M、N、O、P、Q、R、S、T、 U、V、W、X、Y、Z
2. 英小文字: a、b、c、d、e、f、g、h、i、j、 k、l、m、n、o、p、q、r、s、t、u、 v、w、x、y、z

レベル標識 (* level indicator)
特定のタイプのファイルを識別するか、または階層での位置を識別する 2 つの英字。 DATA DIVISION 内のレベル標識には、CD、FD、および SD がある。

レベル番号 (* level-number)
階層構造におけるデータ項目の位置を示すか、またはデータ記述記入項目の特性を示す、2 桁の数字で表されたユーザー定義語。 1 から 49 のレベル番号は、論理レコードの階層構造におけるデータ項目の位置を示す。 1 から 9 のレベル番号は、1 桁の数字として書くことも、0 の後に有効数字を付けて書くこともできる。 レベル番号 66、77、および 88 は、データ記述記入項目の特性を識別する。

ライブラリー名 (* library-name)
COBOL ライブラリーの名前を表すユーザー定義語。与えられたソース・プログラムをコンパイルするためにコンパイラーが使用するライブラリーを識別する。

ライブラリー・テキスト (* library text)
COBOL ライブラリーの中にある一連のテキスト・ワード、コメント行、 区切り文字のスペース、または区切り文字の疑似テキスト区切り文字。

リリアン日 (Lilian date)
グレゴリオ暦の開始以降の日数。 第 1 日は 1582 年 10 月 15 日、金曜日。 リリアン日フォーマットは、 グレゴリオ暦の考案者であるルイジ・リリオにちなんだ名称。

*** LINAGE-COUNTER**
ページ本体内の現在位置を指す値を収めた特殊レジスター。

リンク (link)
(1) リンク接続 (伝送メディア) と、それぞれがリンク接続の終端にある 2 つのリンク・ステーションの組み合わせ。 1 つのリンクは、マルチポイントまたはトークンリング構成において、複数のリンク間で共用できる。 (2) データ項目あるいは 1 つ以上のコンピューター・プログラムの部分を相互接続すること。例えば、リンケージ・エディターによってオブジェクト・プログラムをリンクして共用ライブラリーを作成すること。

LINKAGE SECTION
呼び出し先のプログラムの DATA DIVISION 内のセクションであり、呼び出し側プログラムから使用可能なデータ項目が記述される。 これらのデータ項目は、呼び出し側プログラムおよび呼び出し先プログラムの両方から参照できる。

リテラル (literal)
ストリングを構成するために配列された文字によって、または形象定数を使用することによって、その値が決められる文字ストリング。

リトル・エンディアン (little-endian)
Intel プロセッサーが 2 進データおよび UTF-16 文字を保管するために使用するデフォルト形式。 この形式では、2 進数データ項目の最上位バイトが最上位のアドレスになり、 UTF-16 文字の最上位バイトが最上位のアドレスになる。 ビッグ・エンディアン *(big-endian)* と比較。

ロケール (locale)
プログラム実行環境の一連の属性であり、文化的に重要な考慮事項を示す。例えば、文字コード・ページ、照合シーケンス、日時形式、通貨表記、数値表記、または言語など。

*** LOCAL-STORAGE SECTION**
DATA DIVISION のセクションであり、VALUE 節で割り当てられた値に応じて、呼び出し単位で割り振りまたは解放が行われるストレージを定義する。

論理演算子 (* logical operator)
予約語 AND、OR、または NOT のいずれか。 条件の形成において、AND または OR、あるいはその両方を論理連結語として使用できる。 NOT は論理否定に使用できる。

論理レコード (* logical record)
最も包括的なデータ項目。 レコードのレベル番号は 01。 レコードは、基本項目またはグループ項目のどちらでもよい。 「レコード *(record)*」と同義。

下位終了 (* low-order end)
文字ストリングの右端の文字。

LSQ ファイル・システム (LSQ file system)
LSQ ファイル・システムは LINE SEQUENTIAL ファイルのみをサポートする。

M

メインプログラム (main program)
プログラムとサブルーチンからなる階層において、プロセス内でプログラムが実行されたときに最初に制御を受け取るプログラム。

Make ファイル (makefile)
アプリケーションに必要なファイルのリストが収められたテキスト・ファイル。 make ユーティリティーはこのファイルを使用して、ターゲット・ファイルを最新の変更で更新する。

大容量記憶 (* mass storage)
データを順次と非順次の 2 つの方法で編成して保管しておくことができるストレージ・メディア。

大容量記憶装置 (* mass storage device)
磁気ディスクなど、大きな記憶容量を持つ装置。

大容量記憶ファイル (* mass storage file)
大容量記憶メディアに格納されたレコードの集合。

MBCS
マルチバイト文字セット *(MBCS) (multibyte character set (MBCS))* を参照。

メガバイト、MB (* megabyte (MB))
1 メガバイトは 1,048,576 バイトに相当する。

マージ・ファイル (* merge file)
MERGE ステートメントによってマージされるレコードの集まり。 マージ・ファイルは、マージ機能により作成され、マージ機能によってのみ使用できる。

簡略名 (* mnemonic-name)
ENVIRONMENT DIVISION において、指定されたインプリメントする人の名前に関連したユーザー定義語。

モジュール定義ファイル (module definition file)
ロード・モジュール内のコード・セグメントを記述するファイル。

マルチバイト文字 (multibyte character)
マルチバイト文字セット内で 2 バイト以上で表わされる文字。 例えば、2 バイト以上で表わされる DBCS 文字または UTF-8 文字。 UTF-16 文字は、UTF-16 がマルチバイト文字セットでないため、マルチバイト文字ではありません。

マルチバイト文字セット (MBCS) (multibyte character set (MBCS))
さまざまなバイト数で表された文字からなるコード化文字セット。 例えば、EUC (拡張 UNIX コード)、UTF-8、1 バイト文字および 2 バイト EBCDIC 文字または ASCII 文字の混合物からなる文字セットなど。

マルチタスキング (multitasking)
2 つ以上のタスクの並行実行またはインターリーブド実行を可能にする操作モード。

マルチスレッド化 (multithreading)
コンピューター内で複数のパスを使用して実行を行う並行操作。 「マルチプロセッシング *(multiprocessing)*」と同義。

N

名前 (name)
COBOL オペランドを定義する 30 文字を超えないで構成されたワード。

名前空間 (namespace)
XML 名前空間 *(XML namespace)* を参照。

国別文字 (national character)
(1) 国別リテラルまたは USAGE NATIONAL の UTF-16 文字。 (2) UTF-16 で表される任意の文字。

国別文字データ (national character data)
UTF-16 で表されるデータの一般参照。

国別文字位置 (national character position)
文字位置 *(character position)* を参照。

国別データ (national data)
「国別文字データ *(national character data)*」を参照。

国別データ項目 (national data item)
カテゴリー国別、国別編集、または USAGE NATIONAL の数字編集のデータ項目。

国別 10 進数データ項目 (national decimal data item)
暗黙的または明示的に USAGE NATIONAL として記述されており、PICTURE の記号 9、S、P、および V の有効な組み合わせを含んでいる、外部 10 進数データ項目。

国別編集データ項目 (national-edited data item)
少なくとも 1 つの N のインスタンスおよび単純挿入記号 B、0、または / の少なくとも 1 つを含んでいる PICTURE 文字ストリングで記述されている、データ項目。 国別編集データ項目は USAGE NATIONAL を持ちます。

国別浮動小数点データ項目 (national floating-point data item)
暗黙的または明示的に USAGE NATIONAL として記述されており、浮動小数点データ項目を記述する PICTURE 文字ストリングを持っている、外部浮動小数点データ項目。

国別グループ項目 (national group item)
明示的または暗黙的に GROUP-USAGE NATIONAL 節で記述されたグループ項目。 国別グループ項目は、INSPECT、STRING、および UNSTRING などの操作で、カテゴリー国別の基本データ項目として定義されているかのように処理されます。 英数字グループ項目内で USAGE NATIONAL データ項目を定義するのとは対照的に、この処理により、国別文字の埋め込みおよび切り捨てが確実に正しく行われます。 グループ内の基本項目を処理する必要のある操作 (MOVE CORRESPONDING、ADD CORRESPONDING、および INITIALIZE など) の場合、国別グループは グループ・セマンティクスを使用して処理されます。

ネイティブ英数字データ項目 (native alphanumeric data item)
(XML 文書の場合) NATIVE 句で記述されたカテゴリー英数字データ項目、または CHAR(NATIVE) オプションを有効にしてコンパイルされたカテゴリー英数字データ項目。 データ項目のエンコードは、有効なランタイム・ロケールの ASCII または UTF-8 コード・ページである。

固有文字セット (* native character set)
OBJECT-COMPUTER 段落で指定されたコンピューターに関連した、インプリメントする人が定義した文字セット。

固有照合シーケンス (* native collating sequence)
OBJECT-COMPUTER 段落で指定されたコンピューターに関連した、インプリメントする人が定義した照合シーケンス。

複合否定条件 (* negated combined condition)
論理演算子 NOT とその直後に括弧で囲んだ複合条件を続けたもの。 条件 *(condition)* および 複合条件 *(combined condition)* も参照。

単純否定条件 (* negated simple condition)
論理演算子 NOT とその直後に単純条件を続けたもの。 条件 *(condition)* および 単純条件 *(simple condition)* も参照。

ネストされたプログラム (nested program)
他のプログラムの中に直接的に含まれているプログラム。

次の実行可能文 (* next executable sentence)
現在のステートメントの実行完了後に制御が移される次の文。

次の実行可能なステートメント (* next executable statement)
現在のステートメントの実行完了後に制御が移される次のステートメント。

次のレコード (* next record)
ファイルの現在のレコードに論理的に続くレコード。

独立項目 (* noncontiguous items)
WORKING-STORAGE SECTION および LINKAGE SECTION 内の基本データ項目で、他のデータ項目と階層上の関係を持たないもの。

非日付データ (nondate)
次の項目のいずれか。

- 日付記述記入項目に DATE FORMAT 節が含まれていないデータ項目
- リテラル
- UNDATE 関数を使用して変換された日付フィールド
- 参照変更された日付フィールド
- 日付フィールド・オペランドを含む特定の算術演算の結果。例えば、2 つの互換日付 フィールドの差

ヌル (null)
無効なアドレスの値をポインター・データ項目に割り当てるために使用される形象定数。 NULL を使えるところならばどこでも、NULLS を使用できる。

数字 (* numeric character)
次のような数字に属する文字。 0、1、2、3、4、5、6、7、8、9。

数値データ項目 (numeric data item)
(1) 記述により内容が数字 0 から 9 より選ばれた文字で表される値に制限されるデータ項目。符号付きである場合、この項目は +、-、または他の表記の演算符号も含むことができます。 (2) カテゴリー数値、内部浮動小数点、または外部浮動小数点のデータ項目。 数値データ項目は、USAGE DISPLAY、NATIONAL、PACKED-DECIMAL、BINARY、COMP、COMP-1、COMP-2、COMP-3、COMP-4、または COMP-5 を持つことができます。

数字編集データ項目 (numeric-edited data item)
印刷出力の際に使用するのに適したフォーマットの数値データを含むデータ項目。 データ項目は、外部 10 進数字の 0 から 9 の数字、小数点、コンマ、通貨符号、符号制御文字、その他の編集記号から構成される。 数字編集項目は、USAGE DISPLAY または USAGE NATIONAL のいずれかで表すことができる。

数字関数 (* numeric function)
クラスとカテゴリーは数字だが、考えられる評価のいくつかにおいて整数関数の要件を満たさないような関数。

数値リテラル (* numeric literal)
1 つ以上の数字から構成されるリテラルで、小数点または代数符号あるいはその両方を含むことができる。 小数点は右端の文字であってはならない。 代数符号がある場合には、それが左端の文字でなければならない。

O

オブジェクト・コード (object code)
コンパイラーまたはアセンブラーからの出力。それ自体が実行可能なマシン・コードか、またはその種のコードの作成を目的として処理するのに適する。

*** OBJECT-COMPUTER**
ENVIRONMENT DIVISION にある段落の名前であり、ここではオブジェクト・プログラムが実行されるコンピューター環境が記述される。

オブジェクト・コンピューター記入項目 (* object computer entry)
ENVIRONMENT DIVISION の OBJECT-COMPUTER 段落内の記入項目。この記入項目には、オブジェクト・プログラムが実行されるコンピューター環境を記述する節が入っている。

記入項目のオブジェクト (* object of entry)
COBOL プログラムの DATA DIVISION 記入項目内の一連のオペランドと予約語であり、その記入項目のサブジェクトの直後に続く。

オブジェクト・プログラム (object program)
問題を解決するためにデータと相互に作用することを目的とする実行可能なマシン言語命令とその他の要素の集合または グループ。 このコンテキストでは、オブジェクト・プログラムとは一般に、COBOL コンパイラーがソース・プログラム定義を操作した 結果得られるマシン言語である。 あいまいになる危険がない場合には、オブジェクト・プログラム という用語の代わりに プログラム というワードだけが使用される。

オブジェクト時 (* object time)
オブジェクト・プログラムが実行されるとき。 実行時 *(run time)* と同義。

古くなったエレメント (* obsolete element)
2002 COBOL 標準 から削除された 85 COBOL 標準 の COBOL 言語エレメント。

ODBC
Open Database Connectivity (ODBC) を参照。

ODO オブジェクト (ODO object)
次の例では、X が OCCURS DEPENDING ON 節のオブジェクト (ODO オブジェクト) である。

```
WORKING-STORAGE SECTION.
01  TABLE-1.
    05  X                   PIC S9.
    05  Y OCCURS 3 TIMES
          DEPENDING ON X    PIC X.
```

ODO オブジェクトの値によって、テーブル内の ODO サブジェクトの数が決まる。

ODO サブジェクト (ODO subject)
上記の例では、Y が OCCURS DEPENDING ON 節の サブジェクト (ODO サブジェクト) である。 テーブル内の ODO サブジェクトの数である Y の値は、X の値によって 決まる。

Open Database Connectivity (ODBC)
さまざまなデータベースおよびファイル・システムのデータへのアクセスを可能にするアプリケーション・プログラミング・インターフェース (API) の仕様。

オープン・モード (* open mode)
OPEN ステートメントが実行されてから、REEL および UNIT 句の指定のない CLOSE ステートメントが実行される前までのファイルの状態。 個々のオープン・モードは、OPEN ステートメントの中で、INPUT、OUTPUT、I-O、または EXTEND のいずれかとして指定する。

オペランド (* operand)
(1) オペランドの一般的な定義は、「操作の対象となるコンポーネント」である。 (2) 本書の目的に沿った言い方をすれば、ステートメントや記入項目の形式中に現れる小文字または日本語で書かれた語 (または語群) はオペランドと見なされ、そのオペランドによって指示されたデータに対して暗黙の参照を行う。

演算、操作 (operation)
オブジェクトに関して要求できるサービス。

演算符号 (* operational sign)
値が正であるか負であるかを示すために数字データ項目または数字リテラルに付けられる代数符号。

オプション・ファイル (optional file)
オブジェクト・プログラムが実行されるたびに必ずしも使用可能でなくてもよいものとして宣言されているファイル。

オプション・ワード (* optional word)
言語を読みやすくする目的でのみ特定の形式で含められる予約語。 このようなワードが表示されている形式をソース単位内で使用する場合、そのワードの有無はユーザーが選択できる。

出力ファイル (* output file)
出力モードまたは拡張モードのいずれかでオープンされるファイル。

出力モード (* output mode)
OUTPUT または EXTEND 句の指定のある OPEN ステートメントが実行されてから、 REEL および UNIT 句の指定のない CLOSE ステートメントが実行される前までのファイルの状態。

出力プロシージャー (* output procedure)
SORT ステートメントの実行中にソート機能が完了した後で制御が渡されるステートメントの集合、または MERGE ステートメントの実行中に、要求があればマージ機能がマージ済みの順序になっているレコードのうち次のレコードを選択できるようになった後で制御が渡されるステートメントの集合。

オーバーフロー条件 (overflow condition)
ある演算結果の一部が意図した記憶単位の容量を超えた場合に発生する条件。

P

パック 10 進数データ項目 (packed-decimal data item)
内部 *10* 進数データ項目 *(internal decimal data item)* を参照。

埋め込み文字 (padding character)
物理レコード内の未使用文字位置を埋めるのに使用される英数字または国別文字。

ページ (page)
データの物理的分離を表す、出力データの垂直分割。 分離は、内部論理要件または出力メディアの外部特性、あるいはその両方に基づいて行われる。

ページ本体 (* page body)
行を記述できる、または行送りすることができる (またはその両方ができる) 論理ページの部分。

段落 (* paragraph)
PROCEDURE DIVISION では、段落名の後に分離文字ピリオドが続き、その後に 0 個以上の文が続く。IDENTIFICATION DIVISION および ENVIRONMENT DIVISION では、 段落ヘッダーの後に 0 個以上の記入項目が続く。

段落ヘッダー (* paragraph header)
予約語の後に分離文字ピリオドが付いたもので、IDENTIFICATION DIVISION および ENVIRONMENT DIVISION において段落の始まりを示すもの。 IDENTIFICATION DIVISION で許可されている段落ヘッダーは次のとおり。

```
PROGRAM-ID. (Program IDENTIFICATION
   DIVISION)
AUTHOR.
INSTALLATION.
DATE-WRITTEN.
DATE-COMPILED.
SECURITY.
```

ENVIRONMENT DIVISION で許可されている段落ヘッダーは次のとおり。

```
SOURCE-COMPUTER.
OBJECT-COMPUTER.
SPECIAL-NAMES.
REPOSITORY. (Program
   CONFIGURATION SECTION)
FILE-CONTROL.
I-O-CONTROL.
```

段落名 (* paragraph-name)
PROCEDURE DIVISION の中の段落を識別し開始するユーザー定義語。

パラメーター (parameter)
呼び出し側プログラムと呼び出し先プログラムの間で受け渡されるデータ。

句 (* phrase)
連続する 1 つ以上の COBOL 文字ストリングを配列したセットで、COBOL プロシージャー・ステートメントまたは COBOL 節の一部を構成する。

物理レコード (* physical record)
ブロック *(block)* を参照。

ポインター・データ項目 (pointer data item)
アドレス値を保管できるデータ項目。 これらのデータ項目は、USAGE IS POINTER 節を使用してポインターとして明示的に定義される。 ADDRESS OF 特殊レジスターは、ポインター・データ項目として暗黙的に定義されている。 ポインター・データ項目は、他のポインター・データ項目と等価かどうか比較したり、他のポインター・データ項目に内容を移動したりできる。

移植する、ポート (port)
(1) 異なるプラットフォームで実行できるようにコンピューター・プログラムを変更すること。 (2) インターネット・プロトコルでは、Transmission Control Protocol (TCP) プロトコルまたは User Datagram Protocol (UDP) プロトコルと高水準のプロトコルまたはアプリケーションの間の特定の論理結合子。 ポートはポート番号によって識別される。

移植性 (portability)
あるアプリケーション・プラットフォームから別のアプリケーション・プラットフォームに、ソース・プログラムに比較的わずかな変更を加えるだけでアプリケーション・プログラムを移行できる能力。

基本レコード・キー (* prime record key)
索引付きファイルのレコードを固有なものとして識別する内容を持つキー。

優先順位番号 (* priority-number)
セグメンテーションの目的で、PROCEDURE DIVISION 内のセクションを分類するユーザー定義語。 セグメント番号には 0 から 9 までの文字だけしか使用できない。セグメント番号は 1 桁または 2 桁として表すことができる。

プロシージャー (* procedure)
PROCEDURE DIVISION 内にある 1 つの段落または論理的に連続する段落のグループ、あるいは 1 つのセクションまたは論理的に連続するセクションのグループ。

プロシージャー・ブランチ・ステートメント (* procedure branching statement)
ソース・コードの中にステートメントが書かれている順番どおりに次の実行可能ステートメントに制御の移動をせず、別のステートメントに明示的に制御の移動を引き起こすステートメント。 プロシージャー分岐ステートメントは次のとおり。ALTER、CALL、EXIT、EXIT PROGRAM、GO TO、MERGE (OUTPUT PROCEDURE 句付き)、PERFORM および SORT (INPUT PROCEDURE または OUTPUT PROCEDURE 句付き)、XML PARSE。

PROCEDURE DIVISION
COBOL の部の 1 つで、問題を解決するための命令を記述する。

プロシージャー統合 (procedure integration)
COBOL 最適化プログラムの機能の 1 つであり、実行されるプロシージャーまたは含まれ ているプログラムへの呼び出しを単純化する。

PERFORM プロシージャー統合とは、PERFORM ステートメントが、実行されるプロシージャーによって置き換えられるプロセスのこと。 含まれているプログラムのプロシージャー統合とは、含まれているプログラムへの呼び出しがプログラム・コードによって置き換えられるプロセスのこと。

プロシージャー名 (* procedure-name)
PROCEDURE DIVISION の中にある段落またはセクションに名前を付けるために使用される ユーザー定義語。 プロシージャー名は、段落名 (これは修飾することができる) またはセクション名から構成される。

プロシージャー・ポインター (procedure pointer)
入り口点を指すポインターを保管できるデータ項目。 USAGE IS PROCEDURE-POINTER 節を付けて定義したデータ項目が、 プロシージャーへの入り口点のアドレスを収める。

プロシージャー・ポインター・データ項目 (procedure-pointer data item)
入り口点を指すポインターを保管できるデータ項目。 USAGE IS PROCEDURE-POINTER 節で定義されるデータ項目には、プロシージャー入り口点のアドレスが入っている。一般的に、COBOL のプログラムと通信するために使用される。

プロセス (process)
プログラムの全部または一部の実行中に発生する一連のイベント。 複数のプロセスを並行して実行することができ、1 つのプロセス内で実行されるプログラムはリソースを共用することができる。

プログラム (program)
(1) コンピューターで処理するのに適した一連の命令。 処理には、コンパイラーを使用してプログラムの実行準備をすることやランタイム環境を使用してプログラムを実行することが含まれます。 (2) 1 つ以上の相互に関係のあるモジュールの論理アセンブリー。 同じプログラムの複数のコピーを異なるプロセスで実行することができる。

プログラム識別記入項目 (* program identification entry)
IDENTIFICATION DIVISION の PROGRAM-ID 段落内の記入項目であり、プログラム名を指定し、選択されたプログラム属性をプログラムに割り当てる節が入っている。

プログラム名 (program-name)
IDENTIFICATION DIVISION およびプログラム終了マーカーにおいて、COBOL ソース・プログラムを識別するユーザー定義語または英数字リテラル。

プロジェクト (project)
ダイナミック・リンク・ライブラリー (DLL) や他の実行可能ファイル (EXE) などのターゲットを作成するのに必要な、データおよびアクションの完全セット。

疑似テキスト (* pseudo-text)
ソース・プログラムまたは COBOL ライブラリーにおいて、疑似テキスト区切り文字によって区切られた一連のテキスト・ワード、コメント行、または区切り文字スペース (疑似テキスト区切り文字を含まない)。

疑似テキスト区切り文字 (* pseudo-text delimiter)
疑似テキストを区切るために使用される 2 つの連続する等号文字 (==)。

句読文字 (* punctuation character)
以下のセットに属する文字。

文字	意味
,	コンマ
;	セミコロン
:	コロン
.	ピリオド (終止符)
"	引用符
(	左括弧
)	右括弧
	スペース
=	等号

Q

QSAM (待機順次アクセス方式) (QSAM (Queued Sequential Access Method))
基本順次アクセス方式 (BSAM) の拡張版。 この方式を使用する場合、処理を待っている入力データ・ブロック、または処理が終わって補助記憶装置または出力装置への転送を待っている出力データ・ブロックのキューが形成される。

QSAM ファイル・システム (QSAM file system)
QSAM (待機順次アクセス方式) ファイル・システムでは、固定長レコード、可変長レコード、およびスパン・レコードがサポートされ、オプション binary および quote site rdw を指定して (z/OS FTP を使用して) z/OS から AIX または Linux に転送した QSAM ファイルに直接アクセスできる。QSAM ファイルでは、すべての COBOL データ型がレコード内でサポートされる。

修飾データ名 (* qualified data-name)
データ名と、その後に連結語の OF または IN とデータ名修飾子を続けたものが 1 つ以上のセットで続いて構成される ID。

修飾子 (* qualifier)
(1) レベル標識と関連付けられるデータ名または名前であり、参照の際に、別のデータ名 (修飾子に従属する項目の名前) と 一緒に、または条件名と一緒に使用される。 (2) セクション名。 そのセクションの中で指定されている段落名と共に参照する際に使用される。 (3) ライブラリー名。 そのライブラリーと関連付けられたテキスト名と共に参照する際に使用される。

R

ランダム・アクセス (* random access)
キー・データ項目のプログラム指定値を使用して、相対ファイルまたは索引付きファイルから取り出したり、削除したり、またはそこに入れたりする論理レコードを識別するアクセス・モード。

レコード (* record)
論理レコード *(logical record)* を参照。

レコード域 (* record area)
DATA DIVISION の FILE SECTION 内のレコード記述項目で記述されるレコードを処理する目的で割り振られるストレージ域。 FILE SECTION では、レコード域の現行の文字位置の数は、明示または暗黙の RECORD 節によって決められる。

レコード記述 (* record description)
レコード記述項目 *(record description entry)* を参照。

レコード記述項目 (* record description entry)
特定のレコードに関連したデータ記述記入項目全体。 「レコード記述 *(record description)*」と同義。

レコード・キー (record key)
索引付きファイル内のレコードを識別する内容を持つキー。

レコード・キー名 (record-key-name)
索引付きファイルに関連付けられているキーを表すユーザー定義語。

レコード名 (* record-name)
COBOL プログラムの DATA DIVISION 内のレコード記述項目で記述されるレコードに名前を付けるユーザー定義語。

レコード番号 (* record number)
編成が順次であるファイル内のレコードの順序数。

記録モード (recording mode)
ファイル内の論理レコードの形式。 記録モードは、F (固定長)、V (可変長)、S (スパン)、または U (不定形式) とすることができる。

再帰 (recursion)
それ自体を呼び出すプログラム、または、それ自体で呼び出したプログラムのいずれかによって直接あるいは間接に呼び出されるプログラム。

再帰可能 (recursively capable)
PROGRAM-ID ステートメントで RECURSIVE 属性が指定されていれば、プログラムは再帰可能である (再帰的に呼び出すことができる)。

リール (reel)
ストレージ・メディアの個別部分。その大きさはインプリメントする人によって決定され、1 つのファイルの一部、1 つのファイルの全部、または任意の個数のファイルが収容される。 「ユニット *(unit)*」および「ボリューム *(volume)*」と同義。

再入可能 (reentrant)
プログラムまたはルーチンの属性。この属性によって、ロード・モジュールの 1 つのコピーを複数のユーザーが共用できる。

参照形式 (* reference format)
COBOL ソース・プログラムを記述するに際して標準的な方式を提供する形式。

参照変更 (reference modification)
新規のカテゴリー英数字、カテゴリー DBCS、またはカテゴリー国別のデータ 項目を定義する方法であり、USAGE DISPLAY、DISPLAY-1、または NATIONAL データ項目の左端文字および左端文字位置を基準にした長さを指定して定義する方法です。

参照修飾子 (* reference-modifier)
固有のデータ項目を定義する文字ストリングと区切り文字の構文的に正しい組み合わせ。 区切り用の左括弧分離符号、左端の文字位置、分離符号のコロン、長さ (オプション)、および区切り用の右括弧分離符号を含む。

関係 (* relation)
関係演算子 *(relational operator)* または 比較条件 *(relation condition)* を参照。

比較文字 (* relation character)
以下のセットに属する文字。

文字	意味
>	より大きい
<	より小さい
=	に等しい

比較条件 (* relation condition)
ある算術式、データ項目、英数字リテラル、または索引名の値が、他の算術式、データ項目、英数字リテラル、または索引名の 値と特定の関係があるという命題 (それに対して真理値を判別する)。 関係演算子 *(relational operator)* も参照。

関係演算子 (* relational operator)
比較条件の構造で使用される、予約語、比較文字、連続する予約語のグループ、または 連続する予約語と比較文字のグループ。 使用できる演算子とそれらの意味は次のとおり。

文字	意味
IS GREATER THAN	より大きい
IS >	より大きい
IS NOT GREATER THAN	より大きくない (以下)
IS NOT >	より大きくない (以下)
IS LESS THAN	より小さい
IS <	より小さい
IS NOT LESS THAN	より小さくない (以上)
IS NOT <	より小さくない (以上)
IS EQUAL TO	に等しい
IS =	に等しい
IS NOT EQUAL TO	に等しくない
IS NOT =	に等しくない
IS GREATER THAN OR EQUAL TO	より大きいか等しい (以上)
IS >=	より大きいか等しい (以上)
IS LESS THAN OR EQUAL TO	より小さいか等しい (以下)
IS <=	より小さいか等しい (以下)

相対ファイル (* relative file)
相対編成のファイル。

相対キー (* relative key)
相対ファイルの中の論理レコードを識別するための内容を持つキー。

相対編成 (* relative organization)
各レコードが、レコードのファイル内における論理的順序位置を指定する 0 より大きい整数値によって、固有なものとして識別される永続的な論理ファイル構造。

相対レコード番号 (* relative record number)
相対編成ファイル内でのレコードの序数。 この番号は、整数の数値リテラルとして扱われる。

予約語 (* reserved word)
COBOL ソース・プログラムの中で使用することができるが、ユーザー定義語またはシステム名としてプログラムの中で使用されてはならないワードのリスト中に挙げられている COBOL ワード。

リソース (* resource)
オペレーティング・システムの制御下に置かれており、実行中のプログラムによって使用できる機能またはサービス。

結果 ID (* resultant identifier)
算術演算の結果が収められるユーザー定義のデータ項目。

ルーチン (routine)
コンピューターに操作または一連の関連操作を実行させる、COBOL プログラム内の一連のステートメント。

ルーチン名 (* routine-name)
COBOL 以外の言語で記述されたプロシージャーを識別するユーザー定義語。

RSD ファイル・システム (RSD file system)
RSD (レコード順次区切り) ファイル・システムは、順次ファイルをサポートするワークステーション・ファイル・システムである。 RSD ファイルは、固定長または可変長レコードのすべての COBOL データ・タイプをサポートし、ほとんどのファイル・エディターで編集可能であり、他の言語で作成されたプログラムによって読み取ることができる。 このシステムは順次ファイルのみをサポートする。

実行時 (* run time)
オブジェクト・プログラムが実行されるとき。 「オブジェクト時 *(object time)*」と同義。

ランタイム環境 (runtime environment)
COBOL プログラムが実行される環境。

実行単位 (* run unit)
1 つの独立型オブジェクト・プログラム、あるいは COBOL の CALL ステートメントによって相互作用し、実行時に 1 つのエンティティーとして機能する複数のオブジェクト・プログラム。

S

SBCS
1 バイト文字セット *(SBCS) (single-byte character set (SBCS))* を参照。

範囲終了符号 (scope terminator)
PROCEDURE DIVISION の特定のステートメントの終わりを示す COBOL 予約語。これは明示的なもの (例えば、END-ADD など) であることもあれば、暗黙のもの (分離文字ピリオド) であることもある。

セクション (* section)
ゼロ、1 つ、または複数の段落またはエンティティー (セクション本体と呼ばれる) と、その最初のものの前にセクション・ヘッダーが付いているもの。 各セクションは、セクション・ヘッダーとそれに関連するセクション本体から構成される。

セクション・ヘッダー (* section header)
後ろに分離文字ピリオドが付いたワードの組み合わせで あり、ENVIRONMENT、DATA、または PROCEDURE の各部において、セクションの始まりを示すもの。 ENVIRONMENT DIVISION および DATA DIVISION では、 セクション・ヘッダーは、予約語の後に分離文字ピリオドを続けたものから構成される。 ENVIRONMENT DIVISION で許可されているセクション・ヘッダーは次のとおり。

```
CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.
```

DATA DIVISION で許可されているセクション・ヘッダーは次のとおり。

```
FILE SECTION.
WORKING-STORAGE SECTION.
LOCAL-STORAGE SECTION.
LINKAGE SECTION.
```

PROCEDURE DIVISION では、セクション・ヘッダーは、セクション名、その後に続く予約語 SECTION、およびその後の分離文字ピリオドから構成される。

セクション名 (* section-name)
PROCEDURE DIVISION の中にあるセクションに名前を付けるユーザー定義語。

セグメント化 (segmentation)
85 COBOL 標準 分割モジュールに基づく COBOL for Linux の機能。 セグメンテーション機能は、セクション・ヘッダーの優先順位番号を使用して、セクションを固定セグメントまたは独立セグメントに割り当てる。 セグメント種別は、セグメントに含まれるプロシージャーが初期状態の制御を受け取るか、最後に使われた状態の制御を受け取るかに影響を及ぼす。

選択構造 (selection structure)
条件が真であるか偽であるかに応じて、ある一連のステートメントか、または別の一連のステートメントが実行されるというプログラムの処理ロジック。

文 (* sentence)
1 つ以上のステートメントの並びで、その最後のものは、分離文字ピリオドで終了する。

別個にコンパイルされたプログラム (* separately compiled program)
あるプログラムが、その中に含まれているプログラムと一緒に、それ以外のすべてのプログラムとは別個にコンパイルされること。

分離文字 (* separator)
文字ストリングを区切るために使用される、1 文字または連続する 2 文字。

分離文字コンマ (* separator comma)
文字ストリングを区切るために使われる、後ろに 1 つのスペースが続く 1 つのコンマ (,)。

分離文字ピリオド (* separator period)
文字ストリングを区切るために使われる、後ろに 1 つのスペースが続く 1 つのピリオド (.)。

分離文字セミコロン (* separator semicolon)
文字ストリングを区切るために使われる、後ろに 1 つのスペースが続く 1 つのセミコロン (;)。

順序構造 (sequence structure)
一連のステートメントが、順序どおりに実行されるプログラムの処理ロジック。

順次アクセス (* sequential access)
ファイル内のレコードの順序によって規定されている、論理レコードの連続した前後関係順に、論理レコードをファイルから取り出したり、ファイルに書き込んだりするアクセス・モード。

順次ファイル (* sequential file)
順次編成のファイル。

順次編成 (* sequential organization)
レコードがファイルに書き込まれるときに確定されたレコードの前後関係によって識別されるような永続的な論理ファイル構造。

逐次探索 (serial search)
最初のメンバーから始めて最後のメンバーで終わるように、ある集合のメンバーが連続的に検査される探査方法。

SFS ファイル・システム (SFS file system)
CICS の SFS (Structured File Server) ファイル・システムは、順次ファイル・アクセス、相対ファイル・アクセス、およびキー索引付きファイル・アクセスをサポートするレコード単位ファイル・システムである。

共用ライブラリー (shared library)
リンカーによって作成され、少なくとも 1 つのサブルーチンを含み、複数のプロセスで使用できるライブラリー。 プログラムとサブルーチンは従来どおりリンクされるが、異なるサブルーチンに共通する

コードは結合されて 1 つのライブラリー・ファイルに入れられる。このライブラリー・ファイルは実行時にロード可能で、多数のプログラムで共用できる。 この共用ライブラリー・ファイルを識別するキーは、各サブルーチンのヘッダーにある。

符号条件 (* sign condition)
データ項目や算術式の代数値が、0 より小さいか、大きいか、または等しいかという命題で、それに関して真理値が判別できる。

シグニチャー (signature)
ある操作とそのパラメーターの名前。

単純条件 (* simple condition)
以下のセットから選択される任意の単一条件。

- 比較条件
- クラス条件
- 条件名条件
- 切り替え状況条件
- 符号条件

条件 *(condition)* および 単純否定条件 *(negated simple condition)* も参照。

1 バイト文字セット (SBCS)(single-byte character set (SBCS))
各文字が 1 バイトで表現される文字のセット。 *ASCII* および *EBCDIC (*拡張 *2* 進化 *10* 進コード*) (EBCDIC (Extended Binary-Coded Decimal Interchange Code))* も参照。

遊びバイト (レコード内) (slack bytes (within records))
複数の基本データ項目を正しく位置合わせするために、コンパイラーによってデータ項目間に挿入されるバイト。 遊びバイトには意味のあるデータは含まれない。 正しい位置合わせを行うために遊びバイトが必要なときは、SYNCHRONIZED 節によって、コン パイラーに遊びバイトを挿入させる。

遊びバイト (レコード間) (slack bytes (between records))
複数の基本データ項目を正しく位置合わせするために、プログラマーによってファイルのブロック化論理レコードの間に挿入されるバイト。 場合によっては、レコード間に遊びバイトを挿入することによってバッファー内で処理されるレコードのパフォーマンスが改善される。

ソート・ファイル (* sort file)
SORT ステートメントによってソートされるレコードの集まり。ソート・ファイルは、ソート機能によってのみ作成され使用される。

ソート・マージ・ファイル記述記入項目 (* sort-merge file description entry)
DATA DIVISION の FILE SECTION の中にある記入項目。レベル標識 SD と、それに続くファイル名、および、必要に応じて、次に続く一連のファイル節から構成される。

*** SOURCE-COMPUTER**
ENVIRONMENT DIVISION にある段落の名前であり、ここではソース・プログラムがコンパイルされるコンピューター環境が記述される。

コンパイル用コンピューター記入項目 (* source computer entry)
ENVIRONMENT DIVISION の SOURCE-COMPUTER 段落内の記入項目であり、ソース・プログラムがコンパイルされるコンピューター環境を記述する節が入っている。

ソース項目 (* source item)
SOURCE 節によって指定される ID で、印刷可能な項目の値を提供する。

ソース・プログラム (source program)
ソース・プログラムは、他の形式や記号を使用して表現することができるが、本書では、構文的に正しい COBOL ステートメントの集合を常に指している。 COBOL ソース・プログラムは、IDENTIFICATION DIVISION または COPY ステートメントで開始され、指定された場合はプログラム終了マーカーで終了するか、または追加のソース・プログラム行なしで終了する。

ソース単位 (source unit)
COBOL ソース・コードの 1 単位で、個別にコンパイルできる。プログラム。 コンパイル単位 とも呼ばれる。

特殊文字 (special character)
以下のセットに属する文字。

文字	意味
+	正符号
-	負符号 (-) (ハイフン)
*	アスタリスク
/	斜線 (スラッシュ)
=	等号
$	通貨記号
,	コンマ
;	セミコロン
.	ピリオド (小数点、終止符)
"	引用符
'	アポストロフィ
(	左括弧
)	右括弧
>	より大きい
<	より小さい
:	コロン
_	下線

SPECIAL-NAMES
ENVIRONMENT DIVISION にある段落の名前。この段落では、環境名がユーザー指定の簡略名と関連付けられる。

特殊名記入項目 (* special names entry)
ENVIRONMENT DIVISION の SPECIAL-NAMES 段落内の記入項目。この記入項目は、通貨記号を指定したり、小数点を選択したり、シンボリック文字を指定したり、インプリメントする人の名前をユーザー指定の簡略名と関連付けたり、英字名を文字セットまたは照合シーケンスと関連付けたり、クラス名を一連の文字と関連付けたりするための手段を提供する。

特殊レジスター (* special registers)
特定のコンパイラー生成ストレージ域のことで、その基本的な使用法は、具体的な COBOL 機能を使用したときに 作り出される情報を記憶することである。

ステートメント (* statement)
COBOL ソース・プログラムに書かれる、動詞を冒頭に置いた、ワード、リテラル、および区切り記号の構文的に正しい組み合わせ。

STL ファイル・システム (STL file system)
標準言語ファイル・システムは、COBOL 用のネイティブ・ワークステーション・ファイル・システムである。 このシステムは、順次ファイル、相対ファイル、および索引ファイルをサポートする。

構造化プログラミング (structured programming)
コンピューター・プログラムを編成してコーディングするための技法であり、この技法 では、プログラムはセグメントの階層で構成され、それぞれのセグメントには 1 つの 入り口点と 1 つの出口点がある。 制御は、構造の下方へと渡され、階層内のより上位レベルへの無条件分岐は行われない。

記入項目のサブジェクト (* subject of entry)
DATA DIVISION の記入項目内において、レベル標識またはレベル番号の直後に現れるオペランドまたは予約語。

サブプログラム (* subprogram)
呼び出し先プログラム *(called program)* を参照。

添え字 (* subscript)
整数、(オプションで演算子 + または - 付きの整数が後ろにある) データ名、あるいは (オプションで演算子 + または - 付きの整数が後ろにある) 索引名のいずれかによって表されるオカレンス番号。これによりテーブル内の特定のエレメントを識別する。 可変数の引数を認める関数では、添え字付き ID を関数引数として使用する場合は、添え字に ALL を使用することができる。

添え字付きデータ名 (* subscripted data-name)
データ名とその後の括弧で囲まれた 1 つ以上の添え字から構成される ID。

置換文字 (substitution character)
ソース・コード・ページからターゲット・コード・ページへの変換の際に、ターゲット・コード・ページで定義されていない文字を表すのに使用される文字。

サロゲート・ペア (surrogate pair)
UTF-16 形式のユニコードで、共に 1 つのユニコード図形文字を表すエンコード方式ペアの単位。 ペアの最初の単位は上位サロゲート と呼ばれ、第 2 の単位は下位サロゲート と 呼ばれる。 上位サロゲートのコード値の範囲は、X'D800' から X'DBFF' である。 下位サロゲートのコード値の範囲は、X'DC00' から X'DFFF' である。 サロゲート・ペアは、Unicode 16 ビット・コード文字セットで表現できる 65,536 文字より多くの文字を表現できる。

切り替え状況条件 (switch-status condition)
オンまたはオフに設定可能な UPSI スイッチが、特定の状況に設定されているという命題で、これに関して真理値を判別することができる。

シンボリック文字 (* symbolic-character)
ユーザー定義の形象定数を指定するユーザー定義語。

構文 (syntax)
(1) 意味や解釈および使用の方法に依存しない、 文字同士または文字のグループ同士の間の関係。 (2) 言語における表現の構造。 (3) 言語構造を支配する規則。 (4) 記号相互の関係。 (5) ステートメントの構築にかかわる規則。

SYSADATA
ADATA コンパイラー・オプションが有効な場合に生成される追加のコンパイル情報のファイル。

SYSIN
1 次コンパイラー入力ファイル。

SYSLIB
2 次コンパイラー入力ファイル。LIB コンパイラー・オプションが有効な場合に処理される。

SYSPRINT
コンパイラー・リスト・ファイル。

システム名 (* system-name)
オペレーティング環境と連絡し合うために使用される COBOL ワード。

T

テーブル (* table)
DATA DIVISION の中で OCCURS 節によって定義される、論理的に連続するデータ項目の集合。

テーブル・エレメント (* table element)
テーブルを構成する反復項目の集合に属するデータ項目。

テキスト名 (* text-name)
ライブラリー・テキストを識別するユーザー定義語。

テキスト・ワード (* text word)
以下のいずれかの文字から成る COBOL ライブラリー、ソース・プログラム、または疑似テキスト内のマージン A およびマージン R の間の、1 文字または連続した文字のシーケンス。

- スペース以外の区切り記号、疑似テキスト区切り文字、英数字リテラルの開始と終了の区切り文字。ライブラリー、ソース・プログラム、または疑似テキスト内のコンテキストに関係なく、右括弧文字と左括弧文字は常に テキスト・ワードと見なされる。

- 英数字リテラルの場合には、リテラルを囲む左引用符と右引用符を含むリテラル。
- コメント行および区切り記号によって囲まれたワード COPY を除く、その他の連続する一連の COBOL 文字で、区切り記号でもリテラルでもないもの。

スレッド (thread)
プロセスの制御下にあるコンピューター命令のストリーム (プロセス内のアプリケーションによって開始される)。

トークン (token)
COBOL エディターでは、プログラムにおける意味の単位。 トークンには、データ、言語キーワード、ID、またはその他の言語構文の一部を含めることができる。

トップダウン設計 (top-down design)
関連付けられた諸機能が、構造の各レベルで実行されるようにする階層構造を使ったコンピューター・プログラムの設計。

トップダウン開発 (top-down development)
構造化プログラミング *(structured programming)* を参照。

トレーラー・ラベル (trailer-label)
(1) 記録メディア・ユニットのデータ・レコードの後にある、ラベル。 (2)「ファイル終わりラベル *(end-of-file label)*」の同義語。

トラブルシューティング (troubleshoot)
コンピューター・ソフトウェアの使用中に問題を検出し、突き止め、除去すること。

真理値 (* truth value)
2 つの値 (真または偽) のどちらか一方によって、条件評価の結果を表したもの。

U

単項演算子 (* unary operator)
正符号 (+) または負符号 (-)。 算術式の変数や算術式の左括弧の前に置き、それぞれ +1 または -1 を式に乗算する。

Unicode
現代世界の各国の言語で記述されるテキストの交換、処理、表示をサポートする汎用文字エンコード標準。 UTF-8、UTF-16、UTF-32 など、Unicode を表現する複数のエンコード・スキームが ある。 COBOL for Linux では、国別データ・タイプの表記としてリトル・エンディアン・フォーマットの UTF-16 を使用して Unicode をサポートしている。

URI (Uniform Resource Identifier (URI))
リソースを一意に指す文字のシーケンスのことで、COBOL for Linux では、名前空間の ID。URI 構文は、文書「*Uniform Resource Identifier (URI): Generic Syntax*」で定義されています。.

ユニット (unit)
直接アクセスのモジュールであり、その大きさは IBM によって決められている。

不成功の実行 (* unsuccessful execution)
ステートメントの実行が試みられたが、そのステートメントに指定された操作すべてを実行できなかったこと。 あるステートメントの実行不成功は、そのステートメントによって参照されるデータには影響を及ぼさないが、状況表示には影響を与える可能性がある。

UPSI スイッチ (UPSI switch)
ハードウェア・スイッチの機能を実行するプログラム・スイッチ。 UPSI-0 から UPSI-7 の 8 つのスイッチがある。

URI
URI を参照。

ユーザー定義語 (* user-defined word)
節やステートメントの形式を満たすためにユーザーが提供する必要のある COBOL ワード。

V

変数 (* variable)
オブジェクト・プログラムの実行によって変更を受ける可能性のある値を持つデータ項目。 算術式で使われる変数は、数字基本項目でなければならない。

可変長項目 (variable-length item)
OCCURS 節の DEPENDING 句で記述された表を含んだグループ項目。

可変長レコード (* variable-length record)
ファイル記述項目またはソート・マージ・ファイル記述記入項目が、 文字位置の数が可変であるレコードを許容しているファイルに関連付けられているレコード。

可変オカレンス・データ項目 (* variable-occurrence data item)
可変オカレンス・データ項目とは、反復される回数が可変であるテーブル・エレメントを言う。 そのような項目は、そのデータ記述記入項目内に OCCURS DEPENDING ON 節を持っているか、またはそのような項目に従属していなければならない。

可変位置グループ (* variably located group)
同じレコード内の可変長テーブルに続くグループ項目 (可変長テーブルに従属するわけではない)。 グループ項目は、英数字グループでも国別グループでも構いません。

可変位置項目 (* variably located item)
同じレコード内の可変長テーブルに続くデータ項目 (可変長テーブルに従属するわけではない)。

動詞 (* verb)
COBOL コンパイラーまたはオブジェクト・プログラムによってとられる処置を表すワード。

ボリューム (volume)
外部ストレージのモジュール。 テープ装置の場合はリール、直接アクセス装置の場合はユニット。

VSAM ファイル・システム (VSAM file system)
COBOL の順次編成、相対編成、および索引編成をサポートするファイル・システム。

VSAM
STL ファイル・システム または *SFS* ファイル・システム の総称。

W

Web サービス (web service)
特定のタスクを実行し、HTTP や SOAP といったオープン・プロトコルを介してアクセス可能なモジュラー・アプリケーション。

空白文字 (white space)
文書にスペースを挿入する文字。 空白文字には以下のものがある。

- スペース
- 水平タブ
- 復帰
- 改行
- 次の行

Unicode 標準では上記のように呼ばれる。

ウィンドウ表示日付フィールド (windowed date field)
ウィンドウ表示 (2 桁) 年を含む日付フィールド。 日付フィールド *(date field)* および ウィンドウ表示西暦年 *(windowed year)* も参照。

ウィンドウ表示西暦年 (windowed year)
2 桁の年だけから構成される日付フィールド。 この 2 桁の年は、世紀ウィンドウを使用して解釈できる。 例えば、10 は 2010 として解釈できる。 世紀ウィンドウ *(century window)* も参照。 拡張西暦年 *(expanded year)* と比較。

ワード (* word)
ユーザー定義語、システム名、予約語、または関数名を形成する、30 文字を超えない文字ストリング。

*** WORKING-STORAGE SECTION**
独立項目または WORKING-STORAGE レコード、あるいはその両方から構成される、WORKING-STORAGE データ項目を記述する DATA DIVISION のセクション。

ワークステーション (workstation)
コンピューターの総称 (パーソナル・コンピューター、3270 端末、インテリジェント・ワークステーション、および UNIX 端末を含む)。ワークステーションはメインフレームまたはネットワークに接続されることがよくある。

ラッパー (wrapper)
オブジェクト指向コードとプロシージャー指向コード間のインターフェースを提供するオブジェクト。 ラッパーを使用すると、他のシステムがプログラムを再利用したり、プログラムにアクセスしたりできるようになる。

X

x
PICTURE 節内の記号であり、コンピューターの有する文字セットの任意の文字を含めることができる。

XML
Extensible Markup Language。 マークアップ言語を定義するための標準メタ言語。SGML から派生した、SGML のサブセットである。 XML では、SGML の複雑で使用頻度の低い部分が省略され、文書タイプを扱うアプリケーションの作成、構造化情報の作成および管理、異種コンピューター・システム間での 構造化情報の伝送および共用がはるかに容易になっている。 XML を使用するときに、SGML で必要とされるような堅固なアプリケーションや 処理は不要である。 XML は、World Wide Web Consortium (W3C) の主導で開発された。

XML データ (XML data)
XML エレメントを持つ階層構造に編成されたデータ。 データ定義は XML エレメント・タイプ宣言で定義される。

XML 宣言 (XML declaration)
使用している XML のバージョンや文書のエンコードなど、XML 文書の特性を指定する XML テキスト。

XML 文書 (XML document)
W3C XML 仕様で定義される形式のデータ・オブジェクト。

XML 名前空間 (XML namespace)
エレメント名および属性名のコレクションのスコープを制限する、W3C XML 名前空間仕様によって定義されたメカニズム。 一意的に選択された XML 名前空間によって、複数の XML 文書または XML 文書内の複数のコンテキストでエレメント名または属性名が一意的に識別されます。

Y

年フィールド拡張 (year field expansion)
2 桁の年の日付フィールドを、ファイルおよびデータベースの中で完全な 4 桁の年になるように明示的に拡張した後、そのフィールドをプログラムの中で拡張形式で使用する方法。 これは、2 桁の年を使用していたアプリケーションに対して確実に信頼できる日付処理を行う唯一の方法である。

Z

ゾーン 10 進数データ項目 (zoned decimal data item)
暗黙的または明示的に USAGE DISPLAY として記述されており、PICTURE の記号 9、S 、P、および V の有効な組み合わせを含んでいる、外部 10 進数データ項目。 ゾーン 10 進数データ項目の内容は、文字 0 から 9 で表され、必要に応じて符号が付きます。 PICTURE ストリングが符号を指定しており、SIGN IS SEPARATE 節が指定されている場合、符号は 文字 + または - として表されます。 SIGN IS SEPARATE が指定されていない場合、符号は、符号位置の最初の 4 ビットをオーバーレイする 1 つの 16 進数字です (先行または末尾)。

#

77 レベル記述記入項目 (77-level-description-entry)
レベル番号 77 を持つ不連続データ項目を記述するデータ記述記入項目。

85 COBOL 標準 (85 COBOL Standard)
以下の標準によって定義された COBOL 言語。

- 「*ANSI INCITS 23-1985, Programming languages - COBOL*」は「*ANSI INCITS 23a-1989, Programming Languages - COBOL - Intrinsic Function Module for COBOL*」に改訂されました。

- 「*ISO 1989:1985, Programming languages - COBOL*」は 「*ISO/IEC 1989/AMD1:1992, Programming languages - COBOL: Intrinsic function module*」に改訂されました。

2002 COBOL 標準 (2002 COBOL Standard)
以下の標準によって定義された COBOL 言語。

- *INCITS/ISO/IEC 1989-2002, Information technology - Programming languages - COBOL*

2014 COBOL 標準 (2014 COBOL Standard)
以下の標準によって定義された COBOL 言語。

- *INCITS/ISO/IEC 1989:2014, Information technology - Programming languages, their environments and system software interfaces - Programming language COBOL*

資料名リスト

COBOL for Linux の資料

インストール・ガイド GC43-5391-00

言語解説書 SC43-5386-00

プログラミング・ガイド SC43-5390-00

サポート

COBOL for Linux の使用に関して問題がある場合は、IBM サポートの Web サイトを参照してください。このサイトでは最新のサポート情報が提供されています。

関連資料

DB2 for Linux、UNIX、 および Windows

以下の資料が IBM 資料にあります。

- コマンド解説書
- データベース: 管理の概念および構成リファレンス
- *Db2 V11.1 for Linux, UNIX, and Windows の SQL 解説書*

TXSeries for Multiplatforms

- *IBM TXSeries for Multiplatforms の資料*

IBM CICS TX on Cloud

- *IBM CICS TX on Cloud の資料*

Unicode および文字表現

- *Unicode*、www.unicode.org/
- *International Components for Unicode: Converter Explorer*、http://demo.icu-project.org/icu-bin/convexp/
- *Character Data Representation Architecture: Reference and Registry*、http://www-01.ibm.com/software/globalization/cdra/

XML

- *Extensible Markup Language (XML)*、www.w3.org/XML/
- *Namespaces in XML 1.0*、www.w3.org/TR/xml-names/
- *Namespaces in XML 1.1*、www.w3.org/TR/xml-names11/
- *XML specification*、www.w3.org/TR/xml/

© Copyright IBM Corp. 2021

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

[サ行]

[タ行]

[ナ行]

[ハ行]

[マ行]

[ヤ行]

[ラ行]

[ワ行]

[数字]

A

B

C

D

E

F

G

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

[特殊文字]

プログラム番号: 5737-L11

SC43-5390-00