Enterprise COBOL for z/OS

プログラミング・ガイド

バージョン 4 リリース 1

SC88-4744-00
(英文原典：SC23-8529-00)

Enterprise COBOL for z/OS

プログラミング・ガイド

バージョン 4 リリース 1

SC88-4744-00
(英文原典：SC23-8529-00)

お願い

本書および本書で紹介する製品をご使用になる前に、 899 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 Enterprise COBOL for z/OS のバージョン 4 リリース 1 (5655-S71) に適用されます。また、新版において特に断りのない限り、それ以降のすべてのリリースおよび修正レベルにも適用されます。製品のレベルに応じた正しい版を使用していることを確認してください。

出版物のご注文は、IBM 担当員または最寄りの IBM 事業所にお申し付けください。

IBM 発行のマニュアルに関する情報のページ

`http://www.ibm.com/jp/manuals/`

こちらから、日本語版および英語版のオンライン・ライブラリーをご利用いただけます。また、マニュアルに関するご意見やご感想を、上記ページよりお送りください。今後の参考にさせていただきます。

(URL は、変更になる場合があります)

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： SC23–8529–00
Enterprise COBOL for z/OS
Programming Guide
Version 4 Release 1

発　行： 日本アイ・ビー・エム株式会社

担　当： ナショナル・ランゲージ・サポート

第1版第1刷 2008.3

目次

表

前書き

本書について

IBM の最新のホスト COBOL コンパイラーである IBM® Enterprise COBOL for z/OS® について。

IBM COBOL のこのバージョンには、COBOL ビジネス・プロセスと Web 主導型ビジネス・プロセスを以下の方法で統合するための新たな COBOL 機能が追加されています。

- COBOL プログラムのコンポーネント化を単純にし、Java™ コンポーネントとの相互運用を可能にする
- 標準化された形式 (XML や Unicode など) のデータの交換および使用を促進する

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。 z/OS のアクセシビリティー機能は、Enterprise COBOL のアクセシビリティーを提供します。

z/OS の主要アクセシビリティー機能により、ユーザーは以下のことを行うことができます。

- スクリーン・リーダーおよびスクリーン拡大ソフトウェアなどの支援テクノロジー製品の使用
- キーボードのみを使用する、特定の機能または同等の機能の操作
- 色、コントラスト、フォント・サイズなどの表示属性のカスタマイズ

支援機能の使用

支援技術製品は、z/OS のユーザー・インターフェースと連動します。特定のガイダンス情報については、z/OS インターフェースへのアクセスに使用する支援技術製品の資料を参照してください。

ユーザー・インターフェースのキーボード・ナビゲーション

ユーザーは、TSO/E または ISPF を使用して z/OS ユーザー・インターフェースにアクセスできます。 TSO/E または ISPF インターフェースへのアクセスついて詳しくは、以下の資料を参照してください。

- *z/OS TSO/E 入門*
- *z/OS TSO/E ユーザーズ・ガイド*
- *z/OS ISPF ユーザーズ・ガイド 第 1 巻*

上記の資料には、キーボード・ショートカットまたはファンクション・キー (PF キー) の使用方法を含む TSO/E および ISPF の使用方法が記載されています。それぞれの資料では、PF キーのデフォルトの設定値とそれらの機能の変更方法についても説明しています。

本書のアクセシビリティー

IBM System z Enterprise Development Tools & Compilers Information Center (publib.boulder.ibm.com/infocenter/pdthelp/index.jsp) で提供される英語 XHTML フォーマットのこの文書は、スクリーン・リーダーを使用する視覚障害者の方がご利用になれます。

スクリーン・リーダーが、構文図やソース・コード例、ピリオドまたはコンマの PICTURE 記号を含む本文を正しく読み上げるようにするためには、スクリーン・リーダーを、すべての句読点を読み上げるように設定する必要があります。

本書の使い方

本書は、Enterprise COBOL プログラムの作成とコンパイルに役立ちます。さらに、オブジェクト指向クラスおよびメソッドの定義、メソッドの呼び出し、およびプログラムでのオブジェクトの参照を行うのに役立ちます。

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 Enterprise 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、「*IBM Enterprise COBOL 言語解説書*」を参照してください。

Enterprise COBOL へのプログラムの移行については、「*IBM Enterprise COBOL コンパイラーおよびランタイム 移行ガイド*」を参照してください。

Language Environment® により、Enterprise COBOL プログラムの実行に必要なランタイム環境とランタイム・サービスが提供されます。プログラムのリンク・エディットと実行については、「*IBM z/OS 言語環境プログラム・プログラミング・ガイド*」および「*IBM z/OS 言語環境プログラム・プログラミング・リファレンス*」を参照してください。

よく使用される Enterprise COBOL 用語と IBM z/OS 言語環境プログラム用語の比較については、xvii ページの『一般に使用される用語の比較』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。最もよく使用される製品名の省略形を次の表に示します。

使用される用語	長形式
CICS®	CICS Transaction Server
Enterprise COBOL	IBM Enterprise COBOL for z/OS
言語環境プログラム	IBM z/OS 言語環境プログラム

使用される用語	長形式
MVS™	MVS/ESA™
z/OS UNIX®	z/OS UNIX システム・サービス

これらの略語のほかに、「標準 COBOL 85」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 - 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

ISO 規格は、米国標準規格と一致しています。

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、巻末の用語集にリストされています。

一般に使用される用語の比較

IBM z/OS 言語環境プログラムおよび IBM Enterprise COBOL for z/OS の資料で使用される用語、およびどの用語が同等であるかを理解するためには、下の表を参照してください。

言語環境プログラムの用語	対応する Enterprise COBOL の用語
集合体	グループ項目
配列	OCCURS 節を使用して作成されたテーブル
配列エレメント	テーブル・エレメント
エンクレーブ	実行単位
EXTERNAL データ	EXTERNAL 節を使用して定義される WORKING-STORAGE データ
ローカル・データ	非 EXTERNAL データ項目
値によるパラメーターの直接受け渡し	BY VALUE
参照によるパラメーターの間接受け渡し	BY REFERENCE
値によるパラメーターの間接受け渡し	BY CONTENT
ルーチン	プログラム
スカラー	基本項目

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。

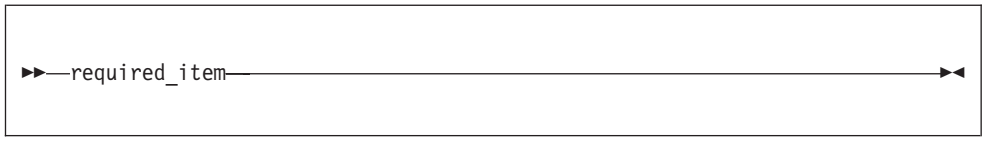
  >>--- 記号は、構文図の開始を示します。

  ---> 記号は、構文図が次の行に続くことを示します。

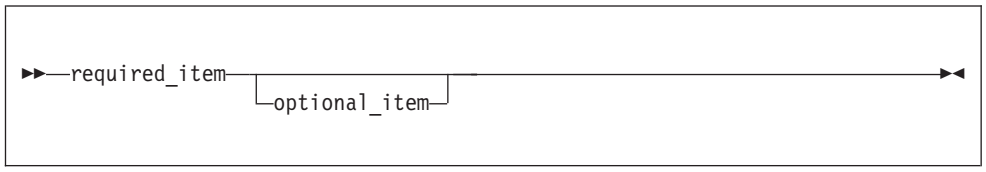
  >--- 記号は、構文図が前の行から続いていることを示します。

  --->< 記号は、構文図の終わりを示します。

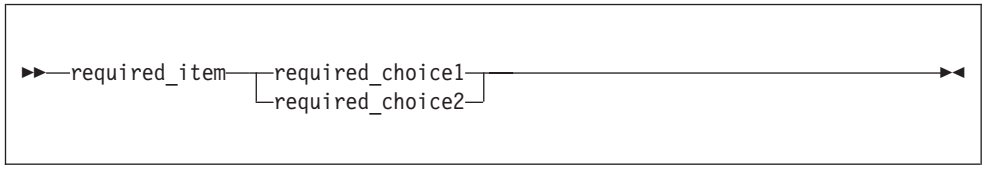
  完全なステートメントではない構文単位の図は、**>---** 記号で始まり、**--->** 記号で終わります。
- 必須項目は、水平線 (メインパス) と同じ高さに示されます。



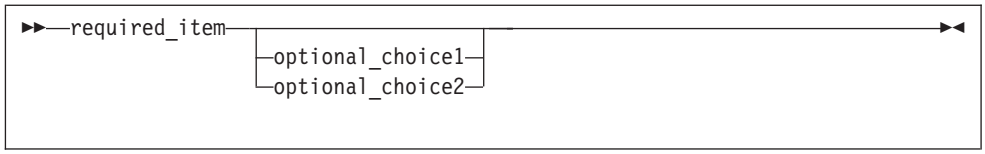
- オプション項目は、メインパスの下側に示されます。



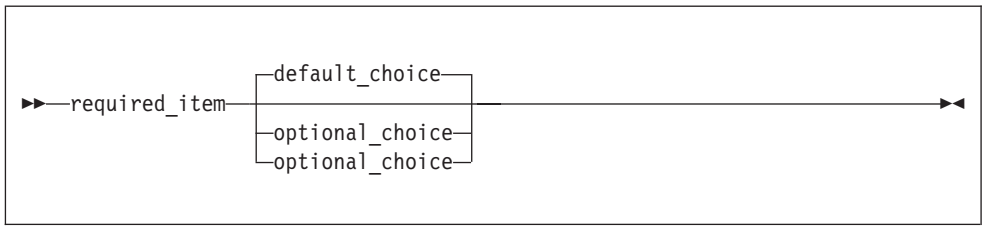
- 複数の項目から選択できる場合には、それらの項目は縦方向に重ねて示されます。それらの項目のうち 1 つを選択しなければならない場合には、スタックのうちの 1 つの項目がメインパス上に示されます。



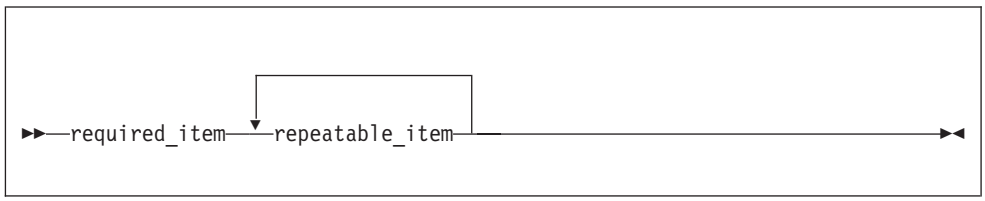
  項目の選択が任意である場合には、スタック全体がメインパスより下に置かれます。



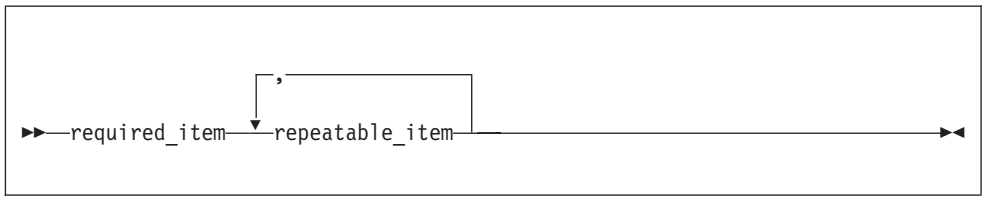
項目のうちの 1 つがデフォルトであれば、それがメインパスより上に示され、残りの選択項目はメインパスより下に示されます。



- メインパス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)。キーワードは、示されているとおりに入力しなければなりません。変数は小文字のイタリックで示されます (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。

ソフトコピー文書およびサポート情報へのアクセス

IBM Enterprise COBOL for z/OS は、ライブラリーの PDF 版および BookManager® 版を製品サイト (www.ibm.com/software/awdtools/cobol/zos/library/) で提供しています。

文書の最新版については、その Web サイトで確認することができます。資料の BookManager 版では、表示技術の違いのため、一部の表や構文図の内容が正しく整列しないことがあります。

サポート情報は、製品サイト (www.ibm.com/software/awdtools/cobol/zos/support/) でも入手できます。

改訂の要約

ここでは、 バージョン 4 で Enterprise COBOL に加えられた主要な変更を示します。本書で解説されている変更には、読者の便宜のため、参照ページが記載されています。 最新の技術上の変更は、PDF 版および BookManager 版の左マージンに縦線 (|) を付けて示してあります。

バージョン 4 リリース 1 (2007 年 12 月)

- Unicode (USAGE NATIONAL) データの操作のパフォーマンスが大幅に向上しました。コンパイラーが、ほとんどの Unicode MOVE 操作および比較で、z/Architecture® ハードウェア命令を生成するようになりました。
- 新しいコンパイラー・オプション (XMLPARSE) によって、 COBOL ライブラリーで使用可能なパーサー (Enterprise COBOL バージョン 3 互換用) または z/OS XML System Services パーサー (401 ページの『XMLPARSE』) のどちらかを選択して構文解析できるようになりました。
- z/OS XML System Services パーサーで文書を構文解析する際に、新しい XML PARSE 機能が使用可能になりました (561 ページの『第 28 章 XML 入力の処理』)。
  - 新しい特殊レジスターおよび新しい XML イベントを使用して名前空間および名前空間接頭部が処理されます。
  - XML PARSE ステートメントの ENCODING 句を使用して文書エンコードを指定できます。

  - Unicode UTF-8 でエンコードされた文書を構文解析できます (585 ページの『UTF-8 でエンコードされた XML 文書の構文解析』)。
  - RETURNING NATIONAL 句を使用して、XML 文書の元のエンコードに関係なく、Unicode で XML 文書フラグメントを受け取ることができます。
  - データ・セット内にある文書を構文解析したり、非常に大きな文書をバッファーごとに構文解析したりできます (572 ページの『XML 文書を 1 セグメントずつ構文解析』)。
- XML GENERATE ステートメントが強化されました (593 ページの『第 29 章 XML 出力の生成』):
  - NAMESPACE 句を使用して名前空間を、NAMESPACE-PREFIX 句を使用して各エレメントに適用される名前空間接頭部を指定できます。
  - ENCODING 句を使用して、生成された文書のコード・ページを指定できます (597 ページの『生成される XML 出力のエンコードの制御』)。
  - XML 文書が、UTF-16、またはさまざまな EBCDIC コード・ページに加えて UTF-8 で生成できるようになりました。
  - WITH ATTRIBUTES 句を使用すると、適格な基本項目が、生成された XML 内で子エレメントとしてではなく属性として表されます。
  - WITH XML-DECLARATION 句を使用すると、XML 宣言が生成されます。
- 新しいコンパイラー・オプション OPTFILE は、データ・セット内からの COBOL コンパイラー・オプションの指定を可能にします (377 ページの『OPTFILE』)。
- コンパイラー・リストが、COPY ステートメント、およびコピーブックが取得されるデータ・セットを相互参照するようになりました (446 ページの『例: XREF 出力: COPY/BASIS 相互参照』)。
- 統合 DB2 コプロセッサー (SQL コンパイラー・オプション) の使用時に、次のような DB2® for z/OS V9 の新しい機能のサポートが有効になりました (467 ページの『DB2 coprocessor』)。
  - 新しい SQL データ・タイプ (XML タイプ、BINARY、VARBINARY、BIGINT) およびファイル参照変数がサポートされました。
  - ラージ・オブジェクト操作の機能拡張である、XML 操作の新しい SQL 構文 MERGE および SELECT FROM MERGE がサポートされました。
  - DB2 処理オプション STDSQL(YES|NO)、NOFOR、および SQL(ALL|DB2) が、SQL コンパイラー・オプションのサブオプションとしてサポートされました (472 ページの『SQL オプションを使用したコンパイル』)。
- 統合 DB2 コプロセッサーの使用時に、次のような COBOL-DB2 アプリケーションに対するいくつかのユーザビリティー機能拡張が使用可能になりました。
  - コンパイラー・リストが機能拡張され、有効になっている DB2 オプション (DB2 for z/OS V9 を使用している場合) を表示するとともに、SQLCA および SQLDA 制御ブロックの拡張を表示するようになりました。
  - コンパイラーをアセンブラー言語プログラムから呼び出す際に、DBRMLIB の代替 DD 名を指定できるようになりました (296 ページの『アセンブラー・プログラムからコンパイラーを開始する』)。
  - LOCAL-STORAGE SECTION または WORKING-STORAGE SECTION を明示的にコーディングする必要がなくなりました。

- デバッグが機能拡張され、デバッグ・ツール V8 をサポートするようになりました。 TEST コンパイラー・オプションの新しいサブオプション EJPD を使用すれば、実動デバッグで デバッグ・ツール コマンド JUMPTO および GOTO が使用可能です。 TEST コンパイラー・オプションは単純化されて、サブオプションが再構成されました。 (392 ページの『TEST』)。

第 1 部 プログラムのコーディング

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。

COBOL クラスまたはメソッドを定義するには、プログラムの場合とは違った方法でいくつかの部を定義することが必要です。

関連タスク
『プログラムの識別』
7 ページの『コンピューター環境の記述』
14 ページの『データの記述』
21 ページの『データの処理』
617 ページの『クラスの定義』
622 ページの『クラス・インスタンス・メソッドの定義』
659 ページの『OO アプリケーションの構造化』

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.  12/21/2007.
Date-Compiled. 12/30/2007.
```

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。
- NAME コンパイラー・オプションを使用した場合は、コンパイルによって作成されるオブジェクト・モジュールを識別するために、その名前が NAME リンケージ・エディターまたはバインダーの制御ステートメントに入れられます。

  ヒント: IBM 製品が使用している接頭部で始まるプログラム名は使用しないでください。名前が以下の接頭部のいずれかで始まるプログラムを使用すると、CALL

ステートメントは意図されたプログラムではなく、IBM ライブラリーまたはコンパイラー・ルーチンを呼び出してしまう可能性があります。

- AFB
- AFH
- CBC
- CEE
- IBM
- IFY
- IGY
- IGZ
- ILB

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
7 ページの『ソース・リストのヘッダーの変更』
『プログラムを再帰的として識別する』
『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』
7 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 (*Enterprise COBOL 言語解説書*)
プログラム名の規則 (*Enterprise COBOL 言語解説書*)

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。THREAD オプションを指定してコンパイルしたプログラムには、RECURSIVE をコーディングする必要があります。

関連タスク
20 ページの『再帰的またはマルチスレッド化されたプログラムでのデータの共用』
515 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
512 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、INITIAL 属性を使用します。

プログラムが初期状態になるのは、以下のとおりです。

- 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
500 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』
503 ページの『静的呼び出しの作成』
504 ページの『動的呼び出しの作成』

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 5655-S71 IBM Enterprise COBOL for z/OS  4.1.0       Date  12/30/2007  Time 15:05:19   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 (*Enterprise COBOL 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、物理的に存在している外部 QSAM、VSAM、または HFS (階層ファイル・システム) データ・セットに関連付ける。

  COBOL 用語のファイルという用語と、オペレーティング・システム (OS) 用語のデータ・セットまたは *HFS ファイル*という用語は、本質的に同じ意味を持ち、ここでの情報では区別なく使用されます。

  顧客情報管理システム (CICS) およびオンライン情報管理システム (IMS™) のメッセージ処理プログラム (MPP) については、ENVIRONMENT DIVISION ヘッダーと、オプションとして、CONFIGURATION SECTION だけをコーディングしてください。 CICS では、COBOL によるファイルの定義は許可されません。IMS では、バッチ・プログラムについてのみ、COBOL によるファイルの定義が許可されます。
- プログラムと外部メディア間でのデータ・レコードの効率的な伝送を制御するために情報を提供する。

『例: FILE-CONTROL の記入項目』

関連タスク

9 ページの『照合シーケンスの指定』
10 ページの『シンボリック文字を定義する』
11 ページの『ユーザー定義のクラスを定義する』
11 ページの『オペレーティング・システムに対してファイルを定義する』

関連参照

セクションおよび段落 (*Enterprise COBOL 言語解説書*)

例: FILE-CONTROL の記入項目

次の表に示す FILE-CONTROL 記入項目の例は、QSAM 順次ファイル、VSAM 索引付きファイル、および行順次ファイル用です。

表 1. **FILE-CONTROL 記入項目**

QSAM ファイル	VSAM ファイル	行順次ファイル
SELECT PRINTFILE[1] ASSIGN TO UPDPRINT[2] ORGANIZATION IS SEQUENTIAL[3] ACCESS IS SEQUENTIAL.[4]	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-VSAM-STATUS.	SELECT PRINTFILE[1] ASSIGN TO UPDPRINT[2] ORGANIZATION IS LINE SEQUENTIAL[3] ACCESS IS SEQUENTIAL.[4]

表 1. **FILE-CONTROL 記入項目** *(続き)*

QSAM ファイル	VSAM ファイル	行順次ファイル
1. SELECT 節は、外部データ・セットと関連付けられる、COBOL プログラム内のファイルを選択します。 2. ASSIGN 節は、プログラムのファイル名を実際のデータ・ファイルの外部名と関連付けます。外部名は、DD ステートメントまたは環境変数を使用して定義することができます。 3. ORGANIZATION 節は、ファイルの編成を記述します。QSAM ファイルの場合、ORGANIZATION 節は任意指定です。 4. ACCESS MODE 節は、レコードを処理する方式 (順次、ランダム、または動的) を定義します。QSAM ファイルおよび行順次ファイルの場合、ACCESS MODE 節はオプションです。これらのファイルの編成は常に順次です。 5. VSAM ファイルの場合、使用する VSAM ファイルのタイプによって、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
167 ページの『第 9 章 QSAM ファイルの処理』
199 ページの『第 10 章 VSAM ファイルの処理』
231 ページの『第 11 章 line-sequential ファイルの処理』
7 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幾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)

10 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: EBCDIC と同じ照合シーケンスを参照します。
- 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 EBCDIC シーケンスの代替。

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
250 ページの『代替照合シーケンスの選択』
152 ページの『国別 (UTF-16) データの比較』

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 EBCDIC 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.
   Source-Computer. IBM-390.
   Object-Computer. IBM-390.
      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
9 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。序数位置を使用して、文字を識別します。位置 1 は、文字 X'00' に対応します。

例えば、バックスペース文字 (EBCDIC アルファベットでは X'16') 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 BACKSPACE IS 23
```

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。)

オペレーティング・システムに対してファイルを定義する

COBOL プログラムで処理するすべてのファイルについて、適切なシステム・データ定義を使用し、ファイルをオペレーティング・システム (OS) に対して定義する必要があります。

オペレーティング・システムに応じて、このシステム・データ定義の形式は以下のいずれかになります。

- MVS JCL の場合は DD ステートメント。
- TSO のもとでは、ALLOCATE コマンド。
- z/OS または UNIX 用の環境変数。この内容は、MVS データ・セットまたは HFS (階層ファイル・システム) 内のファイルのいずれかを定義することができます。

以下の例は、FILE-CONTROL 項目と、システム・データ定義および FILE SECTION 内の FD 項目との関係を示しています。

- JCL DD ステートメント:

```
    (1)
//OUTFILE  DD  DSNAME=MY.OUT171,UNIT=SYSDA,SPACE=(TRK,(50,5))
/*
```

- Environment variable (export command):

```
          (1)
export OUTFILE=DSN(MY.OUT171),UNIT(SYSDA),SPACE(TRK,(50,5))
```

- COBOL コード:

```
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT CARPOOL
      ASSIGN TO OUTFILE (1)
      ORGANIZATION IS SEQUENTIAL.
. . .
DATA DIVISION.
FILE SECTION.
FD CARPOOL       (2)
    LABEL  RECORD STANDARD
    BLOCK  CONTAINS 0 CHARACTERS
    RECORD CONTAINS 80 CHARACTERS
```

(1) ASSIGN 節の *assignment-name* は、DD ステートメントの *ddname* OUTFILE、または export コマンドの環境変数 OUTFILE を指します。

- //OUTFILE DD DSNAME=OUT171 . . .、または
- export OUTFILE= . . .

(2) FILE-CONTROL 記入項目にファイル *file-name* を指定する場合は、次のように、そのファイルを FD 記入項目で記述しなければなりません。

```
SELECT CARPOOL
. . .
FD CARPOOL
```

関連タスク
13 ページの『バッファーおよび装置スペースの最適化』

関連参照
15 ページの『FILE SECTION 記入項目』
file section (*Enterprise COBOL 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、ファイルの名前を実行時に異なる実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 DSNAME 値または DSN または export の PATH 値を変更して、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。

ASSIGN 節の *assignment-name* で使用する名前は、DD ステートメントの DD 名または export コマンドの環境変数と同じでなければなりません。

SELECT 節で使用する *file-name* (SELECT MASTER など) は、FD *file-name* 記入項目と同じでなければなりません。

2 つのファイルが SELECT 節でそれぞれ同じ ddname または環境変数名を使用することはできません。もし使用した場合、結果は予測できません。例えば、DISPLAY の出力先が SYSOUT になっている場合、ファイルに対する SELECT 節で DD 名または環境変数として SYSOUT を使用しないでください。

『例: さまざまな入力ファイルの使用』

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 DD ステートメントまたは export コマンドをコーディングすることで、同じ COBOL プログラムからさまざまなファイルにアクセスすることを示します。

次の SELECT 節を含む COBOL プログラムを考えてみます。

```
SELECT MASTER ASSIGN TO DA-3330-S-MASTERA
```

3 つのファイル、MASTER1、MASTER2、および MASTER3 が入力ファイルとなる可能性があるとします。プログラムを実行する前に、プログラム実行を要求するジョブ・ステップに、次の DD ステートメントの 1 つをコーディングするか、または、プログラムを実行する同じシェルから、次の export コマンドの 1 つを発行します。

```
//MASTERA     DD   DSNAME=MY.MASTER1,. . .
export MASTERA=DSN(MY.MASTER1),. . .

//MASTERA     DD   DSNAME=MY.MASTER2,. . .
export MASTERA=DSN(MY.MASTER2),. . .

//MASTERA     DD   DSNAME=MY.MASTER3,. . .
export MASTERA=DSN(MY.MASTER3),. . .
```

したがって、プログラムで MASTER を参照すると、それは現在 DD 名または環境変数名 MASTERA に割り当てられているファイルへの参照になります。

この例では、PATH(*path*) 形式の export コマンドを使用して HFS の行順次ファイルを参照できないことに注意してください。行順次ファイルでは編成フィールド (S- または AS-) を指定することができないためです。

バッファーおよび装置スペースの最適化

APPLY WRITE-ONLY 節を使用すると、ブロック化可変長レコードの順次ファイルを作成する際に、バッファーおよび装置スペースを最適に使用することができます。

APPLY WRITE-ONLY を指定すると、バッファーは、次のレコードがバッファーの未使用部分に収まらない場合のみ切り捨てられます。APPLY WRITE-ONLY を指定しない場合は、最大サイズのレコードを収容できる十分なスペースが残っていないと、バッファーは切り捨てられます。

APPLY WRITE-ONLY 節が意味を持つのは、ブロック化可変長レコードの順次ファイルの場合だけです。

AWO コンパイラー・オプションは、すべての適格ファイルに暗黙の APPLY WRITE-ONLY 節を適用します。NOAWO コンパイラー・オプションが指定されると、ファイルに APPLY WRITE-ONLY 節が指定されていても、効力はありません。APPLY WRITE-ONLY 節の方が、NOAWO コンパイラー・オプションに優先します。

APPLY-WRITE ONLY 節を使用すると、入力ファイルがバッファー内のデータを処理せずに、レコード域を使用することがあります。これは、入力ファイルと出力ファイルの両方の処理に影響を与える可能性があります。

関連参照
347 ページの『AWO』

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 のセクションの 1 つ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)

Enterprise COBOL 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。

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

関連タスク
『入出力操作でのデータの使用』
19 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 (*Enterprise COBOL 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のフィールドを記述します。
  - レコード全体をレベル 01 記述でコーディングし、次に WORKING-STORAGE SECTION で、レコードのフィールドをより詳しく記述する作業用コピーをコーディングすることができます。レコードを WORKING-STORAGE に入れるには、READ-INTO ステートメントを使用します。処理は、WORKING-STORAGE のデータのコピーに対して行われます。処理されたデータは、WRITE FROM ステートメントによって、FILE SECTION で定義されたレコード域に書き込まれます。
  - 設定されるレコード名は、WRITE および REWRITE ステートメントの対象となります。

- QSAM ファイルの場合のみ、RECORDING MODE 節でレコード形式を設定することができます。RECORDING MODE 節を省くと、コンパイラーは RECORD 節およびレベル 01 レコード記述に基づいてレコード形式を判別します。
- QSAM ファイルの場合は、BLOCK CONTAINS 節でファイルについてのブロック化因数を設定することができます。BLOCK CONTAINS 節を省略すると、デフォルトとしてファイルはブロック化されません。しかし、z/OS のデータ管理機能 (DD ファイル・ジョブ制御ステートメントを含む) で、これをオーバーライドできます。
- 行順次ファイルの場合は、BLOCK CONTAINS 節でファイルについてのブロック化因数を設定することができます。BLOCK CONTAINS 1 RECORDS、または BLOCK CONTAINS *n* CHARACTERS (ここで、*n* は 1 つの論理レコードの長さ (バイト単位) です) をコーディングすると、 WRITE ステートメントにより、レコードは、バッファーに入れられずに、即時にファイルに転送されます。この手法は、各レコードを直ちに (エラー・ログなどに) 書き込みたい場合に有用です。

同じ実行単位内のプログラムは、共通ファイルを共用したり、共通ファイルにアクセスすることができます。これを行う方法は、プログラムがネストされた構造 (含まれている構造) の一部であるか、個別にコンパイルされる (バッチ・シーケンスの一部としてコンパイルされるプログラムを含む) かによって異なります。

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。

関連概念
512 ページの『ネストされたプログラム』

関連タスク
532 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
『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 2. **FILE SECTION 記入項目**

節	定義対象	注
FD	PROCEDURE DIVISION の入出力ステートメント (OPEN、CLOSE、READ、さらに、VSAM の場合は 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* を介して *ddname* と関連付けられます。
BLOCK CONTAINS	物理レコードのサイズ	CHARACTERS 句が指定された場合、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。 QSAM: 指定する場合は、JCL またはデータ・セット・ラベルの情報と一致していなければなりません。BLOCK CONTAINS 0 と指定した場合、または指定を行わなかった場合、ユーザーに代わってシステムが最適ブロック・サイズを決定します。 行順次: WRITE ステートメントのバッファリングを制御するために指定できます。 VSAM: 構文が検査されますが、実行に影響はありません。
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。この節を指定する場合、これは JCL またはデータ・セット・ラベルの情報と一致している必要があります。*n* が 0 である場合には、JCL またはデータ・セット・ラベルに LRECL をコーディングしなければなりません。
RECORD IS VARYING	論理レコード (可変長) のサイズ	整数サイズ (指定した場合) 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。この節を指定する場合、これは JCL またはデータ・セット・ラベルの情報と一致している必要があります。コンパイラーはレコード記述が一致しているか検査します。
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。この節を指定する場合、これは JCL またはデータ・セット・ラベルの情報と一致している必要があります。コンパイラーはレコード記述が一致しているか検査します。
LABEL RECORDS	QSAM ファイルのラベル	VSAM: コメントとして処理されます
STANDARD	ラベルが存在する	QSAM: コメントとして処理されます
OMITTED	ラベルが存在しない	QSAM: コメントとして処理されます

表 2. **FILE SECTION 記入項目** *(続き)*

節	定義対象	注
data-name	ユーザーによって定義されたラベル	QSAM: (オプションの) テープまたはディスクに許可されます
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
LINAGE	論理ページの深さ	QSAM のみ
CODE-SET	ASCII または EBCDIC ファイル	QSAM のみ。 ASCII ファイルが CODE-SET 節で識別されたとき、そのファイルが VS COBOL II、COBOL for OS/390® & VM、または IBM Enterprise COBOL for z/OS を使用して作成されたものでない場合は、対応する DD ステートメントで DCB=(OPTCD=Q. . .) または DCB=(RECFM=D. . .) をコーディングしなければならないことがあります。
RECORDING MODE	物理レコード記述	QSAM のみ

関連概念
194 ページの『QSAM ファイルのラベル』

関連参照
File section (*Enterprise COBOL 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 WORKING-STORAGE は実行単位の開始時に割り振られます。

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 のクラス定義における WORKING-STORAGE については、関連する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またはメソッドの個々の呼び出しに対して割り振られ、プログラムまたはメソッド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、起動または呼び出しのたびに、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。

スレッド化: 複数のスレッドで同時に実行されるプログラムは、それぞれの起動先が WORKING-STORAGE データの単一コピーへのアクセスを共用します。 LOCAL-STORAGE データは起動先ごとに個別のコピーを使用します。

『例: ストレージ・セクション』

関連タスク 500 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』
549 ページの『第 27 章 マルチスレッド化のための COBOL プログラムの準備』
620 ページの『クラス・インスタンス・データ定義用の WORKING-STORAGE SECTION』

関連参照 Working-storage section (*Enterprise COBOL 言語解説書*) Local-storage section (*Enterprise COBOL 言語解説書*)

例: ストレージ・セクション

以下に、WORKING-STORAGE と LOCAL-STORAGE の両方を使用する再帰的プログラムの例を示します。

```
CBL 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
17 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。別々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
20 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』
20 ページの『ネストされたプログラムでのデータの共用』
20 ページの『再帰的またはマルチスレッド化されたプログラムでのデータの共用』
521 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 または INVOKE . . . USING ステートメントを使用してデータを渡してください。

関連タスク
521 ページの『データの受け渡し』

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
512 ページの『ネストされたプログラム』

再帰的またはマルチスレッド化された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、またはプログラムを THREAD コンパイラー・オプションを指定してコンパイルする場合、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、またはプログラムを THREAD コンパイラー・オプションを指定してコンパイルする場合、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。

関連概念
550 ページの『マルチスレッド化』

関連タスク
515 ページの『再帰呼び出しの実行』
553 ページの『マルチスレッド化によるファイルの処理』

関連参照
396 ページの『THREAD』
SET ステートメント (*Enterprise COBOL 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
22 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
743 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 (*Enterprise COBOL 言語解説書*)
USING 句 (*Enterprise COBOL 言語解説書*)
CALL ステートメント (*Enterprise COBOL 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 動詞で始まります。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
24 ページの『コンパイラー指示ステートメント』
24 ページの『範囲終了符号』
『命令ステートメント』
23 ページの『条件ステートメント』
25 ページの『宣言』

関連参照
PROCEDURE DIVISION の構成 (*Enterprise COBOL 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、INVOKE、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
『条件ステートメント』
24 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
22 ページの『命令ステートメント』
『範囲終了符号』

関連タスク
97 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 (*Enterprise COBOL 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、または制御フローについて特定のアクションを行わせます。

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 407 ページの『第 18 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 (*Enterprise COBOL 言語解説書*)

範囲終了符号

範囲終了符号は動詞またはステートメントの終了を示します。範囲終了符号には、明示的なものと暗黙的なものがあります。

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

次のプログラムの断片における 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、次のプログラム断片で、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

23 ページの『条件ステートメント』
22 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
413 ページの『入出力エラーの検出および処理』

関連参照
宣言 (*Enterprise COBOL 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。

関連概念
45 ページの『ストレージとそのアドレス可能度』

関連タスク
『変数、構造、リテラル、および定数の使用』
31 ページの『データ項目への値の割り当て』
41 ページの『画面上またはファイル内での値の表示 (DISPLAY)』
43 ページの『組み込み関数の使用 (組み込み関数)』
44 ページの『テーブル (配列) とポインターの使用』
133 ページの『第 7 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`、`USAGE PROCEDURE-POINTER`、または `USAGE OBJECT REFERENCE` として記述されたデータ項目を定義することもできます。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
『変数の使用』
28 ページの『データ項目とグループ項目の使用』
30 ページの『リテラルの使用』
30 ページの『定数の使用』
31 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 (*Enterprise COBOL 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、データ項目の名前と長さを指定す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宣言できます。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
138 ページの『Unicode および言語文字のエンコード』

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

関連参照
373 ページの『NSYMBOL』
146 ページの『国別データのストレージ』
PICTURE 節 (*Enterprise COBOL 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

138 ページの『Unicode および言語文字のエンコード』
142 ページの『国別グループ』

関連タスク

139 ページの『COBOL での国別データ (Unicode) の使用』
144 ページの『国別グループの使用』

関連参照

15 ページの『FILE SECTION 記入項目』
146 ページの『国別データのストレージ』
グループ項目のクラスおよびカテゴリー (*Enterprise COBOL 言語解説書*)

PICTURE 節 (*Enterprise COBOL 言語解説書*)
MOVE ステートメント (*Enterprise COBOL 言語解説書*)
USAGE 節 (*Enterprise COBOL 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。

値のデータ項目を宣言したり、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"
```

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
138 ページの『Unicode および言語文字のエンコード』

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

関連参照
373 ページの『NSYMBOL』
リテラル (*Enterprise COBOL 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
139 ページの『COBOL での国別データ (Unicode) の使用』
155 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。

以下に例を示します。

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

関連タスク
141 ページの『国別文字形象定数の使用』
155 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 (*Enterprise COBOL 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 3. **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する。
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。

表 3. **プログラム内でのデータ項目の割り当て** *(続き)*

目的	方法
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 • OBJECT REFERENCE データ項目にオブジェクト・インスタンスを関連付ける	SET ステートメントを使用する。

『例: データ項目の初期化』

関連タスク

35 ページの『構造の初期化 (INITIALIZE)』
37 ページの『基本データ項目への値の割り当て (MOVE)』
38 ページの『グループ・データ項目への値の割り当て (MOVE)』
40 ページの『画面またはファイルからの入力の割り当て (ACCEPT)』
111 ページの『データ項目の結合 (STRING)』
114 ページの『データ項目の分割 (UNSTRING)』
39 ページの『算術結果の割り当て (MOVE または COMPUTE)』
122 ページの『データ項目の計算および置換 (INSPECT)』
133 ページの『第 7 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]

identifier-1 **PICTURE**	*identifier-1* **(初期化前)**	*identifier-1* **(初期化後)**
N(3)	004100420031[2]	002000200020[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 (「4040」) 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
```

NATIONAL-3	NATIONAL-1 (初期化前)	NATIONAL-1 (初期化後)
00430044[1]	00410042[2]	00430044[1]
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)
003400350036[1]	003100320033[2]	003000340035[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	003900380037[1]	003000310032[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 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 DATA BY NUM-NATL-3
```

NUM-NATL-3	NUM-EDIT-NATL-1 (初期化前)	NUM-EDIT-NATL-1 (初期化後)
003000310032[1]	0024003100320037[2]	0024002000310032[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

『構造の初期化 (INITIALIZE)』

82 ページの『テーブルの初期化 (INITIALIZE)』

49 ページの『数値データの定義』

関連参照

373 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	C00138300000000425000000000	*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"0020" (国別スペース) が入ります。
- TRANSACTION-OUT の残りの 27 の国別文字の位置には、NX"0030" (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
142 ページの『国別グループ』

関連タスク
82 ページの『テーブルの初期化 (INITIALIZE)』
144 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 (*Enterprise COBOL 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。

```
CBL CODEPAGE(00875)
. . .
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
138 ページの『Unicode および言語文字のエンコード』

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

関連参照 350 ページの『CODEPAGE』
データのクラスおよびカテゴリー (*Enterprise COBOL 言語解説書*)
MOVE ステートメント (*Enterprise COBOL 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

138 ページの『Unicode および言語文字のエンコード』
142 ページの『国別グループ』

関連タスク
37 ページの『基本データ項目への値の割り当て (MOVE)』
144 ページの『国別グループの使用』
147 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 (*Enterprise COBOL 言語解説書*)
MOVE ステートメント (*Enterprise COBOL 言語解説書*)
CORRESPONDING 句 (*Enterprise COBOL 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。以下に例を示します。

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

言語環境プログラムの呼び出し可能サービスを使用すれば、日付、時刻、数値その他の計算の結果をデータ項目に割り当てることができます。言語環境プログラムのサービスは、標準の COBOL CALL ステートメントを介して使用することができ、それらが戻す値は CALL ステートメントのパラメーターに入れて渡されます。例えば、次のステートメントをコーディングして、データ項目の絶対値を見つけるために言語環境プログラムのサービス CEESIABS を呼び出すことができます。

```
Call 'CEESIABS' Using Arg, Feedback-code, Result.
```

この呼び出しの結果、データ項目 Result には、データ項目 Arg の値の絶対値が割り当てられます。データ項目 Feedback-code には、サービスが正常に完了したかどうかを示す戻りコードが入ります。特定の呼び出し可能サービスの要件に従って、正しい記述を使用してすべてのデータ項目を DATA DIVISION で定義しなければなりません。上記の例の場合は、データ項目を次のように定義することができます。

```
77 Arg                Pic s9(9)  Binary.
77 Feedback-code      Pic x(12)  Display.
77 Result             Pic s9(9)  Binary.
```

関連参照
358 ページの『DIAGTRUNC』
組み込み関数 (*Enterprise COBOL 言語解説書*)
言語環境プログラム・プログラミング・リファレンス (呼び出し可能サービス)

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。以下に例を示します。

```
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)。以下に、その例を示します。

  ```
  SYSIN is Names-Input
  ```

  device は、階層ファイル・システム (HFS) のパスを参照する DD 名にすることもできます。この DD 名が定義されていない場合、プログラムが z/OS UNIX 環境で実行されていれば、stdin が入力ソースになります。この DD 名 が定義されておらず、プログラムが z/OS UNIX 環境で実行されていない場合には、ACCEPT ステートメントは失敗します。

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、コンソールからの入力データは CODEPAGE コンパイラー・オプションで指定された EBCDIC コード・ページから国別 (Unicode UTF-16) 表現に変換されます。 ACCEPT ステートメントを使用したときに、国別データの変換が行われるのは、この場合のみです。この場合に変換が行われるのは、画面から入力データがくることが認識されているからです。

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
138 ページの『Unicode および言語文字のエンコード』

関連タスク
148 ページの『英数字または DBCS から国別への変換 (NATIONAL-OF)』

関連参照 350 ページの『CODEPAGE』
ACCEPT ステートメント (*Enterprise COBOL 言語解説書*)
SPECIAL-NAMES 段落 (*Enterprise COBOL 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、次のメッセージをシステム論理出力装置に表示します。

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

データをシステム論理出力装置以外の宛先に書き込みたい場合には、SYSOUT 以外の宛先を指定した UPON 句を使用してください。例えば、次のステートメントは、SYSPUNCH DD ステートメントで指定されたファイルに書き込みを行います。

```
Display "Hello" upon syspunch.
```

SYSPUNCH DD ステートメントを使用して、HFS 内のファイルを指定できます。例えば、次のように定義すると、DISPLAY 出力はファイル /u/*userid*/cobol/demo.lst に書き込まれます。

```
//SYSPUNCH DD PATH='/u/userid/cobol/demo.lst',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),PATHMODE=SIRWXU,
// FILEDATA=TEXT
```

次のステートメントは、ジョブ・ログまたはコンソール、および TSO 画面 (TSO のもとで実行している場合) に書き込みを行います。

```
Display "Hello" upon console.
```

USAGE NATIONAL データ項目の値をコンソールに表示するとき、その値は、CODEPAGE オプションの値に基づいて Unicode (UTF-16) 表現から EBCDIC に変換されます。DISPLAY ステートメントを使用したときに、国別データの変換が行われるのは、この場合のみです。この場合に変換が行われるのは、出力が画面へ送信されることが認識されているからです。

出力の宛先が別の装置であるときに国別データ項目を変換するには、DISPLAY-OF 組み込み関数 (以下の例を参照) を使用します。

```
01 Data-in-Unicode pic N(10) usage national.
. . .
        Display function Display-of(Data-in-Unicode, 00037)
```

関連概念
138 ページの『Unicode および言語文字のエンコード』

関連タスク 42 ページの『システム論理出力装置上でのデータの表示』
42 ページの『WITH NO ADVANCING の使用』

149 ページの『国別の英数字への変換 (DISPLAY-OF)』
454 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 350 ページの『CODEPAGE』
DISPLAY ステートメント (*Enterprise COBOL 言語解説書*)

システム論理出力装置上でのデータの表示

データをシステム論理出力装置に書き込むには、UPON 節を省略するか、または宛先 SYSOUT を指定した UPON 節を使用してください。

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

出力は、OUTDD コンパイラー・オプションで指定された DD 名に送られます。この DD 名に階層ファイル・システムのファイルを指定できます。

OUTDD という DD 名が割り振られておらず、プログラムが z/OS UNIX 環境で実行されていない場合には、SYSOUT=* というデフォルトの DD が割り振られます。
OUTDD という DD 名が割り振られていないときに、プログラムが z/OS UNIX 環境で実行されていれば、_IGZ_SYSOUT 環境変数が次のように使用されます。

未定義または stdout に設定されている
出力は stdout に送られます (ファイル記述子 1)。

stderr に設定されている
出力は stderr に送られます (ファイル記述子 2)。

それ以外 (stdout または stderr 以外に設定されている)
DISPLAY ステートメントが失敗し、重大度 3 の言語環境プログラム条件が起こります。

DISPLAY 出力が stdout または stderr に経路指定された場合、出力はレコード単位に分割されません。出力は改行のない単一の文字ストリームとして書き込まれます。

OUTDD と言語環境プログラムのランタイム・オプションの MSGFILE が同じ DD 名を指定していると、DISPLAY 出力と言語環境プログラム実行時診断の両方が言語環境プログラムのメッセージ・ファイルに経路指定されます。

関連タスク
490 ページの『環境変数の設定およびアクセス』

関連参照
380 ページの『OUTDD』
DISPLAY ステートメント (*Enterprise COBOL 言語解説書*)

WITH NO ADVANCING の使用

WITH NO ADVANCING 句を指定し、出力が DD 名に送られる場合は、印刷制御文字 + (正符号) が次の DISPLAY ステートメントの最初の出力位置に入れられます。 + は ANSI 定義の印刷制御文字であり、これはレコード印刷前の行送りを抑止します。

WITH NO ADVANCING 句を指定した場合に、出力が stdout または stderr に送られるときは、ストリームの終わりに改行文字が追加されません。その後の DISPLAY ステートメントがストリームの終わりに文字を追加する場合があります。

WITH NO ADVANCING を指定せず、出力が DD 名に送られる場合は、印刷制御文字 ' '(スペース) が次の DISPLAY ステートメントの最初の出力位置に入れられます (これはシングル・スペース出力を示します)。

```
DISPLAY "ABC"
DISPLAY "CDEF" WITH NO ADVANCING
DISPLAY "GHIJK" WITH NO ADVANCING
DISPLAY "LMNOPQ"
DISPLAY "RSTUVWX"
```

上記のステートメントをコーディングした場合は、以下の結果が出力装置に送られます。

```
 ABC
 CDEF
+GHIJK
+LMNOPQ
 RSTUVMX
```

印刷される出力は、出力装置が印刷制御文字をどのように解釈するかによって異なります。

WITH NO ADVANCING 句を指定しなかった場合、出力が stdout または stderr に送られるときは、ストリームの終わりに改行文字が追加されます。

関連参照
DISPLAY ステートメント (*Enterprise COBOL 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。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、関数によって戻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

94 ページの『組み込み関数を使用したテーブル項目の処理』
123 ページの『データ項目の変換 (組み込み関数)』
126 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それらを受け渡す。
- 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

75 ページの『テーブルの定義 (OCCURS)』
516 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』

ストレージとそのアドレス可能度

COBOL プログラムを実行するときは、プログラムおよびプログラムで使用されるデータは仮想記憶域にあります。COBOL で使用するストレージは、16MB 境界より下でも 16MB 境界より上でも構いませんが、2GB の境界を超えないようにしてください。このストレージをアドレッシングするときは、24 ビットおよび 31 ビットの 2 つのアドレッシング・モードが使用可能です。

24 ビット・アドレッシングを使用して、16MB 境界より下のストレージをアドレッシングすることができます (ただし、16MB 境界より上のアドレッシングはできません)。31 ビット・アドレッシングを使用すると、16MB 境界より上と 16MB 境界より下のどちらのストレージでもアドレッシングできます。31 ビット・アドレッシングを使用して、*無制限のストレージ* をアドレッシングできます。したがって、16MB 境界より上と 16MB 境界より下の両方を含む、プログラムで使用可能なすべてのストレージをアドレッシングできます。

Enterprise COBOL は、z/OS の 64 ビットの仮想アドレッシング機能を直接には利用しませんが、31 ビット・アドレッシング・モードまたは 24 ビット・アドレッシング・モードで実行している COBOL アプリケーションは、64 ビット z/OS システムで完全にサポートされます。

アドレッシング・モード (AMODE) は、ユーザーのプログラムによってどのハードウェア・アドレッシング・モードがサポートされるかを示す属性です。24 ビット・アドレッシング、31 ビット・アドレッシング、24 ビットまたは 31 ビット・アドレッシングのいずれかを示す属性があります。この属性はそれぞれ AMODE 24、AMODE 31、または AMODE ANY になります。オブジェクト・プログラム、ロード・モジュール、および実行プログラムにはそれぞれ AMODE 属性があります。Enterprise COBOL オブジェクト・プログラムはすべて AMODE ANY です。

常駐モード (RMODE) は、仮想記憶域のどこにプログラムが常駐するかを示すプログラム・ロード・モジュールの属性です。16MB 境界の下、16MB 境界の下または上の 2 つの属性があります。この属性は RMODE 24 または RMODE ANY です。

Enterprise COBOL は、言語環境プログラムのサービスを使用して、実行時に使用されるストレージを制御します。したがって、COBOL コンパイラー・オプションおよび言語環境プログラムのランタイム・オプションは、単独でまたは組み合わせによって、プログラムとデータの AMODE 属性および RMODE 属性に影響を与えます。

DATA プログラムが RENT を指定してコンパイルされている場合、WORKING-STORAGE データ、入出力バッファー、パラメーター・リスト用ストレージのロケーションに影響を及ぼすコンパイラー・オプション。

RMODE 常駐モードに影響を及ぼすコンパイラー・オプション。プログラムが NORENT を指定してコンパイルされている場合、WORKING-STORAGE データ、入出力バッファー、パラメーター・リスト用のストレージのロケーションにも影響を及ぼします。

RENT 再入可能プログラムを生成するためのコンパイラー・オプション。

HEAP ランタイム・ヒープのストレージを制御するランタイム・オプション。例えば、COBOL WORKING-STORAGE は、ヒープ・ストレージから割り振られます。

STACK ランタイム・スタックのストレージを制御するランタイム・オプション。例えば、COBOL LOCAL-STORAGE は、スタック・ストレージから割り振られます。

ALL31 アプリケーションを完全に AMODE 31 で実行できるかどうかを指定するランタイム・オプション。

RMODE の設定

RMODE オプションと RENT オプションは、プログラムの RMODE 属性を決定します。

表 4. **RMODE 属性に対する RMODE および RENT コンパイラー・オプションの影響**

RMODE コンパイラー・オプション	RENT コンパイラー・オプション	RMODE 属性
RMODE(AUTO)	NORENT	RMODE 24
RMODE(AUTO)	RENT	RMODE ANY
RMODE(24)	RENT または NORENT	RMODE 24
RMODE(ANY)	RENT または NORENT	RMODE ANY

リンク・エディットに関する考慮事項: COBOL が生成するオブジェクト・コードに属性 RMODE 24 がある場合には、RMODE 24 でオブジェクト・コードをリンク・エディットする必要があります。 COBOL が生成するオブジェクト・コードに属性 RMODE ANY がある場合には、RMODE ANY または RMODE 24 でオブジェクト・コードをリンク・エディットすることができます。

データの受け渡しに関するストレージ制限

16MB 境界より上のストレージで割り振られたパラメーターを AMODE 24 サブプログラムに渡してはなりません。31 ビット・アドレッシング・モードで実行しているプログラムの場合は、WORKING-STORAGE データおよびパラメーター・リストを強制的に境界より下に配置して、AMODE 24 で実行しているプログラムにデータを渡してください。

- DATA(24) を指定して、再入可能プログラム (RENT) をコンパイルする。
- RMODE(24) または RMODE(AUTO) を指定して、再入不可プログラム (NORENT) をコンパイルする。
- RMODE(ANY) を指定してコンパイルされた再入不可プログラム (NORENT) は、RMODE 24 を指定してリンク・エディットする必要があります。NORENT プログラムに対するデータ域は、プログラムが DATA(24) でコンパイルされている場合でも、プ

ログラムのロード先に応じて、16MB 境界より上または下になります。 DATA オプションは、NORENT を指定してコンパイルされたプログラムに影響を与えません。

データ域のロケーション

再入可能プログラムの場合は、DATA コンパイラー・オプションおよび HEAP ランタイム・オプションによって、データ域のストレージ (WORKING-STORAGE SECTION や FD レコード域など) を 16MB 境界より下から獲得するのか、制限のないストレージから獲得するのかを制御します。プログラムが 16MB 境界より上の仮想記憶域アドレスで 31 ビット・アドレス方式で実行される場合は、プログラムを RENT または RMODE(ANY) でコンパイルしなければなりません。DATA オプションは、NORENT を指定してコンパイルされたプログラムに影響を与えません。

ランタイム・オプション HEAP(,,BELOW) を指定すると、 DATA コンパイラー・オプションの効果はなくなります。WORKING-STORAGE SECTION データ域のストレージは、 16MB 境界より下から割り振られます。しかし、HEAP(,,ANYWHERE) をランタイム・オプションとして指定すると、データ域のストレージは、プログラムを DATA(24) コンパイラー・オプションを使用してコンパイルした場合は 16MB 境界より下から割り振られ、DATA(31) コンパイラー・オプションを使用してコンパイルした場合は、制限のないストレージから割り振られます。

LOCAL-STORAGE データのストレージ

LOCAL-STORAGE データ項目のロケーションは、STACK ランタイム・オプションおよびプログラムの AMODE によって制御されます。LOCAL-STORAGE データ項目は、STACK(,,ANYWHERE) ランタイム・オプションが有効で、プログラムが AMODE 31 で実行されているときは、制限のないストレージで獲得されます。それ以外の場合、LOCAL-STORAGE は 16MB 境界より下で獲得されます。DATA コンパイラー・オプションは、LOCAL-STORAGE データのロケーションに影響を与えません。

外部データに対するストレージ

DATA コンパイラー・オプションは、動的データ域 (WORKING-STORAGE、FD レコード域、およびパラメーター・リスト) のストレージ獲得方法に影響を与えるだけでなく、EXTERNAL データ用のストレージをどこから獲得するかにも影響を与えることがあります。EXTERNAL データに必要なストレージは、次の条件が満たされる場合には、制限のないストレージから獲得されます。

- プログラムが、DATA(31) および RENT コンパイラー・オプション、または RMODE(ANY) および NORENT コンパイラー・オプションでコンパイルされている。
- HEAP(,,ANYWHERE) ランタイム・オプションが有効である。
- ALL31(ON) ランタイム・オプションが有効である。

これ以外の場合はすべて、EXTERNAL データ用のストレージは 16MB 境界より下から獲得されます。ALL31(ON) ランタイム・オプションを指定するためには、実行単位内のすべてのプログラムが 31 ビット・アドレッシング・モードで実行可能でなければなりません。

QSAM 入出力バッファー用のストレージ

DATA コンパイラー・オプションは、QSAM ファイルの入出力バッファーをどこに獲得するかにも影響する場合があります。QSAM ファイルのバッファーの割り振りおよび DATA コンパイラー・オプションについては、以下の関連参照情報を参照してください。

関連概念
507 ページの『AMODE 切り替え』
言語環境プログラム・プログラミング・ガイド
(ヒープ・ストレージの概要: AMODE の考慮事項)

関連タスク
499 ページの『第 24 章 サブプログラムの使用』
521 ページの『第 25 章 データの共用』

関連参照
192 ページの『QSAM ファイル用のバッファーの割り振り』
354 ページの『DATA』
383 ページの『RENT』
385 ページの『RMODE』
735 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』
言語環境プログラム・プログラミング・リファレンス (HEAP, STACK, ALL31)
MVS プログラム管理: ユーザーズ・ガイドおよび解説書

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
『数値データの定義』
51 ページの『数値データの表示』
52 ページの『数値データの保管方法の制御』
61 ページの『非互換データの検査 (数値のクラス・テスト)』
62 ページの『算術の実行』
72 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。

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則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。

57 ページの『例: 数値データおよび内部表現』

関連概念

54 ページの『数値データの形式』
753 ページの『付録 A. 中間結果および算術精度』

関連タスク

51 ページの『数値データの表示』

52 ページの『数値データの保管方法の制御』
62 ページの『算術の実行』
142 ページの『国別数値データ項目の定義』

関連参照

60 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』
146 ページの『国別データのストレージ』
346 ページの『ARITH』
375 ページの『NUMPROC』
SIGN 節 (*Enterprise COBOL 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 と宣言してください。 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。

USAGE NATIONAL を持つ数値データ項目または数字編集データ項目を EBCDIC で表示するには、それらの項目を CONSOLE に向けてください。例えば、上記のコードの Edited-price が USAGE NATIONAL を持っているときに、上記の最後のステートメントが以下のものである場合、プログラムを実行すると $ 1,500.99 が表示されます。

```
Display Edited-price Upon Console
```

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 upon console
```

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。

57 ページの『例: 数値データおよび内部表現』

関連タスク

41 ページの『画面上またはファイル内での値の表示 (DISPLAY)』
『数値データの保管方法の制御』
49 ページの『数値データの定義』
62 ページの『算術の実行』
142 ページの『国別数値データ項目の定義』
147 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 (*Enterprise COBOL 言語解説書*)
BLANK WHEN ZERO 節 (*Enterprise COBOL 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。以下に例を示します。

```
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 節の規則および 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)。

57 ページの『例: 数値データおよび内部表現』

関連概念

54 ページの『数値データの形式』
58 ページの『データ形式の変換』
753 ページの『付録 A. 中間結果および算術精度』

関連タスク

49 ページの『数値データの定義』
51 ページの『数値データの表示』
62 ページの『算術の実行』

関連参照

59 ページの『変換および精度』
60 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*データ項目と呼びます。

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。代わりに Compute-Result-N を EBCDIC 文字で表示するには、それをコンソールに送ってください。

```
Display Compute-Result-N Upon Console
```

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 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。それらを SIZE ERROR 条件についてテストしたり、丸めたりするのは、他のタイプに比べて非効率的です。

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 として宣言したデータ項目は、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 5. **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
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、*Enterprise COBOL 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。左端のビットには符号が入り、次の 7 つのビットには指数が入り、残りの 3 または 7 バイトには仮数が入ります。

COMP-1 および COMP-2 データ項目は、zSeries® の 16 進形式で保管されます。

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

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

関連参照
146 ページの『国別データのストレージ』
397 ページの『TRUNC』
データのクラスおよびカテゴリー (*Enterprise COBOL 言語解説書*)
SIGN 節 (*Enterprise COBOL 言語解説書*)
VALUE 節 (*Enterprise COBOL 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。

表 6. **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

表 6. **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
2 進数	PIC S9999 BINARY PIC S9999 COMP PIC S9999 COMP-4	+ 1234	04 D2
		- 1234	FB 2E
	PIC S9999 COMP-5	+ 12345[1]	30 39
		- 12345[1]	CF C7
	PIC 9999 BINARY PIC 9999 COMP PIC 9999 COMP-4	1234	04 D2
	PIC 9999 COMP-5	60000[1]	EA 60
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 4F
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
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

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。

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 または COMPUTE 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
『変換および精度』
60 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-1 データ項目が 9 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 9 個だけ受け取り、残りの桁は 0 になります。

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。

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 9 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。

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
753 ページの『付録 A. 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: C、A、E、および F

負: D および B

COBOL NUMPROC コンパイラー・オプションは、ゾーン 10 進数データおよび内部 10 進数データの符号処理に影響します。 NUMPROC は、2 進数データ、国別 10 進数データ、および浮動小数点データには影響しません。

NUMPROC(PFD)

X'*sd*' (*s* はサイン表記であり、*d* は数字を表す) が与えられた場合に、NUMPROC(PFD) を使用すると、コンパイラーはデータ内の符号が 3 つの優先符号の 1 つであると想定します。

符号付き正数または 0:
X'C'

符号付き負数:
X'D'

符号なしまたは英数字:
X'F'

この想定に基づいて、コンパイラーは与えられるすべての符号を使用してデータを処理します。優先符号は、必要な場合にのみ生成されます (例えば、符号なしデータが符号付きデータに移動される場合)。NUMPROC(PFD) オプシ

ョンを使用すると、処理時間を節約することができますが、正しい処理を行うためには、データに優先符号を使用しなければなりません。

NUMPROC(NOPFD)

NUMPROC(NOPFD) コンパイラー・オプションが有効であると、コンパイラーはすべての有効な符号構成を受け入れます。受け取り側では、常に、優先符号が生成されます。NUMPROC(NOPFD) を使用すると、NUMPROC(PFD) より効率が低下しますが、優先符号を使用しないデータが存在する可能性があるときには NUMPROC(NOPFD) を使用しなければなりません。

符号なしのゾーン 10 進数送出側が英数字受信に移動されても、符号は未変更のままです (NUMPROC(NOPFD) が有効な場合でも)。

NUMPROC(MIG)

NUMPROC(MIG) が有効であると、コンパイラーは、OS/VS COBOL によって作成されるコードに似たコードを生成します。このオプションは、特に OS/VS COBOL プログラムを IBM Enterprise COBOL for z/OS に移行する場合に有用です。

関連参照 375 ページの『NUMPROC』
405 ページの『ZWB』

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。以下に例を示します。

```
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。例えば、パック 10 進数項目は、数字位置に 16 進値 X'0' から X'9' があり、符号位置 (分離または非分離) に有効な符号値があるかどうか検査されます。

ゾーン 10 進数およびパック 10 進数項目の場合、数値のクラス・テストは、NUMPROC コンパイラー・オプションおよび NUMCLS オプション (インストール時に

設定される) の影響を受けます。インストールの際に使用された NUMCLS 設定を確認するには、システム・プログラマーに問い合わせてください。

NUMCLS(PRIM) がインストール先で有効な場合は、次の表を使用して、コンパイラーが符号に有効と見なす値を見つけてください。

表 7. **NUMCLS(PRIM) および有効な符号**

	NUMPROC(NOPFD)	NUMPROC(PFD)	NUMPROC(MIG)
符号付き	C、D、F	C、D、+0 (正のゼロ)	C、D、F
符号なし	F	F	F
分離符号	+, -	+、-、+0 (正のゼロ)	+, -

NUMCLS(ALT) がインストール先で有効な場合は、次の表を使用して、コンパイラーが符号に有効と見なす値を見つけてください。

表 8. **NUMCLS(ALT) および有効な符号**

	NUMPROC(NOPFD)	NUMPROC(PFD)	NUMPROC(MIG)
符号付き	A から F	C、D、+0 (正のゼロ)	A から F
符号なし	F	F	F
分離符号	+, -	+、-、+0 (正のゼロ)	+, -

関連参照
375 ページの『NUMPROC』

算術の実行

幾つかの COBOL 言語機能 (COMPUTE、算術式、数値組み込み関数、数学呼び出し可能サービス、および日付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。数学関数および日時操作用の言語環境プログラム呼び出し可能サービスは、算術結果をデータ項目に割り当てる手段も提供します。

関連タスク
63 ページの『COMPUTE およびその他の算術ステートメントの使用』
63 ページの『算術式の使用』
64 ページの『数字組み込み関数の使用』
65 ページの『数学用の呼び出し可能サービスの使用』
67 ページの『データ呼び出し可能サービスの使用』

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。幾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。以下に例を示します。

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
70 ページの『固定小数点演算と浮動小数点演算の対比』
753 ページの『付録 A. 中間結果および算術精度』

関連タスク
49 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 9. **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
70 ページの『固定小数点演算と浮動小数点演算の対比』
753 ページの『付録 A. 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 10. **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	CURRENT-DATE	ANNUITY	ACOS	MEAN
MAX	DATE-OF-INTEGER	PRESENT-VALUE	ASIN	MEDIAN
MIN	DATE-TO-YYYYMMDD		ATAN	MIDRANGE
NUMVAL	DATEVAL		COS	RANDOM
NUMVAL-C	DAY-OF-INTEGER		FACTORIAL	RANGE
ORD-MAX	DAY-TO-YYYYDDD		INTEGER	STANDARD-DEVIATION
ORD-MIN	INTEGER-OF-DATE		INTEGER-PART	VARIANCE
	INTEGER-OF-DAY		LOG	
	UNDATE		LOG10	
	WHEN-COMPILED		MOD	
	YEAR-TO-YYYY		REM	
	YEARWINDOW		SIN	
			SQRT	
			SUM	
			TAN	

68 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。

数字組み込み関数の機能の多くは、言語環境プログラム呼び出し可能サービスによっても提供されます。

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

関連参照
346 ページの『ARITH』

数学用の呼び出し可能サービスの使用

ほとんどの COBOL 組み込み関数には、同じ結果を得るのに使用できる、対応する数学用の呼び出し可能サービスがあります。

デフォルト・オプション ARITH(COMPAT) を使用してコンパイルすると、COBOL 浮動小数点組み込み関数は長精度 (64 ビット) の結果を戻します。オプション ARITH(EXTEND) を使用してコンパイルすると、COBOL 浮動小数点組み込み関数 (RANDOM は例外) は拡張精度 (128 ビット) の結果を戻します。

例えば (以下の表の最初の行を考慮します)、ARITH(COMPAT) を使用してコンパイルした場合、CEESDACS は ACOS と同じ結果を戻します。 ARITH(EXTEND) を使用してコンパイルした場合は、CEESQACS が ACOS と同じ結果を戻します。

表 11. **演算組み込み関数と呼び出し可能サービスの互換性**

COBOL 組み込み関数	対応する長精度言語環境プログラム呼び出し可能サービス	対応する拡張精度言語環境プログラム呼び出し可能サービス	組み込み関数と呼び出し可能サービスで結果が同じか
ACOS	CEESDACS	CEESQACS	はい
ASIN	CEESDASN	CEESQASN	はい
ATAN	CEESDATN	CEESQATN	はい
COS	CEESDCOS	CEESQCOS	はい
LOG	CEESDLOG	CEESQLOG	はい
LOG10	CEESDLG1	CEESQLG1	はい
RANDOM[1]	CEERAN0	なし	いいえ
REM	CEESDMOD	CEESQMOD	はい
SIN	CEESDSIN	CEESQSIN	はい
SQRT	CEESDSQT	CEESQSQT	はい
TAN	CEESDTAN	CEESQTAN	はい

1. RANDOM は、ARITH(EXTEND) を使用してコンパイルし、31 ビットの引数を渡した場合でも、長精度 (64 ビット) 浮動小数点結果を戻します。

RANDOM 組み込み関数と CEERAN0 サービスはともに、0 から 1 の範囲の乱数を生成します。ただし、それぞれが独自のアルゴリズムを使用するため、RANDOM と CEERAN0 は同じシードから異なる乱数を生成します。

同じ結果をもたらす関数であっても、組み込み関数と言語環境プログラム呼び出し可能サービスの使用法は異なります。組み込み関数の引数に要求されるデータ型に関する規則の方が、制限が緩くなっています。数字組み込み関数の場合には、任意の数値データ型の引数を使用することができます。しかし、CALL ステートメントを使用して言語環境プログラム呼び出し可能サービスを呼び出す場合は、パラメーターをそのサービスで必要な数値データ型 (通常は COMP-1 または COMP-2) と一致させなければなりません。

組み込み関数と言語環境プログラム呼び出し可能サービスのエラー処理は異なる場合があります。言語環境プログラムの数学サービスを呼び出すときに明示的なフィードバック・トークンを渡す場合は、それぞれの呼び出しの後でそのフィードバック・コードを検査し、エラーを処理するための明示的なアクションを取らなければなりません。しかし、フィードバック・トークンを使用して明示的に OMITTED を呼び出した場合は、トークンを検査する必要はありません。エラーがあれば、言語環境プログラムが自動的にエラーを通知します。

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

関連タスク
745 ページの『言語環境プログラム呼び出し可能サービスの使用』

関連参照
346 ページの『ARITH』

データ呼び出し可能サービスの使用

COBOL の日付組み込み関数と言語環境プログラムの日付呼び出し可能サービスは両方とも、グレゴリオ暦を基本とします。ただし、開始日付は、INTDATE コンパイラー・オプションの設定値によって異なることがあります。

INTDATE(LILIAN) が有効な場合、COBOL は第 1 日として 1582 年 10 月 15 日を使用します。言語環境プログラムは常に、第 1 日として 1582 年 10 月 15 日を使用します。INTDATE(LILIAN) を使用すると、COBOL 組み込み関数と言語環境プログラムの日付呼び出し可能サービスから同じ結果が得られます。次の表は、INTDATE(LILIAN) が有効な場合の結果を比較したものです。

表 12. **日付組み込み関数と呼び出し可能サービスの INTDATE(LILIAN) と互換性**

COBOL 組み込み関数	言語環境プログラムの呼び出し可能サービス	結果
DATE-OF-INTEGER	CEEDATE (ピクチャー・ストリング YYYYMMDD 付き)	互換性あり
DAY-OF-INTEGER	CEEDATE (ピクチャー・ストリング YYYYDDD 付き)	互換性あり
INTEGER-OF-DATE	CEEDAYS	互換性あり
INTEGER-OF-DATE	CEECBLDY	非互換

デフォルト設定である INTDATE(ANSI) が有効な場合、COBOL は第 1 日として 1601 年 1 月 1 日を使用します。次の表は、INTDATE(ANSI) が有効な場合の結果を比較したものです。

表 13. **日付組み込み関数と呼び出し可能サービスの INTDATE(ANSI) と互換性**

COBOL 組み込み関数	言語環境プログラムの呼び出し可能サービス	結果
INTEGER-OF-DATE	CEECBLDY	互換性あり
DATE-OF-INTEGER	CEEDATE (ピクチャー・ストリング YYYYMMDD 付き)	非互換
DAY-OF-INTEGER	CEEDATE (ピクチャー・ストリング YYYYDDD 付き)	非互換
INTEGER-OF-DATE	CEEDAYS	非互換

関連タスク
745 ページの『言語環境プログラム呼び出し可能サービスの使用』

関連参照
366 ページの『INTDATE』

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 =
      関数
                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

125 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

通常、算術演算に上記の浮動小数点についての特性がない場合、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)。以下に例を示します。

```
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。

暗黙の比較 (関係演算子が使用されない) は、単位として扱われません。しかし、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における評価に関しては別々に扱われます。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
```

72 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
762 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)e+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'9F' を示しています。

```
    Currency Sign X'9F' 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、次の表に示すように、使用されているコード・ページに応じて、X'9F' または X'5A' のいずれかです。

表 14. **ユーロ記号の 16 進値**

コード・ページ CCSID	適用される国	変更元	ユーロ記号
1140	米国、カナダ、オランダ、ポルトガル、オーストラリア、ニュージーランド	037	X’9F’
1141	オーストリア、ドイツ	273	X’9F’
1142	デンマーク、ノルウェー	277	X’5A’
1143	フィンランド、スウェーデン	278	X’5A’
1144	イタリア	280	X’9F’
1145	スペイン、ラテンアメリカ - スペイン語圏	284	X’9F’
1146	英国	285	X’9F’
1147	フランス	297	X’9F’
1148	ベルギー、カナダ、スイス	500	X’9F’
1149	アイスランド	871	X’9F’

関連参照
353 ページの『CURRENCY』
CURRENCY SIGN 節 (*Enterprise COBOL 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.
```

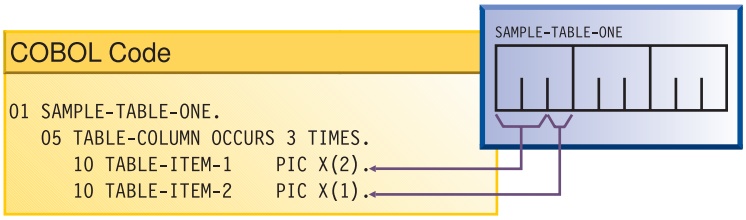
上記の例は、次のような表示出力を作成します。

```
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速度を高めるうえで大切なものです。

関連タスク

77 ページの『テーブルのネスト』
『テーブルの定義 (OCCURS)』
79 ページの『テーブル内の項目の参照』
81 ページの『テーブルに値を入れる方法』
87 ページの『可変長テーブルの作成 (DEPENDING ON)』
90 ページの『テーブルの探索』
94 ページの『組み込み関数を使用したテーブル項目の処理』
728 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 記述で指定することはできません。

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。

93 ページの『例: 二分探索』

関連概念

142 ページの『国別グループ』

関連タスク

77 ページの『テーブルのネスト』
79 ページの『テーブル内の項目の参照』
81 ページの『テーブルに値を入れる方法』
87 ページの『可変長テーブルの作成 (DEPENDING ON)』
144 ページの『国別グループの使用』
92 ページの『二分探索 (SEARCH ALL)』
49 ページの『数値データの定義』

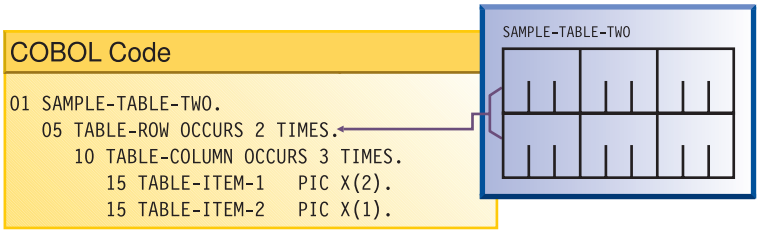
関連参照

OCCURS 節 (*Enterprise COBOL 言語解説書*)

SIGN 節 (*Enterprise COBOL 言語解説書*)
ASCENDING KEY および DESCENDING KEY 句(*Enterprise COBOL 言語解説書*)

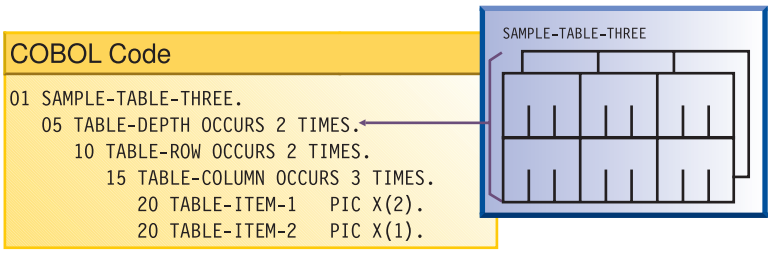
テーブルのネスト

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。以下に例を示します。



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。

78 ページの『例: 添え字付け』
78 ページの『例: 指標付け』

関連タスク

75 ページの『テーブルの定義 (OCCURS)』
79 ページの『テーブル内の項目の参照』
81 ページの『テーブルに値を入れる方法』
87 ページの『可変長テーブルの作成 (DEPENDING ON)』

90 ページの『テーブルの探索』
94 ページの『組み込み関数を使用したテーブル項目の処理』
728 ページの『テーブルの効率的処理』

関連参照
OCCURS 節 (*Enterprise COBOL 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
79 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
80 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
『添え字付け』
80 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。以下に例を示します。

```
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。以下に例を示します。

```
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 2 文字の長さ分、テーブル・エレメント 1 に移動します。

78 ページの『例: 添え字付け』

関連タスク
『索引付け』
81 ページの『テーブルに値を入れる方法』
90 ページの『テーブルの探索』
728 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とき、読み取られた最後の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。そのあ

と、現行の指標値を最後のレコードの指標値と比較することによって、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。そのあと、索引名を SEARCH ステートメントまたは関係条件ステートメントでも使用できるようになります。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。

78 ページの『例: 指標付け』

関連タスク

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

関連参照

INDEXED BY 句 (*Enterprise COBOL 言語解説書*)
INDEX 句 (*Enterprise COBOL 言語解説書*)
SET ステートメント (*Enterprise COBOL 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

82 ページの『テーブルの動的なロード』
89 ページの『可変長テーブルのロード』
82 ページの『テーブルの初期化 (INITIALIZE)』
83 ページの『テーブルの定義時の値の割り当て (VALUE)』
90 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、変更済みの値をテーブルに動的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。

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

関連参照
VARYING 句を指定した PERFORM (*Enterprise COBOL 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。

32 ページの『例: データ項目の初期化』

関連タスク

35 ページの『構造の初期化 (INITIALIZE)』
『テーブルの定義時の値の割り当て (VALUE)』
90 ページの『可変長テーブルへの値の割り当て』
108 ページの『テーブルのループ処理』
28 ページの『データ項目とグループ項目の使用』
144 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 (*Enterprise COBOL 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
『それぞれのテーブル項目の個別の初期化』
『グループ・レベルでのテーブルの初期化』
85 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』
35 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) を宣言します。
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
35 ページの『構造の初期化 (INITIALIZE)』
90 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 (*Enterprise COBOL 言語解説書*)
OCCURS 節 (*Enterprise COBOL 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"00410042" ('AB' の UTF-16 表現) が入り、国別 10 進数 項目 ElementTwoN(1) には、NX"00310032" ('12' の UTF-16 表現) が入ります。以下同様です。

関連参照
OCCURS 節 (*Enterprise COBOL 言語解説書*)
GROUP-USAGE 節 (*Enterprise COBOL 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
90 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 (*Enterprise COBOL 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
     RECORDING MODE F
     BLOCK 0 RECORDS
     RECORD 80 CHARACTERS
     LABEL RECORD STANDARD.
 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

765 ページの『付録 B. 複合 OCCURS DEPENDING ON』

関連タスク 90 ページの『可変長テーブルへの値の割り当て』 『可変長テーブルのロード』 768 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』 130 ページの『データ項目の長さの検出』 *Enterprise COBOL コンパイラーおよびランタイム 移行ガイド*

関連参照

OCCURS DEPENDING ON 節 (*Enterprise COBOL 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
     RECORDING MODE F
     BLOCK 0 RECORDS
     RECORD 80 CHARACTERS
     LABEL RECORD STANDARD.
 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
83 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 (*Enterprise COBOL 言語解説書*)

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
『逐次探索 (SEARCH)』
92 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。

92 ページの『例: 逐次探索』

関連参照
SEARCH ステートメント (*Enterprise COBOL 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。

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。

『例: 二分探索』

関連タスク
75 ページの『テーブルの定義 (OCCURS)』

関連参照
SEARCH ステートメント (*Enterprise COBOL 言語解説書*)
一般比較条件 (*Enterprise COBOL 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。

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、関数の引数の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))
```

『例: 組み込み関数を使用したテーブルの処理』

関連タスク
43 ページの『組み込み関数の使用 (組み込み関数)』
123 ページの『データ項目の変換 (組み込み関数)』
126 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 (*Enterprise COBOL 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
『プログラム・アクションの選択』
106 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
『アクションの選択項目のコーディング』
102 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 (*Enterprise COBOL 言語解説書*)
EVALUATE ステートメント (*Enterprise COBOL 言語解説書*)

アクションの選択項目のコーディング

2 つの処理アクションからいずれかを選択するようにするには、IF . . . ELSE を使用します。 (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
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
『ネストされた IF ステートメントの使用』
99 ページの『EVALUATE ステートメントの使用』
102 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。

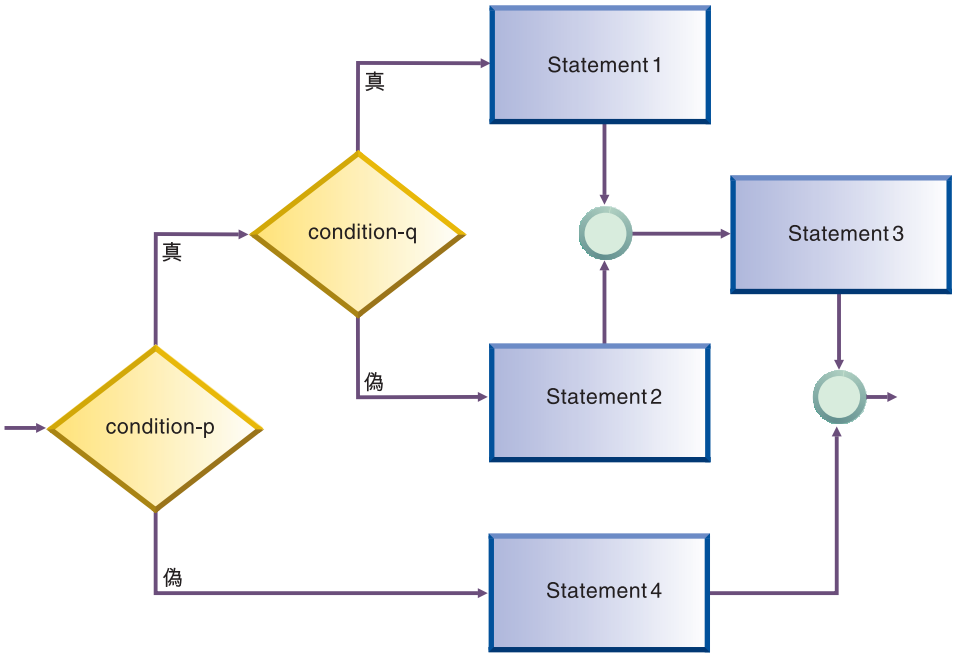
ただし、ネストされた IF ステートメントを多用しないでください。明示範囲終了符号および字下げが役に立つとはいえ、ロジックをたどるのが困難になる可能性があります。プログラムが 2 つを超える値について変数をテストしなければならない場合には、おそらく EVALUATE を使用する方が適切です。

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
97 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 (*Enterprise COBOL 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、幾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。

100 ページの『例: THRU 句を使用した EVALUATE』
101 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。

101 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
97 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 (*Enterprise COBOL 言語解説書*)
一般比較条件 (*Enterprise COBOL 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、幾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.
  Source-Computer. IBM-390.
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。

指定できる条件の一部を以下に示します。

- 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 DBCS
  - IS KANJI
  - IS NOT 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
『スイッチおよびフラグ』

関連タスク
『スイッチおよびフラグの定義』
105 ページの『スイッチとフラグのリセット』
61 ページの『非互換データの検査 (数値のクラス・テスト)』
152 ページの『国別 (UTF-16) データの比較』
157 ページの『有効な DBCS 文字に関するテスト』

関連参照
一般比較条件 (*Enterprise COBOL 言語解説書*)
クラス条件 (*Enterprise COBOL 言語解説書*)
条件名項目の規則 (*Enterprise COBOL 言語解説書*)
符号条件 (*Enterprise COBOL 言語解説書*)
複合条件 (*Enterprise COBOL 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
『スイッチおよびフラグの定義』
105 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。

『例: スイッチ』
『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' なら、Transaction-EOF という名前の条件がオン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
```

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、マスター・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-Master-Record-Paragraph
  When Change-Transaction
    Perform Update-Existing-Record-Paragraph
  When Delete-Transaction
    Perform Delete-Master-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。

『例: スイッチをオンに設定する』
106 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

以下の例は、値 TRUE をレベル 88 項目に移動する 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示しています。

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

次の例は、値をレベル 88 項目に移動する 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示しています。

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
『インラインまたはライン外 PERFORM の選択』
107 ページの『ループのコーディング』
108 ページの『テーブルのループ処理』
109 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 (*Enterprise COBOL 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。

  プログラム内の幾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の方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、別の段落への分岐と暗黙の戻りが必要になります。実行された段落が、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 1 度実行されます。この追加の実行を回避するためには、段落を通常の順次フローの外側 (例えば、GOBACK の後) に置くか、または段落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。

『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 ステートメントを使用すれば、段落を指定した回数だけ実行することができます。

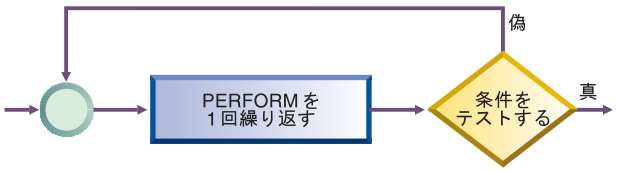
```
PERFORM 010-PROCESS-ONE-MONTH 12 TIMES
INSPECT . . .
```

上記の例では、制御が PERFORM ステートメントに達すると、段落 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段落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 . . .
```

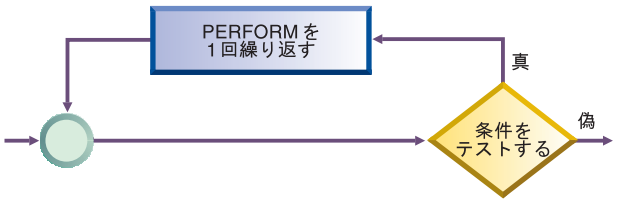
段落を少なくとも 1 回実行し、その後の実行の前にテストを行うようにしたい場合は、PERFORM . . . WITH TEST AFTER . .. UNTIL ステートメントを使用します。このステートメントは、do-until 構造と同等です。



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。それから、条件 WS-DATA-IX = 12 が検査されます。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。しかし、PERFORM . . . THRU ステートメントをコーディングすれば、段落のグループ、1 つのセクション、またはセクションのグループを実行することができます。

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

94 ページの『組み込み関数を使用したテーブル項目の処理』

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

『データ項目の結合 (STRING)』
114 ページの『データ項目の分割 (UNSTRING)』
117 ページの『ヌル終了ストリングの取り扱い』
118 ページの『データ項目のサブストリングの参照』
122 ページの『データ項目の計算および置換 (INSPECT)』
123 ページの『データ項目の変換 (組み込み関数)』
126 ページの『データ項目の評価 (組み込み関数)』
133 ページの『第 7 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、幾つかの 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。

『例: STRING ステートメント』

関連タスク
264 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 (*Enterprise COBOL 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、幾つかの 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

115 ページの『例: UNSTRING ステートメント』

関連概念

138 ページの『Unicode および言語文字のエンコード』

関連タスク

264 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 (*Enterprise COBOL 言語解説書*)
データのクラスおよびカテゴリー (*Enterprise COBOL 言語解説書*)

例: UNSTRING ステートメント

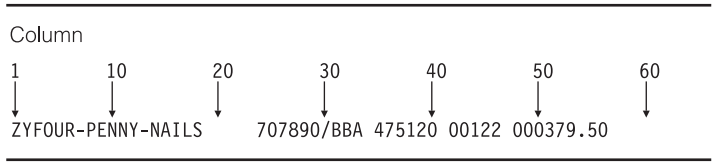
次の例は、選択した情報を入力レコードから転送する UNSTRING ステートメントを示しています。情報の中には、印刷用に編成されているものもあれば、その後の処理用に編成されているものもあります。

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。

- 区切り文字用としてピリオド (.) を DBY-1 に入れます。
- 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

さまざまな手段を使って、ヌル終了ストリング (例えば、C プログラムとの間でやり取りされるストリング) を構成し取り扱うことができます。

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。

118 ページの『例: ヌル終了ストリング』

関連タスク

527 ページの『ヌル終了ストリングの処理』

関連参照

Alphanumeric リテラル (*Enterprise COBOL 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幾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) 望ましいサブストリングの長さ (*文字位置*の数)

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"0032") 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。

121 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。

121 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
『参照修飾子』
138 ページの『Unicode および言語文字のエンコード』

関連タスク
79 ページの『テーブル内の項目の参照』

関連参照
390 ページの『SSRANGE』
参照変更 (*Enterprise COBOL 言語解説書*)
関数定義 (*Enterprise COBOL 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。以下に例を示します。

```
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)
```

121 ページの『例: 参照修飾子としての演算式』
121 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
40 ページの『画面またはファイルからの入力の割り当て (ACCEPT)』

118 ページの『データ項目のサブストリングの参照』
139 ページの『COBOL での国別データ (Unicode) の使用』

関連参照
参照変更 (*Enterprise COBOL 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。以下に例を示します。

```
Move Customer-Record(Function Integer(Function Sqrt(I)): ) to WS-name
```

関連参照
INTEGER (*Enterprise COBOL 言語解説書*)
INTEGER-PART (*Enterprise COBOL 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。

『例: INSPECT ステートメント』

関連概念

138 ページの『Unicode および言語文字のエンコード』

関連参照

INSPECT ステートメント (*Enterprise COBOL 言語解説書*)

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
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・データ項目を他の幾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。

122 ページの『例: INSPECT ステートメント』

関連タスク
『大文字または小文字への変換 (UPPER-CASE、LOWER-CASE)』
『逆順への変換 (REVERSE)』
125 ページの『数値への変換 (NUMVAL、NUMVAL-C)』
126 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』

大文字または小文字への変換 (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。

関連タスク
40 ページの『画面またはファイルからの入力の割り当て (ACCEPT)』
41 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
138 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 および NUMVAL-C 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。

```
01  R       Pic x(20)  Value "- 1234.5678".
01  S       Pic x(20)  Value "  $12,345.67CR".
01  Total   Usage is Comp-1.
. . .
    Compute Total = Function Numval(R) + Function Numval-C(S)
```

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 および NUMVAL-C 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 または NUMVAL-C 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 または NUMVAL-C を使用する場合は、数値データを固定形式で静的に宣言したり、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。以下に例を示します。

```
+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
54 ページの『数値データの形式』
58 ページの『データ形式の変換』
138 ページの『Unicode および言語文字のエンコード』

関連タスク
147 ページの『国別 (Unicode) 表現との間の変換』

関連参照
59 ページの『変換および精度』
346 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
138 ページの『Unicode および言語文字のエンコード』

関連タスク
147 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、データ項目の長さ (USAGE NATIONAL データ項目を含む) を調べるためのものです。
- 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
138 ページの『Unicode および言語文字のエンコード』

関連タスク
『照合シーケンスに関する単一文字の評価』
『最大または最小データ項目の検出』
130 ページの『データ項目の長さの検出』
130 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、要求された文字を戻します。以下に例を示します。

```
INITIALIZE Customer-Name REPLACING ALPHABETIC BY Function Char(65)
```

関連参照
CHAR (*Enterprise COBOL 言語解説書*)
ORD (*Enterprise COBOL 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、照合シーケンスにしたがって評価します。

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。上記の例で ORD-MAX 関数を使用すると、数字関数への参照が有効場所にないので、コンパイラーはエラー・メッセージを出します。以下に、ORD-MAX の有効な使用例を示します。

```
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
62 ページの『算術の実行』
94 ページの『組み込み関数を使用したテーブル項目の処理』
『英数字または国別関数によって戻される可変長結果』

関連参照
MAX (*Enterprise COBOL 言語解説書*)
MIN (*Enterprise COBOL 言語解説書*)
ORD-MAX (*Enterprise COBOL 言語解説書*)
ORD-MIN (*Enterprise COBOL 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長結果

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"0066 0020 0020 0020 0020" (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

127 ページの『最大または最小データ項目の検出』
62 ページの『算術の実行』

関連参照

MAX (*Enterprise COBOL 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 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しか使用できません。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、整数引数が認められる組み込み関数への引数として LENGTH OF 特殊レジスターを使用することができます。(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
62 ページの『算術の実行』
87 ページの『可変長テーブルの作成 (DEPENDING ON)』
94 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
LENGTH (*Enterprise COBOL 言語解説書*)
LENGTH OF (*Enterprise COBOL 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、時刻を秒までしか扱いません。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 (*Enterprise COBOL 言語解説書*)

第 7 章 国際環境でのデータの処理

Enterprise COBOL 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、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:
  - CODEPAGE を使用すると、プログラムで英数字および DBCS データに使用するコード・ページを指定できます。
  - 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。

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
138 ページの『Unicode および言語文字のエンコード』
142 ページの『国別グループ』

関連タスク
139 ページの『COBOL での国別データ (Unicode) の使用』
147 ページの『国別 (Unicode) 表現との間の変換』
151 ページの『UTF-8 データの処理』
151 ページの『中国語 GB 18030 データの処理』
152 ページの『国別 (UTF-16) データの比較』
155 ページの『DBCS サポート用のコーディング』
771 ページの『付録 C. 2 バイト文字セット (DBCS) データの変換』

関連参照
『COBOL ステートメントと国別データ』
137 ページの『組み込み関数と国別データ』
350 ページの『CODEPAGE』
373 ページの『NSYMBOL』
データのクラスおよびカテゴリー (*Enterprise COBOL 言語解説書*)
データ・カテゴリーおよび PICTURE 規則 (*Enterprise COBOL 言語解説書*)
MOVE ステートメント (*Enterprise COBOL 言語解説書*)
一般比較条件 (*Enterprise COBOL 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 15. **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	*identifier-1*, *identifier-2*	*identifier-1* が CODEPAGE コンパイラー・オプションで指定されたネイティブ・コード・ページから変換されるのは、入力が CONSOLE からのものである場合のみです。	40 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。		63 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*		521 ページの『データの受け渡し』

表 15. **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
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。		63 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*		407 ページの『第 18 章 コンパイラー指示ステートメント』
DISPLAY	*identifier-1*	*identifier-1* が EBCDIC に変換されるのは、CONSOLE mnemonic-name が直接的または間接的に指定されている場合のみです。	41 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。		63 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。	32 ページの『例: データ項目の初期化』
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。	122 ページの『データ項目の計算および置換 (INSPECT)』
INVOKE	*identifier-2* または *literal-1* としてのメソッド名。BY VALUE 句の *identifier-3* または *literal-2*。		636 ページの『メソッドの呼び出し (INVOKE)』
MERGE	マージ・キー	COLLATING SEQUENCE 句は適用されません。	248 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。	37 ページの『基本データ項目への値の割り当て (MOVE)』 38 ページの『グループ・データ項目への値の割り当て (MOVE)』

表 15. **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
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。		63 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。	92 ページの『二分探索 (SEARCH ALL)』
SORT	ソート・キー	COLLATING SEQUENCE 句は適用されません。	248 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。	111 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。		63 ページの『COMPUTE およびその他の算術ステートメントの使用』
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。	114 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)		593 ページの『第 29 章 XML 出力の生成』

表 15. **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
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。 XML-NNAMESPACE および XML-NNAMESPACE-PREFIX 特殊レジスターには、国別文字で関連した名前空間 ID および名前空間接頭部 (存在する場合) が含まれています。	561 ページの『第 28 章 XML 入力の処理』

関連タスク
49 ページの『数値データの定義』
51 ページの『数値データの表示』
139 ページの『COBOL での国別データ (Unicode) の使用』
152 ページの『国別 (UTF-16) データの比較』

関連参照 350 ページの『CODEPAGE』
データのクラスおよびカテゴリー (*Enterprise COBOL 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 16. **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	149 ページの『国別の英数字への変換 (DISPLAY-OF)』
LENGTH	Integer	130 ページの『データ項目の長さの検出』
LOWER-CASE、UPPER-CASE	国別	124 ページの『大文字または小文字への変換 (UPPER-CASE、LOWER-CASE)』
NUMVAL, NUMVAL-C	数値	125 ページの『数値への変換 (NUMVAL、NUMVAL-C)』
MAX、MIN	国別	127 ページの『最大または最小データ項目の検出』
ORD-MAX、ORD-MIN	Integer	127 ページの『最大または最小データ項目の検出』
REVERSE	国別	124 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
49 ページの『数値データの定義』
139 ページの『COBOL での国別データ (Unicode) の使用』

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

Unicode および言語文字のエンコード

Enterprise COBOL 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)* と呼ばれる幾つかのエンコード・スキーム (UTF-8、UTF-16、および UTF-32 など) があります。 Enterprise COBOL 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、ビッグ・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'00*nn*'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'004100420043' 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 および 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。 COBOL for Windows® 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 (UTF-16LE) をサポートします。UTF-16LE 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 Enterprise COBOL へ移植する場合、そのデータをビッグ・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必要があります。

関連タスク
147 ページの『国別 (Unicode) 表現との間の変換』

関連参照
146 ページの『国別データのストレージ』
文字セットとコード・ページ (*Enterprise COBOL 言語解説書*)

COBOL での国別データ (Unicode) の使用

Enterprise COBOL では、幾つかの方法で国別 (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
138 ページの『Unicode および言語文字のエンコード』
142 ページの『国別グループ』

関連タスク
『国別データ項目の定義』
140 ページの『国別リテラルの使用』
141 ページの『国別文字形象定数の使用』
142 ページの『国別数値データ項目の定義』
144 ページの『国別グループの使用』
147 ページの『国別 (Unicode) 表現との間の変換』
152 ページの『国別 (UTF-16) データの比較』

関連参照
146 ページの『国別データのストレージ』
データのクラスおよびカテゴリー (*Enterprise COBOL 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
51 ページの『数値データの表示』

関連参照
373 ページの『NSYMBOL』
BLANK WHEN ZERO 節 (*Enterprise COBOL 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"00410042" 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 C1C2 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"C1C2" 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。

関連タスク
30 ページの『リテラルの使用』

関連参照
373 ページの『NSYMBOL』
国別リテラル (*Enterprise COBOL 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。

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 との間の変換) 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 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
147 ページの『国別 (Unicode) 表現との間の変換』
152 ページの『国別 (UTF-16) データの比較』

関連参照
形象定数 (*Enterprise COBOL 言語解説書*)
DISPLAY-OF (*Enterprise COBOL 言語解説書*)
Support for Unicode: Using Unicode Services

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
49 ページの『数値データの定義』
51 ページの『数値データの表示』

関連参照
SIGN 節 (*Enterprise COBOL 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、幾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

38 ページの『グループ・データ項目への値の割り当て (MOVE)』
111 ページの『データ項目の結合 (STRING)』
114 ページの『データ項目の分割 (UNSTRING)』
122 ページの『データ項目の計算および置換 (INSPECT)』
144 ページの『国別グループの使用』

関連参照

GROUP-USAGE 節 (*Enterprise COBOL 言語解説書*)

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
142 ページの『国別グループ』

関連タスク
145 ページの『国別グループを基本項目として使用』
145 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 (*Enterprise COBOL 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 内の値は、国別文字で 06/23/2007 となります。

```
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 2007.
    . . .
    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 06232007 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
38 ページの『グループ・データ項目への値の割り当て (MOVE)』
154 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』
『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 (*Enterprise COBOL 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 17. **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。
EXEC SQL ステートメントのホスト変数	国別グループ項目をホスト変数として指定してください。	国別グループ項目は、実質的には、グループ項目に従属するホスト変数セットの省略表現です。
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

35 ページの『構造の初期化 (INITIALIZE)』
82 ページの『テーブルの初期化 (INITIALIZE)』
37 ページの『基本データ項目への値の割り当て (MOVE)』
38 ページの『グループ・データ項目への値の割り当て (MOVE)』
130 ページの『データ項目の長さの検出』
593 ページの『XML 出力の生成』
471 ページの『SQL ステートメントでの国別グループ項目の使用』

関連参照

修飾 (*Enterprise COBOL 言語解説書*)
RENAMES 節 (*Enterprise COBOL 言語解説書*)

国別データのストレージ

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 18. **エンコード方式と英数字、DBCS、および国別データのサイズ**

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 [1]	EBCDIC	EBCDIC DBCS	UTF-16BE
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

1. 英数字または DBCS データに適用できる EBCDIC コード・ページを指定するには、CODEPAGE コンパイラー・オプションを使用してください。
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

関連概念
138 ページの『Unicode および言語文字のエンコード』

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、CODEPAGE コンパイラー・オプション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
148 ページの『英数字、DBCS、および整数から国別への変換 (MOVE)』
148 ページの『英数字または DBCS から国別への変換 (NATIONAL-OF)』
149 ページの『国別の英数字への変換 (DISPLAY-OF)』
149 ページの『デフォルト・コード・ページのオーバーライド』
152 ページの『国別 (UTF-16) データの比較』

関連参照
350 ページの『CODEPAGE』
150 ページの『変換例外』

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「AB」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'0020') 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
37 ページの『基本データ項目への値の割り当て (MOVE)』
38 ページの『グループ・データ項目への値の割り当て (MOVE)』
51 ページの『数値データの表示』
155 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 (*Enterprise COBOL 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。 CODEPAGE コンパイラー・オプシ

ョンを使用して有効な異なるコード・ページでソースがエンコードされている場合、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。

150 ページの『例: 国別データとの間の変換』

関連タスク
151 ページの『UTF-8 データの処理』
151 ページの『中国語 GB 18030 データの処理』
157 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 350 ページの『CODEPAGE』
NATIONAL-OF (*Enterprise COBOL 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、ソースのコンパイル時に CODEPAGE コンパイラー・オプションで有効にされたコード・ページになります。

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。

150 ページの『例: 国別データとの間の変換』

関連タスク
151 ページの『UTF-8 データの処理』
151 ページの『中国語 GB 18030 データの処理』

関連参照
DISPLAY-OF (*Enterprise COBOL 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、 CODEPAGE オプション値として指定された CCSID 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、CODEPAGE コンパイラー・オプションで有効なコード・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、CODEPAGE コンパイラー・オプションで指定された EBCDIC コード・ページが使用されることを想定しています。

関連参照
350 ページの『CODEPAGE』

変換例外

国別データと英数字データとの間の暗黙変換または明示変換が失敗して、重大度 3 の言語環境プログラム条件が生成されることがあります。

暗黙的または明示的に指定したコード・ページが有効なコード・ページではない場合、失敗することがあります。

ターゲットの CCSID に対になる文字が存在しない文字については、変換例外は発生しません。このような文字はターゲット・コード・ページの*置換文字* に変換されます。

関連参照
350 ページの『CODEPAGE』

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。

```
 CBL CODEPAGE(00037)
* . . .
 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, 00875)
         to Data-in-Unicode
* . . . process Data-in-Unicode here . . .
     Move function Display-of(Data-in-Unicode, 00875)
         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、CCSID 00875 (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
```

上記の MOVE ステートメントは、CCSID 00037 (米国英語) から UTF-16 への変換に基づいて、Data-in-Greek を Unicode 表現に変換します。 Data-in-Greek は CCSID 00875 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。

CODEPAGE コンパイラー・オプションを正しく CCSID 00875 に設定することができた場合 (つまり、プログラムの残りの部分も EBCDIC データをギリシャ語で処理した場合) は、上記と同じ例を次のようにしてコーディングすることができます。

```
 CBL CODEPAGE(00875)
* . . .
 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
```

UTF-8 データの処理

UTF-8 データを処理する必要がある場合は、最初に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。

ASCII または EBCDIC データを UTF-8 に変換するには、次の 2 つのステップを実行する必要があります。

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

147 ページの『国別 (Unicode) 表現との間の変換』
118 ページの『データ項目のサブストリングの参照』
585 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。

GB 18030 文字は、UTF-16 またはコード・ページ CCSID 1392 でエンコードできます。コード・ページ 1392 は、1 文字に 1、2、または 4 バイトを使用する ASCII マルチバイト・コード・ページです。 GB 18030 文字のサブセットは、中国語 ASCII コード・ページ (CCSID 1386) または中国語 EBCDIC コード・ページ (CCSID 1388) でエンコードできます。

Enterprise COBOL は GB 18030 を明示的にはサポートしていませんが、幾つかの方法で GB 18030 文字の処理をサポートします。以下のことが可能です。

- DBCS データ項目を使用して、CCSID 1388 で表される GB 18030 文字を処理します。
- 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 (CCSID 1388 または 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、コード・ページ 1388 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-EBCDIC 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-EBCDIC, 1388) to UnicodeString
*  Process data in Unicode
     Move function Display-of(UnicodeString, 1388) to Chinese-GB18030-String
```

関連タスク

147 ページの『国別 (Unicode) 表現との間の変換』
155 ページの『DBCS サポート用のコーディング』

関連参照

146 ページの『国別データのストレージ』

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

続く各節では、国別データとその他のデータ項目との比較について概説します。詳細については、関連参照を参照してください。

関連タスク
『2 つのクラス国別オペランドの比較』
154 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』
154 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』
154 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』
154 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 (*Enterprise COBOL 言語解説書*)
一般比較条件 (*Enterprise COBOL 言語解説書*)
国別比較 (*Enterprise COBOL 言語解説書*)
グループ比較 (*Enterprise COBOL 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。

同じ長さを持つ 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'0020') が埋め込まれているものとして扱われます。

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
142 ページの『国別グループ』

関連タスク
144 ページの『国別グループの使用』

関連参照
国別比較 (*Enterprise COBOL 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 (*Enterprise COBOL 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
142 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 (*Enterprise COBOL 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
141 ページの『国別文字形象定数の使用』

関連参照
国別比較 (*Enterprise COBOL 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 (*Enterprise COBOL 言語解説書*)

DBCS サポート用のコーディング

IBM Enterprise COBOL for z/OS 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 (DBCS 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
- DBCS コンパイラー・オプション

関連タスク
『DBCS データの宣言』
156 ページの『DBCS リテラルの使用』
157 ページの『有効な DBCS 文字に関するテスト』
157 ページの『DBCS データを含む英数字データ項目の処理』
771 ページの『付録 C. 2 バイト文字セット (DBCS) データの変換』

関連参照 357 ページの『DBCS』

DBCS データの宣言

DBCS データ項目を宣言するには、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 項目の宣言で 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。

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。

関連参照
373 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 (") または単一引用符 (') 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。

シフトアウト (SO) 制御文字 X'0E' 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'0F' 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、英数字リテラルに含まれる DBCS 文字のストリングは、SO および SI 文字で区切る必要があり、SO および SI 文字がシフト・コードとして認識されるためには DBCS コンパイラー・オプションが有効になっている必要があります。

DBCS 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。

DBCS 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
31 ページの『形象定数の使用』

関連参照 357 ページの『DBCS』
373 ページの『NSYMBOL』
383 ページの『QUOTE/APOST』
DBCS リテラル (*Enterprise COBOL 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、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、文字を検査することで行われます。

関連タスク
102 ページの『条件式のコーディング』

関連参照
クラス条件 (*Enterprise COBOL 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
111 ページの『データ項目の結合 (STRING)』
114 ページの『データ項目の分割 (UNSTRING)』
118 ページの『データ項目のサブストリングの参照』
147 ページの『国別 (Unicode) 表現との間の変換』

第 8 章 ファイルの処理

データの読み取りと書き込みは、すべてのプログラムに不可欠なものです。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、結果を示します。

情報のソースおよび結果のターゲットとしては、以下の 1 つ以上の項目を使用できます。

- 別のプログラム
- 直接アクセス記憶装置
- 磁気テープ
- プリンター
- 端末
- カード読取装置または穿孔装置

外部装置上に存在する情報は、物理レコードまたはブロックとして格納されています。レコードまたはブロックは、入力または出力操作時にシステムによって 1 つの単位として処理される情報の集まりです。

COBOL プログラムは、物理レコードを直接処理しません。論理レコードを処理します。論理レコードは、1 つの完全な物理レコードであることもあるし、1 つの物理レコードの一部であることもあるし、1 つ以上の物理レコードの一部または全部を含むこともあります。COBOL プログラムは、論理レコードを、それらが定義されたとおりに正確に処理します。

COBOL では、論理レコードの集合をファイル (プログラムで処理できる情報の列) と言います。

関連概念
『ファイル編成および入出力装置』

関連タスク
161 ページの『ファイル編成およびアクセス・モードの選択』
164 ページの『ファイルの割り振り』
165 ページの『入出力エラーの検査』

ファイル編成および入出力装置

入出力装置によって異なりますが、ファイル編成には、順次、行順次、索引付き、または相対があります。プログラムを設計するときに、使用する装置およびファイル・タイプを決定してください。

以下のファイル編成を選択することができます。

順次ファイル編成
レコードの配置は、ファイル作成時のレコードの入力順によって決まります。それぞれのレコード (最初のレコードは除く) には固有の先行レコード

があり、それぞれのレコード (最後のレコードは除く) には固有の後続レコードがあります。いったん確立されると、これらの関係は変わりません。

順次ファイルについて認められるアクセス (レコード伝送) モードは順次のみです。

行順次ファイル編成

行順次ファイルとは、階層ファイル・システム (HFS) 上にある順次ファイルであり、データとしては文字のみが入っています。各レコードは改行文字で終了します。

行順次ファイルについて認められるアクセス (レコード伝送) モードは順次のみです。

索引付きファイル編成

ファイル内のそれぞれのレコードには特殊フィールドが含まれ、そのフィールドの内容がレコード・キーを形成します。このキーの位置は各レコードで同じです。ファイルの論理配置は、ファイルの索引コンポーネントがレコード・キーによって順序付けて確立します。ファイル内のレコードの実際の物理配置は、COBOL プログラムにとっては重要ではありません。

索引付きファイルは、レコード・キーに加えて、代替索引を使用することもできます。これらのキーにより、レコードに関して別の論理順序付けを使用して、ファイルにアクセスできます。

索引付きファイルについて認められるアクセス (レコード伝送) モードは、順次、ランダム、または動的です。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。

相対ファイル編成

ファイルのレコードは、ファイルの始まりからの相対位置によって識別されます。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。

相対ファイルについて認められるアクセス (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。

IBM Enterprise COBOL for z/OS では、レコードの保管および入出力装置からの取り出しに関するオペレーティング・システムへの要求は、QSAM と VSAM の 2 つのアクセス方式、および UNIX ファイル・システムによって処理されます。

データの保管にどの装置タイプを使用するかによって、使用できるファイル編成の選択に影響する可能性があります。ファイル編成オプションで最も柔軟に選択できるのは、直接アクセス記憶装置の場合です。順次専用の装置の場合、編成オプションは制限されますが、有用な他の特性 (テープの可搬性など) があります。

順次専用装置

端末装置、プリンター、カード読取装置、および穿孔装置は、一度に 1 行を処理するので、*ユニット・レコード装置* と呼ばれます。したがって、ユニット・レコード装置との間で読み取りまたは書き込みを行うときは、プログラムで一度にレコードを 1 つずつ順次に処理しなければなりません。

テープでは、レコードは順次に並べられるので、プログラムでそれらを順次に処理しなければなりません。テープ・ファイルを処理するときは、QSAM 物理順次ファイルを使用してください。テープのレコードは固定長でも可変長でも構いません。

直接アクセス・ストレージ・デバイス

直接アクセス・ストレージ・デバイスには多くのレコードが格納できます。これらの装置に保管されるファイルのレコード配置によって、プログラムがデータを処理できる方法が決まります。直接アクセス装置を使用すると、以下の複数のファイル編成タイプが使用できるため、プログラムにおける柔軟性が高まります。

- 順次 (VSAM または QSAM)
- 行順次 (UNIX)
- 索引付き (VSAM)
- 相対 (VSAM)

関連タスク

164 ページの『ファイルの割り振り』
167 ページの『第 9 章 QSAM ファイルの処理』
199 ページの『第 10 章 VSAM ファイルの処理』
231 ページの『第 11 章 line-sequential ファイルの処理』
『ファイル編成およびアクセス・モードの選択』

ファイル編成およびアクセス・モードの選択

アプリケーションで使用するファイル編成およびアクセス・モードを決定する際に使用できる指針があります。

ファイル編成の選択時に以下の指針を考慮してください。

- アプリケーションがレコード (固定長でも可変長でも) に順次にアクセスのみを行い、既存のレコード間にレコードを挿入しない場合には、QSAM または VSAM 順次ファイルが最も単純なタイプです。
- 印刷可能文字および特定の制御文字のみを含むレコードに順次にアクセスする UNIX アプリケーションを開発している場合には、行順次ファイルが最も適しています。
- アプリケーションが (レコードが固定長でも可変長でも) 順次アクセスとランダム・アクセスの両方を必要とする場合には、VSAM 索引付きファイルが最も柔軟性のあるタイプです。
- アプリケーションがレコードをランダムに挿入および削除する場合には、相対ファイルが適しています。

アクセス・モードの選択時には、以下の指針を考慮してください。

- ファイルのかなりの部分がアプリケーションの中で参照または更新される場合には、順次アクセスの方が、ランダムまたは動的アクセスより速くなります。
- アプリケーションの実行ごとに処理されるレコードの割合が小さい場合は、ランダム・アクセスまたは動的アクセスを使用します。

表 19. **COBOL ファイルのファイル編成、アクセス・モード、レコード・フォーマットの要約**

ファイル編成	順次アクセス	ランダム・アクセス	動的アクセス	固定長	可変長
QSAM (物理順次)	X			X	X
行順次	X			X[1]	X
VSAM 順次 (ESDS)	X			X	X
VSAM 索引付き (KSDS)	X	X	X	X	X
VSAM 相対 (RRDS)	X	X	X	X	X

1. データ自体は可変長形式ですが、COBOL 固定長レコードに読み込んだり、固定長レコードから書き出すことができます。

関連参照
『入出力コーディングの形式』
232 ページの『許可される制御文字』

入出力コーディングの形式

入出力コーディングの一般形式を、以下に示します。コードの後に、ユーザー指定の情報について説明しています。

```
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  PICTURE 99.
. . .
PROCEDURE DIVISION.
    . . .
    OPEN iomode filename                       (9)
    . . .
    READ filename
    . . .
    WRITE recordname
    . . .
    CLOSE filename
    . . .
    STOP RUN.
```

上記のコードの中のユーザー指定の情報は、次のとおりです。

(1) ***filename***

任意の正規 COBOL 名。SELECT 節と FD 項目、および READ、OPEN、CLOSE の各ステートメントでは、同じファイル名を使用しなければなりません。さ

らに、START または DELETE ステートメントを使用する場合にも、ファイル名が必要になります。この名前は、必ずしも、システムに認識されているデータ・セットの実際の名前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。

(2) ***assignment-name***

選択した任意の名前 (COBOL およびシステムの命名規則に従うもの)。名前は、ユーザー定義語である場合には 1 から 30 文字の長さにすることができ、リテラルである場合には 1 から 160 文字の長さにすることができます。*assignment-name* の *name* 部分は、DD ステートメントで、ALLOCATE コマンド (TSO) で、あるいは環境変数として (例えば、export コマンドで) (UNIX) 指定することができます。

(3) ***org***

編成は、SEQUENTIAL、LINE SEQUENTIAL、INDEXED、または RELATIVE にすることができます。この節は、QSAM ファイルについては任意指定です。

(4) ***access***

アクセス・モードは、SEQUENTIAL、RANDOM、または DYNAMIC にすることができます。順次ファイル処理の場合 (行順次ファイルを含む) は、この節を省略することができます。

(5) ***file-status***

COBOL ファイル状況キー。ファイル状況キーは、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) または国別 10 進数 (USAGE NATIONAL) 項目として指定できます。

(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。

(7) ***fieldlength***

フィールドの論理長。

(8) ***type***

ファイルのレコード形式。レコード記入項目をレベル 01 記述以上に分ける場合、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。

(9) ***iomode***

INPUT または OUTPUT モード。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。書き込みだけを行う場合は、OUTPUT または EXTEND 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 (ただし、LINE SEQUENTIAL 編成の場合を除きます)。

関連タスク

167 ページの『第 9 章 QSAM ファイルの処理』
199 ページの『第 10 章 VSAM ファイルの処理』
231 ページの『第 11 章 line-sequential ファイルの処理』

ファイルの割り振り

z/OS または UNIX アプリケーション内のどのタイプのファイル (順次、行順次、索引付き、または相対) についても、DD 名または環境変数名のいずれかを使用して外部名を定義することができます。外部名とは、ASSIGN 節の *assignment-name* 内の名前です。

ファイルが HFS 内にある場合は、DD 定義または環境変数のいずれかを使用して、PATH キーワードでパス名を指定することによって、ファイルを定義することができます。

環境変数名は大文字でなければなりません。その値として使用できる属性は、定義中のファイルの編成によって異なります。

外部名は 2 つの方法のいずれかで定義できるので、COBOL 実行時には、以下のステップを使用してファイルの定義が調べられます。

1. DD 名が明示的に割り振られていれば、それが使用されます。定義は、JCL の DD ステートメント、TSO/E からの ALLOCATE コマンド、またはユーザー開始の動的割り振りから取ることができます。
2. DD 名が明示的に割り振られていないが、同じ名前の環境変数が設定されている場合は、その環境変数の値が使用されます。

   ファイルは、環境変数で指定された属性を使用して動的に割り振られます。最小限、PATH() または DSN() オプションのいずれかを指定しなければなりません。オプションおよび属性はすべて大文字でなければなりません (ただし、 PATH オプションの *path-name* サブオプションは例外で、これには大/小文字の区別があります)。DSN() オプションに一時データ・セット名を指定することはできません。

   以下の場合には、ファイル状況コード 98 が出されます。
   - 環境変数の内容 (ヌルまたはすべてブランクの値を含む) が無効である。
   - ファイルの動的割り振りが失敗する。
   - ファイルの動的割り振り解除が失敗する。

   COBOL 実行時には、各 OPEN ステートメントで環境変数の内容が検査されます。同じ外部名を持つファイルが前の OPEN ステートメントによって動的に割り振られており、その OPEN 以後に環境変数の内容が変更された場合は、実行時に、前の割り振りが動的に割り振り解除され、環境変数に現在設定されているオプションを使用してファイルが再度割り振られます。環境変数の内容が変更されていない場合、実行時には現行の割り振りが使用されます。
3. DD 名も環境変数も定義されていない場合、次のようになります。
   a. 割り振りが QSAM ファイルに対するものであり、CBLQDA ランタイム・オプションが有効であれば、適格ファイルに対して CBLQDA 動的割り振り処理が行われます。この種の「暗黙の」動的割り振りは、実行単位が存続する間持続され、再割り振りすることはできません。
   b. そうでない場合、割り振りは失敗します。

COBOL 実行時には、暗黙の CBLQDA 割り振りを除いて、実行単位の終了時にすべての動的割り振りが割り振り解除されます。

関連タスク
490 ページの『環境変数の設定およびアクセス』
185 ページの『QSAM ファイルの定義および割り振り』
181 ページの『QSAM ファイルの動的作成』
223 ページの『VSAM ファイルの割り振り』

入出力エラーの検査

それぞれの入力または出力ステートメントが実行された後、操作が成功したか失敗したかを示す値にファイル状況キーが更新されます。

FILE STATUS 節を使用して、それぞれの入力または出力ステートメントの後にファイル状況キーを検査し、ゼロ以外のファイル状況コードが戻されている場合にはエラー処理プロシージャーを呼び出してください。 VSAM ファイルの場合、FILE STATUS 節の 2 番目のデータ項目を使用して、追加の VSAM 状況コード情報を取得できます。

入出力操作でのエラーを処理する別の方法として、ERROR (EXCEPTION と同義) 宣言をコーディングする方法があります。

関連タスク
265 ページの『入出力操作でのエラーの処理』
268 ページの『ERROR 宣言のコーディング』
269 ページの『ファイル状況キーの使用』

第 9 章 QSAM ファイルの処理

待機順次アクセス方式 (QSAM) 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次々に配置されます。

プログラムでは、これらのファイルを順次にのみ処理することができ、レコードを、それらがファイルに入っているのと同じ順序で検索します (READ ステートメントを使用して)。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。しかし、直接アクセス・ストレージ・デバイス上の QSAM ファイルは (REWRITE を使用して) 更新することができます (ただし HFS では不可)。

QSAM ファイルは、テープ、直接アクセス・ストレージ・デバイス (DASD)、ユニット・レコード装置、および端末装置上に置くことができます。QSAM 処理は、テーブルおよび中間記憶域に最適です。

QSAM を使用して、HFS 内のバイト・ストリーム・ファイルにアクセスすることもできます。これらのファイルは、レコード構造を持たない 2 進数のバイト単位の順次ファイルです。COBOL プログラムでコーディングしたレコード定義と、読み取りおよび書き込みに使用する変数の長さによって、転送されるデータの量が決まります。

関連概念
194 ページの『QSAM ファイルのラベル』
z/OS DFSMS: Using Data Sets (アクセス方式)

関連タスク
『COBOL での QSAM ファイルおよびレコードの定義』
179 ページの『QSAM ファイルの入出力ステートメントのコーディング』
184 ページの『QSAM ファイルのエラーの処理』
184 ページの『QSAM ファイルの操作』
197 ページの『テープ上の QSAM ASCII ファイルの処理』
198 ページの『ASCII ファイルのラベルの処理』

COBOL での QSAM ファイルおよびレコードの定義

COBOL プログラムのファイルを QSAM ファイルとして定義して、ファイルを外部ファイル名と関連付けるには、FILE-CONTROL 記入項目を使用します。

外部ファイル名 (DD 名または環境変数名) とは、ファイルがオペレーティング・システムに認識されるときに使用される名前です。次の例では、COMMUTER-FILE-MST は、プログラムがファイルに使用する名前であり、COMMUTR は外部名です。

```
FILE-CONTROL.
        SELECT COMMUTER-FILE-MST
        ASSIGN TO S-COMMUTR
        ORGANIZATION IS SEQUENTIAL
        ACCESS MODE IS SEQUENTIAL.
```

ASSIGN 節 *name* には、ファイルが QSAM ファイルである文書の外部名の前に S- を含めることができます。ORGANIZATION 節と ACCESS MODE 節は、ともに任意指定です。

関連タスク
『レコード形式の指定』
176 ページの『ブロック・サイズの設定』

レコード形式の指定

DATA DIVISION の FD 項目では、レコード・フォーマットおよびレコードをブロック化するかどうかの標識をコーディングしてください。関連するレコード記述項目では、*record-name* およびレコード長を指定します。

RECORDING MODE 節に、レコード・フォーマット F、V、S、または U をコーディングできます。 COBOL は、RECORD 節から、またはファイルの FD 記入項目に関連付けられたレコード記述から、レコード形式を判別します。レコードがブロック化されるようにしたい場合には、FD 記入項目に BLOCK CONTAINS 節をコーディングしてください。

次の例は、固定長レコードを持つファイルの場合に FD 記入項目がどのようになるかを示しています。

```
FILE SECTION.
FD  COMMUTER-FILE-MST
    RECORDING MODE IS F
    BLOCK CONTAINS 0 RECORDS
    RECORD CONTAINS 80 CHARACTERS.
01  COMMUTER-RECORD-MST.
    05  COMMUTER-NUMBER        PIC  X(16).
    05  COMMUTER-DESCRIPTION   PIC  X(64).
```

S の記録モードは、HFS 内のファイルではサポートされません。上記の例はこのようなファイルに適しています。

関連概念
169 ページの『論理レコード』

関連タスク
169 ページの『固定長フォーマットの要求』
170 ページの『可変長フォーマットの要求』
173 ページの『スパン形式の要求』
175 ページの『不定形式の要求』
167 ページの『COBOL での QSAM ファイルおよびレコードの定義』

関連参照
15 ページの『FILE SECTION 記入項目』

論理レコード

COBOL では、*論理レコード* という用語を、z/OS QSAM とは多少異なる仕方で使用します。

形式 V および形式 S ファイルの場合、QSAM 論理レコードではレコードのユーザー・データ部分の前に、COBOL 論理レコードの定義には含まれていない 4 バイトの接頭部が入ります。

形式 F および形式 U のファイル、および HFS のバイト・ストリーム・ファイルの場合、QSAM 論理レコードと COBOL 論理レコードの定義は同じです。

本書では、*QSAM 論理レコード* は QSAM 定義を指し、*論理レコード* は COBOL 定義を指しています。

関連参照
170 ページの『形式 F レコードのレイアウト』
171 ページの『形式 V レコードのレイアウト』
174 ページの『形式 S レコードのレイアウト』
176 ページの『形式 U レコードのレイアウト』

固定長フォーマットの要求

固定長レコードは、形式 F になります。この形式を明示的に要求するには、`RECORDING MODE F` を使用します。

`RECORDING MODE` 節は省略することができます。ファイルに関連付けられている最大のレベル 01 レコードの長さが、`BLOCK CONTAINS` 節でコーディングされているブロック・サイズを超えておらず、かつ次のいずれかを行った場合は、コンパイラーは記録モードを `F` と見なします。

- `RECORD CONTAINS` *integer* 節 (形式 1 `RECORD` 節) を使用して、レコード長 (バイト単位) を示します。

  この節を使用すると、ファイルは必ずレコード長 *integer* の固定長形式になります (別の長さを指定した複数のレベル 01 レコード記述がファイルに関連付けられている場合でも)。
- `RECORD CONTAINS` *integer* 節を省略するが、ファイルに関連付けられているすべてのレベル 01 レコード記述項目を、同じ固定サイズで、`OCCURS DEPENDING ON` 節を含まないようにコーディングする。この固定サイズがレコード長になります。

非ブロック化形式 F ファイルでは、論理レコードはブロックと同じになります。

ブロック化形式 F ファイルでは、ブロック内の論理レコードの数 (*ブロック化因数*) はファイル内の (最後のブロックを除く) すべてのブロックで一定です。最後のブロックは少ないことがあります。

HFS 内のファイルはブロック化されることはありません。

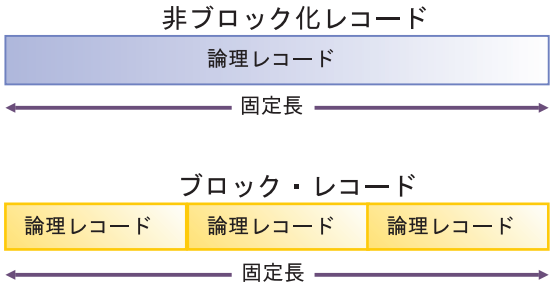
関連概念
169 ページの『論理レコード』

関連タスク
『可変長フォーマットの要求』
173 ページの『スパン形式の要求』
175 ページの『不定形式の要求』
168 ページの『レコード形式の指定』

関連参照
『形式 F レコードのレイアウト』

形式 F レコードのレイアウト:

形式 F の QSAM レコードのレイアウトを以下に示します。



関連概念
169 ページの『論理レコード』

関連タスク
169 ページの『固定長フォーマットの要求』
z/OS DFSMS: Using Data Sets (固定長レコード形式)

関連参照
171 ページの『形式 V レコードのレイアウト』
174 ページの『形式 S レコードのレイアウト』
176 ページの『形式 U レコードのレイアウト』

可変長フォーマットの要求

可変長レコードは、形式 V または形式 D です。形式 D レコードは、ASCII テープ・ファイル上の可変長レコードです。形式 D レコードは、形式 V レコードと同じ方法で処理されます。

いずれの場合も、RECORDING MODE V を使用してください。RECORDING MODE 節は省略することができます。ファイルに関連付けられている最大のレベル 01 レコードの長さが、BLOCK CONTAINS 節でコーディングされているブロック・サイズを超えておらず、かつ次のいずれかを行った場合は、コンパイラーは記録モードを V と見なします。

- RECORD IS VARYING 節 (形式 3 RECORD 節) を使用する。

値を *integer-1* および *integer-2* (RECORD IS VARYING FROM *integer-1* TO *integer-2*) に与えた場合、最大レコード長は、ファイルに関連したレベル 01 レコード記述項目にコーディングされた長さにかかわらず、 *integer-2* にコーディングされた値です。整数サイズは、レコードのデータ項目の USAGE にかかわらず、最小および最大レコード長 (バイト数) を示します。

integer-1 および *integer-2* を省略する場合、最大レコード長は、ファイルに関連付けられた最大のレベル 01 レコード記述項目のサイズであると判別されます。

- RECORD CONTAINS *integer-1* TO *integer-2* 節 (形式 2 RECORD 節) を使用する。*integer-1* および *integer-2* を、ファイルに関連したレベル 01 レコード記述項目の最小長および最大長 (バイト数) と一致させてください。最大レコード長は *integer-2* の値です。
- RECORD 節を省略するが、サイズが異なるか OCCURS DEPENDING ON 節を含む、複数のレベル 01 レコード (ファイルに関連付けられる) をコーディングする。

  最大レコード長は、ファイルに関連付けられた最大のレベル 01 レコード記述項目のサイズであると判別されます。

形式 V ファイルに READ INTO ステートメントを指定すると、そのファイルに関して読み取られたレコード・サイズが、コンパイラーによって生成される MOVE ステートメントで使用されます。したがって、読み取られたレコードがレベル 01 レコード記述に対応していない場合には、予期したとおりの結果が得られないことがあります。MOVE ステートメントに関するその他の規則はすべて適用されます。例えば、READ ステートメントによって読み込まれる形式 V レコードに MOVE ステートメントを指定すると、移動されるレコードのサイズは、そのレベル 01 レコード記述に相当します。

形式 V ファイルに対する READ ステートメントを指定し、その後にレベル 01 レコードの MOVE を続けて指定すると、実際のレコード長が使用されません。プログラムは、レベル 01 レコード記述で記述されたバイト数を移動しようとします。このバイト数が実際のレコード長を超え、プログラムによってアドレッシング可能な領域外に及ぶと、結果は予測できません。レベル 01 レコード記述によって記述されたバイト数が、読み取られた物理レコードより短い場合は、レベル 01 記述を超えるバイトは切り捨てられます。可変長レコードの実際の長さを見つけるには、ファイル定義 (FD) の RECORD 節の形式 3 で *data-name-1* を指定してください。

関連タスク

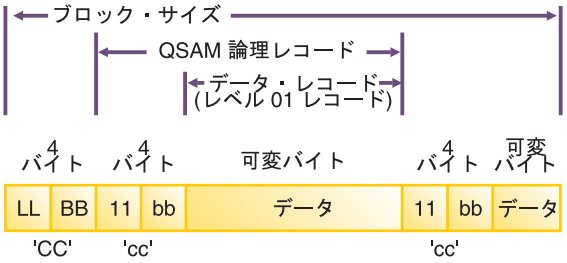
169 ページの『固定長フォーマットの要求』
173 ページの『スパン形式の要求』
175 ページの『不定形式の要求』
168 ページの『レコード形式の指定』

関連参照

15 ページの『FILE SECTION 記入項目』
『形式 V レコードのレイアウト』
Enterprise COBOL コンパイラーおよびランタイム 移行ガイド
(VS COBOL II ランタイムからの移行)

形式 V レコードのレイアウト:

形式 V QSAM レコードでは、データの前に制御フィールドがあります。 QSAM 論理レコード長は、プログラム内で定義されたレコード長に 4 バイト (制御フィールド用の) を加算することによって決まりますが、レコードおよびレコード長の記述の中にこれらの 4 バイトを含めてはなりません。



CC 各ブロックの最初の 4 バイトには、制御情報が入ります。

LL は、ブロックの長さ ('CC' フィールドを含む) を指定する 2 バイトを表します。

BB は、システム使用に予約されている 2 バイトを表します。

cc 各論理レコードの最初の 4 バイトには、制御情報が入ります。

11 は、論理レコード長 ('cc' フィールドを含む) を指定する 2 バイトを表します。

bb は、システム使用に予約されている 2 バイトを表します。

ブロック長は次のように決定されます。

- 非ブロック化形式 V レコード: CC + cc + データ部分
- ブロック化形式 V レコード: CC + 各レコードの cc + 各レコードのデータ部分

オペレーティング・システムは、ファイルの作成時に制御バイトを設定します。この制御バイト・フィールドは、プログラムの DATA DIVISION における論理レコードの記述には現れません。 COBOL は、制御バイトを収容するのに十分な大きさの入出力バッファーを割り振ります。バッファー内のこれらの制御フィールドをプログラムの中で使用することはできません。可変長レコードがユニット・レコード装置に書き込まれるときには、制御バイトは印刷も穿孔もされません。しかし、それらは、その他の外部ストレージ装置や、ストレージのバッファー域には入れられます。 V モード・レコードを入力バッファーから WORKING-STORAGE 域に移動すると、それらは制御バイトなしで移動されます。

HFS 内のファイルはブロック化されることはありません。

関連概念
169 ページの『論理レコード』

関連タスク
170 ページの『可変長フォーマットの要求』

関連参照
170 ページの『形式 F レコードのレイアウト』
174 ページの『形式 S レコードのレイアウト』
176 ページの『形式 U レコードのレイアウト』

スパン形式の要求

スパン・レコード とは、形式 S のレコードで、1 つ以上の物理ブロックに含めることができる QSAM 論理レコードのことです。

磁気テープまたは直接アクセス装置に割り当てられる QSAM ファイルのスパン・レコードには、RECORDING MODE S をコーディングできます。HFS のファイルについてはスパン・レコードを要求してはなりません。RECORDING MODE 節は省略することができます。最大レコード長 (バイト数) に 4 を加えた値が、 BLOCK CONTAINS 節で設定されたブロック・サイズより大きい場合、コンパイラーは記録モードが S であると判定します。

プログラム内の形式 S のファイルの場合、コンパイラーは最大レコード長を形式 V について使用されるのと同じ規則で判別します。この長さは、RECORD 節の使用法に基づいています。

フォーマット S レコードを含んでいるファイルを作成する場合に、レコードがブロックの残りのスペースより大きい場合、COBOL はレコードの 1 セグメントを書き込んでブロックを埋めます。レコードの残りの部分は、その長さに応じて、次の 1 つ以上のブロックに格納されます。COBOL は、32,760 バイトまでの長さの QSAM スパン・レコードをサポートします。

形式 S レコードを持つファイルを検索するときには、プログラムは完全なレコードだけを検索することができます。

形式 S ファイルのメリット: 形式 S レコードを持つファイルを定義すると、外部ストレージを効率よく利用できる上に、論理レコード長を使用してファイルを編成することができます。

- 直接アクセス装置でトラック容量を効率的に使用するためにブロック長を設定することができます。
- 論理レコード長を装置依存の物理ブロック長に合わせて調整する必要がありません。1 つの論理レコードが 2 つ以上の物理ブロックにわたることができます。
- 異なるタイプの直接アクセス・ストレージ間で論理レコードを転送したいときに、柔軟性が増します。

しかし、形式 S ファイルを処理するためには追加のオーバーヘッドが必要になります。

形式 S ファイルおよび READ INTO: 形式 S ファイルに READ INTO ステートメントを指定した場合、コンパイラーは、そのファイルの読み取りにのみ使用するレコードのサイズを使用する MOVE ステートメントを生成します。読み取られたレコードが、レベル 01 レコード記述に対応していない場合には、期待どおりの結果が得られないことがあります。 MOVE ステートメントに関するその他の規則はすべて適用されます。

関連概念

169 ページの『論理レコード』
174 ページの『スパン・ブロック化および非ブロック化ファイル』

関連タスク
169 ページの『固定長フォーマットの要求』
170 ページの『可変長フォーマットの要求』
175 ページの『不定形式の要求』
168 ページの『レコード形式の指定』

関連参照
15 ページの『FILE SECTION 記入項目』
『形式 S レコードのレイアウト』

スパン・ブロック化および非ブロック化ファイル: スパン・ブロック化された QSAM ファイルは、それぞれが 1 つ以上の論理レコードまたは論理レコードのセグメントを含むブロックから構成されます。スパン非ブロック化ファイルは、それぞれが 1 つの論理レコードまたは論理レコードの 1 つのセグメントを含む物理ブロックから構成されます。

スパン・ブロック化ファイルでは、論理レコードは固定長であっても可変長であっても構わず、そのサイズは物理ブロック・サイズと等しくても、より小さくても、より大きくても構いません。論理レコードと物理ブロックのサイズの間に必要とされる関係はありません。

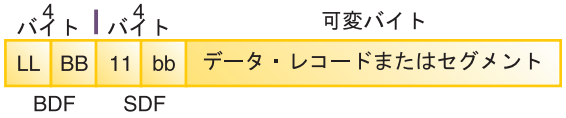
スパン非ブロック化ファイルでは、論理レコードは固定長であっても可変長であっても構いません。物理ブロックが 1 つの論理レコードを含んでいる場合、ブロック長は論理レコード・サイズによって判別されます。論理レコードをセグメンテーションしなければならない場合、システムは、常に、できる限り大きな物理ブロックを書き込みます。論理レコード全体が 1 つのトラックに収まらない場合、システムは論理レコードをセグメンテーションします。

関連概念
169 ページの『論理レコード』

関連タスク
173 ページの『スパン形式の要求』

形式 S レコードのレイアウト:

以下に示すように、スパン・レコードの前には制御フィールドが置かれます。



各ブロックの前には、4 バイトのブロック記述子フィールド (図中の「BDF」) が置かれます。ブロック記述子フィールドは、各物理ブロックの始まりに 1 つだけあります。

ブロック内のレコードの各セグメントの前には、そのセグメントがレコード全体である場合でも、4 バイトのセグメント記述子フィールド (図中の「SDF」) が置かれます。セグメント記述子フィールドは、ブロック内の各レコード・セグメントにつき 1 つあります。セグメント記述子フィールドは、そのセグメントが最初、最後、または中間のいずれかであるかも示します。

これらのフィールドは、COBOL プログラムの DATA DIVISION では記述せず、プログラム内で使用することはできません。

関連タスク
173 ページの『スパン形式の要求』

関連参照
170 ページの『形式 F レコードのレイアウト』
171 ページの『形式 V レコードのレイアウト』
176 ページの『形式 U レコードのレイアウト』

不定形式の要求

形式 U レコードは、未定義または未指定の特性を持ちます。形式 U を使用すると、形式 F または形式 V 仕様に適合しないブロックを処理することができます。

形式 U ファイルを使用するときは、ストレージの各ブロックが 1 つの論理レコードになります。形式 U のファイルを読み取るとブロック全体が 1 レコードとして返されます。形式 U のファイルに書き込むと 1 レコードが 1 ブロックとして書き込まれます。コンパイラーが記録モードを U と判別するのは、RECORDING MODE U をコーディングした場合だけです。

異なるレコード形式で書き込まれたファイルの更新または拡張には、形式 U を使用しないことをお勧めします。フォーマット U を使用して、別のフォーマットで作成されたファイルを更新する場合、データ・セット・ラベルの RECFM 値が変更されるか、またはデータ・セットに別のフォーマットで作成されたレコードが入ることがあります。

レコード長は、RECORD 節が使用される方法に基づいて、プログラムの中で判別されます。

- RECORD CONTAINS *integer* 節 (形式 1 RECORD 節) を使用する場合、ファイルに関連したレベル 01 レコード記述項目の長さにかかわらず、レコード長は *integer* 値になります。整数サイズは、そのデータ・セット項目の USAGE にかかわらず、レコード中のバイト数を示します。
- RECORD IS VARYING 節 (形式 3 RECORD 節) を使用する場合、*integer-1* および *integer-2* をコーディングするかどうかに基づいて、レコード長が決まります。

  integer-1 および *integer-2* (RECORD IS VARYING FROM *integer-1* TO *integer-2*) をコーディングする場合、ファイルに関連したレベル 01 レコード記述項目の長さにかかわらず、最大レコード長は *integer-2* 値です。整数サイズは、レコードのデータ項目の USAGE にかかわらず、最小および最大レコード長 (バイト数) を示します。

  integer-1 および *integer-2* を省略する場合、最大レコード長は、ファイルに関連付けられた最大のレベル 01 レコード記述項目のサイズであると判別されます。
- RECORD CONTAINS *integer-1* TO *integer-2* 節 (形式 2 RECORD 節) を使用する場合に、 *integer-1* および *integer-2* がファイルに関連したレベル 01 レコード記述項目の最小長および最大長 (バイト数) と一致している場合、最大レコード長は *integer-2* 値です。

- RECORD 節を省略する場合、最大レコード長は、ファイルに関連付けられた最大のレベル 01 レコード記述項目のサイズであると判別されます。

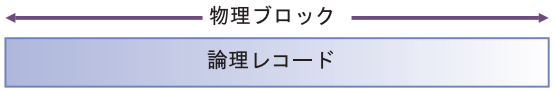
形式 U ファイルおよび READ INTO: 形式 U ファイルに READ INTO ステートメントを指定した場合、コンパイラーは、そのファイルの読み取りにのみ使用するレコードのサイズを使用する MOVE ステートメントを生成します。読み取られたレコードが、レベル 01 レコード記述に対応していない場合には、期待どおりの結果が得られないことがあります。 MOVE ステートメントに関するその他の規則はすべて適用されます。

関連タスク
169 ページの『固定長フォーマットの要求』
170 ページの『可変長フォーマットの要求』
173 ページの『スパン形式の要求』
168 ページの『レコード形式の指定』

関連参照
15 ページの『FILE SECTION 記入項目』
『形式 U レコードのレイアウト』

形式 U レコードのレイアウト:

形式 U では、外部ストレージの各ブロックは 1 つの論理レコードとして扱われます。レコード長やブロック長フィールドはありません。



関連概念
169 ページの『論理レコード』

関連タスク
175 ページの『不定形式の要求』

関連参照
170 ページの『形式 F レコードのレイアウト』
171 ページの『形式 V レコードのレイアウト』
174 ページの『形式 S レコードのレイアウト』

ブロック・サイズの設定

COBOL では、BLOCK CONTAINS 節を使用して物理レコードのサイズを設定します。この節を省略すると、コンパイラーはレコードがブロック化されないものと見なします。

ディスクまたはテープ上の QSAM ファイルをブロック化すると、処理速度が増し、ストレージ要件を最小限にすることができます。z/OS UNIX ファイル (HFS のファイルを含む)、PDSE メンバー、およびスプール・データ・セットをブロック化できますが、そのようにしてもシステムがデータを保管する方法に影響はありません。

BLOCK CONTAINS 節でブロック・サイズを明示的に設定する場合には、装置についての最大ブロック・サイズよりも大きくしてはなりません。BLOCK CONTAINS 節の CHARACTERS 句を指定する場合、レコード中のデータ項目の USAGE にかかわらず、サイズではレコード中のバイト数を示す必要があります。形式 F ファイルについて設定するブロック・サイズは、レコード長の整数倍でなければなりません。

プログラムでテープ上の QSAM ファイルを使用する場合は、少なくとも 12 から 18 バイトの物理ブロック・サイズを使用してください。そうでない場合、以下のいずれかのアクションの実行時にパリティー検査が行われると、ブロックはスキップオーバーされます。

- 12 バイトよりも小さいレコード・ブロックの読み取り
- 18 バイトよりも小さいレコード・ブロックの書き込み

一般に、ブロックが大きいとパフォーマンスは向上します。ブロック・サイズがわずか数キロバイトの場合は、特に効率が低下します。最低でも数十キロバイトのブロック・サイズを選択してください。レコードのブロック化を指定してブロック・サイズを省略すると、装置の使用効率とデータ転送速度を考慮した最適なブロック・サイズがシステムによって選択されます。

z/OS によるブロック・サイズの決定: パフォーマンスを最大化するには、COBOL ソース・プログラム内でブロック化ファイルのブロック・サイズを明示的に設定しないでください。新しいブロック化データ・セットの場合は、z/OS にシステム決定のブロック・サイズを提供させるようにする方が簡単です。このフィーチャーを使用するには、以下のガイドラインに従ってください。

- ソース・プログラムに BLOCK CONTAINS 0 をコーディングする。
- ソース・プログラムに RECORD CONTAINS 0 をコーディングしない。
- JCL DD ステートメントで、BLKSIZE 値をコーディングしない。

明示的なブロック・サイズの設定: ブロック・サイズを明示的に設定したい場合は、以下の指針に従うと、プログラムが最も柔軟になります。

- ソース・プログラムに BLOCK CONTAINS 0 をコーディングする。
- DD 名定義 (JCL DD ステートメント) で、BLKSIZE 値をコーディングする。

z/OS における拡張形式データ・セットの場合、DFSMS™ は物理レコードに 32 バイトのブロック接尾部を追加します。ブロック・サイズを明示的に指定する (JCL または ISPF を介して) 場合には、このブロック接尾部のサイズをブロック・サイズに含めないでください。このブロック接尾部は、プログラムの中で使用することはできません。z/OS DFSMS は、ブロック接尾部を読み取るためのスペースを割り振ります。ただし、直接アクセス装置の 1 つのトラックに収まる拡張フォーマット・データ・セットのブロック数を計算するとき、ブロック・サイズにブロック接尾部のサイズを含める必要があります。

BLOCK CONTAINS 節に直接、または BLOCK CONTAINS *n* RECORDS を使用して間接的に 32760 より大きなブロック・サイズを指定した場合には、データ・セットをテープに定義していない限り、データ・セットの OPEN は、ファイル状況コード 90 で失敗します。

既存のブロック化データ・セットの場合は、次のようにすると最も簡単です。

- ソース・プログラムに BLOCK CONTAINS 0 をコーディングする。
- DD 名定義に BLKSIZE 値をコーディングしません。

DD 名定義から BLKSIZE を省くと、ブロック・サイズは、システムによってデータ・セット・ラベルから自動的に取得されます。

LBI の利用: 大きなブロック・サイズ用のラージ・ブロック・インターフェース (LBI) を使用して、テープ・データ・セットのパフォーマンスを向上させることができます。 LBI が使用可能であれば、COBOL 実行時には、システム決定のブロック・サイズを使用するテープ・ファイルに対して、この機能が自動的に使用されます。LBI は、JCL または BLOCK CONTAINS 節でブロック・サイズが明示的に定義されるファイルでも、使用されます。LBI を使用すると、ブロック・サイズが 32760 (磁気テープ装置でサポートされる場合) を超えても構いません。

LBI はどんな場合にでも使用されるわけではありません。以下の場合には、32760 を超えるブロック・サイズを使用しようとすると、コンパイル時に診断され、OPEN 障害となります。

- スパン・レコード
- OPEN I-O

32760 を超えるブロック・サイズを使用すると、別のシステムでテープを読み取れなくなる場合があります。32760 より大きなブロック・サイズで作成したテープは、32760 より大きなブロック・サイズをサポートする磁気テープ装置を持つシステムでしか読み取ることができません。ファイル、装置、またはオペレーティング・システム・レベルに関して、指定したブロック・サイズが大きすぎると、ランタイム・メッセージが出されます。

システム決定のブロック・サイズを 32760 に限定するには、どこにも BLKSIZE を指定せずに、以下のいずれかの項目を 32760 に設定してください。

- データ・セットの DD ステートメントの BLKSZLIM キーワード
- BLKSZLIM キーワードを使用した、データ・クラスの BLKSZLIM (システム・プログラマーによる設定が必要)
- キーワード TAPEBLKSZLIM を使用して、SYS1.PARMLIB の DEVSUP*xx* メンバーにあるシステムのブロック・サイズ限界 (システム・プログラマーによる設定が必要)

ブロック・サイズ限界は、以下の項目を検査してコンパイラーによって検出される最初のゼロ以外の値です。

BLKSIZE または BLKSZLIM 値をどのソースからも利用できないと、システムは BLKSIZE を 32760 に限定します。その後、32760 を超えるブロック・サイズを以下のいずれかの方法で使用可能にすることができます。

- ファイルに対する DD ステートメントで、32760 より大きい BLKSZLIM 値を指定し、COBOL ソースで BLOCK CONTAINS 0 を使用する。
- DD ステートメントまたは COBOL ソースの BLOCK CONTAINS 節で、32760 より大きな値を BLKSIZE に指定する。

BLKSZLIM は、装置独立です。

ブロック・サイズおよび DCB RECFM サブパラメーター: z/OS のもとでは、DCB RECFM サブパラメーターに S または T オプションをコーディングすることができます。

- 標準ブロック (ファイルの最後のブロックを除き、ファイル内に切り捨てられたブロックまたは埋められていないトラックがないもの) だけを持つ形式 F レコードの場合は、DCB RECFM サブパラメーターで S (標準) オプションを使用してください。S は、テープ上のレコードに関してもサポートされます。これは、レコードが DASD またはテープ上にない場合には無視されます。

  この標準ブロック・オプションを使用すると、特に直接アクセス装置の場合は、入出力のパフォーマンスが向上する可能性があります。
- QSAM ファイルの場合、T (トラック・オーバーフロー) オプションはもう有用ではありません。

関連タスク
167 ページの『COBOL での QSAM ファイルおよびレコードの定義』
z/OS DFSMS: Using Data Sets

関連参照
15 ページの『FILE SECTION 記入項目』
BLOCK CONTAINS 節 (*Enterprise COBOL 言語解説書*)

QSAM ファイルの入出力ステートメントのコーディング

QSAM ファイル、または HFS 内のバイト・ストリーム・ファイルを QSAM (OPEN、READ、WRITE、REWRITE、および CLOSE) を使用して処理するには、次の入出力ステートメントをコーディングします。

OPEN ファイルの処理を開始します。どの QSAM ファイルも、INPUT、OUTPUT、または EXTEND (装置の能力に応じて) としてオープンすることができます。

直接アクセス・ストレージ・デバイス上の QSAM ファイルは、I-O としてオープンすることもできます。 HFS ファイルを I-O として開くことはできません。そうしようとすると、ファイル状況は 37 になります。

READ ファイルからレコードを読み取ります。順次処理では、プログラムは、レコードをファイルの作成時に入力されたのと同じ順序で 1 つずつ読み取ります。

WRITE ファイル内にレコードを作成します。プログラムは、ファイルの終わりに新しいレコードを書き込みます。

REWRITE
レコードを更新します。HFS 内のファイルは REWRITE を使用して更新することはできません。

CLOSE ファイルとプログラムの間の接続を解放します。

関連タスク 180 ページの『QSAM ファイルのオープン』 181 ページの『QSAM ファイルへのレコードの追加』 182 ページの『QSAM ファイルの更新』 182 ページの『QSAM ファイルのプリンターまたはスプール・データ・セットへの書き込み』 183 ページの『QSAM ファイルのクローズ』

関連参照
OPEN ステートメント (*Enterprise COBOL 言語解説書*)
READ ステートメント (*Enterprise COBOL 言語解説書*)
WRITE ステートメント (*Enterprise COBOL 言語解説書*)
REWRITE ステートメント (*Enterprise COBOL 言語解説書*)
CLOSE ステートメント (*Enterprise COBOL 言語解説書*)
ファイル状況キー (*Enterprise COBOL 言語解説書*)

QSAM ファイルのオープン

プログラムで READ、WRITE、または REWRITE ステートメントを使用してファイルのレコードを処理するためには、その前に、そのファイルを OPEN ステートメントでオープンしておかなければなりません。

OPEN ステートメントが機能するのは、以下の両方の条件が真である場合です。

- ファイルが使用可能であるか、または動的に割り振られている。
- DD 名定義またはファイルのデータ・セット・ラベルにコーディングされた*固定ファイル属性* が、SELECT 節および FD 項目でそのファイル用にコーディングされた属性と一致している。

  ファイル編成属性、コード・セット、最大レコード・サイズ、またはレコード・フォーマット (固定または可変) で不一致があると、ファイル状況コードは 39 になり、OPEN ステートメントは失敗します。 HFS のファイルのオープン時には、最大レコード・サイズおよびレコード・フォーマットの不一致はエラーではありません。

  固定長 QSAM ファイルの場合、FD 項目に RECORD CONTAINS 0 をコーディングしても、レコード・サイズ属性は矛盾しません。レコード・サイズは DD ステートメントまたはデータ・セット・ラベルから取られ、OPEN ステートメントは正常に終了します。

プログラムの実行中にファイルが再度オープンされないように、CLOSE WITH LOCK をコーディングしてください。

テープ・ファイルを逆順に処理するには、OPEN ステートメントの REVERSED オプションを使用してください。ファイルは最後に位置付けられ、READ ステートメントでは、データ・レコードが逆順に読み取られます。REVERSED オプションがサポートされるのは、固定長レコードを持つファイルの場合だけです。

関連タスク
181 ページの『QSAM ファイルの動的作成』
189 ページの『ファイル属性をプログラムと一致させる』

関連参照
OPEN ステートメント (*Enterprise COBOL 言語解説書*)

QSAM ファイルの動的作成

ある QSAM ファイルがオペレーティング・システム上で利用可能になっていないときに、COBOL プログラムがそのファイルの作成を指定することがあります。そのような場合、ファイルは動的に作成されます。

(QSAM ファイルが z/OS 上で*使用可能*であると見なされるのは、有効な DD ステートメント、環境変数用の export コマンド、または TSO ALLOCATE コマンドを使用してオペレーティング・システムに認識されている場合です。)そうしないと、ファイルは*利用不能*です。

DD ステートメントの DD 名のミススペルは、DD ステートメントが欠落しているのと同じです。また、無効な値を指定した環境変数は、変数が設定されていないのと同じです。

ランタイム・オプション CBLQDA を使用しており、以下の状況のいずれか 1 つが存在していれば、その QSAM ファイルが暗黙的に作成されます。

- オプション・ファイルが EXTEND または I-O として開かれている。

  オプション・ファイルとは、プログラムが実行されるたびに、必ずしも使用可能である必要はないファイルです。 INPUT、I-O、または EXTEND モードで開かれたファイルは、FILE-CONTROL 段落で SELECT OPTIONAL 節をコーディングすることによって、オプション・ファイルとして定義されます。
- OPTIONAL 句に関係なく、そのファイルが OUTPUT 用にオープンされている。

ファイルは、インストール先で設定されたシステム・デフォルト属性と、プログラムの SELECT 節および FD 記入項目でコーディングされた属性を用いて割り振られます。

この暗黙の割り振りメカニズムを、環境変数を使用して行うファイルの明示的な動的割り振りと混乱しないでください。明示的な動的割り振りでは、有効な環境変数が設定されている必要があります。 CBLQDA サポートが使用されるのは、上述されたように QSAM ファイルが利用できず、有効な環境変数が設定されていない場合だけです。

z/OS のもとでは、CBLQDA オプションを使用して作成されるファイルは一時データ・セットであり、プログラムの実行後は存在しません。

関連タスク
180 ページの『QSAM ファイルのオープン』

QSAM ファイルへのレコードの追加

QSAM ファイルに追加を行うためには、ファイルを EXTEND としてオープンし、WRITE ステートメントを使用して、ファイルの最後のレコードの直後にレコードを追加します。

I-O としてオープンされたファイルにレコードを追加したい場合には、ファイルをいったんクローズしてから、EXTEND としてオープンしなければなりません。

関連参照
READ ステートメント (*Enterprise COBOL 言語解説書*)
WRITE ステートメント (*Enterprise COBOL 言語解説書*)

QSAM ファイルの更新

QSAM ファイルを更新できるのは、それが直接アクセス・ストレージ・デバイスに存在する場合のみです。HFS のファイルを更新することはできません。

次のようにして、既存のレコードを同じ長さの別のレコードで置き換えてください。

1. ファイルを I-O としてオープンします。
2. 既存のレコードを更新するには、REWRITE を使用します。(REWRITE の前の最後のファイル処理ステートメントは、成功した READ ステートメントでなければなりません。)

圧縮フォーマットで割り振った拡張フォーマットのデータ・セットを、I-O としてオープンすることはできません。

関連参照
REWRITE ステートメント (*Enterprise COBOL 言語解説書*)

QSAM ファイルのプリンターまたはスプール・データ・セットへの書き込み

COBOL には、印刷されるページのサイズを制御するため、およびレコードの垂直位置決めを制御するための言語ステートメントが用意されています。

ページ・サイズの制御: 印刷されるページのサイズ (ページの上部マージンと下部マージンの間の行数および脚注域の行数) を制御するには、FD 記入項目の LINAGE 節を使用します。 LINAGE 節を使用すると、COBOL では ADV コンパイラー・オプションも要求されたかのようにファイルを処理します。

LINAGE 節を WRITE BEFOREIAFTER ADVANCING *nn* LINES と組み合わせて使用する場合は、注意して値を設定してください。 ADVANCING *nn* LINES 句が使用されると、COBOL はまず、LINAGE-COUNTER と *nn* の合計を計算します。それ以降のアクションは、*nn* のサイズによって異なります。LINAGE-COUNTER が増加されると、END-OF-PAGE 命令句が実行されます。したがって、LINAGE-COUNTER は、END-OF-PAGE 句の実行時に、現行の脚注域ではなく次の論理ページを指していることがあります。

AT END-OF-PAGE または NOT AT END-OF-PAGE 命令句が実行されるのは、書き込み操作が正常に完了した場合だけです。書き込み操作が成功しなかった場合には、制御が WRITE ステートメントの終わりに渡され、すべての条件句が省略されます。

レコードの垂直位置決めの制御: 印刷ページに書き込むそれぞれのレコードの垂直位置決めを制御するには、WRITE ADVANCING ステートメントを使用します。

BEFORE ADVANCING は、ページが送られる前にレコードを印刷します。 AFTER ADVANCING は、ページが送られた後でレコードを印刷します。

ADVANCING の後に、ページが送られることになる行数を整数 (または *mnemonic-name* を指定した *identifier*) で指定してください。 WRITE ステートメントで、ADVANCING 句を省略した場合、その効果は次のようにコーディングした場合と同じです。

```
AFTER ADVANCING 1 LINE
```

関連参照
WRITE ステートメント (*Enterprise COBOL 言語解説書*)

QSAM ファイルのクローズ

プログラムを QSAM ファイルから切り離すには、CLOSE ステートメントを使用します。既にクローズされているファイルをクローズしようとすると、論理エラーになります。

QSAM ファイルをクローズしなくても、以下の条件のもとでは、ファイルは自動的にクローズされます。ただし、実行単位内の OS/VS COBOL プログラムのいずれかで定義されたファイルを除きます。

- 実行単位が正常に終了すると、実行単位内の COBOL プログラムのいずれかで定義されている、オープン状態のすべてのファイルはクローズされます。
- 実行単位が異常終了した場合は、TRAP(ON) ランタイム・オプションが有効化されていれば、その実行単位内の COBOL プログラムのいずれかで定義されている、オープン状態のすべてのファイルはクローズされます。
- 言語環境プログラムの条件処理が完了し、その条件が起こったルーチン以外のルーチンでアプリケーションが処理を再開する場合は、実行単位の中で COBOL プログラム (再度呼び出されて再入される可能性のある) のいずれかで定義されている、オープン状態のすべてのファイルがランタイムによって閉じられます。

  (条件が処理された後で) プログラムが実行を再開する位置を変更することができます。このためには、言語環境プログラムの CEEMRCR 呼び出し可能サービスを使用して再開カーソルを移動するか、または C longjmp のような言語構造体を使用します。
- COBOL サブプログラムに CANCEL を使用すると、そのプログラムに定義されているオープン状態の非外部ファイルは実行時にクローズされます。
- INITIAL 属性を持つ COBOL サブプログラムが制御権を戻すと、そのプログラムに定義されているオープン状態の非外部ファイルは実行時にクローズされます。
- マルチスレッド・アプリケーションのスレッドが終了すると、同じスレッド内部からオープンした外部ファイルと非外部ファイルはいずれもクローズされます。

このような暗黙の CLOSE 操作が実行されたときは、DATA DIVISION のファイル状況キー・データ項目が設定されますが、EXCEPTION/ERROR および LABEL 宣言は呼び出されません。

エラー: マルチスレッド・アプリケーションで QSAM ファイルをオープンした場合は、ファイルをオープンしたのと同じ実行スレッドからクローズする必要があります。異なるスレッドからファイルをクローズしようとすると、ファイル状況コードの条件 90 でクローズが失敗します。

関連参照
CLOSE ステートメント (*Enterprise COBOL 言語解説書*)

QSAM ファイルのエラーの処理

入力ステートメントまたは出力ステートメントが失敗しても、COBOL がユーザーに代わって訂正処置を行うことはありません。重大エラーではない入出力エラーが発生した後で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。

COBOL は、特定の QSAM 入出力エラーを代行受信して処理するために、以下の方法を提供します。

- ファイル終了句 (AT END)
- EXCEPTION/ERROR 宣言
- FILE STATUS 節
- INVALID KEY 句

FILE STATUS キーまたは宣言をコーディングしなかった場合に、重大な QSAM 処理エラーが起こると、メッセージが出され、言語環境プログラム条件が合図されます。ランタイム・オプション ABTERMENC(ABEND) が指定されていると、この条件の結果として、異常終了が引き起こされます。

FILE STATUS 節または EXCEPTION/ERROR 宣言を使用する場合は、そのファイルに対する DD ステートメントの DCB で EROPT=ACC をコーディングしてください。そうしなければ、COBOL プログラムは、いくつかのエラー条件が発生した後で処理を継続することができなくなります。

FILE STATUS 節を使用する場合は、必ずキーを調べ、キー値に基づいて適切なアクションを取るようにしてください。キーを調べなくても、プログラムが実行を継続できる可能性がありますが、予期したものではない結果になることがあります。

関連タスク
265 ページの『入出力操作でのエラーの処理』

QSAM ファイルの操作

COBOL プログラムで QSAM ファイルを操作するには、定義、割り振り、取得を行うとともに、そのファイル属性がプログラムのものと確実に一致するようにします。また、ストライプ拡張フォーマット QSAM データ・セットを使用して、パフォーマンスの向上に役立てることができます。

関連タスク
185 ページの『QSAM ファイルの定義および割り振り』
187 ページの『QSAM ファイルの検索』
189 ページの『ファイル属性をプログラムと一致させる』
191 ページの『ストライプ拡張形式 QSAM データ・セットの使用』

関連参照
192 ページの『QSAM ファイル用のバッファーの割り振り』

QSAM ファイルの定義および割り振り

DD ステートメントまたは環境変数を使用して、QSAM ファイルまたは HFS のバイト・ストリーム・ファイルを定義できます。これらのファイルの割り振りは、COBOL ファイルの割り振りに関する一般的な規則に従います。

環境変数を使用する場合、名前は大文字でなければなりません。次のいずれかの方法で、MVS データ・セットを指定します。

- DSN(*dataset-name*)
- DSN(*dataset-name*(*member-name*))

dataset-name は完全に修飾されていなければならず、一時データ・セットにすることはできません (つまり、& で開始させないでください)。

制約事項: 環境変数を使用して PDS や PDSE を作成することはできません。

オプションとして、DSN の後に、以下の属性を任意の順序で指定することができます。

- 後処理の値 (NEW、OLD、 SHR、または MOD)
- TRACKS または CYL
- SPACE(*nnn*,*mmm*)
- VOL(*volume-serial*)
- UNIT(*type*)
- KEEP、DELETE、CATALOG、または UNCATALOG
- STORCLAS(*storage-class*)
- MGMTCLAS(*management-class*)
- DATACLAS(*data-class*)

環境変数または DD 定義を使用して、HFS のファイルを定義することができます。そうするには、ASSIGN 節の外部名と一致する名前を使用して、以下のいずれかの項目を定義してください。

- PATH='*absolute-path-name*' および FILEDATA=BINARY を使用する DD 割り振り
- 値 PATH(*pathname*) を持つ環境変数 (ここで、*pathname* は / で始まる絶対パス名です)

COBOL for OS/390 & VM バージョン 2 リリース 2 より前の COBOL のリリースとの互換性を与えるために、HFS ファイルの DD 割り振りを使用するときに FILEDATA=TEXT を指定することもできますが、この使用はお勧めできません。HFS のテキスト・ファイルを処理する場合は、LINE SEQUENTIAL 編成を使用してください。QSAM を使用して HFS のテキスト・ファイルを処理する場合は、環境変数を使用してファイルを定義することはできません。

QSAM ファイルを定義する場合には、以下のパラメーターを使用します。

表 20. **QSAM ファイル割り振り**

目的	使用する DD パラメーター	使用する EV キーワード
ファイルに名前を付ける。	DSNAME (データ・セット名)	DSN
ファイルに割り振る入出力装置のタイプと数量を選択する。	UNIT	タイプの UNIT の場合のみ
ファイルが常駐するボリュームと、ボリュームの取り付けについて指示を与える。	VOLUME (またはシステムに出力ボリュームを選択させる)	VOL
ファイルに必要なスペースのタイプおよび量を割り振る。 (直接アクセス・ストレージ・デバイスの場合のみ)	SPACE	スペースの量 (1 次と 2 次のみ) の場合は、SPACE。スペースのタイプの場合は、TRACKS?または CYL。
ファイルに関連付けられるラベルのタイプおよび内容の一部を指定する。	LABEL	n/a
ジョブ・ステップの完了後にファイルをカタログするか、後続ステップに渡すか、保存するかを指示する。	DISP	NEW、OLD、SHR、MOD に、KEEP、DELETE、CATALOG、または UNCATALOG の指定を追加したもの。
追加したいすべてのデータ制御ブロック情報を完成させる。	DCB サブパラメーター	n/a

QSAM ファイルの一部の情報は常に、FILE-CONTROL 段落、FD エントリー、およびその他の COBOL 節にコード化する必要があります。その他の情報は、出力ファイルについての DD ステートメントまたは環境変数でコーディングする必要があります。入力ファイルの場合、システムはファイル・ラベルから情報を取得することができます (標準ラベル・ファイルの場合)。入力ファイルについての DD ステートメントで DCB 情報が指定されると、それはデータ・セット・ラベル上の情報をオーバーライドします。例えば、新規の直接アクセス装置ファイルに割り振られるスペースの量は、DD ステートメントで SPACE パラメーターを使用して設定することができます。

QSAM ファイルの一部の特性は、COBOL 言語では表現できませんが、ファイルの DD ステートメントで DCB パラメーターを使用してコーディングすることができます。 DCB パラメーターのサブパラメーターで、システムがデータ・セット定義を完成させるのに必要な、以下の項目を含む情報を指定してください。

- コンパイル時に、BLOCK CONTAINS 0 RECORDS がコーディングされた場合の、ブロック・サイズ (BLKSIZE=) (推奨方法)。
- レコードの書き込みまたは読み取りにおいてエラーが発生した場合に実行されるオプション。
- TRACK OVERFLOW または標準ブロック。
- カード読取装置または穿孔装置の操作モード。

DD DUMMY にコーディングされた DCB 属性は、COBOL プログラムの FD 記入項目にコーディングされた属性をオーバーライドしません。

492 ページの『例: 環境変数の設定とアクセス』

関連タスク

176 ページの『ブロック・サイズの設定』
167 ページの『COBOL での QSAM ファイルおよびレコードの定義』
164 ページの『ファイルの割り振り』

関連参照

『QSAM ファイルを作成するためのパラメーター』
MVS プログラム管理: ユーザーズ・ガイドおよび解説書

QSAM ファイルを作成するためのパラメーター

次に示す DD ステートメントのパラメーターは、QSAM ファイルを作成するために頻繁に使用されます。

```
DSNAME=  [ dataset-name                          ]
DSN=     [ dataset-name(member-name)             ]
         [ &&name                                ]
         [ &&name(member-name)                   ]

UNIT=    ( name[,unitcount] )

VOLUME=  ( [PRIVATE] [,RETAIN] [,vol-sequence-num] [,volume-count] ...
VOL=
                              ... [ ,SER=(volume-serial[,volume-serial]...) ] )
                                  [ ,REF=[ dsname                           ]
                                         [ *.ddname                         ]
                                         [ *.stepname.ddname                ]
                                         [ *.stepname.procstep.ddname       ]

SPACE=   ([ TRK                  ],(primary-quantity[,secondary-quantity][,directory-quantity]))
         [ CYL                   ]
         [ average-record-length ]

LABEL=   ( [Data-set-sequence-number,][ NL  ] [ ,EXPDT=[ yyddd     ] ] )
                                      [ SL  ] [        [ yyyy/ddd  ] ]
                                      [ SUL ] [ ,RETPD=xxxx          ]

DISP=    ( [ NEW ] [ ,DELETE ] [ ,DELETE ] )
           [ MOD ] [ ,KEEP   ] [ ,KEEP   ]
                   [ ,PASS   ] [ ,CATLG  ]
                   [ ,CATLG  ]

DCB=     ( subparameter-list )
```

関連タスク

185 ページの『QSAM ファイルの定義および割り振り』

QSAM ファイルの検索

QSAM ファイルは、カタログされていてもいなくても、ジョブ制御ステートメントまたは環境変数を使用して検索することができます。

カタログされたファイル

ボリュームやスペースなど、すべてのデータ・セット情報は、カタログおよびファイル・ラベルに保管されています。コーディングする必要があるの

は、データ・セット名と後処理方法です。DD ステートメントを使用する場合、これは DSNAME パラメーターと DISP パラメーターです。環境変数を使用する場合、これは DSN パラメーターとパラメーター OLD、SHR、または MOD のいずれか 1 つです。

カタログされていないファイル
一部の情報はファイル・ラベルに保管されていますが、*dsname* および後処理方法のほかにユニットおよびボリューム情報をコーディングしなければなりません。

JCL を使用しており、そのファイルを現行ジョブ・ステップまたは現行ジョブ内の前のジョブ・ステップで作成した場合は、前の DD ステートメントを参照してデータ・セット情報の大部分を得ることができます。ただし、DSNAME および DISP はコーディングする必要があります。

関連参照
『QSAM ファイルを検索するためのパラメーター』

QSAM ファイルを検索するためのパラメーター

次に示す DD ステートメントのパラメーターは、前に作成したファイルを検索するために使用されます。

```
DSNAME=  [ dataset-name                    ]
DSN=     [ dataset-name(member-name)       ]
         [ *.ddname                        ]
         [ *.stepname.ddname               ]
         [ &&name                          ]
         [ &&name(member-name)             ]

UNIT=    ( name[,unitcount] )

VOLUME=  ( subparameter-list )
VOL=

LABEL=   ( subparameter-list )


DISP=    ( [ OLD    ] [ ,DELETE  ] [ ,DELETE  ] )
           [ SHR    ] [ ,KEEP    ] [ ,KEEP    ]
           [ MOD    ] [ ,PASS    ] [ ,CATLG   ]
                      [ ,CATLG   ] [ ,UNCATLG ]
                      [ ,UNCATLG ]

DCB=     ( subparameter-list )
```

関連タスク
187 ページの『QSAM ファイルの検索』

ファイル属性をプログラムと一致させる

DD ステートメントまたはデータ・セット・ラベルにコーディングされた固定ファイル属性と、SELECT 節および FD 項目でそのファイル用にコーディングされた属性が整合していない場合、プログラムの OPEN ステートメントが機能しないことがあります。

ファイル編成、レコード・フォーマット (固定長または可変長)、レコード長、またはコード・セットの属性が一致していないと、ファイル状況コードが 39 になり、OPEN ステートメントが失敗します。HFS のファイルについては、例外があります。すなわち、レコード形式およびレコード長が一致していなくても、HFS ではエラーとなりません。

一般的なファイル状況 39 の問題が起こらないようにするには、既存ファイルまたは新規ファイルの処理に関するガイドラインに従ってください。

DD または TSO ALLOCATE コマンドでファイルを使用可能にしていなかった場合でも、COBOL プログラムでファイルが作成されるように指定している場合、Enterprise COBOL はファイルを動的に割り振ります。ファイルがオープンされるときには、プログラム内でコーディングされたファイル属性が使用されます。このため、ファイル属性の矛盾を心配する必要はありません。

JCL または環境変数内の情報は、データ・セット・ラベル内の情報をオーバーライドするので注意してください。

関連タスク
『既存ファイルの処理』
190 ページの『新規ファイルの処理』
180 ページの『QSAM ファイルのオープン』

関連参照
15 ページの『FILE SECTION 記入項目』

既存ファイルの処理

プログラムで既存のファイルを処理するときには、COBOL プログラム内のファイルの記述をデータ・セットのファイル属性と矛盾しないようにコーディングします。以下の指針を使用して、最大レコード長を定義します。

表 21. **QSAM ファイルの最大レコード長**

形式:	指定:
V または S	データ・セットの長さ属性よりも 4 バイト小さくする。
F	データ・セットの長さ属性と同じ値。
U	データ・セットの長さ属性と同じ値。

プログラムで可変長 (フォーマット V) レコードを定義する最も簡単な方法は、 FD 項目で RECORD IS VARYING FROM *integer-1* TO *integer-2* 節を使用して、*integer-2* に適切な値を設定する方法です。レコードのデータ項目の基礎となる USAGE にかかわらず、整数サイズをバイト数で表してください。 例えば、データ・セットの長さ属性を 104 バイト (LRECL=104) にすると想定しましょう。最大レコード長がレベル

01 レコード記述ではなく RECORD IS VARYING 節から判別されることを念頭に置いて、プログラム内で以下のコードを使用して形式 V ファイルを定義することができます。

```
FILE SECTION.
FD  COMMUTER-FILE-MST
    RECORDING MODE IS V
    RECORD IS VARYING FROM 4 TO 100 CHARACTERS.
01  COMMUTER-RECORD-A   PIC X(4).
01  COMMUTER-RECORD-B   PIC X(75).
```

前の例の既存のファイルが、形式 V ではなく形式 U である場合を想定します。104 バイトがすべてユーザー・データである場合、プログラム内で以下のコードを使用してファイルを定義することができます。

```
FILE SECTION.
FD  COMMUTER-FILE-MST
    RECORDING MODE IS U
    RECORD IS VARYING FROM 4 TO 104 CHARACTERS.
01  COMMUTER-RECORD-A   PIC X(4).
01  COMMUTER-RECORD-B   PIC X(75).
```

プログラムで固定長レコードを定義するには、RECORD CONTAINS *integer* 節をコーディングするか、またはこの節は省略し、すべてのレベル 01 レコード記述が同じ固定サイズになるようにコーディングしてください。どちらの場合も、データ・セットの長さ属性の値と等しい値を使用してください。同じプログラムを使用して実行時に種々のファイルを処理しようとする場合に、それらのファイルの固定長が異なっている場合、RECORD CONTAINS 0 をコーディングすることによりレコード長の矛盾を回避してください。

既存のファイルが ASCII データ・セット (DCB=(OPTCD=Q)) である場合には、ファイルについての FD 記入項目で CODE-SET 節を使用しなければなりません。

関連タスク
『新規ファイルの処理』
169 ページの『固定長フォーマットの要求』
170 ページの『可変長フォーマットの要求』
175 ページの『不定形式の要求』
180 ページの『QSAM ファイルのオープン』

関連参照
15 ページの『FILE SECTION 記入項目』

新規ファイルの処理

COBOL プログラムが、プログラム実行前に使用可能にされる新規ファイルにレコードを書き込む場合、DD ステートメント、環境変数、または割り振りでのファイル属性が、プログラム内の属性と矛盾しないようにしてください。

通常、ファイルを事前定義する場合には、最小限のパラメーターをコーディングするだけで済みます。ただし、データ・セットの長さ属性を明示的に設定する必要がある場合 (例えば、ISPF 割り振りパネルを使用している場合、または DD ステートメントがバッチ・ジョブ用であり、その中でプログラムが RECORD CONTAINS 0 を使用する場合)、次の指針に従ってください。

- 形式 V および形式 S ファイルの場合は、プログラム内で定義されているものよりも 4 バイト大きい長さ属性を設定します。
- 形式 F および形式 U ファイルの場合は、プログラム内で定義されているものと同じ長さ属性を設定します。
- ファイルを OUTPUT としてオープンし、それをプリンターに書き込む場合は、ADV コンパイラー・オプションと、プログラムで使用する言語によって、コンパイラーがレコード長に紙送り制御文字のための 1 バイトを追加することがあります。そのような場合には、LRECL 値をコーディングする際に、追加されるバイトを長さに含めてください。

例えば、プログラムに可変長レコードを含むファイルの以下のコードが入っている場合、DD ステートメントまたは割り振りにおける LRECL 値は 54 になります。

```
FILE SECTION.
FD  COMMUTER-FILE-MST
    RECORDING MODE IS V
    RECORD CONTAINS 10 TO 50 CHARACTERS.
01  COMMUTER-RECORD-A   PIC X(10).
01  COMMUTER-RECORD-B   PIC X(50).
```

関連タスク

189 ページの『既存ファイルの処理』
169 ページの『固定長フォーマットの要求』
170 ページの『可変長フォーマットの要求』
175 ページの『不定形式の要求』
180 ページの『QSAM ファイルのオープン』
181 ページの『QSAM ファイルの動的作成』

関連参照

15 ページの『FILE SECTION 記入項目』

ストライプ拡張形式 QSAM データ・セットの使用

ストライプ拡張形式 QSAM データ・セットは、大容量のデータを含んだファイルを処理するアプリケーションや入出力操作に要する時間が全体のパフォーマンスに大きな影響を与えるアプリケーションにメリットをもたらすことがあります。

ストライプ拡張形式 QSAM データ・セットは、複数のボリュームにわたる拡張形式の QSAM データ・セットであるため、並列データ・アクセスを可能にします。

QSAM ストライプ・データ・セットを使用することから最大限の効果を得るためには、z/OS DFSMS が必要とされる数のバッファーを 16MB 境界より上に割り振ることができなければなりません。QSAM ストライプ・データ・セットに割り振られるファイルを含むアプリケーションを開発するときには、これらの指示に従ってください。

- バッファーを 16MB 境界より上に割り振ることができないファイルについては、QSAM ストライプ・データ・セットの使用を避けます。
- ファイルについての FILE-CONTROL 記入項目で RESERVE 節を省略します。これによって、 z/OS DFSMS がデータ・セットについての最適なバッファー数を判別できるようになります。

- DATA(31) および RENT 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し、ロード・モジュールを AMODE(31) にします。
- ファイルが形式 F、形式 V、または形式 U レコードを持つ EXTERNAL ファイルである場合は、ALL31(ON) ランタイム・オプションを指定します。

すべてのストライプ・データ・セットは拡張フォーマット・データ・セットですが、すべての拡張フォーマット・データ・セットがストライプ・データ・セットというわけではないことに注意してください。

関連タスク
z/OS DFSMS: Using Data Sets

関連参照
『QSAM ファイル用のバッファーの割り振り』

QSAM ファイル用のバッファーの割り振り

z/OS DFSMS は、QSAM ファイルに合わせて、16 MB 境界より上または下にファイルの入出力データを保管するためのバッファーを自動的に割り振ります。

ほとんどの QSAM ファイルのバッファーは、16MB 境界より上に割り振られます。例外は次のとおりです。

- AMODE 24 で実行されるプログラム。
- DATA(24) および RENT オプションを指定してコンパイルされたプログラム。
- NORENT および RMODE(24) オプションを指定してコンパイルされたプログラム。
- NORENT および RMODE(AUTO) オプションを指定してコンパイルされたプログラム。
- EXTERNAL ファイル (ALL31(OFF) ランタイム・オプションが指定されている場合)。ALL31(ON) ランタイム・オプションを指定するためには、実行単位内のすべてのプログラムが 31 ビット・アドレッシング・モードで実行可能でなければなりません。
- TSO 端末に割り振られるファイル。
- 形式 S (スパン) レコードを持つファイル (ファイルが以下のいずれかである場合)。
  - EXTERNAL ファイル (ALL31(ON) が指定されている場合でも)
  - I-O-CONTROL 段落の SAME RECORD AREA 節で指定されたファイル
  - I-O としてオープンされ、REWRITE ステートメントを使用して更新されるブロック化ファイル

関連概念
45 ページの『ストレージとそのアドレス可能度』

関連タスク
191 ページの『ストライプ拡張形式 QSAM データ・セットの使用』

QSAM を使用する HFS ファイルへのアクセス

階層ファイル・システム (HFS) 内のバイト・ストリーム・ファイルを、QSAM を使用して ORGANIZATION SEQUENTIAL ファイルとして処理できます。これを行うためには、DD 名または環境変数のいずれか 1 つを ASSIGN 節の *assignment-name* として指定してください。

DD 名 キーワード PATH= および FILEDATA=BINARY によってファイルを識別する DD 割り振り。

環境変数名
ファイルの HFS パスの実行時値を持つ環境変数

以下の制約事項に従ってください。

- スパン・レコード形式はサポートされません。
- OPEN I-O および REWRITE はサポートされません。これらの操作の 1 つを行おうとすると、次のファイル状況条件の 1 つになります。
  - OPEN I-O からは 37。
  - REWRITE からは 47 (ファイルを I-O として正常にオープンできなかったため)。

使用上の注意

- 次のタイプのどちらかの矛盾については、ファイル状況 39 (固定ファイル属性の矛盾) が強制されません。
  - レコード長の矛盾
  - レコード・タイプの矛盾 (固定と可変の違い)
- READ は、ファイルの最大論理レコード・サイズのバイト数を戻します (ただし、最後のレコードは別で、それより短い可能性があります)。

  例えば、ファイル定義に、3、5、および 10 バイトの長さのレベル 01 レコード記述があり、3 つのレコード「abc」、「defgh」、および「ijklmnopqr」をこの順に書き込むとしましょう。このファイルの最初の READ は 'abcdefghij' を戻し、2 番目の READ は 'klmnopqr ' を戻し、3 番目の READ は AT END 条件となります。

COBOL for OS/390 & VM バージョン 2 リリース 2 より前の IBM COBOL のリリースとの互換性を与えるために、HFS ファイルの DD 割り振りを使用するときに FILEDATA=TEXT を指定することもできますが、この使用はお勧めできません。HFS のテキスト・ファイルを処理する場合は、LINE SEQUENTIAL 編成を使用してください。QSAM を使用して HFS のテキスト・ファイルを処理する場合は、環境変数を使用してファイルを定義することはできません。

関連タスク
164 ページの『ファイルの割り振り』
185 ページの『QSAM ファイルの定義および割り振り』
z/OS DFSMS: Using Data Sets (HFS データ・セットの使用)

QSAM ファイルのラベル

ラベルを使用して、磁気テープ、直接アクセス・ボリューム、およびデータ・セットを識別することができます。オペレーティング・システムでは、ラベル処理ルーチンを使用して、ラベルを識別および検査し、ボリュームとデータ・セットの位置を突き止めます。

標準と標準外の 2 種類のラベルがあります。Enterprise COBOL では、標準外ユーザー・ラベルはサポートされません。さらに、標準ユーザー・ラベルには、関連データ・セットに関するユーザー指定の情報も含まれます。

標準ラベルは、ボリューム・ラベルとデータ・セット・ラベルのグループから構成されます。ボリューム・ラベルは、ボリューム上のデータの前または後に置かれ、そのボリュームを識別および記述します。データ・セット・ラベルは、ボリューム上の各データ・セットの前または後に置かれ、そのデータ・セットを識別および記述します。

- データ・セットの前に置かれるデータ・セット・ラベルは、*ヘッダー・ラベル* と呼ばれます。
- データ・セットの後ろに置かれるデータ・セット・ラベルは、*トレーラー・ラベル* と呼ばれます。トレーラー・ラベルは、データ・セット内のブロックのカウントを含む点を除けば、ヘッダー・ラベルと似ています。
- データ・セット・ラベル・グループには、任意で、標準ユーザー・ラベルを組み込むことができます。
- ボリューム・ラベル・グループには、任意で、標準ユーザー・ラベルを組み込むことができます。

関連タスク
『トレーラー・ラベルおよびヘッダー・ラベルの使用』

関連参照
196 ページの『標準ラベルの形式』

トレーラー・ラベルおよびヘッダー・ラベルの使用

ユーザー・ラベルは、データ・セットまたはボリューム (リール) の始まりまたは終わりに達したときに作成、検査、または更新することができます。ボリュームの終わり出口とボリューム開始出口を使用できます。中間トレーラーおよびヘッダーを作成したり、検査することもできます。

データ・セットの各ボリューム上で、最大 8 個のヘッダー・ラベルと 8 個のトレーラー・ラベルを作成、検査、または更新することができます。(QSAM `EXTEND` は、ファイル開始ラベルが処理されない点を除き、`OUTPUT` と同じ働きをします。) ラベルは、マルチボリューム・データ・セットの場合、最初のボリュームに置かれます。このボリュームは、トレーラー・ラベルが作成、検査、または更新される予定である場合には、`CLOSE` として取り付けなければなりません。`INPUT` または `I-O` としてオープンされたファイルのトレーラー・ラベルは、`AT END` 条件に達しているファイルに対して `CLOSE` ステートメントが実行されるときに処理されます。

誤った位置番号を指定してヘッダーまたはトレーラーをコーディングすると、結果は予測できません。(データ管理は、ラベルを強制的に正しい相対位置にすることがあります。)

標準ラベル処理を使用するときには、データ・セットを記述する DD ステートメントで標準およびユーザー・ラベルのラベル・タイプ (SUL) をコーディングしてください。

ユーザー・ラベル・トラックの取得: 直接アクセス・ボリュームに対して SUL の LABEL サブパラメーターを使用すると、データ・セットの作成時に、別個のユーザー・ラベル・トラックが割り振られます。この追加のトラックは初期割り振り時に割り振られ、順次データ・セットの場合はボリュームの終わり (ボリューム切り替え) 時に割り振られます。ユーザー・ラベル・トラック (順次データ・セットの各ボリュームにつき 1 つ) には、ユーザー・ヘッダー・ラベルとユーザー・トレーラー・ラベルの両方が含まれます。LABEL 名がユーザー LABEL 宣言の外側で参照されると、結果は予測できません。

ユーザー・ラベルの処理: USE AFTER LABEL 宣言は、サポートされるファイル上のユーザー・ラベルを処理するためのプロシージャーを提供します。AFTER オプションは、標準ユーザー・ラベルの処理を示します。

ラベルは、ファイルの FD 記入項目で、LABEL RECORDS 節の *data-names* としてリストしてください。

表 22. **QSAM ユーザー・ラベルの処理**

ファイルの オープンのモード	次のような条件の場合:	結果:
INPUT	USE . . . LABEL 宣言が OPEN オプションまたはファイルにコーディングされている。	ラベルが読み取られ、制御が LABEL 宣言に渡されます。
OUTPUT	USE . . . LABEL 宣言が OPEN オプションまたはファイルにコーディングされている。	ラベル用のバッファー域が提供され、制御が LABEL 宣言に渡されます。
INPUT または I-O	AT END 条件に達しているファイルに対して CLOSE ステートメントが実行される。	トレーラー・ラベルを処理するために制御が LABEL 宣言に渡されます。

GO TO MORE-LABELS ステートメントを使用して、特殊な出口を指定することができます。このステートメントの結果として、ラベル DECLARATIVE SECTION から出た場合は、システムは次のいずれかの処置を行います。

- 現行の開始または終了ラベルを書き込んだ後、ラベルをさらに作成するために USE セクションの始まりに再入します。最後のラベルを作成した後、システムはセクションの最後のステートメントを実行することによって終了します。
- 追加の開始または終了ラベルを読み取った後、ラベルをさらに検査するために USE セクションの始まりに再入します。ユーザー・ラベルの処理時には、検査する別のユーザー・ラベルがある場合にのみ、システムはそのセクションに再入します。このため、セクションの最後のステートメントを通り抜けるプログラム・パスは必要ありません。

ユーザー・ラベルについて GO TO MORE-LABELS ステートメントが実行されない場合には、その直後のユーザー・ラベルを検査または作成するための、DECLARATIVE SECTION への再入は行われません。

関連概念
194 ページの『QSAM ファイルのラベル』

標準ラベルの形式

標準ラベルは、EBCDIC または ASCII で記録される 80 文字のレコードです。最初の 4 文字は、常に、ラベルを識別するために使用されます。

表 23. **標準テープ・ラベルの ID**

ID	説明
VOL1	ボリューム・ラベル
HDR1 または HDR2	データ・セット・ヘッダー・ラベル
EOV1 または EOV2	データ・セット・トレーラー・ラベル (ボリュームの終わり)
EOF1 または EOF2	データ・セット・トレーラー・ラベル (データ・セットの終わり)
UHL1 から UHL8	ユーザー・ヘッダー・ラベル
UTL1 から UTL8	ユーザー・トレーラー・ラベル

直接アクセス・ボリュームのラベルの形式は、テープ・ボリューム・ラベル・グループのラベル・グループの形式とほとんど同じです。違いは、初期 DASTO ボリューム・ラベルのデータ・セット・ラベルが、データ・セット制御ブロック (DSCB) から構成されるということだけです。DSCB は、ボリューム目録 (VTOC) の中にあり、スペース割り振りなどの制御情報のほかに、テープ・データ・セット・ヘッダーおよびトレーラー情報と等価の情報を含んでいます。

標準ユーザー・ラベル

標準ラベル・グループの中では、ユーザー・ラベルの指定はオプションです。ユーザー・ヘッダー・ラベル (UHL1-8) およびユーザー・トレーラー・ラベル (UTL1-8) に使用される形式は、次のいずれかで記録される長さ 80 文字のラベルから構成されます。

- EBCDIC (DASD 上または IBM 標準ラベル付きテープ上)
- ASCII または ISO/ANSI ラベル付きテープ

最初の 3 バイトは、以下のいずれのラベルであるかを識別する文字から構成されます。

- ユーザー・ヘッダー・ラベル (データ・セットの始まりにある) を表す UHL
- ユーザー・トレーラー・ラベル (ボリュームの終わりまたはデータ・セットの終わりにある) を表す UTL

次のバイトには、同じタイプのラベルのセットにおけるこのラベルの相対位置が入ります。1 から 8 個のラベルを指定できます。残りの 76 バイトは、ユーザー指定情報から構成されます。

標準ユーザー・ラベルは、QSAM ストライプ・データ・セットについてはサポートされません。

関連概念
194 ページの『QSAM ファイルのラベル』

テープ上の QSAM ASCII ファイルの処理

プログラムが QSAM ASCII ファイルを処理する場合には、ASCII アルファベットを要求し、レコード・フォーマットを定義し、(JCL で) DD 名を定義する必要があります。

さらに、プログラムが ASCII ファイルからの符号付き数値データ項目を処理する場合、別々の符号を持つゾーン 10 進数として数値データを定義してください。すなわち、USAGE DISPLAY として、また SIGN 節の SEPARATE 句で定義してください。

CODEPAGE コンパイラー・オプションは、ASCII テープをサポートするための ASCII と EBCDIC 間での変換に使用されるコード・ページには影響を与えません。ASCII テープのサポートに使用される CCSID がどのように選択されるか、およびデフォルトの CCSID については、z/OS DFSMS の資料を参照してください。

ASCII アルファベットの要求: SPECIAL-NAMES 段落で、ASCII を表す STANDARD-1 をコーディングします。

```
ALPHABET-NAME IS STANDARD-1
```

ファイルの FD 項目に、次のようにコーディングしてください。

```
CODE-SET IS ALPHABET-NAME
```

レコード・フォーマットの定義: QSAM ASCII テープ・ファイルは、以下のいずれかのレコード形式で処理してください。

- 固定長 (形式 F)
- 不定 (形式 U)
- 可変長 (形式 V)

可変長レコードを使用する場合、フォーマット D を明示的に指定することはできません。代わりに RECORDING MODE V を指定してください。この形式情報は、内部で D モードに変換されます。D モード・レコードは、レコードごとに 4 バイトのレコード記述子を持ちます。

DD 名の定義: z/OS のもとでは、ASCII ファイルを処理するには、特別な JCL コーディングが必要です。 DCB パラメーターの以下のサブパラメーターを、DD ステートメントにコーディングしてください。

BUFOFF=[*L*|*n*]

L ブロック長 (ブロック接頭語を含む) が入る 4 バイトのブロック接頭語。

n ブロック接頭語の長さ。

- 入力の場合は、0 から 99

- 出力の場合は、0 または 4

  BLOCK CONTAINS 0 をコーディングした場合には、この値を使用してください。

BLKSIZE=*n*

n ブロックのサイズ (ブロック接頭語の長さを含む)。

LABEL=[AL|AUL|NL]

AL 米国標準規格 (ANS) ラベル

AUL ANS およびユーザー・ラベル

NL ラベルなし

OPTCD=Q

Q この値は、ASCII ファイルの場合には必須であり、ファイルが Enterprise COBOL を使用して作成された場合はデフォルトです。

関連タスク
『ASCII ファイルのラベルの処理』

関連参照
z/OS DFSMS: Using Data Sets (文字データ変換)

ASCII ファイルのラベルの処理

ASCII ファイルの標準ラベル処理は、EBCDIC ファイルの標準ラベル処理と同じです。システムは、ASCII コードを EBCDIC コードに変換してから処理します。

すべての ANS ユーザー・ラベルは任意指定です。ASCII ファイルは、ユーザー・ヘッダー・ラベル (UHL*n*) およびユーザー・トレーラー・ラベル (UTL*n*) を持つことができます。ファイルの始まりおよび終わりにおけるユーザー・ラベルの数に制限はありません。必要な数だけラベルを作成することができます。すべてのユーザー・ラベルは、長さが 80 バイトでなければなりません。

ユーザー・ラベルを作成または検査するには (ユーザー・ラベル出口)、USE AFTER STANDARD LABEL プロシージャーをコーディングしてください。 USE BEFORE STANDARD LABEL プロシージャーを使用することはできません。

テープ上の ASCII ファイルは、以下のようにすることができます。

- ANS ラベル
- ANS およびユーザー・ラベル
- ラベルなし

ASCII テープ上のラベルは、ASCII コードのみでなければなりません。ASCII と EBCDIC の組み合わせが入っているテープは、読み取ることができません。

関連タスク
197 ページの『テープ上の QSAM ASCII ファイルの処理』

第 10 章 VSAM ファイルの処理

仮想記憶アクセス方式 (VSAM) は、直接アクセス・ストレージ・デバイス上のファイルに関するアクセス方式です。 VSAM を使用して、ファイルのロード、ファイルからのレコードの取得、ファイルの更新、およびファイルのレコードの追加、置換、および削除を行うことができます。

VSAM 処理には、QSAM と比較して以下の利点があります。

- 無許可アクセスに対するデータの保護。
- システム間の互換性。
- 装置からの独立性 (ブロック・サイズおよびその他の制御情報に留意する必要がありません)。
- より単純な JCL (システムに必要な情報は統合カタログに入れて提供されます)。
- 索引付きファイル編成または相対ファイル編成を使用する機能。

以下の表は、COBOL 用語や読者がよく知っている用語と VSAM 用語との違いを示しています。

表 24. **VSAM、COBOL、および非 VSAM 用語の比較**

VSAM 用語	COBOL 用語	類似した非 VSAM 用語
データ・セット	ファイル	データ・セット
入力順データ・セット (ESDS)	順次ファイル	QSAM データ・セット
キー順データ・セット (KSDS)	索引付きファイル	ISAM データ・セット
相対レコード・データ・セット (RRDS)	相対ファイル	BDAM データ・セット
制御インターバル		ブロック
制御インターバル・サイズ (CISZ)		ブロック・サイズ
バッファー (BUFNI/BUFND)		BUFNO
アクセス方式制御ブロック (ACB)		データ制御ブロック (DCB)
クラスター (CL)		データ・セット
クラスター定義		データ・セット割り振り
JCL DD ステートメントの AMP パラメーター		JCL DD ステートメントの DCB パラメーター
レコード・サイズ		レコード長

この VSAM 資料では、*ファイル* という用語は、COBOL ファイルまたは VSAM データ・セットのいずれかを指します。

複雑な要件があるか、または VSAM を頻繁に使用する予定である場合には、使用するオペレーティング・システムの VSAM 資料を調べてください。

関連概念

200 ページの『VSAM ファイル』

関連タスク

202 ページの『VSAM ファイル編成およびレコードの定義』
208 ページの『VSAM ファイルの入出力ステートメントのコーディング』
217 ページの『VSAM ファイルでのエラー処理』
218 ページの『パスワードによる VSAM ファイルの保護』
219 ページの『z/OS および UNIX のもとでの VSAM データ・セットの操作』
227 ページの『VSAM パフォーマンスの向上』

関連参照

z/OS DFSMS: Using Data Sets
データ・セットに対する z/OS DFSMS マクロ命令
z/OS DFSMS カタログのためのアクセス方式サービス・プログラム

VSAM ファイル

VSAM データ・セットの物理編成は、他のアクセス方式で使用されている編成とは大きく異なっています。

VSAM データ・セットは、制御インターバル (CI) および制御域 (CA) で保持されます。CI と CA のサイズは、通常、アクセス方式によって判別され、それらが使用される方法について意識することはありません。

VSAM では、次の 3 種類のファイル編成を使用できます。

VSAM 順次ファイル編成
(VSAM *ESDS (入力順データ・セット)* 編成とも呼ばれます。) VSAM 順次ファイル編成では、レコードは入力された順番に保管されます。

VSAM 入力順データ・セットは、QSAM 順次ファイルと同等です。レコードの順序は固定されます。

VSAM 索引付きファイル編成
(VSAM *KSDS (キー順データ・セット)* 編成とも呼ばれます。) VSAM 索引付きファイル (KSDS) では、レコードは、組み込み基本キー・フィールドの照合シーケンスの定義に従って配列されます。基本キーは、レコード内の 1 つ以上の連続する文字から構成されます。基本キーは、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。レコードの基本キーは、例えば、従業員番号や送り状番号にすることができます。

VSAM 相対ファイル編成
(VSAM 固定長または可変長 *RRDS (相対レコード・データ・セット)* 編成とも呼ばれます。) VSAM 相対レコード・セット (RRDS) には、相対キーによって順序付けされたレコードが入ります。*相対キー* とは、ファイルの始まりからのレコードの相対的な位置を表す相対レコード番号です。相対レコード番号は、固定長または可変長レコードを識別します。

VSAM 固定長 RRDS では、レコードはストレージ内の一連の固定長スロットに入れられます。各スロットは、相対レコード番号に関連付けられています。例えば、10 個のスロットが入っている固定長 RRDS では、最初のスロットは相対レコード番号 1 であり、10 番目のスロットは相対レコード番号 10 です。

VSAM 可変長 RRDS では、レコードはそれらの相対レコード番号に従って順序付けられます。レコードの保管および検索は、相対レコード番号の設定に従って行われます。

本書では、*VSAM 相対レコード・データ・セット* (または *RRDS*) という用語は、特に区別する必要がない限り、固定長レコードを持つ相対レコード・データ・セットと可変長レコードを持つ相対レコード・データ・セットの両方を指すために使用されています。

次の表は、種々のタイプの VSAM データ・セットの特性を比較したものです。

表 25. **VSAM データ・セット・タイプの比較**

特性	**入力順データ・セット (ESDS)**	**キー順データ・セット (KSDS)**	**相対レコード・データ・セット (RRDS)**
レコードの順序	書き込まれた順序	キー・フィールドによる照合シーケンス	相対レコード番号の順序
アクセス	順次	索引を通してキー順	相対レコード番号順 (キーのように処理される)
代替索引	COBOL ではサポートされませんが、1 つ以上の代替索引を持つことができます。	1 つ以上の代替索引を持つことができます。	代替索引を持つことはできません。
レコードの相対バイト・アドレス (RBA) と相対レコード番号 (RRN)	RBA を変更することはできません。	RBA を変更できます。	RRN を変更することはできません。
レコード追加用のスペース	データ・セットの終わりにあるスペースが使用されます。	レコードを挿入し、それらの長さを適切に変更するためには、分散フリー・スペースが使用されます。	固定長 RRDS の場合は、データ・セット内の空のスロットが使用されます。 可変長 RRDS の場合は、分散フリー・スペースが使用され、追加されるレコードの長さは適切に変更されます。
レコード削除用のスペース	レコードは削除できませんが、それのスペースを同じ長さのレコードのために再利用することはできます。	削除または短縮されたレコードのスペースは、制御インターバルで自動的に再利用されます。	削除されたレコードのスペースを再利用することができます。
スパン・レコード	スパン・レコードを持つことができます。	スパン・レコードを持つことができます。	スパン・レコードを持つことはできません。
作業ファイルとしての再利用	代替索引を持っているか、キー範囲と関連付けられているか、またはボリューム当たりのエクステントが 123 個を超えている場合を除き、作業ファイルとして再利用できます。	代替索引を持っているか、キー範囲と関連付けられているか、またはボリューム当たりのエクステントが 123 個を超えている場合を除き、作業ファイルとして再利用できます。	再利用することができます。

関連タスク

202 ページの『VSAM ファイルの順次編成の指定方法』

203 ページの『VSAM ファイルの索引編成の指定方法』
204 ページの『VSAM ファイルの相対編成の指定方法』
220 ページの『VSAM ファイルの定義』

VSAM ファイル編成およびレコードの定義

ENVIRONMENT DIVISION の FILE-CONTROL 段落の項目を使用して、COBOL プログラムにおける VSAM ファイルのファイル編成およびアクセス・モードを定義します。

DATA DIVISION の FILE SECTION で、そのファイルのファイル記述 (FD) 記入項目をコーディングします。関連するレコード記述項目では、*record-name* およびレコード長を定義します。レコードの論理サイズは、RECORD 節でコーディングします。

重要: Enterprise COBOL プログラムでは、アクセス方式サービス・プログラムで定義してからでなければ、VSAM データ・セットを処理できません。

表 26. **VSAM ファイル編成、アクセス・モード、およびレコード・フォーマット**

ファイル編成	順次アクセス	ランダム・アクセス	動的アクセス	固定長	可変長
VSAM 順次 (ESDS)	はい	いいえ	いいえ	はい	はい
VSAM 索引付き (KSDS)	はい	はい	はい	はい	はい
VSAM 相対 (RRDS)	はい	はい	はい	はい	はい

関連タスク
『VSAM ファイルの順次編成の指定方法』
203 ページの『VSAM ファイルの索引編成の指定方法』
204 ページの『VSAM ファイルの相対編成の指定方法』
205 ページの『VSAM ファイルのアクセス・モードの指定』
206 ページの『VSAM ファイルのレコード長の定義』
269 ページの『ファイル状況キーの使用』
271 ページの『VSAM 状況コードの使用 (VSAM ファイルのみ)』
220 ページの『VSAM ファイルの定義』

VSAM ファイルの順次編成の指定方法

COBOL プログラム内では、VSAM ESDS ファイルは、ORGANIZATION IS SEQUENTIAL 節で識別してください。順次ファイル内のレコードは、順次にだけしかアクセス (読み取りまたは書き込み) することができません。

レコードをファイルに入れた後は、それを短くしたり、長くしたり、削除したり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。

次の例は、VSAM 順次ファイル (ESDS) の代表的な FILE-CONTROL 記入項目を示しています。

```
SELECT S-FILE
    ASSIGN TO SEQUENTIAL-AS-FILE
    ORGANIZATION IS SEQUENTIAL
    ACCESS IS SEQUENTIAL
    FILE STATUS IS FSTAT-CODE VSAM-CODE.
```

関連概念
200 ページの『VSAM ファイル』

VSAM ファイルの索引編成の指定方法

ORGANIZATION IS INDEXED 節を使用して、COBOL プログラムの VSAM KSDS ファイルを示してください。 RECORD KEY 節を使用して、レコードの基本キーをコーディングしてください。代替キーおよび代替索引を使用することもできます。

```
RECORD KEY IS data-name
```

上の例で、*data-name* は、DATA DIVISION のレコード記述項目で定義したときの基本キー・フィールドの名前です。基本キー・データ項目は、クラス英字、英数字、DBCS、数値、または国別にすることができます。 USAGE NATIONAL を持っている場合、基本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データ項目にすることもできます。レコード・キーの照合は、キーのクラスやカテゴリーに関係なく、キーの 2 進値に基づいて行われます。

次の例は、動的にアクセスされる VSAM 索引付きファイル (KSDS) の場合のステートメントを示します。基本キー COMMUTER-NO のほかに、代替キー LOCATION-NO を指定します。

```
SELECT I-FILE
    ASSIGN TO INDEXED-FILE
    ORGANIZATION IS INDEXED
    ACCESS IS DYNAMIC
    RECORD KEY IS IFILE-RECORD-KEY
    ALTERNATE RECORD KEY IS IFILE-ALTREC-KEY
    FILE STATUS IS FSTAT-CODE VSAM-CODE.
```

関連概念
200 ページの『VSAM ファイル』

関連タスク
『代替キーの使用』
204 ページの『代替索引の使用』

関連参照
RECORD KEY 節 (*Enterprise COBOL 言語解説書*)
データのクラスおよびカテゴリー (*Enterprise COBOL 言語解説書*)

代替キーの使用

基本キーのほかに、VSAM KSDS ファイル用の 1 つ以上の代替キーをコーディングすることができます。代替キーを使用すれば、索引付きファイルにアクセスして、基本キー順序以外の順序でレコードを読み取ることができます。

代替キーは固有である必要はありません。重複してもよいように代替キーがコーディングされている場合、複数のレコードにアクセスできます。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。

COBOL プログラムで代替キーを定義するには、ALTERNATE RECORD KEY 節を使用します。

```
ALTERNATE RECORD KEY IS data-name
```

上の例で、*data-name* は、DATA DIVISION のレコード記述項目で定義したときの代替キー・フィールドの名前です。代替キー・データ項目は、基本キー・データ項目のように、クラス英字、英数字、DBCS、数値、または国別にすることができます。代替キーの照合は、キーのクラスやカテゴリーに関係なく、キーの 2 進値に基づいて行われます。

代替索引の使用

VSAM KSDS ファイルで代替索引を使用するためには、アクセス方式サービスを使用して、*代替索引 (AIX)* と呼ばれるデータ・セットを定義する必要があります。

AIX には、指定された代替キーのそれぞれの値について 1 つのレコードが入ります。レコードは、代替キー値により順次に配列されます。各レコードには、代替キー値が入っている関連索引付きファイル内のすべてのレコードの対応する基本キーが含まれます。

関連タスク
221 ページの『代替索引の作成』

VSAM ファイルの相対編成の指定方法

COBOL プログラム内では、VSAM RRDS ファイルは、ORGANIZATION IS RELATIVE 節を使用して識別してください。各論理レコードを相対レコード番号に関連付けるには、RELATIVE KEY IS 節を使用します。

次の例は、相対キーに入れられた値によってランダムにアクセスされる相対レコード・データ・セット (RRDS) を示しています。

```
SELECT R-FILE
    ASSIGN TO RELATIVE-FILE
    ORGANIZATION IS RELATIVE
    ACCESS IS RANDOM
    RELATIVE KEY IS RFILE-RELATIVE-KEY
    FILE STATUS IS FSTAT-CODE VSAM-CODE.
```

ランダム化ルーチンを使用して、各レコード内のキー値をそのレコードの相対レコード番号と関連付けることができます。レコード・キーを相対レコード番号に変換する技法は数多くありますが、最もよく使用されるのは除算 / 剰余技法です。この技法では、キーをデータ・セット内のスロットの数で割って、商と剰余を出します。剰余に 1 を加算すると、結果は有効な相対レコード番号になります。

VSAM RRDS では、代替索引はサポートされません。

関連概念
200 ページの『VSAM ファイル』
『固定長および可変長 RRDS』

関連タスク
『可変長 RRDS の使用』
220 ページの『VSAM ファイルの定義』

固定長および可変長 RRDS

固定長レコードを含む RRDS では、各レコードがそれぞれ 1 つのスロットを占めます。レコードの保管と検索は、そのスロットの相対レコード番号に従って行われます。可変長の RRDS にはスロットがありません。代わりに、定義したフリー・スペースを使用して、より効率的なレコード挿入を行うことができます。

固定長レコードを含む RRDS をロードするときには、スロットをスキップし、それらを空にしておくこともできます。可変長レコードを含む RRDS をロードするときは、相対レコード番号をスキップすることができます。

可変長 RRDS の使用

可変長レコードを含んだ相対レコード・データ・セット (RRDS) を使用する場合には、できる限り、 VSAM 可変長 RRDS を使用する必要があります。・サポートを使用する場合、VSAM KSDS を使用して、可変長 RRDS をシミュレートします。

以下の手順を実行します。

1. ORGANIZATION IS RELATIVE 節でファイルを定義します。
2. 可変長サイズでレコードを記述するには、FD 記入項目を使用します。
3. NOSIMVRD ランタイム・オプションを使用します。
4. アクセス方式サービス・プログラムを介して、VSAM ファイルを RRDS として定義します。

関連タスク
220 ページの『VSAM ファイルの定義』

関連参照
z/OS DFSMS カタログのためのアクセス方式サービス・プログラム

VSAM ファイルのアクセス・モードの指定

VSAM 順次ファイルのレコードは、順次でしかアクセスできません。 VSAM 索引付きファイルおよび相対ファイルの中のレコードは、順次、ランダム、または動的の 3 つの方法でアクセスすることができます。

順次アクセスの場合は、FILE-CONTROL 記入項目で ACCESS IS SEQUENTIAL をコーディングします。その場合、索引付きファイルのレコードは、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。

ランダム・アクセスの場合は、FILE-CONTROL 記入項目で ACCESS IS RANDOM をコーディングします。その場合、索引付きファイルのレコードは、キー・フィールドに

入れられた値に従ってアクセスされます。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。

動的アクセスの場合は、FILE-CONTROL 記入項目で ACCESS IS DYNAMIC をコーディングします。動的アクセスは、同じプログラムの中での順次とランダムの混合アクセスです。動的アクセスを使用すると、順次処理とランダム処理の両方を実行する 1 つのプログラムを作成して、あるレコードは順次にアクセスし、別のレコードはキーによってアクセスすることができます。

『例: VSAM ファイルでの動的アクセスの使用』

関連タスク
213 ページの『VSAM ファイルからのレコードの読み取り』

例: VSAM ファイルでの動的アクセスの使用

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。

プログラムが、VSAM ファイルの動的アクセスを使用して、時間給 $15.00 から $20.00 の従業員と $25.00 以上の従業員を処理する場合、プログラムは以下のことを行います。

1. 1500 のキーに基づいて最初のレコードをランダムに検索します (ランダム検索 READ を使用して)。
2. 給与フィールドが 2000 を超えるまで、順次に読み取ります (READ NEXT を使用して)。
3. 2500 のキーに基づいて、次のレコードをランダム検索します。
4. ファイルの終わりになるまで、順次に読み取ります。

関連タスク
213 ページの『VSAM ファイルからのレコードの読み取り』

VSAM ファイルのレコード長の定義

VSAM レコードは、固定長または可変長に定義できます。 COBOL は、RECORD 節と、ファイルの FD 記入項目に関連付けられたレコード記述から、レコード形式を判別します。

VSAM ファイルについては、ブロック化の概念は意味を持たないため、BLOCK CONTAINS 節は省略して構いません。この節は構文検査されますが、プログラムの実行方法には影響を及ぼしません。

関連タスク
207 ページの『固定長レコードの定義』
207 ページの『可変長レコードの定義』
Enterprise COBOL コンパイラーおよびランタイム 移行ガイド

関連参照
15 ページの『FILE SECTION 記入項目』

固定長レコードの定義

VSAM レコードを固定長として定義するには、次のコーディング・オプションのいずれかを使用します。

表 27. **VSAM 固定長レコードの定義**

RECORD 節	**節形式**	**レコード長**	**コメント**
RECORD CONTAINS *integer* をコーディングする。	1	*integer-3* バイト の長さでサイズが固定。	ファイルに関連するレベル 01 レコード記述項目の長さは重要ではありません。
RECORD 節を省略するが、ファイルに関連付けられたすべてのレベル 01 レコードを同じサイズとしてコーディングし、OCCURS DEPENDING ON 節には何もコーディングしません。		コーディングされた固定サイズ。	

関連参照
RECORD 節 (*Enterprise COBOL 言語解説書*)

可変長レコードの定義

VSAM レコードを可変長として定義するには、次のコーディング・オプションのいずれかを使用します。

表 28. **VSAM 可変長レコードの定義**

RECORD 節	**節形式**	**最大レコード長**	**コメント**
RECORD IS VARYING FROM *integer-6* TO *integer-7* をコーディングする。	3	*integer-7* バイト	ファイルに関連するレベル 01 レコード記述項目の長さは重要ではありません。
RECORD IS VARYING をコーディングする。	3	ファイルに関連する最大のレベル 01 レコード記述項目のサイズ。	コンパイラーが最大レコード長を決定します。
RECORD CONTAINS *integer-4* TO *integer-5* をコーディングする。	2	*integer-5* バイト	最小レコード長は、*integer-4* バイトです。
RECORD 節を省略するが、サイズが異なるか OCCURS DEPENDING ON 節を含む、ファイルに関連付けられた複数のレベル 01 レコードをコーディングする。		ファイルに関連する最大のレベル 01 レコード記述項目のサイズ。	コンパイラーが最大レコード長を決定します。

形式 V ファイルに READ INTO ステートメントを指定すると、そのファイルに関して読み取られたレコード・サイズが、コンパイラーによって生成される MOVE ステートメントで使用されます。したがって、読み取られたレコードがレベル 01 レコード記述に対応していない場合には、予期したとおりの結果が得られないことがあります。MOVE ステートメントに関するその他の規則はすべて適用されます。例えば、READ ステートメントによって読み込まれた形式 V レコードに対して MOVE ステートメントを指定すると、レコードのサイズはそのレベル 01 レコード記述に相当します。

関連参照
RECORD 節 (*Enterprise COBOL 言語解説書*)

VSAM ファイルの入出力ステートメントのコーディング

以下の COBOL ステートメントを使用して、VSAM ファイルを処理します。

OPEN VSAM データ・セットを処理のために COBOL プログラムに接続します。

WRITE ファイルにレコードを追加するか、またはファイルをロードします。

START READ NEXT ステートメントのためにクラスターにおける現在場所を設定します。

START はレコードを取り出しません。現行レコード・ポインターを設定するだけです。

READ および READ NEXT
ファイルからレコードを取り出します。

REWRITE
レコードを更新します。

DELETE 索引付きファイルおよび相対ファイルからのみ、レコードを論理的に除去します。

CLOSE VSAM データ・セットをプログラムから切り離します。

以下のすべての要因によって、特定の VSAM データ・セットに使用できる入出力ステートメントが決まります。

- アクセス・モード (順次、ランダム、または動的)
- ファイル編成 (ESDS、KSDS、または RRDS)
- OPEN ステートメントのモード (INPUT、OUTPUT、I-O、または EXTEND)

次の表は、順次ファイル (ESDS) のステートメントとオープン・モードの可能な組み合わせを示しています。 X は、列の上部に示されているオープン・モードでステートメントを使用できることを示します。

表 29. **VSAM 順次ファイル用入出力ステートメント**

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

次の表は、索引付き (KSDS) ファイルおよび相対 (RRDS) ファイルに関して使用できるステートメントとオープン・モードの可能な組み合わせを示しています。 X は、列の上部に示されているオープン・モードでステートメントを使用できることを示します。

表 30. **VSAM 相対ファイルおよび索引付きファイル用入出力ステートメント**

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	

FILE STATUS 節でコーディングしたフィールドは VSAM によって、それぞれの入出力ステートメントの後で、操作の成功または失敗を示すために更新されます。

関連概念
『ファイル位置標識』

関連タスク
『ファイルのオープン (ESDS、KSDS、または RRDS)』
213 ページの『VSAM ファイルからのレコードの読み取り』
214 ページの『VSAM ファイル内のレコードの更新』
215 ページの『VSAM ファイルへのレコードの追加』
216 ページの『VSAM ファイル内のレコードの置換』
216 ページの『VSAM ファイルからのレコードの削除』
217 ページの『VSAM ファイルのクローズ』

関連参照
ファイル状況キー (*Enterprise COBOL 言語解説書*)

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、および READ NEXT ステートメントによって設定されます。

その後の READ または READ NEXT 要求は、設定されたファイル位置標識の位置を使用し、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。

関連タスク
213 ページの『VSAM ファイルからのレコードの読み取り』

ファイルのオープン (ESDS、KSDS、または RRDS)

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 ステートメントでオープンしておかなければなりません。

ファイルの可用性および作成は、OPEN 処理、オプション・ファイル、およびファイル状況コード 05 と 35 に影響を与えます。例えば、EXTEND、I-O、または INPUT モードで、オプションでも利用可能な状態にもないファイルを開いた場合、ファイル状況 35 になり、OPEN ステートメントは失敗します。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が作成され、ファイル状況 05 が返されます。

OPEN 操作が正しく機能するのは、ファイルの DD ステートメントまたはデータ・セット・ラベルで固定ファイル属性を指定し、しかも COBOL プログラムの SELECT 節および FD 記入項目でそのファイルについて一貫性のある属性を指定する場合だけです。以下の項目が一致していないと、ファイル状況コード 39 が戻され、OPEN ステートメントは失敗に終わります。

- ファイル編成の属性 (順次、相対、または索引付き)

- 基本レコード・キー
- 代替レコード・キー
- 最大レコード・サイズ
- レコード・タイプ (固定長または可変長)

VSAM ファイルについての OPEN ステートメントをコーディングする方法は、ファイルが空である (レコードが入ったことがない) か、ロード済みであるかによって異なります。ファイルのタイプがどちらであっても、プログラムでは、それぞれの OPEN ステートメントの後でファイル状況キーを検査しなければなりません。

関連タスク 『空のファイルのオープン』 213 ページの『ロード済みファイル (レコードが入っているファイル) のオープン』

関連参照 212 ページの『VSAM ファイルにレコードをロードするために使用されるステートメント』

空のファイルのオープン

レコードをまったく含んでいないファイル (空ファイル) を開くには、OPEN ステートメントの形式を使用してください。

開こうとするファイルのタイプに応じて、次のいずれかのステートメントを使用してください。

- ESDS ファイルの場合は、OPEN OUTPUT。
- KSDS および RRDS ファイルの場合は、OPEN OUTPUT または OPEN EXTEND。(どちらのコーディングでも効果は同じです。) ファイルをランダム・アクセスまたは動的アクセス用にコーディングし、それがオプション・ファイルであれば、OPEN I-O を使用することができます。

*オプション・ファイル*とは、プログラムが実行されるたびに、必ずしも使用可能である必要はないファイルです。 INPUT、I-O、または OUTPUT モードで開かれるファイルは、FILE-CONTROL 段落の SELECT OPTIONAL 節で定義することにより、オプションとして定義できます。

順次的なファイルの初期ロード: ファイルの初期ロードとは、レコードを初めてファイルに書き込むことを意味します。これは、以前のすべてのレコードが削除されたファイルにレコードを書き込むことと同じではありません。 VSAM ファイルを初期ロードするには、以下のようにしてください。

1. ファイルをオープンします。
2. 順次処理を行います (ACCESS IS SEQUENTIAL)。 (順次処理は、ランダム処理や動的処理よりも速く行われます。)
3. WRITE を使用して、ファイルにレコードを加えます。

OPEN OUTPUT を使用して VSAM ファイルをロードすると、プログラムのパフォーマンスが著しく向上します。OPEN I-O または OPEN EXTEND を使用すると、プログラムのパフォーマンスに悪い影響があります。

VSAM 索引付きファイルを順次にロードすると、順次処理ではユーザー定義のフリー・スペースが保持されるため、ロードとその後の処理の両方のパフォーマンスが最適化されます。将来の追加がより効率的になります。

ACCESS IS SEQUENTIAL を使用する場合は、レコードを RECORD KEY の昇順に書き込まなければなりません。

VSAM 相対ファイルを順次にロードすると、レコードは相対レコード番号の昇順にファイルに入れられます。

ランダム的または動的なファイルの初期ロード: ランダム処理または動的処理でファイルをロードできますが、これらの処理は順次処理ほど効率的ではありません。VSAM がランダム処理または動的処理をサポートしていないので、COBOL が、OPEN OUTPUT または OPEN I-O とともに ACCESS IS RANDOM または ACCESS IS DYNAMIC を使用できるようにするために余分な処理を実行しなければなりません。これらのステップにより、ファイルの使用準備が整い、ファイルはロード済みファイル状況になります (少なくとも 1 度使用されたことがあるからです)。

ファイルが使用できるよう準備するための余分なオーバーヘッドに加えて、ランダム処理ではユーザー定義のフリー・スペースが考慮されません。その結果、将来の追加処理が非効率的になる恐れがあります。順序処理では、ユーザー定義のフリー・スペースが保持されます。

拡張フォーマット VSAM データ・セットをロードする場合、z/OS DFSMS システム管理バッファリングによりローカル共用リソース (LSR) へのバッファリングが設定されていると、OPEN に対してファイル状況 30 が発生します。この場合に VSAM データ・セットを正常にロードするには、VSAM データ・セットがシステム管理バッファリングを迂回するよう DD AMP パラメーターで ACCBIAS=USER を指定してください。

アクセス方式サービス・プログラムによる VSAM データ・セットのロード: アクセス方式サービス・プログラムの REPRO コマンドを使用して VSAM データ・セットをロードまたは更新することができます。可能なときはいつでも、REPRO を使用してください。

関連タスク
213 ページの『ロード済みファイル (レコードが入っているファイル) のオープン』

関連参照
『VSAM ファイルにレコードをロードするために使用されるステートメント』
z/OS DFSMS カタログのためのアクセス方式サービス・プログラム (REPRO)

VSAM ファイルにレコードをロードするために使用されるステートメント

レコードを VSAM ファイルにロードするには、以下に示すステートメントを使用してください。

表 31. **VSAM ファイルにレコードをロードするために使用されるステートメント**

除算	ESDS	KSDS	RRDS
ENVIRONMENT DIVISION	SELECT ASSIGN FILE STATUS PASSWORD ACCESS MODE	SELECT ASSIGN ORGANIZATION IS INDEXED RECORD KEY ALTERNATE RECORD KEY FILE STATUS PASSWORD ACCESS MODE	SELECT ASSIGN ORGANIZATION IS RELATIVE RELATIVE KEY FILE STATUS PASSWORD ACCESS MODE
DATA DIVISION	FD 記入項目	FD 記入項目	FD 記入項目
PROCEDURE DIVISION	OPEN OUTPUT OPEN EXTEND WRITE CLOSE	OPEN OUTPUT OPEN EXTEND WRITE CLOSE	OPEN OUTPUT OPEN EXTEND WRITE CLOSE

関連タスク
211 ページの『空のファイルのオープン』
214 ページの『VSAM ファイル内のレコードの更新』

ロード済みファイル (レコードが入っているファイル) のオープン

既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、または OPEN EXTEND を使用してください。

VSAM 入力順または相対レコード・ファイルを EXTEND としてオープンする場合には、追加されたレコードは、ファイル内の最後の既存のレコードの後に置かれます。

VSAM キー順ファイルを 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。

関連タスク
211 ページの『空のファイルのオープン』
219 ページの『z/OS および UNIX のもとでの VSAM データ・セットの操作』

関連参照 212 ページの『VSAM ファイルにレコードをロードするために使用されるステートメント』 *z/OS DFSMS カタログのためのアクセス方式サービス・プログラム*

VSAM ファイルからのレコードの読み取り

ファイルからレコードを検索 (READ) するには、READ ステートメントを使用します。レコードを読み取るためには、ファイルを INPUT または I-O としてオープンしていなければなりません。プログラムでは、それぞれの READ の後でファイル状況キーを検査しなければなりません。

VSAM 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。

VSAM 索引付きファイルおよび相対レコード・ファイルの中のレコードは、次のように検索することができます。

順次　索引付きファイルの場合は、ファイル位置標識のファイル位置標識から、使用中のキー (RECORD KEY または ALTERNATE RECORD KEY) の昇順に従い、相対ファイルの場合は、相対レコード位置の昇順に従って

ランダム
READ 要求の前に、RECORD KEY または ALTERNATE RECORD KEY または RELATIVE KEY をどのように設定するかによって、任意の順序で

動的　順次とランダムの混合で

動的アクセスの場合、順次検索には READ NEXT を使用し、ランダム検索 (キーによる) には READ を使用することによって、特定のレコードの直接読み取りと、いくつかのレコードの順次読み取りを切り替えることができます。

特定のレコードから順次に読み取りを行いたい場合には、READ NEXT ステートメントの前に START ステートメントを使用して、ファイル位置標識を、特定のレコードを指すように設定してください。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。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。

```
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、VSAM 索引付きファイルに対して直接 READ が実行されると、その代替キーを持つデータ・セット (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データ・セット・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。同じ代替キー値を持つ、読み取るべきレコードがほかにある場合には、02 のファイル状況コードが戻されます。そのキー値を持つ最後のレコードが読み取られると、00 のコードが戻されます。

関連概念
210 ページの『ファイル位置標識』

関連タスク
205 ページの『VSAM ファイルのアクセス・モードの指定』

VSAM ファイル内のレコードの更新

VSAM ファイルを更新するには、これらの PROCEDURE DIVISION ステートメントを使用します。

表 32. **VSAM ファイルのレコードを更新するためのステートメント**

アクセス方式	ESDS	KSDS	RRDS
ACCESS IS SEQUENTIAL	OPEN EXTEND WRITE CLOSE または OPEN I-O READ REWRITE 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	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	OPEN I-O READ NEXT WRITE REWRITE START DELETE CLOSE	OPEN I-O READ NEXT WRITE REWRITE START DELETE CLOSE
ACCESS IS DYNAMIC (ランダム処理)	適用されない	OPEN I-O READ WRITE REWRITE DELETE CLOSE	OPEN I-O READ WRITE REWRITE DELETE CLOSE

関連参照 212 ページの『VSAM ファイルにレコードをロードするために使用されるステートメント』

VSAM ファイルへのレコードの追加

COBOL WRITE ステートメントを使用すると、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することができます。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・サイズよりも大きいものであってはなりません。プログラムでは、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なければなりません。

順次のレコード追加: OUTPUT または EXTEND としてオープンされた VSAM 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書か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には、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 節に RELATIVE KEY データ項目を組み込むと、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。

ランダムまたは動的なレコード追加: レコードを索引付きデータ・セットに書き込むとき、ACCESS IS RANDOM または ACCESS IS DYNAMIC であれば、レコードは任意の順序で書き込むことができます。

VSAM ファイル内のレコードの置換

VSAM ファイル内のレコードを置換するには、I-O として開いたファイルに対して REWRITE を使用してください。ファイルが、I-O として開かれていない場合には、レコードは再書き込みされず、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 を発行します。
- 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 を発行します。

VSAM 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 I-O を開き、DELETE ステートメントを使用します。順次ファイルに対して DELETE を使用することはできません。

ACCESS IS SEQUENTIAL を使用する場合、またはファイルにスパン・レコードが含まれている場合には、まず削除すべきレコードをプログラムで読み取らなければなりません。その後、読み取られたレコードを DELETE で除去します。DELETE の前の READ が成功しなかった場合には、削除は行われず、状況キー値が 92 に設定されます。

ACCESS IS RANDOM または ACCESS IS DYNAMIC を使用する場合は、削除すべきレコードを前もってプログラムで読み取る必要はありません。レコードを削除するには、削除するレコードのキーを RECORD KEY データ項目に移動してから、DELETE を出します。プログラムでは、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なければなりません。

VSAM ファイルのクローズ

プログラムを VSAM ファイルから切り離すには、CLOSE ステートメントを使用します。既にクローズされているファイルをクローズしようとすると、論理エラーになります。それぞれの CLOSE 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。

VSAM ファイルをクローズしなくても、以下の条件のもとでは、ファイルは自動的にクローズされます。ただし、実行単位内の OS/VS COBOL プログラムのいずれかで定義されたファイルを除きます。

- 実行単位が正常に終了すると、その実行単位内の COBOL プログラムのいずれかで定義されている、オープン状態のすべてのファイルはクローズされます。
- 実行単位が異常終了した場合は、TRAP(ON) ランタイム・オプションが設定されていれば、その実行単位内の COBOL プログラムのいずれかで定義されているオープン状態のファイルはすべてクローズされます。
- 言語環境プログラム条件処理が完了し、その条件が起こったルーチン以外のルーチンでアプリケーションが再開すると、実行単位内の COBOL プログラムのうち、再度呼び出されて再入される可能性のあるプログラムに定義されているオープン・ファイルはクローズされます。

  条件が処理された後でプログラムが実行を再開する位置を変更することができます。この変更を行うには、例えば、CEEMRCR 呼び出し可能サービスを使用して再開カーソルを移動するか、または C longjmp ステートメントのような言語構造体を使用してください。
- COBOL サブプログラムに CANCEL を出すと、そのプログラム内で定義されているオープン状態の非外部ファイルがすべてクローズされます。
- INITIAL 属性を持つ COBOL サブプログラムが制御権を戻すと、そのプログラム内で定義されているオープン状態の非外部ファイルがすべてクローズされます。
- マルチスレッド・アプリケーションのスレッドが終了すると、同じスレッド内部からオープンした外部ファイルと非外部ファイルはいずれもクローズされます。

このような暗黙の CLOSE 操作が実行されたときは、DATA DIVISION のファイル状況キー・データ項目が設定されますが、EXCEPTION/ERROR および LABEL 宣言は呼び出されません。

エラー: マルチスレッド・アプリケーションで VSAM ファイルをオープンした場合は、ファイルをオープンしたのと同じ実行スレッドからクローズする必要があります。異なるスレッドからファイルをクローズしようとすると、ファイル状況コードの条件 90 でクローズが失敗します。

VSAM ファイルでのエラー処理

入出力ステートメントの操作が失敗しても、COBOL がユーザーに代わって訂正処置を行うことはありません。

VSAM ファイルを扱う際の OPEN および CLOSE エラーはすべて、それがプログラム内の論理エラーであっても外部ストレージ・メディア上の入出力エラーであって

も、また、たとえ DECLARATIVE も FILE STATUS 節もコーディングされていなくても、制御を COBOL プログラムに戻します。

他の入出力ステートメントの操作の失敗については、重大レベルより軽度のエラーの後でプログラムに実行を継続させるかどうかを選択します。

COBOL は、特定の VSAM 入出力エラーを代行受信して処理するために、以下の方法を提供します。

- ファイル終了句 (AT END)
- EXCEPTION/ERROR 宣言
- FILE STATUS 節 (ファイル状況キーおよび VSAM 状況コード)
- INVALID KEY 句

プログラム内に定義するそれぞれの VSAM ファイルごとに状況キーを定義しなければなりません。それぞれの入力要求または出力要求 (特に、OPEN および CLOSE) の後で、状況キーの値を検査してください。

ファイル状況キーや宣言をコーディングしなかった場合、重大な VSAM 処理エラーが起こると、メッセージが出され、言語環境プログラム条件が信号で伝えられます。ランタイム・オプション ABTERMENC(ABEND) が指定されていると、この条件の結果として、異常終了が引き起こされます。

関連タスク
265 ページの『入出力操作でのエラーの処理』
271 ページの『VSAM 状況コードの使用 (VSAM ファイルのみ)』

関連参照
データ・セットに対する z/OS DFSMS マクロ命令
(VSAM マクロの戻りコードおよび理由コード)

パスワードによる VSAM ファイルの保護

z/OS システムで推奨するセキュリティー機構は RACF® ですが、Enterprise COBOL では、無許可アクセスや無許可更新を防止するため、VSAM ファイルに明示パスワードを使用することもサポートします。

明示パスワードを使用するには、FILE-CONTROL 段落で PASSWORD 節をコーディングしてください。この節は、ファイルのカタログ記入項目に読み取りまたは更新パスワードが入っている場合にのみ使用してください。

- カタログ記入項目に読み取りパスワードが入っている場合には、PASSWORD 節が FILE-CONTROL 段落で使用され、DATA DIVISION で記述されていない限り、ファイルは COBOL プログラム内でオープンおよびアクセスできません。ファイルがオープンされるときには、参照された *data-name* に有効なパスワードが入っていなければなりません。
- カタログ記入項目に更新パスワードが入っている場合には、ファイルはオープンおよびアクセスすることはできますが、PASSWORD 節が FILE-CONTROL 段落で使用され、DATA DIVISION で記述されていない限り、更新できません。

- カタログ記入項目に読み取りパスワードと更新パスワードの両方が入っている場合には、プログラム内でファイルを読み取るときと更新するときの両方に更新パスワードを使用してください。

プログラムでレコードの検索だけを行い、更新を行わない場合には、読み取りパスワードしか必要ありません。プログラムでファイルをロードまたは更新する場合には、カタログされた更新パスワードを指定しなければなりません。

索引付きファイルの場合は、RECORD KEY の PASSWORD データ項目に有効なパスワードが含まれていなければ、ファイルを正常にオープンすることはできません。

VSAM 索引付きファイルをパスワード保護する場合、完全にパスワード保護するためには、すべての代替索引もパスワード保護しなければなりません。それぞれの代替索引はそれ自体のパスワードを持つため、PASSWORD 節を置く場所が重要になります。PASSWORD 節は、それが適用されるキー節の直後になければなりません。

例: VSAM 索引付きファイルのパスワード保護

以下に、パスワード保護のある VSAM 索引付きファイルについて使用される COBOL コードの例を示します。

```
. . .
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT LIBFILE
      ASSIGN TO PAYMAST
      ORGANIZATION IS INDEXED
      RECORD KEY IS EMPL-NUM
         PASSWORD IS BASE-PASS
      ALTERNATE RECORD KEY IS EMPL-PHONE
         PASSWORD IS PATH1-PASS
. . .
WORKING-STORAGE SECTION.
01  BASE-PASS          PIC X(8) VALUE "25BSREAD".
01  PATH1-PASS         PIC X(8) VALUE "25ATREAD".
```

z/OS および UNIX のもとでの VSAM データ・セットの操作

VSAM ファイルを z/OS および UNIX のもとで使用する場合、アクセス方式サービス・プログラム (IDCAMS) コマンド、環境変数、および JCL (ジョブ制御言語) に関していくつかの特別なコーディング上の考慮事項があります。

VSAM ファイルが *使用可能* となるのは、以下のすべての条件が真である場合です。

- VSAM ファイルが、アクセス方式サービス・プログラムを使用して定義されている。
- DD ステートメント、環境変数、または ALLOCATE コマンドを指定することによって、プログラムで VSAM ファイルを定義している。
- 既にレコードが入っている。

VSAM ファイルは、定義されていても、レコードが入ったことがなければ、*利用不能* です。

VSAM 順次ファイルに対する OPEN ステートメントの完了時には、必ず戻りコード 0 が戻されます。

ファイルを空にするには、アクセス方式サービス・プログラム REPRO コマンドを使用してください。この方法でレコードを削除すると、ファイルの最も高い相対バイト・アドレス (RBA) が 0 にリセットされます。ファイルは事実上空になり、COBOL にとってはそれがレコードを含んだことがないかのように見えます。

関連タスク
11 ページの『オペレーティング・システムに対してファイルを定義する』
『VSAM ファイルの定義』
221 ページの『代替索引の作成』
223 ページの『VSAM ファイルの割り振り』
225 ページの『RLS による VSAM ファイルの共用』

VSAM ファイルの定義

Enterprise COBOL の VSAM 入力順、キー順、相対レコードの各データ・セットは、アクセス方式サービス・プログラム (IDCAMS) を使用してそれらを定義してからでなければ処理できません。

VSAM クラスター とは、VSAM データ・セットの論理定義であり、1 つまたは 2 つのコンポーネントを持っています。

- VSAM クラスターのデータ・コンポーネントには、データ・レコードが入れられます。
- VSAM キー順クラスターの索引コンポーネントは、索引レコードから構成されます。

VSAM データ・セット (クラスター) を定義するには、DEFINE CLUSTER アクセス方式サービス・コマンドを使用してください。このプロセスには、データ転送を行わずに統合カタログ内に項目を作成することが含まれます。クラスターについては、以下の情報を定義してください。

- 記入項目の名前。
- この定義およびそのパスワードを入れるカタログの名前 (デフォルト名を使用できます)。
- 編成 (順次、索引付き、または相対)。
- データ・セットが占有する装置およびボリューム。
- データ・セットに必要なスペース。
- レコード・サイズおよび制御インターバル・サイズ (CISIZE)。
- 将来のアクセスに必要なパスワード (もしあれば)。

クラスター内のデータ・セットの種類によっては、クラスターごとに以下の情報も定義してください。

- VSAM 索引付きデータ・セット (KSDS) の場合は、レコード内の基本キーの長さおよび位置を指定してください。
- VSAM 固定長相対レコード・データ・セット (RRDS) の場合は、最大の COBOL レコード・サイズより大か等しいレコード・サイズを指定してください。

```
DEFINE CLUSTER NUMBERED
RECORDSIZE(n,n)
```

このようにデータ・セットを定義した場合、すべてのレコードは、固定スロット・サイズ *n* まで埋め込まれます。 RECORD 節の RECORD IS VARYING ON *data-name* 形式を使用した場合、WRITE または REWRITE は、DEPENDING ON *data-name* で指定された長さを、VSAM によって転送されるレコードの長さとして使用します。このデータは、その後、固定スロット・サイズに埋め込まれます。 READ ステートメントは必ず、固定スロット・サイズを DEPENDING ON *data-name* で戻します。

- VSAM 可変長相対レコード・データ・セット (RRDS) の場合は、予期される平均サイズ COBOL レコードと予期される最大サイズ COBOL レコードを指定してください。

```
DEFINE CLUSTER NUMBERED
RECORDSIZE(avg,m)
```

予期される平均サイズ COBOL レコードは、予期される最大サイズ COBOL レコードより小さくなければなりません。

関連タスク
『代替索引の作成』
223 ページの『VSAM ファイルの割り振り』
204 ページの『VSAM ファイルの相対編成の指定方法』

関連参照
z/OS DFSMS カタログのためのアクセス方式サービス・プログラム

代替索引の作成

代替索引は、複数のキーを使用するデータ・セット内のレコードへのアクセスを提供するものです。索引付きデータ・セット (KSDS) の基本索引キーと同じ方法でレコードにアクセスします。

代替索引の使用を計画している場合には、以下のことを知っておかなければなりません。

- 索引が関連付けられるデータ・セット (基本クラスター) のタイプ。
- キーが固有であるか固有でないか。
- 索引をパスワード保護するかどうか。
- 代替索引使用のパフォーマンスの側面。

代替索引は、実際には、VSAM データ・セットのキーへのポインターを含む VSAM データ・セットであるため、代替索引および代替索引パス (代替索引と基本索引の間の関係を確立するエンティティー) を定義しなければなりません。代替索引を定義した後、代替索引とその基本クラスターの間の関係 (パス) を確立するカタログ記入項目を作成してください。このパスにより、代替キーを介して基本クラスターのレコードにアクセスすることが可能になります。

代替索引を使用するには、以下のステップに従ってください。

1. DEFINE ALTERNATEINDEX コマンドを使用して、代替索引を定義します。このコマンドの中では、以下の項目を指定します。
   - 代替索引の名前。
   - 関連する VSAM 索引付きデータ・セットの名前。
   - 代替索引のレコードにおける位置と、代替索引が固有であるかどうか。
   - データ・セットの変更時に代替索引を更新するかどうか。
   - この定義およびそのパスワードを入れるカタログの名前 (デフォルト名を使用できます)。

   COBOL プログラムでは、代替索引は、FILE-CONTROL 段落の ALTERNATE RECORD KEY 節によってのみ識別されます。 ALTERNATE RECORD KEY 定義は、カタログ記入項目の定義と一致していなければなりません。カタログしたパスワード項目は、ALTERNATE RECORD KEY 句の後に直接コーディングしなければなりません。
2. DEFINE PATH コマンドを使用して、代替索引を基本クラスター (代替索引を介してアクセスするデータ・セット) に関連付けます。このコマンドの中では、以下の項目を指定します。
   - パスの名前。
   - パスが関連付けられている代替索引。
   - 代替索引を含むカタログの名前。

   基本クラスターと代替索引は、同じカタログ内の項目によって記述されます。
3. VSAM 索引付きデータ・セットをロードします。
4. BLDINDEX コマンドを使用して (一般に)、代替索引を作成します。入力ファイルを索引付きデータ・セット (基本クラスター) として識別し、出力ファイルを代替索引またはそのパスとして識別します。BLDINDEX は、VSAM 索引付きデータ・セット (つまり、基本クラスター) のすべてのレコードを読み取り、代替索引を作成するのに必要なデータを取り出します。

   あるいは、ランタイム・オプション AIXBLD を使用して、実行時に代替索引を作成することもできます。しかし、これはパフォーマンスに悪影響を及ぼすことがあります。

『例: 代替索引の項目』

関連タスク
204 ページの『代替索引の使用』

関連参照
言語環境プログラム・プログラミング・リファレンス (AIXBLD (COBOL のみ))

例: 代替索引の項目

次の例では、COBOL FILE-CONTROL 記入項目と、2 つの代替索引を持つ VSAM 索引付きファイルに対する DD ステートメントまたは環境変数の関係を示しています。

JCL を使用する場合:

```
//MASTERA     DD   DSNAME=clustername,DISP=OLD      (1)
//MASTERA1    DD   DSNAME=path1,DISP=OLD           (2)
//MASTERA2    DD   DSNAME=path2,DISP=OLD           (3)
```

環境変数を使用する場合:

```
export MASTERA=DSN(clustername),OLD                (1)
export MASTERA=DSN(path1),OLD                      (2)
export MASTERA=DSN(path2),OLD                      (3)
. . .
FILE-CONTROL.
  SELECT MASTER-FILE ASSIGN TO MASTERA             (4)
      RECORD KEY IS EM-NAME
      PASSWORD IS PW-BASE                          (5)
      ALTERNATE RECORD KEY IS EM-PHONE             (6)
          PASSWORD IS PW-PATH1
      ALTERNATE RECORD KEY IS EM-CITY              (7)
          PASSWORD IS PW-PATH2.
```

(1) 基本クラスター名は *clustername* です。

(2) 最初の代替索引パスの名前は *path1* です。

(3) 2 番目の代替索引パスの名前は *path2* です。

(4) 基本クラスターの DD 名または環境変数名は、ASSIGN 節で指定されます。

(5) パスワードはその索引の直後に続きます。

(6) キー EM-PHONE は、最初の代替索引に関連します。

(7) キー EM-CITY は、2 番目の代替索引に関連します。

関連タスク

221 ページの『代替索引の作成』

VSAM ファイルの割り振り

VSAM データ・セットはすべて、アクセス方式サービス・プログラム DEFINE コマンドを介して事前定義し、カタログしなければなりません。 VSAM データ・セットに関する情報の大部分は、カタログに入っているので、最小限の DD または環境変数情報のみを指定するだけで構いません。

VSAM ファイル (索引付き、相対、および順次) の割り振りは、COBOL ファイルの割り振りに関する一般的な規則に従います。

環境変数を使用して VSAM ファイルを割り当てる場合、変数名は大文字でなければなりません。通常、関係がある変数は、入力およびデータ・バッファーだけです。以下のオプションを示された順番で指定する必要があります。他のオプションを指定してはなりません。

1. DSN(*dsname*)。ここで、*dsname* は基本クラスターの名前です。
2. OLD または SHR

VSAM ファイルおよび対応する export コマンドに必要な基本 DD ステートメントは、次のとおりです。

```
//ddname   DD   DSN=dsname,DISP=SHR,AMP=AMORG
export  evname="DSN(dsname),SHR"
```

いずれの場合も、*dsname* は、アクセス方式サービス・プログラムの DEFINE CLUSTER コマンドまたは DEFINE PATH コマンドで使用した名前と同じ名前でなければなりません。データ・セットは既にカタログされているため、DISP は OLD または SHR でなければなりません。JCL を使用する際に MOD を指定すると、そのデータ・セットは OLD として扱われます。

AMP は、データ・セットについてプログラムによって提供される情報を補足する VSAM JCL パラメーターです。AMP が有効になるのは、プログラムが VSAM ファイルをオープンしたときです。AMP パラメーターを介して設定されたパラメーターは、カタログ内の情報またはプログラムによって提供される情報に優先します。AMP パラメーターは、次の場合にのみ必要です。

- ダミー VSAM データ・セットを使用する場合。次はその例です。

```
//ddname  DD   DUMMY,AMP=AMORG
```

- 追加の索引またはデータ・バッファーを要求する場合。次はその例です。

```
//ddname   DD  DSN=VSAM.dsname,DISP=SHR,
//             AMP=('BUFNI=4,BUFND=8')
```

環境変数を使用して VSAM データ・セットを割り振る場合は、AMP を指定することはできません。

VSAM 基本クラスターの場合は、SELECT 節の後の ASSIGN 節で指定した名前と同じシステム名 (DD 名または環境変数名) を指定してください。

COBOL プログラムで代替索引を使用する場合は、基本クラスターのシステム名 (DD ステートメントまたは環境変数) だけでなく、各代替索引パスごとのシステム名も指定する必要があります。プログラム内で代替索引パスのシステム名を明示的に宣言するための言語メカニズムはありません。このため、それぞれの代替索引パスのシステム名 (DD 名または環境変数名) を以下のガイドラインに従って作成する必要があります。

- 基本クラスター名に整数を連結します。
- プログラム内のファイルに定義された最初の代替レコード (FILE-CONTROL 段落の ALTERNATE RECORD KEY 節) と関連付けられたパスに 1 を使用します。
- そのファイルのその後の代替レコード定義と関連付けられたパスごとに 1 ずつ増やします。

例えば、基本クラスターのシステム名が ABCD である場合、プログラム内のファイルに定義された最初の代替索引パスは ABCD1、2 番目の代替索引パスのシステム名は ABCD2 となり、以後同様に続きます。

基本クラスター・システム名とシーケンス番号を合わせた長さが 8 文字を超える場合は、システム名の基本クラスター部分の右側が切り捨てられ、連結結果が 8 文字になるよう切り詰められます。例えば、基本クラスターのシステム名が ABCDEFGH である場合、最初の代替索引パスのシステム名は ABCDEFG1、10 番目の代替索引パスのシステム名は ABCDEF10 となり、以後同様に続きます。

関連タスク

164 ページの『ファイルの割り振り』

関連参照
MVS プログラム管理: ユーザーズ・ガイドおよび解説書

RLS による VSAM ファイルの共用

VSAM JCL パラメーター RLS を使用することによって、VSAM とのレコード・レベル共用を指定することができます。 RLS を指定することは、COBOL プログラム実行時に RLS モードを要求する唯一の方法です。

整合性を保つ読み取りプロトコルが必要なときは RLS=CR を使用し、読み取り保全性プロトコルが必要でないときは RLS=NRI を使用してください。環境変数を使用して VSAM データ・セットを割り振る場合は、RLS を指定することはできません。

関連タスク
『RLS モードでの VSAM ファイルに関する更新問題の回避』
226 ページの『RLS モードでの VSAM ファイルのエラーの処理』

関連参照
226 ページの『RLS を使用する際の制約事項』

RLS モードでの VSAM ファイルに関する更新問題の回避

VSAM データ・セットが I-O (更新) 用に RLS モードで開かれるとき、指定されている RLS の値 (RLS=CR または RLS=NRI) にかかわらず、最初の READ によりレコードの排他ロックが行われます。

COBOL ファイルが ACCESS RANDOM と定義されている場合、VSAM は、 WRITE または REWRITE ステートメントの発行後あるいは 別のレコードに対する READ ステートメントの発行後に、レコードに対する排他ロックを解除します。WRITE または REWRITE の処理が実行されると、VSAM は即時にレコードを書き出します。

しかし、COBOL ファイルが ACCESS DYNAMIC として定義されている場合、VSAM は、WRITE や REWRITE ステートメントの後でも、READ ステートメントの後でもレコードの排他ロックを解放しません。ただし、I-O ステートメントにより VSAM が別の制御インターバル (CI) へ移動させられる場合には解放されます。この結果、WRITE または REWRITE が実行されても、処理が別の CI に移動してロックが解除されるまで、VSAM はレコードを書き込みません。 ACCESS DYNAMIC を使用する際に、レコードの即時の書き出しや排他ロックの即時の解除 (あるいはその両方) を実現する 1 つの方法は、CI ごとに 1 レコードのみを許容する VSAM データ・セットを定義することです。

RLS=CR を指定すると、レコードはロックされ、別のレコードに対する別の READ が要求されるまで、そのレコードへの更新を妨げることができます。読み取り中のレコードに関するロックが有効な間、他のユーザーは同じレコードに対する READ を要求することはできますが、読み取りロックが解除されるまで、そのレコードを更新することはできません。 RLS=NRI を指定した場合、入力の READ が発行されたときにはロックは有効になりません。別のユーザーがそのレコードを更新する可能性があります。

RLS=CR のロッキング規則により、アプリケーションはレコード・ロックが使用可能になるのを待つ可能性があります。この待機によって、入力の READ がスローダウ

ンことがあります。 RLS=CR を使用するためには、アプリケーションのロジックを修正することが必要な場合があります。アプリケーションが複数の更新環境で正しく機能することを確認するまでは、回復不能範囲を更新するバッチ・ジョブには、RLS パラメーターを使用しないでください。

VSAM データ・セットを RLS モードで INPUT または I-O 処理用にオープンするときには、OPEN または START を出すのは READ の *直前* が良いでしょう。OPEN または START と READ の間に遅延があると、 OPEN または START の実行後に、アプリケーションが位置を定めたレコードの前の位置に他のユーザーがレコードを追加する可能性が高くなります。COBOL 実行時には、OPEN が要求された時点で VSAM データ・セットの開始位置が明示的に指し示されますが、READ が遅れると、他のユーザーによって追加されたレコードによって、VSAM データ・セットの実際の開始位置が変わる可能性があります。

RLS を使用する際の制約事項

RLS モードを使用すると、VSAM クラスター属性およびランタイム・オプションにいくつかの制約事項が適用されます。

以下の制約事項に留意してください。

- VSAM ファイルをオープンするとき、VSAM クラスター属性 KEYRANGE および IMBED はサポートされません。
- VSAM クラスター属性 REPLICATE は推奨されません。これは、システム規模のバッファー・プールと、ストレージ階層内の潜在的に大きな CF キャッシュ構造によって、利点が打ち消されるためです。
- VSAM ファイルをオープンするとき、VSAM が空のパスのオープンを認めないため、AIXBLD ランタイム・オプションはサポートされません。代替索引データ・セットを作成するために AIXBLD ランタイム・オプションが必要である場合には、VSAM データ・セットを非 RLS モードでオープンしなければなりません。
- SIMVRD ランタイム・オプションは VSAM ファイルに対してはサポートされません。
- 一時データ・セットは使用できません。

RLS モードでの VSAM ファイルのエラーの処理

アプリケーションが RLS モードの VSAM データ・セットにアクセスする場合には、*それぞれの* 要求の後でファイル状況および VSAM フィードバック・コードを検査するようにしてください。

入出力処理時にアプリケーションが「SMSVSAM サーバー利用不能」に直面した場合には、VSAM ファイルの再オープンを試行する前にそれを明示的にクローズしてください。VSAM は、その障害を表す戻りコード 16 を生成します。フィードバック・コードはありません。COBOL プログラムでは、VSAM 戻りコード 16 に関して 2 番目のファイル状況域の最初の 2 バイトを検査することができます。COBOL 実行時には、OPEN 処理中にエラーが発生すると、IGZ0205W メッセージが生成され、ファイルは自動的にクローズされます。

その他のすべての RLS モード・エラーは、VSAM 戻りコード 4、8、または 12 を戻します。

関連タスク
271 ページの『VSAM 状況コードの使用 (VSAM ファイルのみ)』

VSAM パフォーマンスの向上

多くの場合、COBOL および VSAM のパフォーマンスの調整を担当するのはシステム・プログラマーです。アプリケーション・プログラマーとしては、以下にリストされている VSAM の局面を制御することができます。

表 33. **VSAM パフォーマンスを改善する方法**

VSAM の局面	行うことができること	理由の説明とコメント
アクセス方式サービス・プログラムの呼び出し	IDCAMS を使用した、代替索引の事前作成	
バッファリング	順次アクセスの場合は、もっと多くのデータ・バッファーを要求し、ランダム・アクセスの場合は、もっと多くの索引バッファーを要求します。 ACCESS IS DYNAMIC の場合は、BUFND と BUFNI の両方を指定します。 アプリケーションが対話式に実行される場合を除いて、追加のバッファーのコーディングを避けます。さらに、入出力遅延が原因であることが考えられる応答時間問題が生じた場合にのみ、バッファーをコーディングするようにします。	デフォルトは、1 つの索引バッファー (BUFNI) と 2 つのデータ・バッファー (BUFND) です。
アクセス方式サービス・プログラムを使用してのレコードのロード	以下の場合は、アクセス方式サービス・プログラムの REPRO コマンドを使用します。 • ターゲット索引付きデータ・セットに既にレコードが入っている。 • 入力順次データ・セットに、更新されるかまたは索引付きデータ・セットに挿入されるレコードが入っている。 COBOL プログラムを使用してファイルをロードする場合は、OPEN OUTPUT および ACCESS SEQUENTIAL を使用します。	これらの条件のもとでは、REPRO コマンドは、COBOL プログラムと同じ速さまたはそれよりも速く、索引付きデータ・セットを更新することができます。

表 33. **VSAM パフォーマンスを改善する方法** *(続き)*

VSAM の局面	行うことができること	理由の説明とコメント
ファイル・アクセス・モード	最高のパフォーマンスを得るためには、順次にレコードにアクセスします。	動的アクセスは、順次アクセスよりは効率が低く、ランダム・アクセスよりは効率的です。ランダム・アクセスでは、VSAM がそれぞれの要求について索引にアクセスしなければならないため、EXCP が増加します。
キーの設計	レコード内のキーを、高位部分が比較的一定になり、低位部分が頻繁に変わるように設計します。	この方法を使用すると、キーが最適に圧縮されます。
複数の代替索引	複数の代替索引の使用を避けます。	更新は基本パスを介して適用されなければならず、複数の代替パスを介して反映されることになるため、複数の代替索引を使用すると、パフォーマンスが低下する可能性があります。
相対ファイル編成	VSAM 可変長相対データ・セットではなく、VSAM 固定長相対データ・セットを使用します。	VSAM 固定長相対データ・セットは、VSAM 可変長相対データ・セットと比較して、スペース的には効率がよくありませんが、実行時の効率はよくなります。
制御インターバル・サイズ (CISZ)	システム・プログラマーに、VSAM データ・セットのデータ・アクセスおよび将来の成長性に関する情報を提供してください。この情報から、システム・プログラマーは最適な制御インターバル・サイズ (CISZ) および FREESPACE サイズ (FSPC) を判別することができます。 制御域 (CA) の分割を最小限にするために、CISZ および FSPC に適切な値を選択します。クラスターに対して LISTCAT ALL コマンドを出して、現在の CA 分割の数を診断してから、すべての CA 分割を定期的に省略するためにクラスターを (EXPORT、IMPORT、または REPRO を使用して) 圧縮することができます。	VSAM は、直接アクセス記憶装置 (DASD) の使用アルゴリズムに最も適するように CISZ を計算しますが、これがアプリケーションにとっては効率的でないことがあります。 4K の平均 CISZ はほとんどのアプリケーションに適しています。CISZ をより小さくすることは、挿入を犠牲にして (つまり、CISZ 分割がより多くなり、結果としてデータ・セット内のスペースがより多くなる)、ランダム処理の検索をより速くすることを意味します。CISZ をより大きくすると、各 READ についてチャネル間で転送されるデータがより多くなります。これは、大きな OS BLKSIZE と同様に、順次処理の場合に効率がよりよくなります。 多くの制御域 (CA) 分割は、VSAM のパフォーマンスにとっては不利です。FREESPACE 値は、ファイルの使用法に応じて、CA 分割に影響を与える場合があります。

関連タスク

205 ページの『VSAM ファイルのアクセス・モードの指定』

z/OS DFSMS: Using Data Sets (リソース・プールの作成、フリー・スペースの最適なパーセントの選択)

関連参照

z/OS DFSMS カタログのためのアクセス方式サービス・プログラム

第 11 章 line-sequential ファイルの処理

行順次ファイルは、階層ファイル・システム (HFS) に常駐し、印刷可能文字および特定の制御文字のみをデータとして含みます。各レコードは、EBCDIC 改行文字 (X'15') で終了します (この文字はレコードの長さには含まれません)。

これらは順次ファイルなので、各レコードは入力順序に従って次々に入れられます。プログラムでは、これらのファイルを順次にのみ処理することができ、レコードを、それらがファイルに入っているのと同じ順序で検索します (READ ステートメントを使用して)。新しいレコードは、前のレコードの後に入れられます。

プログラムで行順次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 内でファイルを識別および記述する
- PROCEDURE DIVISION 内でファイル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
『COBOL での行順次ファイルおよびレコードの定義』
232 ページの『行順次ファイルの構造の記述』
234 ページの『行順次ファイル用の入出力ステートメントのコーディング』
237 ページの『行順次ファイルのエラーの処理』
233 ページの『行順次ファイルの定義および割り振り』
UNIX システム・サービス・ユーザーズ・ガイド

関連参照
232 ページの『許可される制御文字』

COBOL での行順次ファイルおよびレコードの定義

COBOL プログラムのファイルを行順次ファイルとして定義し、そのファイルを対応する外部ファイル名 (DD 名または環境変数名) に関連付けるには、 ENVIRONMENT DIVISION の FILE-CONTROL 段落を使用します。

外部ファイル名とは、ファイルがオペレーティング・システムに認識されるときに使用される名前です。次の例では、COMMUTER-FILE は、プログラムがファイルに使用する名前であり、COMMUTR は外部名です。

```
FILE-CONTROL.
    SELECT COMMUTER-FILE
    ASSIGN TO COMMUTR
    ORGANIZATION IS LINE SEQUENTIAL
    ACCESS MODE IS SEQUENTIAL
    FILE STATUS IS ECODE.
```

外部名の前に、ASSIGN *assignment-name* 節に編成フィールド (S- または AS-) を含めてはなりません。ACCESS 句と FILE STATUS 節はオプションです。

関連タスク
『行順次ファイルの構造の記述』
234 ページの『行順次ファイル用の入出力ステートメントのコーディング』
233 ページの『行順次ファイルの定義および割り振り』

関連参照
『許可される制御文字』

許可される制御文字

印刷可能文字以外で行順次ファイルに入れることができるのは、次の表に示す制御文字だけです。 16 進値は、EBCDIC で示されています。

16 進値	制御文字
X'05'	水平タブ
X'0B'	垂直タブ
X'0C'	用紙送り
X'0D'	復帰
X'0E'	DBCS シフトアウト
X'0F'	DBCS シフトイン
X'15'	改行
X'16'	バックスペース
X'2F'	アラーム

改行文字は、レコード区切り文字として扱われます。他の制御文字はデータとして扱われ、レコードの一部分です。

関連タスク
231 ページの『COBOL での行順次ファイルおよびレコードの定義』

行順次ファイルの構造の記述

FILE SECTION で、そのファイルのファイル記述 (FD) 記入項目をコーディングします。関連するレコード記述項目では、*record-name* およびレコード長を定義します。

RECORD 節を使用して、レコードの論理サイズ (バイト単位) をコーディングしてください。行順次ファイルはストリーム・ファイルです。行順次ファイルの本質は文字中心であるので、物理レコードは可変長です。

次の例は、行順次ファイルの FD 記入項目を示しています。

固定長レコードの場合:

```
FILE SECTION.
FD  COMMUTER-FILE
    RECORD CONTAINS 80 CHARACTERS.
```

```
01  COMMUTER-RECORD.
    05  COMMUTER-NUMBER        PIC  X(16).
    05  COMMUTER-DESCRIPTION   PIC  X(64).
```

可変長レコードの場合:

```
FILE SECTION.
FD  COMMUTER-FILE
    RECORD VARYING FROM 16 TO 80 CHARACTERS.
01  COMMUTER-RECORD.
    05  COMMUTER-NUMBER        PIC  X(16).
    05  COMMUTER-DESCRIPTION   PIC  X(64).
```

同一の固定サイズをコーディングし、どのレベル 01 レコード記述項目の OCCURS DEPENDING ON 節もファイルと関連付けられていない場合には、その固定サイズが論理レコード長になります。しかし、レコードの終わりにあるブランクはファイルに書き込まれないので、物理レコードは可変長になります。

関連タスク

231 ページの『COBOL での行順次ファイルおよびレコードの定義』
234 ページの『行順次ファイル用の入出力ステートメントのコーディング』
『行順次ファイルの定義および割り振り』

関連参照

データ部 -- ファイル記述項目 (*Enterprise COBOL 言語解説書*)

行順次ファイルの定義および割り振り

DD ステートメントまたは環境変数を使用して、HFS の行順次ファイルを定義できます。これらのファイルの割り振りは、COBOL ファイルの割り振りに関する一般的な規則に従います。

行順次ファイルを定義するには、ASSIGN 節の外部名と一致する名前を持つ、DD 割り振りまたは環境変数をコーディングしてください。

- DD 割り振り:
  - PATH='*absolute-path-name*' を指定する DD ステートメント
  - PATH('*absolute-path-name*') を指定する TSO 割り振り

  以下のオプションを指定することもできます。
  - PATHOPTS
  - PATHMODE
  - PATHDISP
- PATH(*absolute-path-name*) の値を持つ環境変数。これ以外の値を指定することはできません。

  例えば、*assignment-name* が COMMUTR である COBOL ファイル用として、プログラムに HFS ファイル /u/myfiles/commuterfile を使用させるには、次のコマンドを使用できます。

  ```
  export COMMUTR="PATH(/u/myfiles/commuterfile)"
  ```

関連タスク
164 ページの『ファイルの割り振り』
231 ページの『COBOL での行順次ファイルおよびレコードの定義』

関連参照
MVS プログラム管理: ユーザーズ・ガイドおよび解説書

行順次ファイル用の入出力ステートメントのコーディング

行順次ファイルを処理するには、下記の入出力ステートメントをコーディングします。

OPEN ファイルの処理を開始します。

行順次ファイルは、INPUT、OUTPUT、または EXTEND としてオープンすることができます。行順次ファイルを I-O としてオープンすることはできません。

READ ファイルからレコードを読み取ります。

順次処理では、プログラムは、ファイルの作成時に入力されたのと同じ順序で、レコードを次々と読み取ります。

WRITE ファイルにレコードを作成します。

プログラムは、ファイルの終わりに新しいレコードを書き込みます。

CLOSE ファイルとプログラムの接続を解放します。

関連タスク
231 ページの『COBOL での行順次ファイルおよびレコードの定義』
232 ページの『行順次ファイルの構造の記述』
『行順次ファイルのオープン』
235 ページの『行順次ファイルからのレコードの読み取り』
235 ページの『行順次ファイルへのレコードの追加』
236 ページの『行順次ファイルのクローズ』
237 ページの『行順次ファイルのエラーの処理』

関連参照
OPEN ステートメント (*Enterprise COBOL 言語解説書*)
READ ステートメント (*Enterprise COBOL 言語解説書*)
WRITE ステートメント (*Enterprise COBOL 言語解説書*)
CLOSE ステートメント (*Enterprise COBOL 言語解説書*)

行順次ファイルのオープン

プログラムは、READ または WRITE ステートメントを使用してファイルのレコードを処理する前に、OPEN ステートメントでファイルをオープンしなければなりません。 OPEN ステートメントは、ファイルが使用可能であるか、または動的に割り振られていれば、機能します。

プログラムの実行中にファイルが再度オープンされないように、CLOSE WITH LOCK をコーディングしてください。

関連タスク
『行順次ファイルからのレコードの読み取り』
『行順次ファイルへのレコードの追加』
236 ページの『行順次ファイルのクローズ』
233 ページの『行順次ファイルの定義および割り振り』

関連参照
OPEN ステートメント (*Enterprise COBOL 言語解説書*)
CLOSE ステートメント (*Enterprise COBOL 言語解説書*)

行順次ファイルからのレコードの読み取り

行順次ファイルから読み取りを行うには、ファイルをオープンして、READ ステートメントを使用します。プログラムは、ファイルの作成時に入力されたのと同じ順序でレコードを次々と読み取ります。

ファイル・レコード内の文字は、以下のいずれかの条件が起こるまで、一度に 1 文字ずつレコード域に読み込まれます。

- レコード区切り文字 (EBCDIC 改行文字) が検出される。

  区切り文字は廃棄され、レコード域の残りの部分はスペースで埋められます。(レコード域はファイル・レコードより長くなります。)
- レコード域全体が文字で埋められる。

  次の未読文字がレコード区切り文字であると、その文字は廃棄されます。次の READ は、次のレコードの先頭文字から読み取りを行います。(レコード域はファイル・レコードと同じ長さになります。)

  これ以外の場合、次の未読文字が、次の READ によって読み取られる先頭文字になります。(レコード域はファイル・レコードより短くなります。)
- ファイルの終わりが検出される。

  レコード域の残りは、スペースで埋められます。 (レコード域はファイル・レコードより長くなります。)

関連タスク
234 ページの『行順次ファイルのオープン』
『行順次ファイルへのレコードの追加』
236 ページの『行順次ファイルのクローズ』
233 ページの『行順次ファイルの定義および割り振り』

関連参照
OPEN ステートメント (*Enterprise COBOL 言語解説書*)
WRITE ステートメント (*Enterprise COBOL 言語解説書*)

行順次ファイルへのレコードの追加

行順次ファイルに追加を行うためには、ファイルを EXTEND としてオープンし、WRITE ステートメントを使用して、ファイルの最後のレコードの直後にレコードを追加します。

レコード域の終わりにあるブランクは除去され、レコード区切り文字が追加されます。レコード域内の文字 (先頭文字から、追加されたレコード区切り文字まで) は、1 つのレコードとしてファイルに書き込まれます。

行順次ファイルに書き込まれるレコードは、USAGE DISPLAY および DISPLAY-1 項目だけを含んでいなければなりません。ゾーン 10 進数データ項目は、符号なしにするか、または符号付きの場合には SIGN 節の SEPARATE 句で宣言する必要があります。

関連タスク
234 ページの『行順次ファイルのオープン』
235 ページの『行順次ファイルからのレコードの読み取り』
『行順次ファイルのクローズ』
233 ページの『行順次ファイルの定義および割り振り』

関連参照
OPEN ステートメント (*Enterprise COBOL 言語解説書*)
WRITE ステートメント (*Enterprise COBOL 言語解説書*)

行順次ファイルのクローズ

プログラムを行順次ファイルから切り離すには、CLOSE ステートメントを使用します。既にクローズされているファイルをクローズしようとすると、論理エラーになります。

行順次ファイルをクローズしなくても、以下の条件のもとでは、ファイルは自動的にクローズされます。

- 実行単位が正常に終了したとき。
- 実行単位が異常終了したときは、TRAP(ON) ランタイム・オプションが設定されている場合。
- 言語環境プログラム条件処理が完了し、その条件が起こったルーチン以外のルーチンでアプリケーションが再開すると、実行単位内の COBOL プログラムのうち、再度呼び出されて再入される可能性のあるプログラムに定義されているオープン・ファイルはクローズされます。

  (条件が処理された後で) プログラムが再開する位置を変更することができます。これは、言語環境プログラムの CEEMRCR 呼び出し可能サービスで再開カーソルを移動するか、または C longjmp 呼び出しのような HLL 言語構造体を使用して行います。

これらの暗黙の CLOSE 操作が実行されると、ファイル状況コードが設定されますが、EXCEPTION/ERROR 宣言は呼び出されません。

関連タスク
234 ページの『行順次ファイルのオープン』
235 ページの『行順次ファイルからのレコードの読み取り』
235 ページの『行順次ファイルへのレコードの追加』
233 ページの『行順次ファイルの定義および割り振り』

関連参照
CLOSE ステートメント (*Enterprise COBOL 言語解説書*)

行順次ファイルのエラーの処理

入出力ステートメントが失敗しても、COBOL がユーザーに代わって訂正処置を取るわけではありません。入出力ステートメントが失敗した後で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。

COBOL は、特定の行順次入出力エラーを代行受信して処理するために、以下の言語エレメントを提供します。

- ファイル終了句 (AT END)
- EXCEPTION/ERROR 宣言
- FILE STATUS 節

これらの技法を使用しなかった場合に、入出力処理でエラーが起こると、言語環境プログラム条件が発生します。

FILE STATUS 節を使用する場合は、必ずキーを調べ、キー値に基づいて適切なアクションを取るようにしてください。キーを調べなくても、プログラムが実行を継続できる可能性がありますが、予期したものではない結果になることがあります。

関連タスク
234 ページの『行順次ファイル用の入出力ステートメントのコーディング』
265 ページの『入出力操作でのエラーの処理』

第 12 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。

Enterprise COBOL では、ソートおよびマージ用の IBM ライセンス・プログラムは DFSORT™ または同等のプログラムでなければなりません。DFSORT に言及している場所では、任意の同等のソートまたはマージ製品を使用することができます。

SORT または MERGE ステートメントを含む COBOL プログラムは、16MB 境界より上または下のいずれにでも常駐させることができます。

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。

制約事項:

- z/OS UNIX のもとでは、SORT または MERGE ステートメントを含む COBOL プログラムを実行することはできません。この制約には、BPXBATCH も含まれます。
- THREAD オプションを指定してコンパイルしたプログラムで SORT ステートメントまたは MERGE ステートメントを使用することはできません。これには、オブジェクト指向構文を使用するプログラムとマルチスレッド・アプリケーションも含まれます。これらはいずれも THREAD オプションを必要とします。

関連概念
『ソートおよびマージ・プロセス』

関連タスク
241 ページの『ソートまたはマージ・ファイルの記述』
241 ページの『ソートまたはマージへの入力の記述』
244 ページの『ソートまたはマージからの出力の記述』
247 ページの『ソートまたはマージの要求』
252 ページの『ソートまたはマージの成否の判断』
252 ページの『ソートまたはマージ操作の途中停止』
253 ページの『FASTSRT を使用してのソートのパフォーマンスの向上』
256 ページの『ソート動作の制御』
DFSORT アプリケーション・プログラミング・ガイド

関連参照 261 ページの『CICS SORT アプリケーションの制約事項』
SORT ステートメント (*Enterprise COBOL 言語解説書*)
MERGE ステートメント (*Enterprise COBOL 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 の 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
248 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 (*Enterprise COBOL 言語解説書*)
MERGE ステートメント (*Enterprise COBOL 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。以下に、その例を示します。

   ```
   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。以下に、その例を示します。

   ```
   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。また、このファイルに DD 名定義を提供する必要もありません。

関連参照
15 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。以下に、その例を示します。

   ```
   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。以下に、その例を示します。

   ```
   DATA DIVISION.
   FILE SECTION.
   FD  Input-File
   ```

```
       LABEL RECORDS ARE STANDARD
       BLOCK CONTAINS 0 CHARACTERS
       RECORDING MODE IS F
       RECORD CONTAINS 100 CHARACTERS.
   01  Input-Record   PIC X(100).
```

関連タスク

243 ページの『入力プロシージャーのコーディング』
247 ページの『ソートまたはマージの要求』

関連参照

15 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
     Label Records Are Standard
     Block Contains 0 Characters
     Recording Mode is F
     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

247 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

246 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 (*Enterprise COBOL 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。以下に、その例を示します。

   ```
   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。以下に、その例を示します。

   ```
   DATA DIVISION.
   FILE SECTION.
   FD  Output-File
       LABEL RECORDS ARE STANDARD
       BLOCK CONTAINS 0 CHARACTERS
       RECORDING MODE IS F
       RECORD CONTAINS 100 CHARACTERS.
   01  Output-Record   PIC X(100).
   ```

関連タスク
『出力プロシージャーのコーディング』
247 ページの『ソートまたはマージの要求』

関連参照
15 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。

DFSORT のコーディング: DFSORT の使用時に、 COBOL プログラムが実行を終了する前に RETURN ステートメントが AT END 条件を検出しなかった場合、 SORT ステートメントは異常終了して DFSORT メッセージ IEC025A が出されることがあります。この状態を回避するには、必ず RETURN ステートメントを AT END 句と一緒にコーディングするようにしてください。加えて、AT END 条件が検出されるまで RETURN ステートメントが実行されるようにもしてください。AT END 条件が起こるのは、最後のレコードがソート作業ファイルからプログラムに戻され、後続の RETURN ステートメントが実行された後です。

『例: DFSORT を使用する際の出力プロシージャーのコーディング』

関連参照
246 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 (*Enterprise COBOL 言語解説書*)

例: DFSORT を使用する際の出力プロシージャーのコーディング

次の例は、プログラムが実行を終了する前に RETURN ステートメントが AT END 条件に出合うようにしたコーディング手法を示しています。AT END 句をコーディングした RETURN ステートメントは、AT END 条件が起こるまで実行されます。

```
IDENTIFICATION DIVISION.
DATA DIVISION.
FILE SECTION.
SD  OUR-FILE.
01  OUR-SORT-REC.
    03  SORT-KEY               PIC X(10).
    03  FILLER                 PIC X(70).
. . .
WORKING-STORAGE SECTION.
01  WS-SORT-REC                PIC X(80).
01  END-OF-SORT-FILE-INDICATOR  PIC X VALUE 'N'.
    88  NO-MORE-SORT-RECORDS          VALUE 'Y'.
. . .
PROCEDURE DIVISION.
A-CONTROL SECTION.
    SORT OUR-FILE ON ASCENDING KEY SORT-KEY
      INPUT PROCEDURE IS B-INPUT
      OUTPUT PROCEDURE IS C-OUTPUT.
    . . .
B-INPUT SECTION.
    MOVE . . .. . .. TO WS-SORT-REC.
    RELEASE OUR-SORT-REC FROM WS-SORT-REC.
    . . .
C-OUTPUT SECTION.
    DISPLAY 'STARTING READS OF SORTED RECORDS: '.
    RETURN OUR-FILE
      AT END
        SET NO-MORE-SORT-RECORDS TO TRUE.
    PERFORM WITH TEST BEFORE UNTIL NO-MORE-SORT-RECORDS
      IF SORT-RETURN = 0 THEN
        DISPLAY 'OUR-SORT-REC = ' OUR-SORT-REC
        RETURN OUR-FILE
```

```
        AT END
          SET NO-MORE-SORT-RECORDS TO TRUE
      END-IF
    END-PERFORM.
```

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出力プロシージャー、および MERGE によって呼び出されるそれぞれの出力プロシージャーには、以下の制約事項が適用されます。

- 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、標準リンケージ規約に従う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してはなりません。
- 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。
- 言語環境プログラムの条件処理では、入力または出力プロシージャーの中でユーザー作成条件処理ルーチンを設定することは許可されません。

関連タスク

243 ページの『入力プロシージャーのコーディング』
244 ページの『出力プロシージャーのコーディング』
言語環境プログラム・プログラミング・ガイド
(リンク・エディットおよび実行の準備)

ソートおよびマージ・データ・セットの定義

z/OS のもとで DFSORT を使用するには、実行時 JCL に DD ステートメントをコーディングして、必要なデータ・セットを記述しなければなりません。

ソートまたはマージ作業域
少なくとも 3 つのデータ・セットを定義します。SORTWK01、SORTWK02、SORTWK03、...、SORTWK*nn* (*nn* は 99 以下) などです。これらのデータ・セットは HFS にあってはなりません。

SYSOUT データ・セット名を変更しない限り、ソート診断メッセージ用にこのデータ・セットを定義します。(名前は、SORT-CONTROL データ・セット内の OPTION 制御ステートメントの MSGDDN キーワードを使用して、または SORT-MESSAGE 特殊レジスターを使用して変更してください。)

SORTCKPT
ソートまたはマージでチェックポイントを設定する場合に、このデータ・セットを定義します。

入出力 必要に応じて、入出力データ・セットを定義します。

SORTLIB (DFSORT ライブラリー)
ソート・モジュールが入ったライブラリー (例えば、SYS1.SORTLIB) を定義します。

関連タスク
256 ページの『ソート動作の制御』
260 ページの『DFSORT によるチェックポイント・リスタートの使用』

可変長レコードのソート

ソートへの入力ファイルに可変長レコードが入っていても、ソート作業ファイルは可変長として定義しないと可変長になりません。

コンパイラーがソート作業ファイルを可変長と見なすのは、そのファイルの SD 記入項目で以下のいずれかのエレメントをコーディングした場合です。

- RECORD IS VARYING 節
- サイズが異なるレコードを定義する 2 つ以上のレコード記述 (あるいは、OCCURS DEPENDING ON 節を含むレコード)

SD 記入項目では RECORDING MODE 節は認められないため、ソート作業ファイルについて RECORDING MODE V を使用することはできません。

パフォーマンスの考慮: 可変長ファイルに対するソートのパフォーマンスを向上させるには、SMS= 制御カードまたは SORT-MODE-SIZE 特殊レジスターで、入力ファイルに最も頻繁に出現するレコード長 (モーダル長) を指定します。

関連タスク
258 ページの『制御ステートメントによる DFSORT デフォルトの変更』
256 ページの『ソート動作の制御』

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、および SELECT 節で宣言された入力ファイル (1 つ以上) の名前を使用してください。

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。以下に例を示します。

```
SORT Sort-Work-1
        ON ASCENDING KEY Sort-Key-1
        USING Input-File
        GIVING Output-File.
```

SORT . . . USING または MERGE . . . USING では、コンパイラーは、ファイルを開き、レコードを読み取り、レコードをソートまたはマージ・プログラムに解放し、そして入力ファイルを閉じるための入力プロシージャーを生成します。 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 では、コンパイラーは、ファイルを開き、レコードを返し、レコードを書き込み、そしてファイルを閉じるための出力プロシージャーを生成します。 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 または MERGE ステートメントの USING または GIVING ファイルは、HFS に常駐する順次ファイルにすることができます。

242 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、それらの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したい場合は、SORT . . . INPUT PROCEDURE を使用してください。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したい場合は、 SORT . . . OUTPUT PROCEDURE を使用してください。以下に例を示します。

```
SORT Sort-Work-1
        ON ASCENDING KEY Sort-Key-1
        INPUT PROCEDURE EditInputRecords
        OUTPUT PROCEDURE FormatData.
```

249 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したい場合は、MERGE . . . OUTPUT PROCEDURE を使用してください。以下に例を示します。

```
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。

関連タスク 246 ページの『ソートおよびマージ・データ・セットの定義』

関連参照
SORT ステートメント (*Enterprise COBOL 言語解説書*)
MERGE ステートメント (*Enterprise COBOL 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。

キーの最大数は規定されていませんが、キーはレコード記述の最初の 4092 バイト内になければなりません。キーの全長が 4092 バイトを超えてはなりません。ただし、EQUALS キーワードが DFSORT OPTION 制御ステートメントにコーディングされている場合は別で、その場合はキーの全長が 4088 バイトを超えてはなりません。

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、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。

関連参照
DFSORT アプリケーション・プログラミング・ガイド (SORT 制御ステートメント)
SORT ステートメント (*Enterprise COBOL 言語解説書*)
MERGE ステートメント (*Enterprise COBOL 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

247 ページの『ソートまたはマージの要求』

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、EBCDIC または ASCII 照合シーケンス、または別の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、 EBCDIC 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。

ASCII ファイルをソートまたはマージするときには、ASCII 照合シーケンスを要求しなければなりません。そうするには、SORT または MERGE ステートメントの COLLATING SEQUENCE 句をコーディングし、 *alphabet-name* を SPECIAL-NAMES 段落の STANDARD-1 として定義してください。

関連タスク

9 ページの『照合シーケンスの指定』
248 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 (*Enterprise COBOL 言語解説書*)
SORT ステートメント (*Enterprise COBOL 言語解説書*)
データのクラスおよびカテゴリー (*Enterprise COBOL 言語解説書*)

ウィンドウ表示日付フィールドに基づいたソート

使用している DFSORT のバージョンが Y2PAST オプションをサポートする場合は、ウィンドウ表示日付フィールドをソート・キーとして指定することができます。このオプションがサポートされていれば、DFSORT は、ウィンドウ表示日付シーケンスに基づいてソートまたはマージすることができます。

ウィンドウ表示日付フィールドに基づいてソートするためには、DATE FORMAT 節を使用してウィンドウ表示日付フィールドを定義し、そのフィールドをソート・キーとして使用してください。DFSORT は、コンパイル単位で使用された同じ世紀ウィンドウを使用します。YEARWINDOW コンパイラー・オプションを使用して、世紀ウィンドウを指定してください。

DFSORT は年末尾型ウィンドウ表示日付フィールドをサポートしますが、コンパイラー自体は、MERGE または SORT 以外のステートメントの年末尾型ウィンドウ表示日付フィールドについては、自動ウィンドウ操作を行いません。

関連概念
694 ページの『2000 年言語拡張 (MLE)』

関連タスク
710 ページの『日付別のソートおよびマージ』

関連参照
404 ページの『YEARWINDOW』
DATE FORMAT 節 (*Enterprise COBOL 言語解説書*)
DFSORT アプリケーション・プログラミング・ガイド (OPTION 制御ステートメント: Y2PAST)

同じキーを持つレコードのオリジナル・シーケンスの保持

同じ照合レコードの順序を入力から出力まで保持することができます。

以下のいずれかの手法を使用します。

- EQUALS オプションをデフォルトとして指定して DFSORT をインストールします。
- 実行時に、IGZSRTCD データ・セットに EQUALS キーワードがある OPTION カードを提供します。
- SORT ステートメントで WITH DUPLICATES IN ORDER 句を使用します。これにより、IGZSRTCD データ・セットの OPTION カードに EQUALS キーワードが追加されます。

  OPTION カードで NOEQUALS キーワード と DUPLICATES 句の両方は使用しないでください。そうしなければ、実行単位が終了します。

関連参照
DFSORT アプリケーション・プログラミング・ガイド (OPTION 制御ステートメント)

ソートまたはマージの成否の判断

DFSORT プログラムは、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。以下に例を示します。

```
    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ます。

デフォルトでは、DFSORT 診断メッセージは SYSOUT データ・セットに送られます。このデフォルトを変更したい場合は、DFSORT の OPTION 制御カードの MSGDDN パラメーターを使用するか、SORT-MESSAGE 特殊レジスターを使用してください。

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。

関連タスク
256 ページの『NOFASTSRT によるソート・エラーの検査』
256 ページの『ソート動作の制御』

関連参照 *DFSORT アプリケーション・プログラミング・ガイド* (DFSORT メッセージおよび戻りコード)

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、次の暗黙の RELEASE または RETURN (USING または GIVING ファイルに対して、あるレコードの読み取りまたは書き込みの後で起こる) が実行された直後に停止されます。

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。

FASTSRT を使用してのソートのパフォーマンスの向上

FASTSRT コンパイラー・オプションを使用すると、ほとんどのソート操作のパフォーマンスが向上します。 FASTSRT を使用した場合は、DFSORT プロダクトが (Enterprise COBOL の代わりに)、SORT . . . USING および SORT . . . GIVING ステートメントで指定した入出力ファイルに入出力操作を実行します。

コンパイラーは通知メッセージを出し、FASTSRT がパフォーマンスを向上させることができるステートメントを指摘します。

使用上の注意

- FASTSRT を使用する場合は、DFSORT オプション SORTIN または SORTOUT は使用できません。FASTSRT コンパイラー・オプションは、USING または GIVING ファイルとして使用している行順次ファイルには適用されません。
- FASTSRT を使用する場合は、ファイル状況を指定しても、ソート操作中はファイル状況は無視されます。

関連参照
363 ページの『FASTSRT』
『JCL に関する FASTSRT の要件』
『ソート入出力ファイルに関する FASTSRT の要件』

JCL に関する FASTSRT の要件

実行時 JCL では、ソート作業ファイル (SORTWK*nn*) を、テープ・データ・セットではなく、直接アクセス装置に割り当てをしてください。

入出力ファイルの場合、DD ステートメントの DCB パラメーターは FD 記述と一致していなければなりません。

ソート入出力ファイルに関する FASTSRT の要件

FASTSRT を指定しても、FASTSRT の要件が満たされていないと、コンパイラーはメッセージを出し、代わりに COBOL ランタイムが入出力を実行します。その場合、プログラムはパフォーマンスの向上を望めなくなります。

FASTSRT を使用するためには、ソートへの入力ファイルおよびソートからの出力ファイルを、次のように記述し、処理してください。

- USING 句では 1 つの入力ファイルのみ指定することができます。 GIVING 句では 1 つの出力ファイルのみ指定することができます。
- 入力ファイルに対して入力プロシージャーを使用したり、出力ファイルに対して出力プロシージャーを使用したりすることはできません。

  入力または出力プロシージャーを使用する代わりに、次の DFSORT 制御ステートメントを使用できます。

  - INREC
  - OUTFILE
  - OUTREC
  - INCLUDE
  - OMIT
  - STOPAFT
  - SKIPREC
  - SUM

  多くの DFSORT の機能では、入力または出力プロシージャーでよく使用されるのと同様の操作を実行します。代わりに適切な DFSORT 制御ステートメントをコーディングし、それらを IGZSRTCD または SORTCNTL データ・セットに入れてください。
- 出力 FD 記入項目に LINAGE 節をコーディングしないでください。
- ソートで使用する FD に適用するために、INPUT 宣言 (入力ファイル用)、OUTPUT 宣言 (出力ファイル用)、またはファイル特定宣言 (入力ファイルまたは出力ファイル用) をコーディングしてはなりません。
- 可変長の相対ファイルを入力ファイルまたは出力ファイルとして使用してはなりません。
- 行順次ファイルを入力ファイルまたは出力ファイルとして使用してはなりません。
- 入力ファイルまたは出力ファイルのいずれかについて、SD と FD 記入項目のレコード記述では、両方とも同じ形式 (固定長または可変長) を定義しなければなりません。また、SD と FD の最大レコード・サイズでは、同じレコード長を定義しなければなりません。

出力ファイルに RELATIVE KEY 節をコーディングしても、これはソートでは設定されません。

パフォーマンスのヒント: 入出力レコードをブロック化すると、ソート処理のパフォーマンスが大幅に向上する可能性があります。

QSAM の要件

- QSAM ファイルのレコード形式は、固定長、可変長、またはスパンでなければなりません。
- QSAM 入力ファイルは空であっても構いません。

- 入力と出力の両方に同じ QSAM ファイルを使用する場合は、2 つの異なる DD ステートメントを使用してこのファイルを記述しなければなりません。例えば、FILE-CONTROL SECTION では、次のようにコーディングすることができます。

  ```
  SELECT FILE-IN ASSIGN INPUTF.
  SELECT FILE-OUT ASSIGN OUTPUTF.
  ```

  DATA DIVISION で、FILE-IN と FILE-OUT の両方についての FD 記入項目を持つことになります。この場合、FILE-IN と FILE-OUT は、それらの名前を除いて同じです。

  PROCEDURE DIVISION では、SORT ステートメントを次のように記述します。

  ```
  SORT file-name
    ASCENDING KEY data-name-1
    USING FILE-IN GIVING FILE-OUT
  ```

  そうすると JCL では、データ・セット INOUT がカタログされていると想定して、次のようにコーディングします。

  ```
  //INPUTF  DD DSN=INOUT,DISP=SHR
  //OUTPUTF DD DSN=INOUT,DISP=SHR
  ```

  これに反して、USING 句と GIVING 句に同じファイル名をコーディングするか、または入力ファイルと出力ファイルに同じ DD 名を割り当てた場合は、そのファイルは入力と出力のいずれかについて FASTSRT に受け入れられますが、両方については受け入れられません。ファイルが入力について FASTSRT に不適格となるような条件が特になければ、ファイルは入力について FASTSRT に受け入れられますが、出力については受け入れられません。ファイルが入力について FASTSRT に不適格である場合でも、出力について FASTSRT に適格である可能性があります。

FASTSRT に対して適格である QSAM ファイルは、SORT ステートメントの実行中に COBOL プログラムからアクセスすることができます。例えば、ファイルが入力で FASTSRT に使用されている場合には、それを出力プロシージャーでアクセスすることができ、ファイルが出力で FASTSRT に使用されている場合には、それを入力プロシージャーでアクセスすることができます。

VSAM の要件

- VSAM 入力ファイルは空であってはなりません。
- VSAM ファイルをパスワード保護することはできません。
- USING 句と GIVING 句の両方で同じ VSAM ファイルを指定することはできません。
- FASTSRT に対して適格である VSAM ファイルは、SORT ステートメントの処理が完了するまで、COBOL プログラムからアクセスすることはできません。例えば、ファイルが入力で FASTSRT に適格である場合には、それを出力プロシージャーでアクセスしてはなりません。逆も同様です。(そのような処理を行うと、OPEN は失敗します。)

関連タスク
DFSORT アプリケーション・プログラミング・ガイド

NOFASTSRT によるソート・エラーの検査

NOFASTSRT オプションを使用してコンパイルすると、ソート・プロセスは、SORT ステートメントの USING または GIVING 句で参照されるファイルに関して、オープン、クローズ、または入出力操作時にエラーの有無を検査しません。このため、SORT が正常に完了したかどうかを調べることが必要になる場合があります。

必要なコードは、次の表に示されているように、USING および GIVING 句で参照されるファイルに FILE STATUS 節または ERROR 宣言がコーディングされているかどうかによって異なります。

表 34. **NOFASTSRT によるソート・エラーの検査のメソッド**

FILE STATUS 節の有無	ERROR 宣言の有無	必要なコード
いいえ	いいえ	特別なコーディングは不要です。ソート・プロセス時に障害が起こると、プログラムは異常終了します。
はい	いいえ	SORT ステートメントの後で SORT-RETURN 特殊レジスターをテストし、ファイル状況キーをテストします。(ファイル状況コードは設定されますが、COBOL はこれを検査できないため、ファイル状況検査を完了したい場合にはお勧めしません。)
どちらでも可	はい	ERROR 宣言で、SORT-RETURN 特殊レジスターを 16 に設定して、ソート・プロセスを停止し、それが失敗したことを示します。SORT ステートメントの後で、SORT-RETURN 特殊レジスターをテストします。

関連タスク

252 ページの『ソートまたはマージの成否の判断』
269 ページの『ファイル状況キーの使用』
268 ページの『ERROR 宣言のコーディング』
252 ページの『ソートまたはマージ操作の途中停止』

ソート動作の制御

ソートの前に特殊レジスターに値を挿入するか、またはコンパイラー・オプションを使用することにより、ソート動作の幾つかの局面を制御できます。また、制御ステートメントやキーワードの選択を行える場合もあります。

ソートの後で特殊レジスターの内容を調べることによって、ソート動作を検査することができます。

以下の表は、特殊レジスターまたはコンパイラー・オプション、および同等のソート制御ステートメント・キーワード (使用可能な場合) を使用して影響を及ぼすことができるソート動作の局面をリストしています。

表 35. **ソート動作を制御する方法**

設定またはテスト対象	特殊レジスターまたはコンパイラー・オプションを使用する場合	制御ステートメント (該当する場合は、およびキーワード) を使用する場合
予約される主記憶域の量	SORT-CORE-SIZE 特殊レジスター	OPTION (キーワード RESINV)
使用される主記憶域の量	SORT-CORE-SIZE 特殊レジスター	OPTION (キーワード MAINSIZE または MAINSIZE=MAX)
可変長レコードを持つファイルのレコードのモーダル長	SORT-MODE-SIZE 特殊レジスター	SMS=*nnnnn*
ソート制御ステートメント・データ・セットの名前 (デフォルト IGZSRTCD)	SORT-CONTROL 特殊レジスター	なし
ソート・メッセージ・ファイルの名前 (デフォルト SYSOUT)	SORT-MESSAGE 特殊レジスター	OPTION (キーワード MSGDDN)
ソート・レコードの数	SORT-FILE-SIZE 特殊レジスター	OPTION (キーワード FILSZ)
ソート完了コード	SORT-RETURN 特殊レジスター	なし
日付フィールドに基づいてソートまたはマージするための世紀ウィンドウ	YEARWINDOW コンパイラー・オプション	OPTION (キーワード Y2PAST)
ソート・キーまたはマージ・キーとして使用されるウィンドウ表示日付フィールドの形式	(PICTURE、USAGE、および DATE FORMAT 節から得られる)	SORT (キーワード FORMAT=Y2*x*)

ソート特殊レジスター: SORT-CONTROL は、ソート制御ステートメント・ファイルの DD 名が入れられる、8 文字の COBOL 特殊レジスターです。 IGZSRTCD というデフォルトの DD 名を使用したくない場合は、ソート制御ステートメントが含まれているデータ・セットの DD 名を SORT-CONTROL に割り当ててください。

SORT-CORE-SIZE、SORT-FILE-SIZE、SORT-MESSAGE、および SORT-MODE-SIZE 特殊レジスターは、それらにデフォルト以外の値を割り当てた場合には、SORT インターフェースで使用されます。しかし、実行時には、ソート制御ステートメント・データ・セット内の制御ステートメントのパラメーターは、特殊レジスター内の対応する設定をオーバーライドし、その影響に対するメッセージが出されます。

SORT-RETURN 特殊レジスターを使用すると、ソートまたはマージが正常に実行されたかどうかを判断したり、ソートまたはマージ操作を途中で停止することができます。

プログラム内に設定したソート特殊レジスターごとに、コンパイラー警告メッセージ (W レベル) が出されます。

関連タスク

252 ページの『ソートまたはマージの成否の判断』
252 ページの『ソートまたはマージ操作の途中停止』
258 ページの『制御ステートメントによる DFSORT デフォルトの変更』

259 ページの『ソート・ファイル用のスペースの割り振り』
DFSORT アプリケーション・プログラミング・ガイド
(DFSORT プログラム制御ステートメントの使用)

関連参照
『IGZSRTCD データ・セットのデフォルトの特性』

制御ステートメントによる DFSORT デフォルトの変更

ソート・パフォーマンスを向上させるために DFSORT のシステム・デフォルトを変更したい場合には、実行時データ・セット IGZSRTCD に含めた制御ステートメントを介して DFSORT に情報を渡してください。

(リストされた順序で) IGZSRTCD に含めることができる制御ステートメントは、次のとおりです。

1. SMS=*nnnnn*。*nnnnn* は、最も出現頻度の高いレコード・サイズの長さ (バイト単位) です。(SD ファイルが可変長である場合にのみ使用します。)
2. OPTION (キーワード SORTIN または SORTOUT を除く)。
3. その他の DFSORT 制御ステートメント (SORT、MERGE、RECORD、END を除く)。

制御ステートメントは、2 桁目から 71 桁目の間にコーディングしてください。行をコンマで終わらせ、次の行を新しいキーワードで開始すれば、制御ステートメント・レコードを継続することができます。レコードではラベルもコメントも使用することはできず、レコード自体を DFSORT コメント・ステートメントにすることはできません。

関連タスク
256 ページの『ソート動作の制御』
DFSORT アプリケーション・プログラミング・ガイド
(DFSORT プログラム制御ステートメントの使用)

関連参照
『IGZSRTCD データ・セットのデフォルトの特性』

IGZSRTCD データ・セットのデフォルトの特性

IGZSRTCD データ・セットは任意指定です。デフォルトは、LRECL=80、BLKSIZE=400、および DD 名 IGZSRTCD です。

SORT-CONTROL 特殊レジスターにコーディングすることによって、別の DD 名を使用することができます。SORT-CONTROL データ・セットの DD 名を定義したときに、メッセージ IGZ0027W を受け取った場合は、OPEN 障害が起こっており、問題を調べる必要があります。

関連タスク
256 ページの『ソート動作の制御』

ソートまたはマージ操作のためのストレージの割り振り

DFSORT のインストール時に設定される特定のパラメーターによって、DFSORT が使用するストレージの量が決まります。一般的に、DFSORT が使用できるストレージが多いほど、ソートまたはマージ操作の実行速度が速くなります。

しかし、DFSORT インストールで、領域内のすべてのフリー・スペースを COBOL 操作用に割り振ってはなりません。プログラムの実行時には、以下のために使用できるストレージが必要です。

- 入力または出力プロシージャーから動的に呼び出される COBOL プログラム
- 言語環境プログラムのランタイム・ライブラリー・モジュール
- 入力または出力プロシージャーで使用するために領域にロードされる可能性があるデータ管理モジュール
- これらのモジュールによって獲得されるストレージ

ソートまたはマージのある特定の実行のために、インストール時に設定された DFSORT 保管値をオーバーライドすることができます。これを行うには、ソート制御ステートメント・データ・セット内の OPTION 制御ステートメントに MAINSIZE および RESINV キーワードをコーディングするか、または SORT-CORE-SIZE 特殊レジスターを使用してください。

領域内のすべてのフリー・スペースが完全に COBOL プログラムのソート操作に使用されるかぎり、ストレージ割り振りをオーバーライドすることがないように注意してください。

関連タスク
256 ページの『ソート動作の制御』
DFSORT 導入およびカスタマイズ

関連参照
DFSORT アプリケーション・プログラミング・ガイド
(OPTION 制御ステートメント)

ソート・ファイル用のスペースの割り振り

NOFASTSRT または入力プロシージャーを使用する場合は、ソートするファイルのサイズを DFSORT は認識しません。このため、大きいファイルをソートするときにスペース不足状態に陥ったり、小さいファイルをソートするときにリソースの過剰割り振りが発生する可能性があります。

このような状況が発生した場合は、SORT-FILE-SIZE 特殊レジスターを使用して、ソートに必要なリソースの量 (例えば、ワークスペースまたは ハイパースペース) を DFSORT が判別できるようにします。SORT-FILE-SIZE を、妥当な入力レコード見積数に設定します。この値は、FILSZ=E*n* の値として DFSORT に渡されます。

関連タスク
256 ページの『ソート動作の制御』
243 ページの『入力プロシージャーのコーディング』
DFSORT アプリケーション・プログラミング・ガイド

DFSORT によるチェックポイント・リスタートの使用

z/OS のもとで DFSORT の実行中に取られたチェックポイントは、DFSORT によって取られたものでない限り、再始動には使用できません。 SORT または MERGE ステートメントの実行中に COBOL プログラムによって取られたチェックポイントは無効です。すなわち、そのような再始動は検出されますが、取り消されます。

ソートまたはマージ操作中にチェックポイントを取るには、以下のステップを行ってください。

1. SORTCKPT についての DD ステートメントを JCL に追加します。
2. I-O-CONTROL 段落で RERUN 節をコーディングします。

   RERUN ON *assignment-name*
3. ソート制御ステートメント・データ・セット (デフォルトの DD 名 IGZSRTCD) 内の OPTION 制御ステートメントで CKPT (または CHKPT) キーワードをコーディングします。

関連概念
683 ページの『第 32 章 割り込みおよびチェックポイント・リスタート』

関連タスク
258 ページの『制御ステートメントによる DFSORT デフォルトの変更』
683 ページの『チェックポイントの設定』

CICS のもとでのソート

CICS のもとでサポートされる IBM ソート・プロダクトはありません。しかし、SORT ステートメント (および CICS のもとで稼働する独自のソート・プログラム) を使用すれば、少量のデータをソートすることができます。

SORT ステートメントには、入力プロシージャーと出力プロシージャーの両方を指定しなければなりません。入力プロシージャーでは、RELEASE ステートメントを使用して、ソートが実行される前にレコードを COBOL プログラムからソート・プログラムに転送します。出力プロシージャーでは、RETURN ステートメントを使用して、ソートが実行された後でレコードをソート・プログラムから COBOL プログラムに転送します。

関連タスク
243 ページの『入力プロシージャーのコーディング』
244 ページの『出力プロシージャーのコーディング』
454 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
261 ページの『CICS SORT アプリケーションの制約事項』
463 ページの『CICS 予約語テーブル』

CICS SORT アプリケーションの制約事項

CICS のもとで実行され、SORT ステートメントを使用する COBOL アプリケーションには、いくつかの制約事項が適用されます。

それらの制約事項は次のとおりです。

- USING または GIVING 句を含む SORT ステートメントはサポートされません。
- ソート制御データ・セットはサポートされません。SORT-CONTROL 特殊レジスターのデータは無視されます。
- 入力または出力プロシージャーで以下の CICS コマンドを使用すると、予測できない結果をもたらす可能性があります。
  - CICS LINK
  - CICS XCTL
  - CICS RETURN
  - CICS HANDLE
  - CICS IGNORE
  - CICS PUSH
  - CICS POP

  上記以外の CICS コマンドは、NOHANDLE または RESP オプションを使用する場合には、使用できます。NOHANDLE または RESP を使用しないと、予測できない結果が生じる恐れがあります。

関連参照

463 ページの『CICS 予約語テーブル』

第 13 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。

Enterprise COBOL 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。

- ユーザー要求ダンプ
- 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
- エラー処理用のユーザー作成ルーチン

関連タスク

264 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』
265 ページの『算術演算でのエラーの処理』
265 ページの『入出力操作でのエラーの処理』
275 ページの『プログラム呼び出し時のエラーの処理』
275 ページの『エラー処理用のルーチンの作成』

ダンプの要求

言語環境プログラム呼び出し可能サービス CEE3DMP への呼び出しをコーディングすることによって、プログラム内の事前に指定した任意のポイントで、言語環境プログラム ランタイム環境およびメンバー言語ライブラリーの定様式ダンプを取ることができます。

```
77  Title-1          Pic x(80)   Display.
77  Options          Pic x(255)  Display.
01  Feedback-code    Pic x(12)   Display.
. . .
    Call "CEE3DMP" Using Title-1, Options, Feedback-code
```

定様式ダンプに記号変数を組み込むためには、TEST コンパイラー・オプションを使用してコンパイルし、 CEE3DMP の VARIABLES サブパラメーターを使用します。さらに、ランタイム・オプションを介して、選択したエラー状態に合ったダンプを作成するように要求することができます。

プログラム内の事前に指定した任意のポイントでシステム・ダンプを取ることができます。ゼロのクリーンアップ値を指定した 言語環境プログラム・サービス CEE3ABD を呼び出して、クリーンアップなしの異常終了を要求します。この呼び出し可能サービスは実行単位を即刻終了させ、異常終了が出されると、システム・ダンプが要求されます。

関連参照
392 ページの『TEST』
言語環境プログラム・デバッグのガイド
言語環境プログラム・プログラミング・リファレンス (CEE3DMP-- ダンプの生成)

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。			

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

浮動小数点指数オーバーフローは、浮動小数点算術計算の値を zSeries の浮動小数点オペランド形式で表すことができない場合に起こります。このタイプのオーバーフローは SIZE ERROR を起こしません。代わりに、異常終了が起こります。異常終了を代行受信し、独自のエラー・リカバリー論理を提供するために、ユーザー作成条件処理ルーチンをコーディングすることができます。

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。

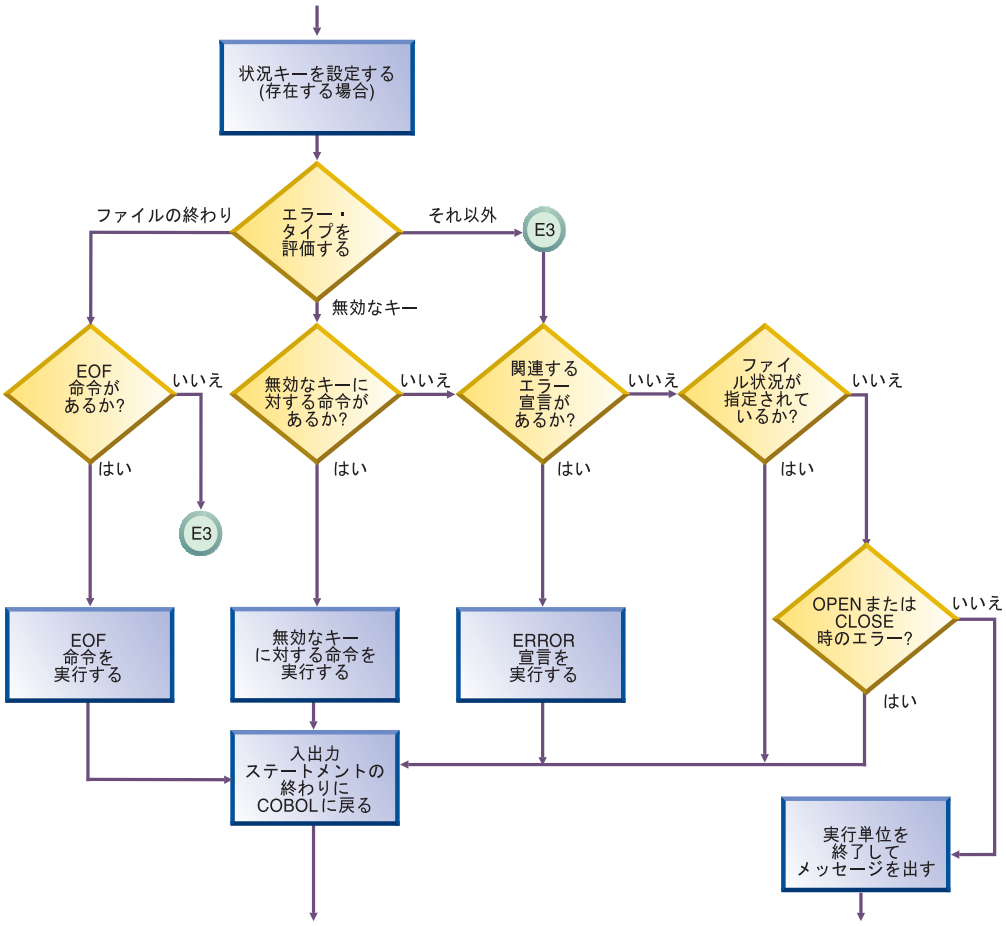
特定の入力または出力の状態またはエラーの代行受信および処理には、以下のいずれかの技法を使用することができます。

- 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 ステートメント上の命令ステートメント句

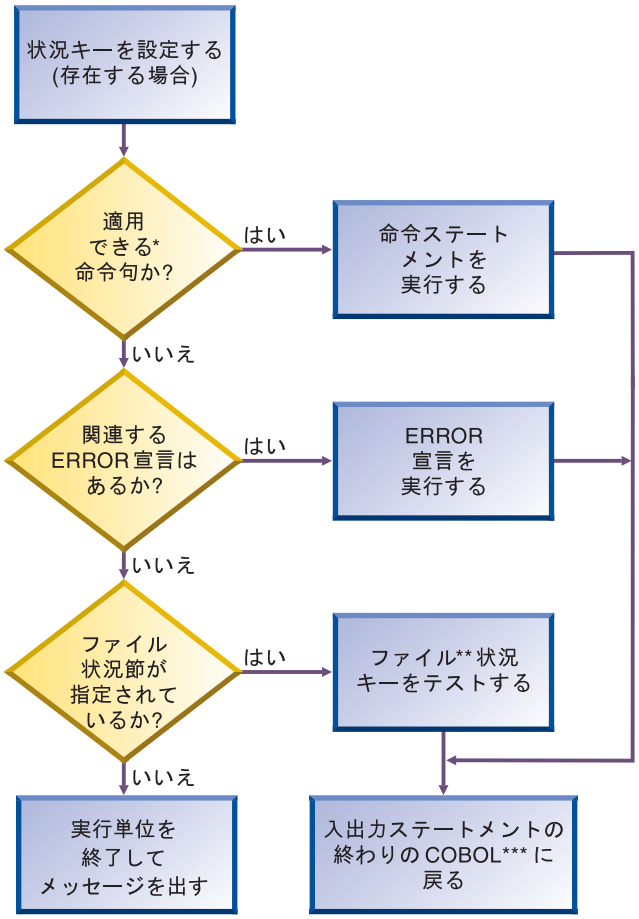
  VSAM ファイルについては、FILE STATUS 節を指定した場合は、VSAM 状況コードをテストして、エラー処理論理に対してプログラムを指示することもできます。
- INVALID KEY 句

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、重大度 3 の言語環境プログラム条件がシグナル通知され、この条件が処理されない場合には、実行単位が終了します。

次の図は、VSAM 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。



次の図は、QSAM または行順次ファイルの入力または出力エラー後のロジック・フローを示しています。このエラーは、REEL/UNIT 節が指定された READ ステートメント、WRITE ステートメント、または CLOSE ステートメントから生じた可能性があります (QSAM のみ)。



*QSAM の場合の可能な句は、AT END、AT END-OF-PAGE、および INVALID KEY です。行順次の場合は、AT END です。

**ファイル状況キーをテストするコードを書く必要があります。

*** COBOL プログラムの実行は、エラーを引き起こした入出力ステートメントの後で継続されます。

関連タスク

268 ページの『ファイルの終わり条件 (AT END) の使用』
268 ページの『ERROR 宣言のコーディング』
269 ページの『ファイル状況キーの使用』
184 ページの『QSAM ファイルのエラーの処理』
271 ページの『VSAM 状況コードの使用 (VSAM ファイルのみ)』
237 ページの『行順次ファイルのエラーの処理』
273 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 (*Enterprise COBOL 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、マスター・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、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する必要があります。

関連参照
AT END 句 (*Enterprise COBOL 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、入出力エラーの発生後、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。

マルチスレッド化: マルチスレッドのアプリケーションで入出力宣言をコーディングするときは、デッドロックを避けます。入出力操作の結果として制御が入出力宣言に移動した場合、その宣言内のステートメントが実行されている間は、そのファイルと関連付けられた自動逐次化ロックが保持されます。宣言内に入出力操作をコーディングすると、ロジックは以下のサンプルに示すデッドロックに陥ることがあります。

```
Declaratives.
D1 section.
Use after standard error procedure on F1
    Read F2.
    . . .
D2 section.
Use after standard error procedure on F2
    Read F1.
    . . .
End declaratives.
    . . .
    Rewrite R1.
    Rewrite R2.
```

このプログラムを 2 つのスレッドで実行している場合、次の一連のイベントが発生するおそれがあります。

1. スレッド 1: 再書き込み R1 が F1 に対するロックを獲得し、入出力エラーが発生する。
2. スレッド 1: F1 に対するロックを保持したまま、宣言 D1 に入る。
3. スレッド 2: 再書き込み R2 が F2 に対するロックを獲得し、入出力エラーが発生する。
4. スレッド 2: 宣言 D2 に入る。
5. スレッド 1: 宣言 D1 から F2 を読み取り、スレッド 2 によって保持されている F2 のロックに対し待機する。
6. スレッド 2: 宣言 D2 から F1 を読み取り、スレッド 1 によって保持されている F1 ロックに対し待機する。
7. デッドロック。

関連参照

EXCEPTION/ERROR 宣言 (*Enterprise COBOL 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):

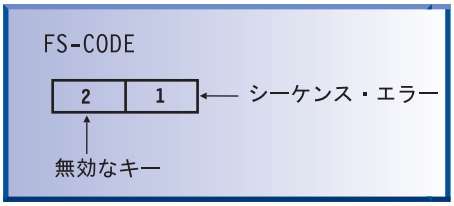
  ```
  WORKING-STORAGE SECTION.
  01  data-name-1  PIC 9(2)  USAGE NATIONAL.
  ```

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。



各ファイルごとに、次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。

VSAM ファイルの場合、さらに 2 番目の ID を FILE STATUS 節にコーディングして、VSAM 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。

ファイル状況キーは単独でも、INVALID KEY オプション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。

271 ページの『例: ファイル状況キー』

関連タスク
『VSAM 状況コードの使用 (VSAM ファイルのみ)』

関連参照
FILE STATUS 節 (*Enterprise COBOL 言語解説書*)
ファイル状況キー (*Enterprise COBOL 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 MASTERFILE ASSIGN TO AS-MASTERA
    FILE STATUS IS MASTER-CHECK-KEY
    . . .
DATA DIVISION.
. . .
WORKING-STORAGE SECTION.
01  MASTER-CHECK-KEY       PIC X(2).
. . .
PROCEDURE DIVISION.
    OPEN INPUT MASTERFILE
    IF MASTER-CHECK-KEY NOT = "00"
        DISPLAY "Nonzero file status returned from OPEN " MASTER-CHECK-KEY
    . . .
```

VSAM 状況コードの使用 (VSAM ファイルのみ)

要求の処理を正確に特定する上で、COBOL ファイル状況コードは一般的過ぎることがよくあります。 FILE STATUS 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、VSAM 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。

```
FILE STATUS IS data-name-1  data-name-8
```

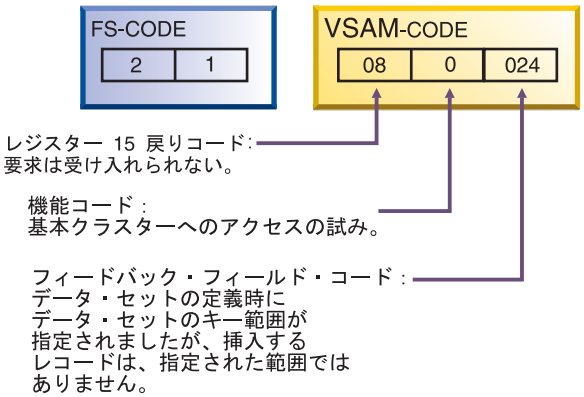
上記のデータ項目 *data-name-1* は COBOL ファイル状況キーを指定します。これは、2 文字の英数字または国別データ項目として、あるいは 2 桁のゾーン 10 進数または国別 10 進数項目として定義されます。

データ項目 *data-name-8* は VSAM 状況コードを指定します。これは、3 個の従属 2 バイト 2 進数フィールドを持つ、6 バイトの英数字グループ・データ項目として定義されます。 VSAM 状況コードは、COBOL ファイル状況キーが 0 でないときに、意味のある値を含みます。

data-name-8 は、以下の VSAM-CODE の場合のように WORKING-STORAGE SECTION 内で定義できます。

```
01 RETURN-STATUS.
    05 FS-CODE              PIC X(2).
    05 VSAM-CODE.
       10 VSAM-R15-RETURN   PIC S9(4) Usage Comp-5.
       10 VSAM-FUNCTION     PIC S9(4) Usage Comp-5.
       10 VSAM-FEEDBACK     PIC S9(4) Usage Comp-5.
```

Enterprise COBOL は、*data-name-8* を使用して、VSAM から提供される情報を渡します。次の例では、FS-CODE が *data-name-1* に相当し、VSAM-CODE が *data-name-8* に相当します。



『例: VSAM 状況コードの検査』

関連参照
FILE STATUS 節 (*Enterprise COBOL 言語解説書*)
ファイル状況キー (*Enterprise COBOL 言語解説書*)
データ・セットに対する *z/OS DFSMS* マクロ命令
(VSAM マクロの戻りコードおよび理由コード)

例: VSAM 状況コードの検査

次の例は、索引付きファイルを読み取り (5 番目のレコードで開始する)、入力または出力要求があるたびにその後でファイル状況キーを検査し、ファイル状況キーがゼロ以外であれば VSAM 状況コードを表示するものです。

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 VSAMFILE ASSIGN TO VSAMFILE
    ORGANIZATION IS INDEXED
    ACCESS DYNAMIC
    RECORD KEY IS VSAMFILE-KEY
    FILE STATUS IS FS-CODE VSAM-CODE.
DATA DIVISION.
FILE SECTION.
FD  VSAMFILE
    RECORD  30.
01  VSAMFILE-REC.
    10 VSAMFILE-KEY             PIC X(6).
    10 FILLER                   PIC X(24).
WORKING-STORAGE SECTION.
01  RETURN-STATUS.
    05 FS-CODE                  PIC XX.
    05 VSAM-CODE.
       10 VSAM-RETURN-CODE      PIC S9(2) Usage Binary.
       10 VSAM-COMPONENT-CODE   PIC S9(1) Usage Binary.
       10 VSAM-REASON-CODE      PIC S9(3) Usage Binary.
```

```
PROCEDURE DIVISION.
    OPEN  INPUT VSAMFILE.
    DISPLAY "OPEN INPUT VSAMFILE FS-CODE: " FS-CODE.

    IF FS-CODE NOT = "00"
       PERFORM VSAM-CODE-DISPLAY
       STOP RUN
    END-IF.

    MOVE "000005" TO VSAMFILE-KEY.
    START VSAMFILE KEY IS EQUAL TO VSAMFILE-KEY.
    DISPLAY "START VSAMFILE KEY="  VSAMFILE-KEY
            " FS-CODE: "  FS-CODE.
    IF FS-CODE NOT = "00"
       PERFORM VSAM-CODE-DISPLAY
    END-IF.

    IF FS-CODE = "00"
       PERFORM READ-NEXT UNTIL FS-CODE NOT = "00"
    END-IF.

    CLOSE VSAMFILE.
    STOP RUN.

READ-NEXT.
    READ VSAMFILE NEXT.
    DISPLAY "READ NEXT VSAMFILE FS-CODE: " FS-CODE.
    IF FS-CODE NOT = "00"
       PERFORM VSAM-CODE-DISPLAY
    END-IF.
    DISPLAY VSAMFILE-REC.

VSAM-CODE-DISPLAY.
    DISPLAY "VSAM-CODE ==>"
            " RETURN: "  VSAM-RETURN-CODE,
            " COMPONENT: "  VSAM-COMPONENT-CODE,
            " REASON: "  VSAM-REASON-CODE.
```

以下に、VSAM 状況コード情報を検査するプログラム例からの出力例を示しています。

```
OPEN INPUT VSAMFILE FS-CODE: 00
START VSAMFILE KEY=000005 FS-CODE: 00
READ NEXT VSAMFILE FS-CODE: 00
000005 THIS IS RECORD NUMBER 5
READ NEXT VSAMFILE FS-CODE: 00
000006 THIS IS RECORD NUMBER 6
READ NEXT VSAMFILE FS-CODE: 10
VSAM-CODE ==> RETURN: 08 COMPONENT: 2 REASON: 004
```

INVALID KEY 句のコーディング

INVALID KEY 句を、VSAM 索引付きおよび相対ファイルに対する、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。

INVALID KEY 句は、QSAM ファイルに対する WRITE 要求に組み込むこともできますが、QSAM ファイルの場合、句の意味が限定されます。この句が使用されるのは、いっぱいになっているディスクに書き込もうとした場合に限られます。

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。
- 入出力動詞に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。

『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。

マスター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、マスター・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、マスター・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。

顧客レコードのマスター・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 MASTER-CUSTOMER-NUMBER、および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 MASTER-CUSTOMER-NUMBER
READ MASTER-CUSTOMER-FILE INTO WS-CUSTOMER-RECORD
  INVALID KEY
    DISPLAY "MASTER 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、ロード・モジュール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、呼び出されるプログラムの可用性についてのみ適用されます。呼び出されるプログラムの実行中にエラーが発生した場合、ON EXCEPTION 句は実行されません。

関連タスク

Enterprise COBOL コンパイラーおよびランタイム 移行ガイド

エラー処理用のルーチンの作成

ON EXCEPTION 句、ON SIZE ERROR 句、またはその他の言語構成要素を使用すると、プログラムの実行中に発生する可能性のあるエラー条件の大部分を処理することができます。しかし、マシン・チェックなどの異常条件が発生すると、通常、アプリケーションは異常終了します。

しかし、Enterprise COBOL および 言語環境プログラムでは、そのような条件が発生したときにユーザー作成プログラムが制御を獲得できる方法を提供しています。言語環境プログラム条件処理を使用して、独自のエラー処理ルーチンを COBOL で書くことができます。それらは、プログラムの報告、分析、あるいは修正でさえも行うことができ、プログラムが実行を再開できるようにします。

言語環境プログラムがユーザー作成のエラー・プログラムに制御を渡すようにさせるには、まずその入り口点を 言語環境プログラムに対して示し、登録する必要があります。 PROCEDURE-POINTER データ項目を使用して、プロシージャー入り口点の入り口アドレスを言語環境プログラム・サービスに渡すことができます。

関連タスク

516 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 2 部 プログラムのコンパイルおよびデバッグ

第 14 章 z/OS のもとでのコンパイル

Enterprise COBOL プログラムは、z/OS のもとでは、ジョブ制御言語 (JCL)、TSO コマンド、CLIST、または ISPF パネルを使用してコンパイルすることができます。

JCL を使ったコンパイルの場合、IBM は一連のカタログ式プロシージャーを用意しています。これを使用すれば、書かなければならない JCL コーディングの量を減らすことができます。カタログ式プロシージャーが要件に合わない場合は、独自の JCL を書くことができます。JCL を使用して、単一のプログラムをコンパイルすること、または複数のプログラムをバッチ・ジョブの中でコンパイルすることができます。

TSO のもとでコンパイルするときは、TSO コマンド、CLIST、または ISPF パネルを使用することができます。

また、`cob2` コマンドを使用すると、z/OS UNIX シェルでもコンパイルできます。

Enterprise COBOL コンパイラーを呼び出すツールまたはインターフェースを開発している場合には、Enterprise COBOL コンパイラーをアセンブラー・プログラムから開始してください。

コンパイル・ステップの一部として、コンパイルに必要なデータ・セットを定義し、プログラムおよび求める出力に必要なコンパイラー・オプションを指定する必要があります。

コンパイラーは、COBOL プログラムを、コンピューターが処理できる言語 (オブジェクト・コード) に変換します。さらに、コンパイラーは、ソース・ステートメント内のエラーをリストして、プログラムのデバッグおよび調整に役立つ補足情報を提供します。コンパイルを制御するには、コンパイラー指示ステートメントおよびコンパイラー・オプションを使用してください。

プログラムをコンパイルした後、コンパイルの結果を検討して、コンパイラー検出エラーがあれば、それを訂正する必要があります。

関連タスク
282 ページの『JCL を使用したコンパイル』
294 ページの『TSO のもとでのコンパイル』
319 ページの『第 15 章 UNIX のもとでのコンパイル』
296 ページの『アセンブラー・プログラムからコンパイラーを開始する』
298 ページの『コンパイラー入出力の定義』
306 ページの『z/OS のもとでのコンパイラー・オプションの指定』
310 ページの『複数プログラムのコンパイル (バッチ・コンパイル)』
314 ページの『ソース・プログラムのエラーの訂正』

関連参照 407 ページの『第 18 章 コンパイラー指示ステートメント』 298 ページの『z/OS のもとでコンパイラーによって使用されるデータ・セット』 308 ページの『z/OS のもとでのコンパイラー・オプションおよびコンパイラー出力』

JCL を使用したコンパイル

コンパイルで JCL の情報 (ジョブ記述、コンパイラーを呼び出すステートメント、および必要なデータ・セットの定義 (HFS ファイルのディレクトリー・パス (存在する場合) など)) を組み込みます。

z/OS のもとでプログラムをコンパイルする最も簡単な方法は、カタログ式プロシージャーを使用する JCL をコーディングすることです。*カタログ式プロシージャー*とは、*プロシージャー・ライブラリー* (SYS1.PROCLIB) と呼ばれる区分データ・セットにある一連のジョブ制御ステートメントです。

次の JCL は、カタログ式プロシージャーの一般形式を示します。

```
//jobname  JOB  parameters
//stepname EXEC [PROC=]procname[,{PARM=|PARM.stepname=}'options']
//SYSIN    DD   data-set parameters
. . .           (source program to be compiled)
/*
//
```

カタログ式プロシージャーを使用してオブジェクト指向のプログラムをコンパイルするときには、追加の考慮事項があります。

540 ページの『例: プロシージャー型 DLL アプリケーションのサンプル JCL』

関連タスク
『カタログ式プロシージャーの使用』
292 ページの『プログラムをコンパイルするための JCL の作成』
306 ページの『z/OS のもとでのコンパイラー・オプションの指定』
312 ページの『バッチ・コンパイルでのコンパイラー・オプションの指定』
538 ページの『DLL を作成するためのプログラムのコンパイル』

関連参照
298 ページの『z/OS のもとでコンパイラーによって使用されるデータ・セット』

カタログ式プロシージャーの使用

カタログ式プロシージャーは、JCL の EXEC ステートメントに指定してください。

例えば、次の JCL は、Enterprise COBOL プログラムをコンパイルし、必要なデータ・セットを定義するために、IBM 提供のカタログ式プロシージャー IGYWC を呼び出します。

```
//JOB1      JOB1
//STEPA     EXEC PROC=IGYWC
//COBOL.SYSIN    DD *
000100 IDENTIFICATION DIVISION
           * (the source code)
. . .
/*
```

ソース・コードの後の /* は省略できます。ソース・コードがデータ・セットに格納されている場合は、SYSIN DD * を、データ・セットを記述する適切なパラメーターに置き換えてください。

これらのプロシージャーは、z/OS の一部であるすべてのジョブ・スケジューラーで使用することができます。スケジューラーは、不要なパラメーターを検出すると、それらを無視するか、または代替パラメーターで置き換えます。

コンパイラー・オプションがプロシージャーで明示的に指定されない場合、インストール時に設定されたデフォルト・オプションが適用されます。これらのデフォルト・オプションは、必要なオプションを含めた EXEC を使用してオーバーライドすることができます。

対応する DD ステートメントをオーバーライドすれば、階層ファイル・システム内のデータ・セットを指定することができます。ただし、指定するコンパイラー・ユーティリティー・ファイル (SYSUT*x*) およびコピー・ライブラリー (SYSLIB) は MVS データ・セットでなければなりません。

カタログ式プロシージャーの呼び出し、EXEC ステートメントのオーバーライドと追加、および DD ステートメントのオーバーライドと追加の詳細は、言語環境プログラム情報の中で記載されています。

関連タスク
言語環境プログラム・プログラミング・ガイド

関連参照 『コンパイル・プロシージャー (IGYWC)』 285 ページの『コンパイルおよびリンク・エディット用プロシージャー (IGYWCL)』 286 ページの『コンパイル、リンク・エディット、実行用のプロシージャー (IGYWCLG)』 287 ページの『コンパイル、ロード、実行用のプロシージャー (IGYWCG)』 288 ページの『コンパイル、プリリンク、リンク・エディット用のプロシージャー (IGYWCPL)』 289 ページの『コンパイル、プリリンク、リンク・エディット、実行用のプロシージャー (IGYWCPLG)』 290 ページの『プリリンクおよびリンク・エディット用のプロシージャー (IGYWPL)』 291 ページの『コンパイル、プリリンク、ロード、実行用のプロシージャー (IGYWCPG)』 *MVS プログラム管理: ユーザーズ・ガイドおよび解説書*

コンパイル・プロシージャー (IGYWC)

IGYWC は、プログラムをコンパイルする単一ステップのカタログ式プロシージャーです。オブジェクト・モジュールを作成します。コンパイラーを呼び出す他のどのカタログ式プロシージャーにおいても、コンパイル・ステップは類似しています。

次の DD ステートメントを入力ストリームで指定し、ソース・プログラムの位置を示す必要があります。

```
//COBOL.SYSIN DD  *       (or appropriate parameters)
```

コンパイルするプログラムでコピーブックを使用する場合は、COPY ステートメントで指定されている SYSLIB または他のライブラリーに対して DD ステートメントも指定する必要があります。以下に例を示します。

```
//COBOL.SYSLIB  DD  DISP=SHR,DSN=DEPT88.BOBS.COBLIB
```

```
//IGYWC  PROC  LNGPRFX='IGY.V4R1M0',SYSLBLK=3200
//*
//*  COMPILE A COBOL PROGRAM
//*
```

```
//*  PARAMETER  DEFAULT VALUE    USAGE
//*   SYSLBLK   3200             BLKSIZE FOR OBJECT DATA SET
//*   LNGPRFX   IGY.V4R1M0       PREFIX FOR LANGUAGE DATA SET NAMES
//*
//*  CALLER MUST SUPPLY //COBOL.SYSIN DD . . .
//*
//COBOL  EXEC PGM=IGYCRCTL,REGION=2048K
//STEPLIB  DD  DSNAME=&LNGPRFX..SIGYCOMP,          (1)
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&LOADSET,UNIT=SYSDA,
//             DISP=(MOD,PASS),SPACE=(TRK,(3,3)),
//             DCB=(BLKSIZE=&SYSLBLK)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT4   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT5   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))        (2)
//SYSUT6   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT7   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
```

(1) STEPLIB は、インストール先によって異なります。

(2) SYSUT5 は、LIB オプションを使用した場合にのみ必要です。

『例: HFS を使用するコンパイル用の JCL』

例: HFS を使用するコンパイル用の JCL:

次のジョブは、IGYWC というプロシージャーを使用して、階層ファイル・システム (HFS) に置かれている COBOL プログラム demo.cbl をコンパイルします。このジョブは、生成されたコンパイラー・リスト demo.lst、オブジェクト・ファイル demo.o、および SYSADATA ファイル demo.adt を HFS に書き込みます。

```
//HFSDEMO JOB ,
// TIME=(1),MSGLEVEL=(1,1),MSGCLASS=H,CLASS=A,REGION=50M,
// NOTIFY=&SYSUID,USER=&SYSUID
//COMPILE EXEC IGYWC,
// PARM.COBOL='LIST,MAP,RENT,FLAG(I,I),XREF,ADATA'
//SYSPRINT DD PATH='/u/userid/cobol/demo.lst',    (1)
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),              (2)
// PATHMODE=SIRWXU,                               (3)
// FILEDATA=TEXT                                  (4)
//SYSLIN DD PATH='/u/userid/cobol/demo.o',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=SIRWXU
//SYSADATA DD PATH='/u/userid/cobol/demo.adt',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=SIRWXU
//SYSIN DD PATH='/u/userid/cobol/demo.cbl',
// PATHOPTS=ORDONLY,
// FILEDATA=TEXT,
// RECFM=F
```

(1) PATH には、HFS ファイルのパス名を指定します。

(2) PATHOPTS は、ファイルのアクセス (読み取り、読み取り / 書き込みなど) を示し、ファイルの状況 (付加、作成、切り捨てなど) を設定します。

(3) PATHMODE は、ファイルの作成時に設定される許可、すなわちファイル・アクセス属性を示します。

(4) FILEDATA は、データをテキストまたはバイナリーのいずれとして扱うかを指定します。

この例に示すコンパイル用 DD ステートメントでは、オーバーライドとして、HFS (PATH='*hfs-directory-path*') と MVS データ・セット (DSN=*traditional-data-set-name*) の組み合わせを使用できます。ただし、コンパイラー・ユーティリティー・ファイル (DD ステートメント SYSUT*x*) および COPY ライブラリー (DD ステートメント SYSLIB) は、MVS データ・セットでなければなりません。

関連参照
UNIX システム・サービス・コマンド解説書
MVS JCL 解説書
298 ページの『z/OS のもとでコンパイラーによって使用されるデータ・セット』

コンパイルおよびリンク・エディット用プロシージャー (IGYWCL)

IGYWCL プロシージャーは、プログラムのコンパイルとリンク・エディットを行う 2 ステップのカタログ式プロシージャーです。

COBOL ジョブ・ステップは、リンケージ・エディターまたはバインダーへの入力となるオブジェクト・モジュールを生成します。他のオブジェクト・モジュールを追加することもできます。次の DD ステートメントを入力ストリームで指定し、ソース・プログラムの位置を示す必要があります。

```
//COBOL.SYSIN DD  *       (or appropriate parameters)
```

プログラムがコピーブックを使用する場合は、COPY ステートメントで指定されている SYSLIB または他のライブラリーに対して DD ステートメントも指定する必要があります。以下に例を示します。

```
//COBOL.SYSLIB  DD  DISP=SHR,DSN=DEPT88.BOBS.COBLIB

//IGYWCL PROC  LNGPRFX='IGY.V4R1M0',SYSLBLK=3200,
//             LIBPRFX='CEE',
//             PGMLIB='&&GOSET',GOPGM=GO
//*
//*  COMPILE AND LINK EDIT A COBOL PROGRAM
//*
//*  PARAMETER  DEFAULT VALUE    USAGE
//*   LNGPRFX   IGY.V4R1M0       PREFIX FOR LANGUAGE DATA SET NAMES
//*   SYSLBLK   3200             BLOCK SIZE FOR OBJECT DATA SET
//*   LIBPRFX   CEE              PREFIX FOR LIBRARY DATA SET NAMES
//*   PGMLIB    &&GOSET          DATA SET NAME FOR LOAD MODULE
//*   GOPGM     GO               MEMBER NAME FOR LOAD MODULE
//*
//*  CALLER MUST SUPPLY //COBOL.SYSIN DD . . .
//*
//COBOL  EXEC PGM=IGYCRCTL,REGION=2048K
//STEPLIB  DD  DSNAME=&LNGPRFX..SIGYCOMP,                (1)
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&LOADSET,UNIT=SYSDA,
//             DISP=(MOD,PASS),SPACE=(TRK,(3,3)),
//             DCB=(BLKSIZE=&SYSLBLK)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT4   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT5   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))             (2)
//SYSUT6   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT7   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//LKED   EXEC PGM=HEWL,COND=(8,LT,COBOL),REGION=1024K
//SYSLIB   DD  DSNAME=&LIBPRFX..SCEELKED,                (3)
//             DISP=SHR
```

```
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&LOADSET,DISP=(OLD,DELETE)
//         DD  DDNAME=SYSIN
//SYSLMOD  DD  DSNAME=&PGMLIB(&GOPGM),
//             SPACE=(TRK,(10,10,1)),
//             UNIT=SYSDA,DISP=(MOD,PASS)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(TRK,(10,10))
```

(1) STEPLIB は、インストール先によって異なります。

(2) SYSUT5 は、LIB オプションを使用した場合にのみ必要です。

(3) SYSLIB は、インストール先によって異なります。

コンパイル、リンク・エディット、実行用のプロシージャー (IGYWCLG)

IGYWCLG は、プログラムのコンパイル、リンク・エディット、および実行を行う 3 ステップのカタログ式プロシージャーです。

COBOL ジョブ・ステップは、リンケージ・エディターまたはバインダーへの入力となるオブジェクト・モジュールを生成します。他のオブジェクト・モジュールを追加することもできます。 COBOL プログラムでデータ・セットを参照する場合は、これらのデータ・セットを定義する DD ステートメントも指定する必要があります。次の DD ステートメントを入力ストリームで指定し、ソース・プログラムの位置を示す必要があります。

```
//COBOL.SYSIN DD  *       (or appropriate parameters)
```

プログラムがコピーブックを使用する場合は、COPY ステートメントで指定されている SYSLIB または他のライブラリーに対して DD ステートメントも指定する必要があります。以下に例を示します。

```
//COBOL.SYSLIB  DD  DISP=SHR,DSN=DEPT88.BOBS.COBLIB
```

```
//IGYWCLG PROC LNGPRFX='IGY.V4R1M0',SYSLBLK=3200,
//             LIBPRFX='CEE',GOPGM=GO
//*
//*  COMPILE, LINK EDIT AND RUN A COBOL PROGRAM
//*
//*  PARAMETER  DEFAULT VALUE    USAGE
//*   LNGPRFX   IGY.V4R1M0       PREFIX FOR LANGUAGE DATA SET NAMES
//*   SYSLBLK   3200             BLKSIZE FOR OBJECT DATA SET
//*   LIBPRFX   CEE              PREFIX FOR LIBRARY DATA SET NAMES
//*   GOPGM     GO               MEMBER NAME FOR LOAD MODULE
//*
//*  CALLER MUST SUPPLY //COBOL.SYSIN DD . . .
//*
//COBOL  EXEC PGM=IGYCRCTL,REGION=2048K
//STEPLIB  DD  DSNAME=&LNGPRFX..SIGYCOMP,                    (1)
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&LOADSET,UNIT=SYSDA,
//             DISP=(MOD,PASS),SPACE=(TRK,(3,3)),
//             DCB=(BLKSIZE=&SYSLBLK)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT4   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT5   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))                  (2)
//SYSUT6   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT7   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//LKED   EXEC PGM=HEWL,COND=(8,LT,COBOL),REGION=1024K
```

```
//SYSLIB   DD  DSNAME=&LIBPRFX..SCEELKED,                      (3)
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&LOADSET,DISP=(OLD,DELETE)
//         DD  DDNAME=SYSIN
//SYSLMOD  DD  DSNAME=&&GOSET(&GOPGM),SPACE=(TRK,(10,10,1)),
//             UNIT=SYSDA,DISP=(MOD,PASS)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(TRK,(10,10))
//GO     EXEC PGM=*.LKED.SYSLMOD,COND=((8,LT,COBOL),(4,LT,LKED)),
//             REGION=2048K
//STEPLIB  DD  DSNAME=&LIBPRFX..SCEERUN,                       (1)
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
```

(1) STEPLIB は、インストール先によって異なります。

(2) SYSUT5 は、LIB オプションを使用した場合にのみ必要です。

(3) SYSLIB は、インストール先によって異なります。

コンパイル、ロード、実行用のプロシージャー (IGYWCG)

IGYWCG は、プログラムのコンパイル、ロード、および実行を行う 2 ステップのカタログ式プロシージャーです。

COBOL ジョブ・ステップは、ローダーへの入力となるオブジェクト・モジュールを作成します。COBOL プログラムでデータ・セットを参照する場合は、これらのデータ・セットを定義する DD ステートメントを指定する必要があります。次の DD ステートメントを入力ストリームで指定し、ソース・プログラムの位置を示す必要があります。

```
//COBOL.SYSIN DD  *       (or appropriate parameters)
```

プログラムがコピーブックを使用する場合は、COPY ステートメントで指定されている SYSLIB または他のライブラリーに対して DD ステートメントも指定する必要があります。以下に例を示します。

```
//COBOL.SYSLIB  DD  DISP=SHR,DSN=DEPT88.BOBS.COBLIB
```

```
//IGYWCG PROC  LNGPRFX='IGY.V4R1M0',SYSLBLK=3200,
//             LIBPRFX='CEE'
//*
//*  COMPILE, LOAD AND RUN A COBOL PROGRAM
//*
//*  PARAMETER  DEFAULT VALUE    USAGE
//*   LNGPRFX   IGY.V4R1M0       PREFIX FOR LANGUAGE DATA SET NAMES
//*   SYSLBLK   3200             BLKSIZE FOR OBJECT DATA SET
//*   LIBPRFX   CEE              PREFIX FOR LIBRARY DATA SET NAMES
//*
//*  CALLER MUST SUPPLY //COBOL.SYSIN DD . . .
//*
//COBOL  EXEC PGM=IGYCRCTL,REGION=2048K
//STEPLIB  DD  DSNAME=&LNGPRFX..SIGYCOMP,                  (1)
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&LOADSET,UNIT=SYSDA,                (2)
//             DISP=(MOD,PASS),SPACE=(TRK,(3,3)),
//             DCB=(BLKSIZE=&SYSLBLK)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT4   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
```

```
//SYSUT5   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))                (3)
//SYSUT6   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT7   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//GO     EXEC PGM=LOADER,COND=(8,LT,COBOL),REGION=2048K
//SYSLIB   DD  DSNAME=&LIBPRFX..SCEELKED,                  (4)
//             DISP=SHR
//SYSLOUT  DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&LOADSET,DISP=(OLD,DELETE)
//STEPLIB  DD  DSNAME=&LIBPRFX..SCEERUN,                   (1)
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
```

(1) STEPLIB は、インストール先によって異なります。

(2) SYSLIN は HFS に常駐できます。

(3) SYSUT5 は、LIB オプションを使用した場合にのみ必要です。

(4) SYSLIB は、インストール先によって異なります。

コンパイル、プリリンク、リンク・エディット用のプロシージャー (IGYWCPL)

IGYWCPL は、プログラムのコンパイル、プリリンク、およびリンク・エディットを行う 3 ステップのカタログ式プロシージャーです。

次の DD ステートメントを入力ストリームで指定し、ソース・プログラムの位置を示す必要があります。

```
SYSIN DD  *       (or appropriate parameters)
```

プログラムがコピーブックを使用する場合は、COPY ステートメントで指定されている SYSLIB または他のライブラリーに対して DD ステートメントも指定する必要があります。以下に例を示します。

```
//COBOL.SYSLIB  DD  DISP=SHR,DSN=DEPT88.BOBS.COBLIB
```

```
//IGYWCPL PROC LNGPRFX='IGY.V4R1M0',SYSLBLK=3200,
//             LIBPRFX='CEE',PLANG=EDCPMSGE,
//             PGMLIB='&&GOSET',GOPGM=GO
//*
//*  COMPILE, PRELINK AND LINK EDIT A COBOL PROGRAM
//*
//*  PARAMETER  DEFAULT VALUE    USAGE
//*   LNGPRFX   IGY.V4R1M0       PREFIX FOR LANGUAGE DATA SET NAMES
//*   SYSLBLK   3200             BLOCK SIZE FOR OBJECT DATA SET
//*   LIBPRFX   CEE              PREFIX FOR LIBRARY DATA SET NAMES
//*   PLANG     EDCPMSGE         PRELINKER MESSAGES MODULE
//*   PGMLIB    &&GOSET          DATA SET NAME FOR LOAD MODULE
//*   GOPGM     GO               MEMBER NAME FOR LOAD MODULE
//*
//*  CALLER MUST SUPPLY //COBOL.SYSIN DD . . .
//*
//COBOL  EXEC PGM=IGYCRCTL,REGION=2048K
//STEPLIB  DD  DSNAME=&LNGPRFX..SIGYCOMP,                (1)
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&LOADSET,UNIT=SYSDA,
//             DISP=(MOD,PASS),SPACE=(TRK,(3,3)),
//             DCB=(BLKSIZE=&SYSLBLK)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
```

```
//SYSUT4   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT5   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))              (2)
//SYSUT6   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT7   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//PLKED   EXEC PGM=EDCPRLK,PARM='',COND=(8,LT,COBOL),
//             REGION=2048K
//STEPLIB  DD  DSNAME=&LIBPRFX..SCEERUN,
//             DISP=SHR
//SYSMSGS  DD  DSNAME=&LIBPRFX..SCEEMSGP(&PLANG),
//             DISP=SHR
//SYSLIB   DD  DUMMY
//SYSIN    DD  DSN=&&LOADSET,DISP=(OLD,DELETE)
//SYSMOD   DD  DSNAME=&&PLKSET,UNIT=SYSDA,DISP=(NEW,PASS),
//             SPACE=(32000,(100,50)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSDEFSD DD  DUMMY
//SYSOUT   DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//*
//LKED   EXEC PGM=HEWL,COND=(8,LT,COBOL),REGION=1024K
//SYSLIB   DD  DSNAME=&LIBPRFX..SCEELKED,                (3)
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&PLKSET,DISP=(OLD,DELETE)
//         DD  DDNAME=SYSIN
//SYSLMOD  DD  DSNAME=&PGMLIB(&GOPGM),
//             SPACE=(TRK,(10,10,1)),
//             UNIT=SYSDA,DISP=(MOD,PASS)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(TRK,(10,10))
```

(1) STEPLIB は、インストール先によって異なります。

(2) SYSUT5 は、LIB オプションを使用した場合にのみ必要です。

(3) SYSLIB は、インストール先によって異なります。

コンパイル、プリリンク、リンク・エディット、実行用のプロシージャー (IGYWCPLG)

IGYWCPLG は、プログラムのコンパイル、プリリンク、リンク・エディット、および実行を行う 4 ステップのカタログ式プロシージャーです。

次の DD ステートメントを入力ストリームで指定し、ソース・プログラムの位置を示す必要があります。

```
SYSIN DD  *       (or appropriate parameters)
```

プログラムがコピーブックを使用する場合は、COPY ステートメントで指定されている SYSLIB または他のライブラリーに対して DD ステートメントも指定する必要があります。以下に例を示します。

```
//COBOL.SYSLIB  DD  DISP=SHR,DSN=DEPT88.BOBS.COBLIB
```

```
//IGYWCPLG PROC LNGPRFX='IGY.V4R1M0',SYSLBLK=3200,
//              PLANG=EDCPMSGE,
//              LIBPRFX='CEE',GOPGM=GO
//*
//*  COMPILE, PRELINK, LINK EDIT, AND RUN A COBOL PROGRAM
//*
//*  PARAMETER  DEFAULT VALUE    USAGE
//*   LNGPRFX   IGY.V4R1M0       PREFIX FOR LANGUAGE DATA SET NAMES
//*   SYSLBLK   3200             BLKSIZE FOR OBJECT DATA SET
//*   PLANG     EDCPMSGE         PRELINKER MESSAGES MODULE
//*   LIBPRFX   CEE              PREFIX FOR LIBRARY DATA SET NAMES
//*   GOPGM     GO               MEMBER NAME FOR LOAD MODULE
```

```
//*
//*  CALLER MUST SUPPLY //COBOL.SYSIN DD . . .
//*
//COBOL  EXEC PGM=IGYCRCTL,REGION=2048K
//STEPLIB  DD  DSNAME=&LNGPRFX..SIGYCOMP,                    (1)
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&LOADSET,UNIT=SYSDA,
//             DISP=(MOD,PASS),SPACE=(TRK,(3,3)),
//             DCB=(BLKSIZE=&SYSLBLK)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT4   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT5   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))                  (2)
//SYSUT6   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT7   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//PLKED   EXEC PGM=EDCPRLK,PARM='',COND=(8,LT,COBOL),
//             REGION=2048K
//STEPLIB  DD  DSNAME=&LIBPRFX..SCEERUN,
//             DISP=SHR
//SYSMSGS  DD  DSNAME=&LIBPRFX..SCEEMSGP(&PLANG),
//             DISP=SHR
//SYSLIB   DD  DUMMY
//SYSIN    DD  DSN=&&LOADSET,DISP=(OLD,DELETE)
//SYSMOD   DD  DSNAME=&&PLKSET,UNIT=SYSDA,DISP=(NEW,PASS),
//             SPACE=(32000,(100,50)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSDEFSD DD  DUMMY
//SYSOUT   DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//*
//LKED   EXEC PGM=HEWL,COND=(8,LT,COBOL),REGION=1024K
//SYSLIB   DD  DSNAME=&LIBPRFX..SCEELKED,                    (3)
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&PLKSET,DISP=(OLD,DELETE)
//         DD  DDNAME=SYSIN
//SYSLMOD  DD  DSNAME=&&GOSET(&GOPGM),SPACE=(TRK,(10,10,1)),
//             UNIT=SYSDA,DISP=(MOD,PASS)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(TRK,(10,10))
//GO     EXEC PGM=*.LKED.SYSLMOD,COND=((8,LT,COBOL),(4,LT,LKED)),
//             REGION=2048K
//STEPLIB  DD  DSNAME=&LIBPRFX..SCEERUN,
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
```

(1) STEPLIB は、インストール先によって異なります。

(2) SYSUT5 は、LIB オプションを使用した場合にのみ必要です。

(3) SYSLIB は、インストール先によって異なります。

プリリンクおよびリンク・エディット用のプロシージャー (IGYWPL)

IGYWPL カタログ式プロシージャーは、プログラムのプリリンクおよびリンク・エディットを行う 2 ステップのプロシージャーです。

```
//IGYWPL PROC  PLANG=EDCPMSGE,SYSLBLK=3200,
//             LIBPRFX='CEE',
//             PGMLIB='&&GOSET',GOPGM=GO
//*
//*  PRELINK AND LINK EDIT A COBOL PROGRAM
//*
```

```
//*  PARAMETER  DEFAULT VALUE    USAGE
//*   PLANG     EDCPMSGE         PRELINK MESSAGES MEMBER NAME
//*   SYSLBLK   3200             BLKSIZE FOR OBJECT DATA SET
//*   LIBPRFX   CEE              PREFIX FOR LIBRARY DATA SET NAMES
//*   PGMLIB    &&GOSET          DATA SET NAME FOR LOAD MODULE
//*   GOPGM     GO               MEMBER NAME FOR LOAD MODULE
//*
//*  CALLER MUST SUPPLY //PLKED.SYSIN DD . . .
//*
//PLKED   EXEC PGM=EDCPRLK,PARM='',
//             REGION=2048K
//STEPLIB  DD  DSNAME=&LIBPRFX..SCEERUN,                (1)
//             DISP=SHR
//SYSMSGS  DD  DSNAME=&LIBPRFX..SCEEMSGP(&PLANG),
//             DISP=SHR
//SYSLIB   DD  DUMMY
//SYSMOD   DD  DSNAME=&&PLKSET,UNIT=SYSDA,DISP=(NEW,PASS),
//             SPACE=(32000,(100,50)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=&SYSLBLK)
//SYSDEFSD DD  DUMMY
//SYSOUT   DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//*
//LKED   EXEC PGM=HEWL,COND=(4,LT,PLKED),REGION=1024K
//SYSLIB   DD  DSNAME=&LIBPRFX..SCEELKED,               (2)
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=*.PLKED.SYSMOD,DISP=(OLD,DELETE)
//         DD  DDNAME=SYSIN
//SYSLMOD  DD  DSNAME=&PGMLIB(&GOPGM),SPACE=(TRK,(10,10,1)),
//             UNIT=SYSDA,DISP=(MOD,PASS)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(TRK,(10,10))
//SYSIN    DD  DUMMY
```

(1) STEPLIB は、インストール先によって異なります。

(2) SYSLIB は、インストール先によって異なります。

コンパイル、プリリンク、ロード、実行用のプロシージャー (IGYWCPG)

IGYWCPG は、プログラムのコンパイル、プリリンク、ロード、および実行を行う 4 ステップのカタログ式プロシージャーです。

次の DD ステートメントを入力ストリームで指定し、ソース・プログラムの位置を示す必要があります。

```
//COBOL.SYSIN DD  *       (or appropriate parameters)
```

プログラムがコピーブックを使用する場合は、COPY ステートメントで指定されている SYSLIB または他のライブラリーに対して DD ステートメントも指定する必要があります。以下に例を示します。

```
//COBOL.SYSLIB  DD  DISP=SHR,DSN=DEPT88.BOBS.COBLIB
```

```
//IGYWCPG PROC LNGPRFX='IGY.V4R1M0',SYSLBLK=3200,
//             PLANG=EDCPMSGE,
//             LIBPRFX='CEE'
//*
//*  COMPILE, PRELINK, LOAD, AND RUN A COBOL PROGRAM
//*
//*  PARAMETER  DEFAULT VALUE    USAGE
//*   LNGPRFX   IGY.V4R1M0       PREFIX FOR LANGUAGE DATA SET NAMES
//*   SYSLBLK   3200             BLKSIZE FOR OBJECT DATA SET
//*   PLANG     EDCPMSGE         PRELINKER MESSAGES MODULE
```

```
//*   LIBPRFX   CEE              PREFIX FOR LIBRARY DATA SET NAMES
//*
//*  CALLER MUST SUPPLY //COBOL.SYSIN DD . . .
//*
//COBOL   EXEC PGM=IGYCRCTL,REGION=2048K
//STEPLIB  DD  DSNAME=&LNGPRFX..SIGYCOMP,                  (1)
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&LOADSET,UNIT=SYSDA,
//             DISP=(MOD,PASS),SPACE=(TRK,(3,3)),
//             DCB=(BLKSIZE=&SYSLBLK)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT4   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT5   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))                (2)
//SYSUT6   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT7   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//PLKED   EXEC PGM=EDCPRLK,PARM='',COND=(8,LT,COBOL),
//             REGION=2048K
//STEPLIB  DD  DSNAME=&LIBPRFX..SCEERUN,
//             DISP=SHR
//SYSMSGS  DD  DSNAME=&LIBPRFX..SCEEMSGP(&PLANG),
//             DISP=SHR
//SYSLIB   DD  DUMMY
//SYSIN    DD  DSN=&&LOADSET,DISP=(OLD,DELETE)
//SYSMOD   DD  DSNAME=&&PLKSET,UNIT=SYSDA,DISP=(NEW,PASS), (3)
//             SPACE=(32000,(100,50)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSDEFSD DD  DUMMY
//SYSOUT   DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//*
//GO      EXEC PGM=LOADER,COND=(8,LT,COBOL),REGION=2048K
//SYSLIB   DD  DSNAME=&LIBPRFX..SCEELKED,                  (4)
//             DISP=SHR
//SYSLOUT  DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&PLKSET,DISP=(OLD,DELETE)
//STEPLIB  DD  DSNAME=&LIBPRFX..SCEERUN,
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
```

(1) STEPLIB は、インストール先によって異なります。

(2) SYSUT5 は、LIB オプションを使用した場合にのみ必要です。

(3) SYSMOD は HFS に常駐できます。

(4) SYSLIB は、インストール先によって異なります。

プログラムをコンパイルするための JCL の作成

カタログ式プロシージャーでは、より複雑なプログラムに必要となる柔軟性が得られない場合は、独自のジョブ制御ステートメントを作成してください。次の例では、プログラムのコンパイルに使用される JCL の一般形式を示します。

```
//jobname  JOB  acctno,name,MSGCLASS=1               (1)
//stepname EXEC PGM=IGYCRCTL,PARM=(options)          (2)
//STEPLIB  DD   DSNAME=IGY.V4R1M0.SIGYCOMP,DISP=SHR  (3)
//SYSUT1   DD   UNIT=SYSDA,SPACE=(subparms)          (4)
//SYSUT2   DD   UNIT=SYSDA,SPACE=(subparms)
//SYSUT3   DD   UNIT=SYSDA,SPACE=(subparms)
//SYSUT4   DD   UNIT=SYSDA,SPACE=(subparms)
//SYSUT5   DD   UNIT=SYSDA,SPACE=(subparms)
```

```
//SYSUT6   DD   UNIT=SYSDA,SPACE=(subparms)
//SYSUT7   DD   UNIT=SYSDA,SPACE=(subparms)
//SYSPRINT DD   SYSOUT=A                              (5)
//SYSLIN   DD   DSNAME=MYPROG,UNIT=SYSDA,             (6)
//              DISP=(MOD,PASS),SPACE=(subparms)
//SYSIN    DD   DSNAME=dsname,UNIT=device,            (7)
              VOLUME=(subparms),DISP=SHR
```

(1) JOB ステートメントは、ジョブの始まりを示します。

(2) EXEC ステートメントは、Enterprise COBOL コンパイラー (IGYCRCTL) を呼び出すことを指定します。

(3) この DD ステートメントは、Enterprise COBOL コンパイラーが常駐するデータ・セットを定義します。

(4) SYSUT DD ステートメントは、コンパイラーがソース・プログラムを処理するために使用するユーティリティー・データ・セットを定義します。
SYSUT ファイルはすべて、直接アクセス記憶装置上に置かなければなりません。

(5) SYSPRINT DD ステートメントは、LIST や MAP などのオプションからの出力を受け取るデータ・セットを定義します。SYSOUT=A は、システム出力装置を宛先とするデータ・セットの標準指定です。

(6) SYSLIN DD ステートメントは、OBJECT オプション (オブジェクト・モジュール) からの出力を受け取るデータ・セットを定義します。

(7) SYSIN DD ステートメントは、ジョブ・ステップ (ソース・コード) への入力として使用するデータ・セットを定義します。

以下のデータ・セットのコンパイル用 DD ステートメントでは、HFS (PATH='*hfs-directory-path*') と MVS データ・セット (DSN=*traditional-data-set-name*) の組み合わせを使用できます。

- ソース・ファイル
- オブジェクト・ファイル
- リスト
- ADATA ファイル
- デバッグ・ファイル
- 実行可能モジュール

ただし、コンパイラー・ユーティリティー・ファイル (DD ステートメント SYSUT*x*) および COPY ライブラリー (DD ステートメント SYSLIB) は、MVS データ・セットでなければなりません。

『例: コンパイル用のユーザー作成 JCL』
540 ページの『例: プロシージャー型 DLL アプリケーションのサンプル JCL』

関連参照
MVS プログラム管理: ユーザーズ・ガイドおよび解説書

例: コンパイル用のユーザー作成 JCL

次の例は、基本 JCL を適応させることができる可能性をいくつか示します。

```
//JOB1     JOB                                          (1)
//STEP1    EXEC PGM=IGYCRCTL,PARM='OBJECT'              (2)
//STEPLIB  DD   DSNAME=IGY.V4R1M0.SIGYCOMP,DISP=SHR
//SYSUT1   DD   UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2   DD   UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3   DD   UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT4   DD   UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT5   DD   UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT6   DD   UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT7   DD   UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSPRINT DD   SYSOUT=A
//SYSLIN   DD   DSNAME=MYPROG,UNIT=SYSDA,
//              DISP=(MOD,PASS),SPACE=(TRK,(3,3))
//SYSIN    DD   *                                       (3)
000100 IDENTIFICATION DIVISION.
. . .
/*                                                      (4)
```

(1) JOB1 は、ジョブの名前です。

(2) STEP1 は、ジョブ内で唯一のジョブ・ステップの名前です。この EXEC ステートメントは、生成されたオブジェクト・コードを (リンク・ステップへの入力として使用する) ディスクまたはテープに入れることも指定しています。

(3) アスタリスクは、入力ストリームの中で入力データ・セットが続くことを示します。

(4) 区切りステートメント /* は、入力ストリームの中で、データとその後の制御ステートメントとを区切ります。

TSO のもとでのコンパイル

TSO のもとでは、TSO コマンド、コマンド・リスト (CLIST)、REXX™ EXEC、または ISPF を使用すると、従来の MVS データ・セットを使用するプログラムをコンパイルすることができます。 HFS ファイルを使用するプログラムの場合は、TSO コマンドまたは REXX EXEC を使用してコンパイルすることができます。

いずれの方式を使用する場合も、以下のように、データ・セットを割り振り、コンパイルを要求しなければなりません。

1. ALLOCATE コマンドを使用してデータ・セットを割り振ります。

   どのコンパイルでも、作業データ・セット (SYSUT*n*) と、SYSIN および SYSPRINT データ・セットを割り振らなければなりません。

   特定のコンパイラー・オプションを指定する場合は、他のデータ・セットも割り振る必要があります。例えば、TERMINAL コンパイラー・オプションを指定した場合には、 SYSTERM データ・セットを割り振って、端末でコンパイラー・メッセージを受け取る必要があります。

   データ・セットはどんな順序で割り振っても構いません。ただし、コンパイルを開始する前に、必要なすべてのデータ・セットを割り振っておく必要があります。

2. READY プロンプトで CALL コマンドを使用してコンパイルを要求します。

   ```
   CALL 'IGY.V4R1M0.SIGYCOMP(IGYCRCTL)'
   ```

ALLOCATE コマンドおよび CALL コマンドは、TSO コマンド行に指定することができます。また、HFS ファイルを使用していない場合は、これらのコマンドを CLIST に指定することができます。

SYSUT*x* ユーティリティー・データ・セットおよび SYSLIB ライブラリーを除いて、どのコンパイラー・データ・セットにも HFS ファイルを割り振ることができます。 ALLOCATE ステートメントの形式は、次のとおりです。

```
Allocate File(SYSIN) Path('/u/myu/myap/std/prog2.cbl')
Pathopts(ORDONLY) Filedata(TEXT)
```

『例: TSO のもとでコンパイルするための ALLOCATE および CALL』
296 ページの『例: TSO のもとでコンパイルするための CLIST』

関連参照
298 ページの『z/OS のもとでコンパイラーによって使用されるデータ・セット』

例: TSO のもとでコンパイルするための ALLOCATE および CALL

次の例は、TSO のもとでコンパイルするときの ALLOCATE および CALL コマンドの指定方法を示します。

```
[READY]
ALLOCATE FILE(SYSUT1) CYLINDERS SPACE(1 1)
[READY]
ALLOCATE FILE(SYSUT2) CYLINDERS SPACE(1 1)
[READY]
ALLOCATE FILE(SYSUT3) CYLINDERS SPACE(1 1)
[READY]
ALLOCATE FILE(SYSUT4) CYLINDERS SPACE(1 1)
[READY]
ALLOCATE FILE(SYSUT5) CYLINDERS SPACE(1 1)
[READY]
ALLOCATE FILE(SYSUT6) CYLINDERS SPACE(1 1)
[READY]
ALLOCATE FILE(SYSUT7) CYLINDERS SPACE(1 1)
[READY]
ALLOCATE FILE(SYSPRINT) SYSOUT
[READY]
ALLOCATE FILE(SYSTERM) DATASET(*)
[READY]
ALLOCATE FILE(SYSLIN) DATASET(PROG2.OBJ) NEW TRACKS SPACE(3,3)
[READY]
ALLOCATE FILE(SYSIN) DATASET(PROG2.COBOL) SHR
[READY]
CALL 'IGY.V4R1M0.SIGYCOMP(IGYCRCTL)' 'LIST,NOCOMPILE(S),OBJECT,FLAG(E,E),TERMINAL'
  .
   (COBOL listings and messages)
  .
[READY]
FREE FILE(SYSUT1,SYSUT2,SYSUT3,SYSUT4,SYSUT5,SYSUT6,SYSUT7,SYSPRINT,SYSTERM,+
SYSIN,SYSLIN)
[READY]
```

例: TSO のもとでコンパイルするための CLIST

次の例は、TSO のもとでコンパイルするための CLIST を示しています。 FREE コマンドは必要ありません。しかし、上手なプログラミングを行うには、ファイルを割り振る前に、それらを解放しておく必要があります。

```
PROC 1 MEM
CONTROL LIST
FREE (SYSUT1)
FREE (SYSUT2)
FREE (SYSUT3)
FREE (SYSUT4)
FREE (SYSUT5)
FREE (SYSUT6)
FREE (SYSUT7)
FREE (SYSPRINT)
FREE (SYSIN)
FREE (SYSLIN)
ALLOC F(SYSPRINT) SYSOUT
ALLOC F(SYSIN) DA(COBOL.SOURCE(&MEM))  SHR REUSE
ALLOC F(SYSLIN) DA(COBOL.OBJECT(&MEM)) OLD REUSE
ALLOC F(SYSUT1) NEW SPACE(5,5) TRACKS UNIT(SYSDA)
ALLOC F(SYSUT2) NEW SPACE(5,5) TRACKS UNIT(SYSDA)
ALLOC F(SYSUT3) NEW SPACE(5,5) TRACKS UNIT(SYSDA)
ALLOC F(SYSUT4) NEW SPACE(5,5) TRACKS UNIT(SYSDA)
ALLOC F(SYSUT5) NEW SPACE(5,5) TRACKS UNIT(SYSDA)
ALLOC F(SYSUT6) NEW SPACE(5,5) TRACKS UNIT(SYSDA)
ALLOC F(SYSUT7) NEW SPACE(5,5) TRACKS UNIT(SYSDA)
CALL 'IGY.V4R1M0.SIGYCOMP(IGYCRCTL)'
```

アセンブラー・プログラムからコンパイラーを開始する

動的起動で ATTACH または LINK マクロを使用して、アセンブラー・プログラム内から Enterprise COBOL コンパイラーを開始することができます。コンパイラー・オプションおよび処理時に使用するデータ・セットの DD 名を指定する必要があります。

以下に例を示します。

symbol {LINK|ATTACH} EP=IGYCRCTL,PARAM=(*optionlist*[,*ddnamelist*]),VL=1

EP コンパイラーのシンボル名を指定します。制御プログラム (ライブラリー・ディレクトリー記入項目からの) が、プログラムが実行を開始すべき入り口点を決定します。

PARAM アセンブラー・プログラムからコンパイラーに渡されるアドレス・パラメーターをサブリストとして指定します。

アドレス・パラメーター・リストの最初のフルワードには、COBOL*optionlist* のアドレスが入っています。2 番目のフルワードには、*ddnamelist* のアドレスが入っています。3 番目と 4 番目のフルワードには、NULL パラメーターまたは 0 が入っています。

optionlist

コンパイル用に指定された COBOL オプションが入っている可変長リストのアドレスを指定します。リストを提示しない場合でも、このアドレスは指定しなければなりません。

optionlist はハーフワード境界から始めなければなりません。高位の 2 バイトには、このリストの残りの部分にあるバイト数のカウントが含まれます。オプションが指定されない場合、このカウントは 0 でなければなりません。*optionlist* は、各フィールドが次のフィールドとコンマで区切られて、自由形式です。ブランクまたはゼロは使用できません。コンパイラーは最初の 100 文字のみを認識します。

ddnamelist

コンパイラー処理中に使用されるデータ・セットの代替の DD 名を含んでいる可変長リストのアドレスを指定します。標準 DD 名を使用する場合は、*ddnamelist* を省略することができます。

ddnamelist はハーフワード境界から始めなければなりません。高位の 2 バイトには、このリストの残りの部分にあるバイト数のカウントが含まれます。8 バイト未満の名前は、左寄せにしてブランクで埋め込む必要があります。代替 DD 名がリストから省略されている場合、標準名が想定されます。名前が省略されている場合、8 バイトの記入項目には 2 進数 0 が入っていなければなりません。リストを短縮して、終わりから名前を省略することができます。

指定された SYSUT*n* データ・セットはすべて直接アクセス記憶装置上になければならず、物理順次編成をもっていなければなりません。これらのデータ・セットが HFS に常駐していてはなりません。

次の表に、*ddnamelist* 内の 8 バイトの記入項目のシーケンスを示します。

代替 DD 名 8 バイト項目	代替 DD 名が置き換えられる名前
1	SYSLIN
2	適用されない
3	適用されない
4	SYSLIB
5	SYSIN
6	SYSPRINT
7	SYSPUNCH
8	SYSUT1
9	SYSUT2
10	SYSUT3
11	SYSUT4
12	SYSTERM
13	SYSUT5
14	SYSUT6
15	SYSUT7
16	SYSADATA
17	SYSJAVA
18	SYSDEBUG
19	SYSMDECK
20	SYSOPTF
21	DBRMLIB

VL アドレス・パラメーター・リストの最後のフルワードの符号ビットを 1 に設定するように指定します。

コンパイラーは、処理を完了すると、レジスター 15 に戻りコードを書き込みます。

関連タスク
『コンパイラー入出力の定義』

関連参照 『z/OS のもとでコンパイラーによって使用されるデータ・セット』 308 ページの『z/OS のもとでのコンパイラー・オプションおよびコンパイラー出力』

コンパイラー入出力の定義

コンパイラーが作業を行うために使用する数種類のデータ・セットを定義しなければなりません。コンパイラーは、入力データ・セットおよびライブラリーを使用して、さまざまなタイプの出力 (オブジェクト・コード、リスト、およびメッセージを含む) を作成します。また、コンパイラーは、コンパイル時にユーティリティー・データ・セットも使用します。

関連タスク 302 ページの『ソース・コード・データ・セットの定義 (SYSIN)』 302 ページの『コンパイラー・オプション・データ・セットの定義 (SYSOPTF)』 303 ページの『ソース・ライブラリーの指定 (SYSLIB)』 303 ページの『出力データ・セットの定義 (SYSPRINT)』 303 ページの『コンパイラー・メッセージの端末への送信 (SYSTERM)』 304 ページの『オブジェクト・コードの作成 (SYSLIN または SYSPUNCH)』 304 ページの『関連データ・ファイル (SYSADATA) の定義』 305 ページの『Java ソース出力ファイル (SYSJAVA) の作成』 305 ページの『デバッグ・データ・セットの定義 (SYSDEBUG)』 306 ページの『ライブラリー処理出力ファイル (SYSMDECK) の定義』

関連参照 『z/OS のもとでコンパイラーによって使用されるデータ・セット』 308 ページの『z/OS のもとでのコンパイラー・オプションおよびコンパイラー出力』

z/OS のもとでコンパイラーによって使用されるデータ・セット

次の表は、コンパイラーが使用する各データ・セットの機能、装置要件、および許可される装置クラスをリストしたものです。

表 36. **コンパイラー・データ・セット**

タイプ	ddname	関数	必要か?	装置要件	許可される装置クラス	HFS に常駐できるか?
入力	SYSIN[1]	ソース・プログラムの読み取り	はい	カード読取装置; 中間記憶装置	任意	はい
	SYSOPTF	コンパイラー・オプションの読み取り	OPTFILE が有効な場合	カード読取装置; 中間記憶装置; 直接アクセス	任意	はい
	SYSLIB または他のコピー・ライブラリー [1]	ユーザー・ソース・ライブラリー (PDS または PDSE) の読み取り	プログラムに COPY または BASIS ステートメントが含まれている場合 (LIB は必須)	直接アクセス	SYSDA	いいえ
ユーティリティー	SYSUT1、SYSUT2、SYSUT3、SYSUT4、SYSUT6[2]	コンパイル時にコンパイラーによって使用される作業データ・セット	はい	直接アクセス	SYSDA	いいえ
	SYSUT5[2]	コンパイル時にコンパイラーによって使用される作業データ・セット	プログラムに COPY、REPLACE、または BASIS ステートメントが含まれている場合 (LIB は必須)	直接アクセス	SYSDA	いいえ
	SYSUT7[2]	リストを作成するためにコンパイラーによって使用される作業データ・セット	はい	直接アクセス	SYSDA	いいえ

表 36. **コンパイラー・データ・セット** *(続き)*

タイプ	ddname	関数	必要か?	装置要件	許可される装置クラス	HFS に常駐できるか?
出力	SYSPRINT[1]	ストレージ・マップ、リスト、およびメッセージの書き込み	はい	プリンター; 中間記憶装置	SYSSQ、SYSDA、標準出力クラス A	はい
	SYSTERM	進行メッセージおよび診断メッセージの書き込み	TERM が有効な場合	出力装置; TSO 端末		はい
	SYSPUNCH	オブジェクト・コードの作成	DECK が有効な場合	カード穿孔装置; 直接アクセス	SYSSQ、SYSDA	はい
	SYSLIN	コンパイラーからの出力およびリンケージ・エディターまたはバインダーへの入力としてのオブジェクト・モジュール・データ・セットの作成	OBJECT が有効な場合	直接アクセス	SYSSQ、SYSDA	はい
	SYSADATA	関連データ・ファイル・レコードの書き込み	ADATA が有効な場合	出力装置		はい
	SYSJAVA	クラス定義用の生成済み Java ソース・ファイルの作成	クラス定義をコンパイルする場合	(HFS ファイルでなければならない)		はい
	SYSUDUMP、SYSABEND、または SYSMDUMP	ダンプの書き込み	DUMP が有効な場合 (まれにしか使用されない)	直接アクセス	SYSDA	はい
	SYSDEBUG	オブジェクト・モジュールとは別のデータ・セットへの記号デバッグ情報テーブルの書き込み	TEST(. . .,SEP,. . .) が有効な場合	直接アクセス	SYSDA	はい
	SYSMDECK	COPY、BASIS、REPLACE、および EXEC SQL INCLUDE ステートメントの拡張の書き込み	MDECK が有効な場合	直接アクセス	SYSDA	はい

1. EXIT オプションを使用して、これらのデータ・セットからユーザー出口を提供することができます。
2. これらのデータ・セットは単一ボリュームでなければなりません。

関連参照

301 ページの『論理レコード長とブロック・サイズ』
362 ページの『EXIT』

論理レコード長とブロック・サイズ

作業データ・セット (SYSUT*n*) および HFS ファイル以外のコンパイラー・データ・セットの場合は、DCB パラメーターの BLKSIZE サブパラメーターを使用してブロック・サイズを設定することができます。この値は、データ・セットが常駐する装置に許されているものでなければなりません。設定する値は、データ・セットが固定長か可変長かによって異なります。

固定長レコード (RECFM=F または RECFM=FB) の場合、LRECL は論理レコード長で、BLKSIZE は LRECL × *n* に等しくなります (*n* はブロック化因数)。

次の表は、固定長データ・セットに定義されている値を示しています。一般には、これらの値を変更してはなりませんが、以下のデータ・セットの場合は値を変更できます。

- SYSDEBUG: リストされた範囲内で、任意の LRECL を指定することができます。推奨値は 1024 です。
- SYSPRINT、SYSDEBUG: BLKSIZE=0 を指定でき、その結果として、システム決定のブロック・サイズになります。

表 37. **固定長コンパイラー・データ・セットのブロック・サイズ**

データ・セット	RECFM	LRECL (バイト)	BLKSIZE[1]
SYSDEBUG[2]	F または FB	80 から 1024[3]	LRECL x *n*
SYSIN	F または FB	80	80 x *n*
SYSLIB または他のコピー・ライブラリー	F または FB	80	80 x *n*
SYSLIN	F または FB	80	80 x *n*
SYSMDECK	F または FB	80	80 x *n*
SYSOPTF	F または FB	80	80 x *n*
SYSPRINT[2]	F または FB	133	133 x *n*
SYSPUNCH	F または FB	80	80 x *n*
SYSTERM	F または FB	80	80 x *n*

1. *n* = ブロック化因数
2. BLKSIZE=0 を指定すると、システムがブロック・サイズを決定します。
3. SYSDEBUG のデフォルトの LRECL は 1024 です。

可変長レコード (RECFM=V) の場合、LRECL は論理レコード長で、BLKSIZE は LRECL + 4 に等しくなります。

表 38. **可変長コンパイラー・データ・セットのブロック・サイズ**

データ・セット	RECFM	LRECL (バイト)	BLKSIZE (バイト) 最小許容値
SYSADATA	VB	1020	1024

ソース・コード・データ・セットの定義 (SYSIN)

以下に示す SYSIN DD ステートメントを使用して、ソース・コードが入ったデータ・セットを定義します。

```
//SYSIN  DD  DSNAME=dsname,UNIT=SYSSQ,VOLUME=(subparms),DISP=SHR
```

ソース・コードまたは BASIS ステートメントを入力ストリームに直接入れることができます。そうするには、次の SYSIN DD ステートメントを使用してください。

```
//SYSIN    DD *
```

ソース・コードまたは BASIS ステートメントは、DD * ステートメントの後ろに続ける必要があります。コンパイルの後に別のジョブ・ステップが続いている場合は、そのステップの EXEC ステートメントが /* ステートメントまたは最後のソース・ステートメントの後に続く必要があります。

コンパイラー・オプション・データ・セットの定義 (SYSOPTF)

以下に示す SYSOPTF DD ステートメントをコーディングすることによって、COBOL プログラムのコンパイラー・オプションが入ったデータ・セットを定義します。

```
//SYSOPTF  DD  DSNAME=dsname,UNIT=SYSDA,VOLUME=(subparms),DISP=SHR
```

コンパイラー・オプション・データ・セットを使用するには、OPTFILE を、コンパイラー呼び出しオプションとして、またはソース・プログラムの PROCESS または CBL ステートメントで、指定します。

SYSOPTF データ・セット内で以下を行います。

- 呼び出しオプション、または PROCESS または CBL ステートメントのコンパイラー・オプションで使用するものと同じ構文を使用して、桁 2 から桁 72 に、フリー・フォームで、コンパイラー・オプションを指定します。
- 行がコメントとして扱われるようにするには、桁 1 にアスタリスク (*) をコーディングします。
- オプションとして、桁 73 から 80 に、シーケンス番号をコーディングします。これらの桁は無視されます。

また、OPTFILE オプションを使用してコンパイルする場合は、SYSOPTF DD ステートメントの後ろの入力ストリームに直接、コンパイラー・オプションを配置することができます。

```
//COB      EXEC PGM=IGYCRCTL,PARM='OPTFILE'
//SYSOPTF  DD DATA,DLM=@@
 SSRANGE   ARITH(COMPAT)
 OPTIMIZE
 . . .
@@
//SYSIN    DD . . .
```

複数のコンパイラー・オプション・データ・セットがある場合には、以下のように、複数の SYSOPTF DD ステートメントを連結することができます。

```
//SYSOPTF  DD  DSNAME=dsname1, . . .
//         DD  DSNAME=dsname2, . . .
```

連結の後ろのデータ・セットにあるコンパイラー・オプションが、連結のそれより前にあるデータ・セットのオプションよりも優先されます。

関連参照
301 ページの『論理レコード長とブロック・サイズ』
377 ページの『OPTFILE』

ソース・ライブラリーの指定 (SYSLIB)

プログラムに COPY または BASIS ステートメントが含まれている場合には、SYSLIB DD ステートメントを使用します。これらの DD ステートメントは、ソース・コード内の COPY ステートメントまたは入力ストリーム内の BASIS ステートメントによって要求されるデータが入っているライブラリー (区分データ・セット) を定義します。

```
//SYSLIB  DD  DSNAME=copylibname,DISP=SHR
```

複数のコピー・ライブラリーまたは基本ライブラリーがある場合には、次のように複数の DD ステートメントを連結します。

```
//SYSLIB  DD  DSNAME=PROJECT.USERLIB,DISP=SHR
//        DD  DSNAME=SYSTEM.COPYX,DISP=SHR
```

ライブラリーは直接アクセス記憶装置に置かれます。 JCL を使用して、または TSO のもとで、コンパイルする場合、ライブラリーが HFS に入っていてはなりません。

NOLIB オプションが有効な場合は、SYSLIB DD ステートメントは必要ありません。

出力データ・セットの定義 (SYSPRINT)

DD 名 SYSPRINT を使用して、リストを生成します。このリストには、PARM パラメーターのデフォルト・オプションまたは要求されたオプションの結果 (すなわち、診断メッセージ、オブジェクト・コード・リスト) が入れられます。

出力は、SYSOUT データ・セット、プリンター、直接アクセス・ストレージ・デバイス、または磁気テープ装置に送ることができます。以下に例を示します。

```
//SYSPRINT DD SYSOUT=A
```

SYSPRINT データ・セットは、順次データ・セット、PDS または PDSE メンバー、あるいは HFS ファイルにすることができます。 SYSPRINT データ・セットのレコード・フォーマット、レコード長、およびブロック・サイズの指定方法の詳細については、下の関連参照を参照してください。

関連参照
301 ページの『論理レコード長とブロック・サイズ』

コンパイラー・メッセージの端末への送信 (SYSTERM)

TSO でコンパイルしている場合には、SYSTERM データ・セットを定義して、コンパイラー・メッセージを端末に送信することができます。

```
ALLOC F(SYSTERM) DA(*)
```

SYSTERM は、他のさまざまな方法で定義することができます (SYSOUT データ・セット、ディスク上のデータ・セット、HFS 内のファイル、別の印刷クラスなどに)。

オブジェクト・コードの作成 (SYSLIN または SYSPUNCH)

OBJECT コンパイラー・オプションを使用すると、オブジェクト・コードを、従来の MVS データ・セットまたは HFS ファイルとしてディスクに保管したり、あるいはテープに保管することができます。コンパイラーは、SYSLIN DD ステートメントまたは SYSPUNCH DD ステートメントで定義されたファイルを使用します。

```
//SYSLIN   DD  DSNAME=dsname,UNIT=SYSDA,
//             SPACE=(subparms),DISP=(MOD,PASS)
```

SYSLIN DD ステートメントの DISP パラメーターを使用して、オブジェクト・コード・データ・セットが以下のうちどのように処理されるかを示すことができます。

- リンケージ・エディターまたはバインダーに渡される
- カタログされる
- 保存される
- カタログされた既存のライブラリーに追加される

上記の例では、データが作成され、別のジョブ・ステップであるリンケージ・エディターまたはバインダーのジョブ・ステップに渡されます。

インストール先で、DECK オプションおよび SYSPUNCH DD ステートメントを使用することができます。B は、穿孔データ・セットの標準出力クラスです。

```
//SYSPUNCH DD  SYSOUT=B
```

NOOBJECT オプションが有効な場合、SYSLIN DD ステートメントは必要ではありません。NODECK オプションが有効な場合は、SYSPUNCH DD ステートメントは必要ではありません。

関連参照
376 ページの『OBJECT』
358 ページの『DECK』

関連データ・ファイル (SYSADATA) の定義

ADATA コンパイラー・オプションを使用する場合には、SYSADATA ファイルを定義します。

```
//SYSADATA DD DSNAME=dsname,UNIT=SYSDA
```

SYSADATA ファイルは、コンパイル中に収集されたプログラムに関する情報が入っている特定のレコード・タイプを含む順次ファイルです。このファイルは、従来の MVS データ・セットまたは HFS ファイルにすることができます。

関連参照
345 ページの『ADATA』

Java ソース出力ファイル (SYSJAVA) の作成

オブジェクト指向プログラムをコンパイルする場合は、SYSJAVA DD ステートメントを追加します。生成された Java ソース・ファイルは、SYSJAVA DD 名に書き込まれます。

```
//SYSJAVA   DD PATH='/u/userid/java/Classname.java',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=SIRWXU,
// FILEDATA=TEXT
```

SYSJAVA ファイルは、HFS に入っている必要があります。

関連タスク 334 ページの『JCL または TSO/E でのオブジェクト指向アプリケーションのコンパイル』

デバッグ・データ・セットの定義 (SYSDEBUG)

JCL または TSO からコンパイルするときに、TEST(. . .,SEP,. . .) コンパイラー・オプションを指定すると、シンボリック・デバッグ情報テーブルは、 SYSDEBUG DD ステートメントで指定されるデータ・セットに書き込まれます。

```
//SYSDEBUG DD  DSNAME=dsname,UNIT=SYSDA
```

SYSDEBUG データ・セットは、順次データ・セット、PDS または PDSE メンバー、あるいは HFS ファイルにすることができます。 SYSDEBUG データ・セットのレコード・フォーマット、レコード長、およびブロック・サイズの指定方法の詳細については、論理レコード長およびブロック・サイズに関する下記の関連参照を参照してください。

言語環境プログラム は、ダンプ・サービスのために SYSDEBUG を使用します。データ・セットの名前は、SYSDEBUG COBOL デバッグ・ファイル・ユーザー出口 (IGZIUXB) を使用して実行時に変更することができます。デバッグ・ツール は、SET DEFAULT LISTINGS コマンド、ユーザー出口 EQAUEDAT、または EQADEBUG DD ステートメントを使用して、名前を変更したデータ・セットに送信することができます。

DD 名 SYSDEBUG に指定したデータ・セット名は、複数の IBM 製品 (言語環境プログラム、デバッグ・ツール、障害アナライザー、および アプリケーション・パフォーマンス・アナライザー など) が使用する可能性があります。詳細については、これらの個別製品の資料を参照してください。

関連タスク
言語環境プログラム・カスタマイズ (COBOL デバッグ・ファイル名の修正)
Debug Tool User's Guide (Debug Tool が COBOL および PL/I の個別デバッグ・ファイルの場所を探索する方法)

関連参照
301 ページの『論理レコード長とブロック・サイズ』
392 ページの『TEST』

ライブラリー処理出力ファイル (SYSMDECK) の定義

MDECK コンパイラー・オプションを使用する場合には、SYSMDECK ファイルを定義します。

```
//SYSMDECK DD DSNAME=dsname,UNIT=SYSDA
```

SYSMDECK ファイルには、ライブラリー処理 (COPY、BASIS、REPLACE、および EXEC SQL INCLUDE ステートメントの拡張) の出力が含まれます。このファイルは、従来の MVS データ・セットまたは HFS ファイルにすることができます。

関連参照
371 ページの『MDECK』

z/OS のもとでのコンパイラー・オプションの指定

コンパイラーは、デフォルトのコンパイラー・オプションを使用してインストールされます。コンパイラーのインストール時に、システム・プログラマーは、例えば、パフォーマンスの向上やある程度の標準の維持を確実に行うために、コンパイラー・オプションの設定を固定することができます。固定されたコンパイラー・オプションはオーバーライドできません。

固定されていないオプションについては、以下のいずれかの方法でコンパイラー・オプションを指定して、デフォルト設定をオーバーライドすることができます。

- COBOL ソースの PROCESS または CBL ステートメントにコーディングする。
- コンパイラーの開始時に組み込む (JCL の EXEC ステートメントの PARM パラメーター、または TSO のコマンド行のいずれか)。
- SYSOPTF データ・セットに組み込んで、上記のいずれかの方法で、OPTFILE コンパイラー・オプションを指定する。

コンパイラーは、以下の優先順位で高位から下位へ、オプションを認識します。

1. 設置場所で固定されているインストール先デフォルト
2. バッチ内の最初のプログラムで有効にされた BUFSIZE、LIB、OUTDD、SIZE、および SQL コンパイラー・オプションの値
3. IDENTIFICATION DIVISION の前に置かれた PROCESS (または CBL) ステートメント
4. コンパイラー呼び出しで指定されたオプション (JCL PARM パラメーターまたは TSO CALL コマンド)
5. 固定されていないインストール先デフォルト

この優先順位の順序は、対立するオプションまたは互いに排他的なオプションが指定されたときにどのオプションが有効かも決定します。

SYSOPTF データ・セットのオプションの優先順位は、OPTFILE コンパイラー・オプションを指定した場所に依存します。例えば、PROCESS ステートメントで OPTFILE を指定した場合、SYSOPTF オプションによってコンパイラーの呼び出し時に指定したオプションが置き換えられます。詳細については、OPTFILE オプションに関する下の関連参照を参照してください。

大部分のオプションは対になっているため、いずれか一方を選択します。例えば、相互参照リストのオプションの対は XREFINOXREF です。相互参照リストが必要な場合は、XREF を指定します。そうでない場合には、NOXREF を指定します。

サブパラメーターがあるオプションもあります。例えば、リストで 1 ページ当たり 44 行が必要な場合には、LINECOUNT(44) と指定します。

308 ページの『例: JCL によるコンパイラー・オプションの指定』
308 ページの『例: TSO のもとでのコンパイラー・オプションの指定』

関連タスク
302 ページの『コンパイラー・オプション・データ・セットの定義 (SYSOPTF)』
『PROCESS (CBL) ステートメントによるコンパイラー・オプションの指定』
312 ページの『バッチ・コンパイルでのコンパイラー・オプションの指定』

関連参照 308 ページの『z/OS のもとでのコンパイラー・オプションおよびコンパイラー出力』 341 ページの『第 17 章 コンパイラー・オプション』 344 ページの『矛盾するコンパイラー・オプション』 377 ページの『OPTFILE』

PROCESS (CBL) ステートメントによるコンパイラー・オプションの指定

COBOL プログラムの PROCESS ステートメントにコンパイラー・オプションをコーディングすることができます。これは、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。

CBL/PROCESS ステートメントの構文

シーケンス・フィールドをコーディングしない場合は、PROCESS ステートメントを 1 から 66 桁目で始めることができます。シーケンス・フィールドは 1 から 6 桁目に指定することができます。シーケンス・フィールドが使用される場合には、そこに 6 文字が含まれていなければならず、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 から 66 桁目の間で開始することができます。

CBL を PROCESS の同義語として使用することができます。 CBL は 1 から 70 桁目の間で開始することができます。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 から 70 桁目の間で開始することができます。

PROCESS 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 ステートメントを使用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、PROCESS 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 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。

プログラミングの編成で、COBOL コンパイラーのデフォルト・オプション・モジュールを使用して、PROCESS ステートメントを使用することを禁止することができます。編成で許可されていない PROCESS ステートメントが COBOL プログラムで見つかった場合、COBOL コンパイラーはエラー診断を生成します。

関連参照
CBL (PROCESS) ステートメント (*Enterprise COBOL 言語解説書*)

例: JCL によるコンパイラー・オプションの指定

次の例は、z/OS のもとで JCL を使用してコンパイラー・オプションを指定する方法を示します。

```
. . .
//STEP1     EXEC PGM=IGYCRCTL,
//               PARM='LIST,NOCOMPILE(S),OBJECT,FLAG(E,E)'
```

例: TSO のもとでのコンパイラー・オプションの指定

次の例は、TSO のもとでコンパイラー・オプションを指定する方法を示します。

```
. . .
[READY]
CALL 'SYS1.LINKLIB(IGYCRCTL)' 'LIST,NOCOMPILE(S),OBJECT,FLAG(E,E)'
```

z/OS のもとでのコンパイラー・オプションおよびコンパイラー出力

コンパイラーがソース・プログラムの処理を完了すると、有効であったコンパイラー・オプションに応じて、1 つ以上の出力が作成されています。

表 39. **z/OS** *のもとでのコンパイラー出力のタイプ*

コンパイラー・オプション	コンパイラー出力	出力のタイプ
ADATA	コンパイル中のプログラムに関する情報	関連データ・ファイル
DLL	DLL サポートで使用可能なオブジェクト・モジュール	オブジェクト
DUMP	コンパイルが異常終了した場合は、システム・ダンプ (SYSUDUMP、SYSABEND、または SYSMDUMP DD ステートメントが必要)。まれにしか使用されません。	リスト
EXPORTALL	DLL のエクスポートされた記号	オブジェクト
FLAG	コンパイラーがプログラムで検出したエラーのリスト	リスト
LIST	マシン言語およびアセンブラー言語でのオブジェクト・コードのリスト	リスト
MAP	プログラムのデータ項目のマップ	リスト
MDECK	プログラムのライブラリー処理ステートメントの拡張	ライブラリー処理サイド・ファイル

表 39. z/OS のもとでのコンパイラー出力のタイプ (続き)

コンパイラー・オプション	コンパイラー出力	出力のタイプ
NUMBER	ユーザー提供の行番号がリストに示される	リスト
COMPILE を指定した OBJECT または DECK	オブジェクト・コード	オブジェクト
OFFSET	オブジェクト・コードの相対アドレスのマップ	リスト
OPTIMIZE	OBJECT が有効な場合は、最適化オブジェクト・コード	オブジェクト
RENT	OBJECT が有効な場合は、再入可能オブジェクト・コード	オブジェクト
SOURCE	ソース・プログラムのリスト	リスト
SQL	DB2 バインド処理用の SQL ステートメントおよびホスト変数情報	データベース要求モジュール (DBRM)
SSRANGE	テーブル内の参照を検査するための余分のコード	オブジェクト内
TERMINAL	端末に送信される進行メッセージと診断メッセージ	端末
TEST(HOOK)	デバッグ・ツールのコンパイルされたフック	オブジェクト内の余分のコード
TEST(NOSEP)	デバッグ・ツールおよび定様式ダンプのための情報テーブル	オブジェクト
TEST(SEP)	デバッグ・ツールおよび定様式ダンプのための情報テーブル	独立したデバッグ・ファイル
VBREF	ソース・プログラム内の動詞の相互参照リスト	リスト
XREF	プロシージャー、プログラム、およびデータの名前に関するソート済み相互参照リスト	リスト

コンパイルからのリスト出力は、SYSPRINT で定義されたデータ・セットに入れられ、オブジェクト出力は SYSLIN または SYSPUNCH に入れられます。進行メッセージと診断メッセージは、SYSPRINT データ・セットに入れるだけでなく、SYSTERM データ・セットに送ることができます。データベース要求モジュール (DBRM) は、DBRMLIB 内に定義されるデータ・セットです。独立したデバッグ・ファイルは、SYSDEBUG 内に定義されるデータ・セットです。

コンパイル時に作成されたリストを保管してください。デバッグまたは調整が必要になったときに、作業のテスト段階でこのリストを使用することができます。

コンパイル後、次に行うのは、コンパイラーがプログラムで見つけたエラーを修正することです。エラーが 1 つも見つからなかった場合には、プロセスの次のステップである、プログラムのリンク・エディットまたはバインドに進むことができます。 (コンパイラー・オプションを使用してオブジェクト・コードの生成を抑止した場合、オブジェクト・コードを入手するには再コンパイルする必要があります。)

関連タスク
言語環境プログラム・プログラミング・ガイド
(言語環境プログラムのもとでのリンク・エディットおよび実行の準備)

関連参照
315 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』
341 ページの『第 17 章 コンパイラー・オプション』

複数プログラムのコンパイル (バッチ・コンパイル)

コンパイラーを 1 回呼び出すだけで、一連の別個の COBOL プログラムをコンパイルすることができます。このコンパイルから作成されたオブジェクト・プログラムをリンクして、1 つのロード・モジュールまたは複数の別個のロード・モジュールにすることができます (これは NAME コンパイラー・オプションで制御されます)。

バッチ・ジョブの一部として複数のプログラムをコンパイルするときは、以下を行う必要があります。

- 1 つのロード・モジュールを作成するのか、複数のロード・モジュールを作成するのかを決定する。
- シーケンスの中の各プログラムを終了させる。
- コンパイラー・オプションを指定する (バッチ・ジョブ内のプログラムで指定されたコンパイラー・オプションの影響を認識する)。

別個のロード・モジュールを作成するには、それぞれのモジュール・セットの前に NAME コンパイラー・オプションを付けなければなりません。コンパイラーが NAME コンパイラー・オプションを検出すると、次の NAME コンパイラー・オプションが見つかるまで、そのシーケンス内の最初のプログラムおよび後続のすべてのプログラムが 1 つのロード・モジュールにリンク・エディットされます。それから、NAME オプションを使用してコンパイルされる一連のプログラムは、それぞれ 1 つの分離したロード・モジュールに入れられます。

シーケンス内の各プログラムは、END PROGRAM マーカーで終了させる必要があります (ただし、バッチの最後のプログラムは例外であり、END PROGRAM マーカーは省略可能です)。あるいは、シーケンスの中の各プログラムの前に CBL または PROCESS ステートメントを付けることができます。

プログラム (プログラム・シーケンスの最後のプログラムを除く) で END PROGRAM マーカーを省略すると、そのシーケンス内の次のプログラムが直前のプログラムにネストされます。以下のいずれかの状況では、エラーとなる可能性があります。

- この結果ネストされるプログラムの中に PROCESS ステートメントがある。
- CBL ステートメントがシーケンス番号域 (桁 1 から 6) に完全にコーディングされていない。

  CBL ステートメントがシーケンス番号域 (桁 1 から 6) に完全にコーディングされている場合は、それはソース・ステートメント行のラベルと見なされるため、その CBL ステートメントに関するエラー・メッセージは出されません。

311 ページの『例: バッチ・コンパイル』

関連タスク

312 ページの『バッチ・コンパイルでのコンパイラー・オプションの指定』

関連参照

372 ページの『NAME』

例: バッチ・コンパイル

次の例は、IGYWCL カタログ式プロシージャーを 1 回呼び出して、3 つのプログラム (PROG1、PROG2、および PROG3) のバッチ・コンパイルおよび 2 つのロード・モジュールの作成を行う方法を示しています。

次のステップが発生します。

- PROG1 と PROG2 は一緒にリンク・エディットされて、PROG2 という名前の 1 つのロード・モジュールが形成されます。このロード・モジュールの入り口点は、デフォルトによって、ロード・モジュールの最初のプログラムである PROG1 が使用されます。
- PROG3 は単独でリンク・エディットされ、PROG3 という名前のロード・モジュールが作成されます。この場合は、ロード・モジュール内の唯一のプログラムであるので、入り口点も PROG3 です。

```
//jobname  JOB  acctno,name,MSGLEVEL=1
//stepname EXEC IGYWCL
//COBOL.SYSIN    DD *
010100 IDENTIFICATION DIVISION.
010200  PROGRAM-ID PROG1.
        . . .
019000  END PROGRAM PROG1.
020100 IDENTIFICATION DIVISION.
020200  PROGRAM-ID PROG2.
        . . .
029000  END PROGRAM PROG2.
  CBL NAME
030100 IDENTIFICATION DIVISION.
030200  PROGRAM-ID PROG3.
        . . .
039000  END PROGRAM PROG3.
/*
//LKED.SYSLMOD DD DSN=&&GOSET                    (1)
/*
//P2       EXEC PGM=PROG2
//STEPLIB  DD   DSN=&&GOSET,DISP=(SHR,PASS)      (2)
. . . (3)
/*
//P3       EXEC PGM=PROG3
//STEPLIB  DD   DSN=&&GOSET,DISP=(SHR,PASS)      (2)
. . . (4)
/*
//
```

(1) LKED ステップ SYSLMOD のデータ・セット名が、一時名 &&GOSET に変更されます。メンバー名は指定しません。

(2) 一時データ・セット &&GOSET は、コンパイル済みプログラムを実行するために、ステップ P2 および P3 の STEPLIB として使用されます。言語環境プログラムのライブラリーが共用ストレージに常駐していない場合は、さらに、ライブラリー・データ・セットを STEPLIB の DD ステートメントとして追加する必要があります。

(3) PROG1 および PROG2 の実行に必要なその他の DD ステートメントと入力を追加しなければなりません。

(4) PROG3 の実行に必要なその他の DD ステートメントと入力を追加しなければなりません。

関連参照
言語環境プログラム・プログラミング・ガイド
(IBM 提供のカタログ式プロシージャー)

バッチ・コンパイルでのコンパイラー・オプションの指定

バッチ・シーケンスの各プログラムのコンパイラー・オプションは、プログラムの実行前またはコンパイラーの起動時に CBL または PROCESS ステートメントを使用して指定することができます。

CBL または PROCESS ステートメントが現行プログラムに指定されていると、コンパイラーは、最初のプログラムの前に有効なオプションと一緒に CBL または PROCESS ステートメントを解決します。現行プログラムに CBL または PROCESS ステートメントが含まれていない場合は、コンパイラーは、前のプログラムで有効であったオプションの設定を使用します。

特定のコンパイラー・オプションは、バッチ・シーケンスの中の各プログラムのコンパイラー・オプション設定の優先順位に影響を及ぼすことに注意してください。コンパイラー・オプションは、以下の優先順位 (高位から下位) で、認識されます。

1. 設置場所で固定されているインストール先デフォルト。
2. バッチ内の最初のプログラムで有効にされた BUFSIZE、LIB、OUTDD、SIZE、および SQL コンパイラー・オプションの値。
3. 現行プログラムに対する CBL または PROCESS ステートメント上のオプション (ある場合)。
4. コンパイラー呼び出しで指定されたオプション (JCL PARM または TSO CALL)。
5. 固定されていないインストール先デフォルト

バッチ・シーケンス内のいずれかのプログラムで BUF、LIB、OUTDD、SIZE、または SQL オプションが必要な場合は、バッチ・シーケンス内の最初のプログラムでそのオプションを有効にする必要があります。(BASIS、COPY、または REPLACE ステートメントを処理するとき、コンパイラーは、バッチ処理のすべてのプログラムを 1 つの入力ファイルとして処理します。)

バッチで LIB オプションを指定する場合は、バッチ・コンパイル時に NUMBER および SEQUENCE オプションを変更することはできません。コンパイラーは、LIB オプションのもとで NUMBER および SEQUENCE を処理するとき、バッチ内のすべてのプログラムを 1 つの入力ファイルとして扱います。このため、入力ファイル全体のシーケンス番号が昇順になっていなければなりません。

コンパイラーが CBL または PROCESS ステートメントの LANGUAGE オプションをエラーと診断すると、言語選択は、コンパイラーが最初の CBL または PROCESS ステートメントを検出する前に有効であった状態に戻します。バッチ・コンパイル時に有効な言語は、その環境での CBL または PROCESS ステートメントの処理規則に従います。

313 ページの『例: バッチ・コンパイルでのオプションの優先順位』
313 ページの『例: バッチ・コンパイルでの LANGUAGE オプション』

例: バッチ・コンパイルでのオプションの優先順位

次のサンプル・リストは、バッチ・コンパイルでのコンパイラー・オプションの優先順位を示しています。

```
PP 5655-S71 IBM Enterprise COBOL for z/OS  4.1.0    Date 12/30/2007. . .
Invocation parameters:
NOTERM
PROCESS(CBL) statements:
CBL CURRENCY,FLAG(I,I)
Options in effect:  All options are installation defaults unless otherwise noted:
    NOADATA
      ADV
      QUOTE
      ARITH(COMPAT)
    NOAWO
      BUFSIZE(4096)
    . . .
      CURRENCY      Process option PROGRAM 1
    . . .
      FLAG(I,I)     Process option PROGRAM 1
    . . .
    NOTERM          INVOCATION option
    . . .
End of compilation for program 1
. . .

PP 5655-S71 IBM Enterprise COBOL for z/OS  4.1.0    Date 12/30/2007. . .
PROCESS(CBL) statements:
CBL APOST
Options in effect:
    NOADATA
      ADV
    APOST           Process option PROGRAM 2
      ARITH(COMPAT)
    NOAWO
      BUFSIZE(4096)
    . . .
    NOCURRENCY      Installation default option for PROGRAM 2
    . . .
      FLAG(I)       Installation default option
    . . .
    NOTERM          INVOCATION option remains in effect
    . . .
End of compilation for program 2
```

例: バッチ・コンパイルでの LANGUAGE オプション

次の例は、バッチ環境での LANGUAGE コンパイラー・オプションの動作を示しています。デフォルトのインストール・オプションは ENGLISH (省略形は EN) で、呼び出しオプションは XX (存在しない言語) です。

```
CBL LANG(JP),FLAG(I,I),APOST,SIZE(MAX)  (1)
    IDENTIFICATION DIVISION.            (2)
    PROGRAM-ID.  COMPILE1.
    .  .  .
    END PROGRAM  COMPILE1.
CBL LANGUAGE(YY)                        (3)
CBL SIZE(2048K),LANGUAGE(JP),LANG(!!)   (4)
    IDENTIFICATION DIVISION.            (2)
    PROGRAM-ID.  COMPILE2.
    .  .  .
    END PROGRAM  COMPILE2.
    IDENTIFICATION DIVISION.
    PROGRAM-ID.  COMPILE3.
```

```
    . . .
    END PROGRAM  COMPILE3.
CBL LANGUAGE(JP),LANGUAGE(YY)          (5)
    . . .
```

(1) インストールのデフォルトは EN です。呼び出しオプションは XX (存在しない言語) でした。 EN は有効な言語です。

(2) CBL ステートメントの走査後、JP が有効な言語になります。

(3) CBL は、言語を EN にリセットします。 YY は、JP によって置き換えられるため無視されます。

(4) !! 英数字ではないため廃棄されます。

(5) CBL は、言語を EN にリセットします。 YY は JP を置き換えますが、存在しません。

プログラム COMPILE1 の場合、コンパイラーが呼び出しオプションを走査するときには、デフォルト言語 ENGLISH (EN) が有効です。XX は存在しない言語 ID なので、診断メッセージは英大/小文字混合で出されます。デフォルトの EN は、コンパイラーが CBL ステートメントを走査するときにも有効なままです。CBL ステートメントの認識されないオプション APOST は、英大/小文字混合で診断されます。これは、CBL ステートメントが処理を完了しておらず、EN が最後の有効な言語オプションであったからです。コンパイラーが CBL オプションを処理した後は、有効となる言語は日本語 (JP) です。

プログラム COMPILE2 では、最初のプログラムが使用される前に有効な言語は英語なので、コンパイラーは、大/小文字混合の CBL ステートメント・エラーを診断します。LANGUAGE オプションが複数指定された場合は、最後に指定された有効な言語だけが使用されます。この例では、最後の有効な言語は日本語 (JP) です。したがって、コンパイラーが CBL オプションの処理を終了すると、日本語が有効な言語になります。CBL および PROCESS ステートメントのオプションを日本語で診断したい場合は、COMPILE1 より前に有効な言語が日本語でなければなりません。

プログラム COMPILE3 に CBL ステートメントはありません。したがって、有効な言語である日本語 (JP) を以前のコンパイルから継承します。

コンパイラーは、COMPILE3 をコンパイルした後は、CBL ステートメントのために、有効な言語を英語 (EN) にリセットします。CBL ステートメントの言語オプションは、最後に指定された 2 文字の英数字言語 ID である YY を解決します。YY は存在しない言語であるため、有効な言語は英語のままです。

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、すべてのエラーを修正する必要はありません。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、それらのエラーを除去するために、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 つのレベルのエラーとは対照的に、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 (データ・セット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の問題) 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
『コンパイル・エラー・メッセージのリストの生成』

関連参照
『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』

コンパイル・エラー・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説明の完全な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できます。プログラムの残りの部分は省略します。

```
Identification Division.
Program-ID. ErrMsg.
```

関連参照
『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』
316 ページの『コンパイル・エラー・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ます。エラーが見つかるたびに、コンパイラーはメッセージを出します。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。

リスト内の各メッセージは、以下の情報を提供します。

- エラーの性質
- エラーを検出したコンパイラー・フェーズ
- エラーの重大度レベル

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。

プログラム (行番号で順序付けされた) の中で検出されたコンパイル・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エラーの要約は、リストの下部に表示されます。

関連タスク
314 ページの『ソース・プログラムのエラーの訂正』
315 ページの『コンパイル・エラー・メッセージのリストの生成』

関連参照
『コンパイル・エラー・メッセージの形式』
317 ページの『コンパイル・エラー・メッセージの重大度コード』
363 ページの『FLAG』

コンパイル・エラー・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 IGY*ppxxxx-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、それらは元のソース・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 このメッセージが COBOL コンパイラーから出されたものであることを識別する接頭部。

pp コンパイラーのどのフェーズまたはサブフェーズでエラーが発見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 エラー・メッセージを識別する 4 桁の数字。

l エラー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
『コンパイル・エラー・メッセージの重大度コード』
363 ページの『FLAG』

コンパイル・エラー・メッセージの重大度コード

コンパイラーが検出できるエラーは、重大度に応じて次の 5 つのカテゴリーに分けられます。

表 40. **コンパイル・エラー・メッセージの重大度コード**

メッセージのレベル	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。

第 15 章 UNIX のもとでのコンパイル

z/OS UNIX のもとでは、cob2 コマンドを使用して、Enterprise COBOL プログラムをコンパイルします。 z/OS UNIX のもとでは、z/OS のもとでコンパイルできるすべての COBOL プログラムをコンパイルすることができます。 COBOL コンパイラーによって生成されたオブジェクト・コードは、z/OS のもとで実行することができます。

コンパイル・ステップの一部として、コンパイルに必要なファイルを定義し、プログラムおよび求める出力に必要なコンパイラー・オプションおよびコンパイラー指示ステートメントを指定します。

コンパイラーの主な仕事は、COBOL プログラムを、コンピューターで処理できる言語 (オブジェクト・コード) に変換することです。さらに、コンパイラーは、ソース・ステートメント内のエラーをリストして、プログラムのデバッグおよび調整に役立つ補足情報を提供します。

関連タスク 『UNIX のもとでの環境変数の設定』 320 ページの『UNIX のもとでのコンパイラー・オプションの指定』 321 ページの『cob2 コマンドを使用したコンパイルおよびリンク』 327 ページの『スクリプトを使用したコンパイル』 329 ページの『UNIX のもとでの OO アプリケーションのコンパイル、リンク、実行』

関連参照 298 ページの『z/OS のもとでコンパイラーによって使用されるデータ・セット』 308 ページの『z/OS のもとでのコンパイラー・オプションおよびコンパイラー出力』

UNIX のもとでの環境変数の設定

*環境変数*は、文字ストリングに関連付けられる名前であり、プログラム環境のいくつかの変化する側面を定義します。環境変数を使用して、プログラム (コンパイラーを含む) が必要とする値を設定します。

コンパイラーに必要な環境変数は、export コマンドを使用して設定します。例えば、SYSLIB 変数を設定するには、シェルまたはスクリプト・ファイルから export コマンドを出してください。

```
export SYSLIB=/u/mystuff/copybooks
```

環境変数に割り当てる値には、他の環境変数または変数自体を含めることができます。これらの変数の値が適用されるのは、export コマンドを出すシェルからコンパイルする場合だけです。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。環境変数名は大文字でなければなりません。

コンパイラーが使用するために設定できる環境変数は、次のとおりです。

COBOPT
コンパイラー・オプションをブランクまたはコンマで区切って指定します。サブオプションはコンマで区切ってください。変数値の始まりまたは終わり

にあるブランクは無視されます。オプションのリストに、ブランクまたは z/OS UNIX シェルにとって意味のある文字が含まれている場合には、リストを引用符で囲んでください。以下に、その例を示します。

```
export COBOPT="TRUNC(OPT) XREF"
```

SYSLIB

COPY ステートメントで明示的なライブラリー名を指定しなかった場合は、COBOL コピーブックの検索の際に使用するディレクトリーへのパスを指定します。複数のパスはコロンで区切ってください。パスは、export コマンドの最初のパスから最後のパスへと順番に評価されます。同じ名前の複数のファイルを持つ変数を設定した場合は、そのファイルの最初に見つかったコピーが使用されます。

COPY ステートメントに明示的なライブラリー名がコード化されていない場合、コンパイラーは、以下の順序でコピーブックを検索します。

1. 現行ディレクトリー
2. -I cob2 オプションで指定されたパス
3. SYSLIB 環境変数で指定されたパス

library-name

COPY ステートメントで明示的なライブラリー名を指定する場合の、コピー元のディレクトリー・パスを指定します。この環境変数名は、プログラム内の *library-name* と同じです。それぞれのライブラリーごとに環境変数を設定しなければなりません。そうでない場合は、エラーになります。環境変数名 *library-name* は大文字でなければなりません。

text-name

テキストのコピー元のファイルの名前を指定します。この環境変数名は、プログラム内の *text-name* と同じです。環境変数名 *text-name* は大文字でなければなりません。

関連タスク

『UNIX のもとでのコンパイラー・オプションの指定』
321 ページの『cob2 コマンドを使用したコンパイルおよびリンク』
490 ページの『環境変数の設定およびアクセス』

関連参照

407 ページの『第 18 章 コンパイラー指示ステートメント』
341 ページの『第 17 章 コンパイラー・オプション』
COPY ステートメント (*Enterprise COBOL 言語解説書*)

UNIX のもとでのコンパイラー・オプションの指定

コンパイラーは、デフォルトのコンパイラー・オプションを使用してインストールされ、セットアップされます。コンパイラーのインストール時に、システム・プログラマーは、コンパイラー・オプションの設定を固定して、確実にパフォーマンスを向上させたり、ある特定の標準を維持したりすることができます。インストール先で固定されたコンパイラー・オプションはどれもオーバーライドすることはできません。

固定されていないオプションについては、以下のいずれかの方法でコンパイラー・オプションを指定して、デフォルト設定をオーバーライドすることができます。

- COBOL ソースの PROCESS または CBL ステートメントにコーディングする。
- cob2 コマンドの -q オプションを指定する。
- COBOPT 環境変数を設定する。

コンパイラーは、上記の優先順位 (高位から下位の順) に従ってオプションを認識します。この優先順位の順序は、対立するオプションまたは互いに排他的なオプションが指定された場合に有効となるオプションも決定します。cob2 コマンドを使用してコンパイルするときは、コンパイラー・オプションは、以下の優先順位 (最高位から最下位へ) の順序で認識されます。

1. オーバーライド不能として固定されているインストール先デフォルト。
2. バッチ・コンパイルの最初のプログラムで有効にされた BUFSIZE、LIB、SQL、OUTDD、および SIZE オプションの値。
3. COBOL ソース・プログラム内の PROCESS または CBL ステートメントで指定された値。
4. cob2 コマンドの -q オプション・ストリングで指定された値。
5. COBOPT 環境変数で指定された値。
6. 固定されていないインストール先デフォルト

制約事項:

- z/OS UNIX で、SQL コンパイラー・オプションを使用しないでください。

  z/OS UNIX のもとでは、独立した SQL プリコンパイラーも組み込みの SQL コプロセッサーも稼働しません。

- OPTFILE オプションは、z/OS UNIX で cob2 コマンドを使用してコンパイルした場合、無視されます。

  代わりに、COBOPT 環境変数を使用できます。これは、OPTFILE に同等の機能を提供します。

関連タスク 307 ページの『PROCESS (CBL) ステートメントによるコンパイラー・オプションの指定』 319 ページの『UNIX のもとでの環境変数の設定』 『cob2 コマンドを使用したコンパイルおよびリンク』

関連参照
344 ページの『矛盾するコンパイラー・オプション』
341 ページの『第 17 章 コンパイラー・オプション』

cob2 コマンドを使用したコンパイルおよびリンク

z/OS UNIX シェルから COBOL プログラムのコンパイルおよびリンクを行うには、cob2 コマンドを使用します。オプションおよび入力ファイル名は (オプションと名前をスペースで区切って) どのような順序で指定しても構いません。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。

複数のファイルをコンパイルする (バッチ・コンパイル) には、複数のソース・ファイル名を指定してください。

z/OS UNIX の場合、COBOL プログラムをコンパイルするには、RENT オプションが必要になります。cob2 コマンドは、COBOL コンパイラー・オプション RENT および TERM を自動的に組み込みます。

cob2 コマンドは、標準の MVS 探索順序で見つかった COBOL コンパイラーを呼び出します。 COBOL コンパイラーが LNKLST にインストールされていない場合、または複数レベルの IBM COBOL コンパイラーがシステムにインストールされている場合、STEPLIB 環境変数で、使用したいコンパイラー PDS を指定できます。例えば、次のステートメントは IGY.V4R1M0 をコンパイラー PDS として指定します。

```
export STEPLIB=IGY.V4R1M0.SIGYCOMP
```

cob2 コマンドは、リンク・ステップには z/OS UNIX シェル・コマンド c89 を暗黙的に使用します。 c89 は、リンカー (z/OS プログラム管理バインダー) へのシェル・インターフェースです。

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。

サフィックス .cbl を持つファイルだけがコンパイラーに渡されます。 cob2 は他のすべてのファイルをリンカーに渡します。

COBOL ソース・プログラム *file*.cbl のコンパイルから要求したリスト出力は、*file*.lst に書き込まれます。リンカーから要求したリスト出力は、stdout に書き込まれます。

リンカーによって、最初のメインプログラムから実行が開始されます。

関連タスク
『UNIX のもとでの DLL の作成』
330 ページの『UNIX のもとでのオブジェクト指向アプリケーションの準備』
UNIX システム・サービス・ユーザーズ・ガイド

関連参照
324 ページの『cob2 の構文およびオプション』
326 ページの『cob2 入出力ファイル』
UNIX システム・サービス・コマンド解説書

UNIX のもとでの DLL の作成

z/OS UNIX シェルから DLL を作成するには、cob2 オプション -bdll を指定する必要があります。

```
cob2 -o mydll -bdll mysub.cbl
```

cob2 -bdll を指定すると、次のようになります。

- COBOL コンパイラーは、コンパイラー・オプション DLL、EXPORTALL、および RENT (これらは DLL に必要なものです) を使用します。

- リンク・ステップは、DLL によってエクスポートされるそれぞれの名前ごとに IMPORT 制御ステートメントを含んでいる、DLL 定義サイド・ファイルを作成します。

DLL 定義サイド・ファイルの名前は、出力ファイル名に基づきます。出力名にサフィックスが付いている場合は、そのサフィックスを *x* で置き換えて、サイド・ファイル名が作られます。例えば、出力ファイル名が foo.dll であれば、サイド・ファイル名は foo.x になります。

後で DLL を呼び出すモジュールを作成するときに DLL 定義サイド・ファイルを使用するには、リンクする必要のある他の任意のオブジェクト・ファイル (*file*.o) と一緒にサイド・ファイルを指定してください。例えば、次のコマンドは、myappl.cbl をコンパイルし、DLL オプションを使用して myappl.o が DLL を参照できるようにし、リンクを行ってモジュール myappl を作成します。

```
cob2 -o myappl -qdll myappl.cbl mydll.x
```

『例: UNIX のもとでの cob2 によるコンパイルおよびリンク』

関連タスク
537 ページの『第 26 章 DLL または DLL アプリケーションの作成』
538 ページの『DLL を作成するためのプログラムのコンパイル』

関連参照
324 ページの『cob2 の構文およびオプション』
326 ページの『cob2 入出力ファイル』

例: UNIX のもとでの cob2 によるコンパイルおよびリンク

次の例では、cob2 の使用法を示します。

- 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
  ```

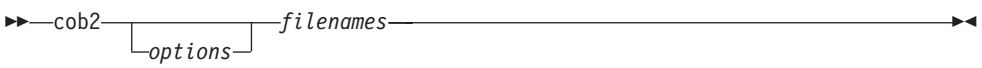
  このコマンドは、alpha.o と beta.o を作成し、その後に alpha.o、beta.o、および COBOL ライブラリーをリンクします。リンク・ステップが成功すると、gamma という名前の実行可能プログラムが作成されます。

- 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
  ```

cob2 の構文およびオプション

cob2 コマンドの構文



cob2 コマンドでは以下のオプションを使用することができます。 (cob2 を大文字で使用しないでください。)

-b*xxx* ストリング *xxx* をパラメーターとしてリンカーに渡します。*xxx* は、コンマで区切られた、*name=value* 形式のリンカー・オプションのリストです。名前と値の両方を (以下に示す特別な場合を除いて) 略さずにフルスペルで書く必要があります。名前と値は、大/小文字が区別されません。-b と *xxx* の間にスペースを入れてはなりません。

オプションの値を指定しなかった場合は、デフォルト値として YES が使用されます。ただし、以下のオプションは例外で、これらは指定されたデフォルト値を持っています。

- LIST=NOIMPORT
- ALIASES=ALL
- COMPAT=CURRENT
- DYNAM=DLL

xxx に対する 1 つの特殊値は dll で、実行可能モジュールが DLL であることを指定します。このストリングはリンカーに渡されません。

-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 cob2 動作を制御します。戻りコードが *n* 以下であれば、cob2 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。コンパイラーから戻された戻りコードが *n* より大きい場合は、cob2 は同じ戻りコードで終了します。リンク・ステップの cob2 によって c89 コマンドが暗黙的に呼び出された場合は、c89 コマンドからの終了値が、cob2 コマンドからの戻りコードとして使用されます。

デフォルトは -comprc_ok=4 です。

-e *xxx* モジュールの入り口点として使用するプログラムの名前を指定します。-e を指定しなかった場合、デフォルト入り口点は、cob2 コマンド呼び出しでファイル名として指定された最初のプログラム (*file*.cbl) またはオブジェクト・ファイル (*file*.o) になります。

-g デバッグに備えてプログラムを準備します。サブオプションなしで TEST オプションを指定するのと等価です。

-I*xxx* *library-name* が指定されていない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。

複数のパスを指定するには、複数の -I オプションを使用するか、単一の -I オプション値の中に複数のパス名をコロンで区切って指定してください。

COPY ステートメントに明示的なライブラリー名がコード化されていない場合、コンパイラーは、以下の順序でコピーブックを検索します。

1. 現行ディレクトリー
2. -I cob2 オプションで指定されたパス
3. SYSLIB 環境変数で指定されたパス

COPY ステートメントを使用する場合には、LIB コンパイラー・オプションを必ず有効にしなければなりません。

-L *xxx* -l オペランドで指定されたアーカイブ・ライブラリーを探索するために使用するディレクトリー・パスを指定します。

-l *xxx* リンカーのためのアーカイブ・ライブラリーの名前を指定します。cob2 コマンドは、lib*xxx*.a という名前を、-L オプションで指定されたディレクトリーで探索し、そのあと通常の探索順序で探索します。 (このオプションは小文字の「エル」であって、大文字の「アイ」ではありません。)

-o *xxx* オブジェクト・モジュール *xxx* を指定します。-o オプションが使用されない場合、オブジェクト・モジュールの名前は a.out になります。

-q*xxx* *xxx* をコンパイラーに渡します (*xxx* は、ブランクまたはコンマで区切られたコンパイラー・オプションのリストです)。

括弧がオプションまたはサブオプションの一部である場合、またはブランクを使用してオプションを区切る場合は、*xxx* を引用符で囲んでください。-q と *xxx* の間にスペースを入れないでください。

-v コンパイル・ステップおよびリンク・ステップで cob2 によって出される生成済みコマンド (渡されるオプションを含む) を表示し、それらを実行します。以下に、出力例を示します。

```
cob2 -v -o mini -qssrange mini.cbl
compiler: ATTCRCTL PARM=RENT,TERM,SSRANGE /u/userid/cobol/mini.cbl
PP 5655-S71 IBM Enterprise COBOL for z/OS  4.1.0  in progress ...
End of compilation 1,  program mini,  no statements flagged.
linker: /bin/c89 -o mini -e // mini.o
```

-# 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。

関連タスク

321 ページの『cob2 コマンドを使用したコンパイルおよびリンク』
322 ページの『UNIX のもとでの DLL の作成』
319 ページの『UNIX のもとでの環境変数の設定』

cob2 入出力ファイル

cob2 コマンドを使用するときは、入力ファイル名として以下のファイルを指定することができます。

表 41. **cob2 コマンドへの入力ファイル**

ファイル名	説明	コメント
file.cbl	コンパイルおよびリンクされる COBOL ソース・ファイル	cob2 オプション -c を指定した場合は、リンクされません。
file.a	アーカイブ・ファイル	リンク・エディット・フェーズで使用できるように、ar コマンドによって作成されます。
file.o	リンク・エディットされるオブジェクト・ファイル	COBOL コンパイラー、C/C++ コンパイラー、またはアセンブラーによって作成することができます。
file.x	DLL 定義サイド・ファイル	ダイナミック・リンク・ライブラリー (DLL) を参照するアプリケーションのリンク・エディット・フェーズで使用されます。

cob2 コマンドを使用すると、以下のファイルが現行ディレクトリーに作成されます。

表 42. **cob2 コマンドからの出力ファイル**

ファイル名	説明	コメント
ファイル	実行可能モジュールまたは DLL	cob2 オプション -o *file* を指定した場合に、リンカーによって作成されます。
a.out	実行可能モジュールまたは DLL	cob2 オプション -o を指定しなかった場合に、リンカーによって作成されます。
file.adt	入力 COBOL ソース・プログラム *file*.cbl に対応する関連データ (ADATA) ファイル	コンパイラー・オプション ADATA を指定した場合に、コンパイラーによって作成されます。
file.dbg	入力 COBOL ソース・プログラム *file*.cbl に対応する、デバッグ・ツール用の記号情報テーブル	コンパイラー・オプション TEST(. . .,SEP,. . .) を指定した場合に、コンパイラーによって作成されます。
file.dek	ライブラリー処理からの拡張 COBOL ソース出力	コンパイラー・オプション MDECK を指定した場合に、コンパイラーによって作成されます。
file.lst	入力 COBOL ソース・プログラム *file*.cbl に対応するリスト・ファイル	コンパイラーによって作成されます。
file.o	入力 COBOL ソース・プログラム *file*.cbl に対応するオブジェクト・ファイル	コンパイラーによって作成されます。
file.x	DLL 定義サイド・ファイル	*file*.dll という名前の DLL を作成する場合に、cob2 リンク・フェーズで作成されます。

表 42. **cob2 コマンドからの出力ファイル** *(続き)*

ファイル名	説明	コメント
class.java	Java クラス定義 (ソース)	クラス定義をコンパイルしたときに作成されます。

関連タスク

321 ページの『cob2 コマンドを使用したコンパイルおよびリンク』

関連参照

345 ページの『ADATA』
371 ページの『MDECK』
392 ページの『TEST』
UNIX システム・サービス・コマンド解説書

スクリプトを使用したコンパイル

シェル・スクリプトを使用して cob2 タスクを自動化するためには、無効なストリングをシェルが cob2 に渡さないようにするため、オプション構文を慎重にコーディングしてください。

次のようにスクリプトでオプション・ストリングをコーディングします。

- コンパイラー・サブオプションを指定するには、左括弧および右括弧ではなく、それぞれ、等号とコロンを使用してください。例えば、-qOPT(FULL),XREF の代わりに、コード -qOPT=FULL:,XREF と指定します。
- コンパイラー・オプションのサブオプションを区切るのに単一引用符が必要な場合は、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・ストリングの中でブランクを使用しないでください。

第 16 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行

オブジェクト指向 (OO) アプリケーションのコンパイル、リンク、および実行は、z/OS UNIX 環境で行うことが推奨されます。関連タスクで説明されている制限はありますが、標準のバッチ JCL または TSO/E コマンドを使用して、OO COBOL アプリケーションをコンパイル、リンク、または実行することも可能です。

関連タスク 『UNIX のもとでの OO アプリケーションのコンパイル、リンク、実行』 334 ページの『JCL または TSO/E でのオブジェクト指向アプリケーションのコンパイル、リンク、および実行』 339 ページの『IBM SDK for z/OS、Java 2 Technology Edition の使用』

UNIX のもとでの OO アプリケーションのコンパイル、リンク、実行

z/OS UNIX 環境で、オブジェクト指向アプリケーションのコンパイル、リンク、および実行を行うとき、アプリケーション・コンポーネントは、HFS に存在します。これらは、z/OS UNIX シェル・コマンドを使用してコンパイルおよびリンクし、シェル・コマンド・プロンプトで、または JCL または TSO/E から BPXBATCH ユーティリティーを使用して、実行します。

関連タスク
『UNIX のもとでのオブジェクト指向アプリケーションのコンパイル』
330 ページの『UNIX のもとでのオブジェクト指向アプリケーションの準備』
332 ページの『UNIX のもとでのオブジェクト指向アプリケーションの実行』

UNIX のもとでのオブジェクト指向アプリケーションのコンパイル

z/OS UNIX シェルでオブジェクト指向アプリケーションをコンパイルする場合、`cob2` コマンドを使用して、 COBOL クライアント・プログラムおよびクラス定義をコンパイルし、`javac` コマンドを使用して、 Java クラス定義をコンパイルしてプロシージャーの *バイトコード* (サフィックス .class) を生成します。

`INVOKE` ステートメントやクラス定義などの OO 構文を含む COBOL ソース・コードや、Java のサービスを使用する COBOL ソース・コードをコンパイルするには、`RENT`、`DLL`、`THREAD`、および `DBCS` コンパイラー・オプションを使用する必要があります。 (`RENT` および `DBCS` オプションはデフォルトです。)

クラス定義を含む COBOL ソース・ファイルは、他のクラスまたはプログラム定義を含むことはできません。

COBOL クラス定義をコンパイルすると、2 つの出力ファイルが生成されます。

- クラス定義のオブジェクト・ファイル (.o)。
- COBOL クラス定義に対応するクラス定義を含む Java ソース・プログラム (.java)。この生成済みの Java クラス定義は、決して編集しないでください。

COBOL クラス定義を変更する場合は、更新された COBOL クラス定義を再コンパイルして、オブジェクト・ファイルと Java クラス定義の両方を再生成しなければなりません。

COPY ステートメントを使用して COBOL クライアント・プログラムまたはクラス定義にファイル JNI.cpy を組み込む場合は、COBOL インストール・ディレクトリー (通常 /usr/lpp/cobol/include) の include サブディレクトリーをコピーブックの検索順序に指定します。 cob2 コマンドの -I オプションを使用するか、SYSLIB 環境変数を設定することによって、include サブディレクトリーを指定することができます。

関連タスク 319 ページの『第 15 章 UNIX のもとでのコンパイル』
『UNIX のもとでのオブジェクト指向アプリケーションの準備』
332 ページの『UNIX のもとでのオブジェクト指向アプリケーションの実行』
490 ページの『環境変数の設定およびアクセス』
663 ページの『JNI サービスへのアクセス』

関連参照 324 ページの『cob2 の構文およびオプション』
357 ページの『DBCS』
359 ページの『DLL』
383 ページの『RENT』
396 ページの『THREAD』

UNIX のもとでのオブジェクト指向アプリケーションの準備

オブジェクト指向 COBOL アプリケーションをリンクするには、cob2 コマンドを使用します。

オブジェクト指向 COBOL クライアント・プログラムを実行できるよう準備するには、オブジェクト・ファイルと次の 2 つの DLL サイド・ファイルをリンクして、実行可能モジュールを作成します。

- libjvm.x。IBM Java 2 Software Development Kit で提供されています。
- igzcjava.x。cobol ディレクトリーの lib サブディレクトリーにおいて、HFS で提供されています。通常、完全なパスは /usr/lpp/cobol/lib/igzcjava.x です。この DLL サイド・ファイルは、SCEELIB PDS (言語環境プログラムの一部) のメンバー IGZCJAVA としても使用可能です。

COBOL クラス定義を実行できるように準備する。

1. 上記 2 つの DLL サイド・ファイルを使用して、オブジェクト・ファイルをリンクし、実行可能 DLL モジュールを作成します。

   結果として生成される DLL モジュールの名前は、lib*Classname*.so にする必要があります。ここで、*Classname* は外部クラス名です。クラスがパッケージの一部であり、外部クラス名にピリオド (.) が使用されている場合は、DLL モジュール名ではピリオドを下線に変更する必要があります。例えば、Account クラスが com.acme パッケージの一部である場合、外部クラス名 (クラスの REPOSITORY 段落記入項目に定義されている) は com.acme.Account であり、このクラスの DLL モジュール名は libcom_acme_Account.so でなければなりません。

2. 生成された Java ソースを Java コンパイラーでコンパイルして、クラス・ファイル (.class) を生成します。

Classname に対応するクラス定義が格納されている COBOL ソース・ファイル *Classname*.cbl の場合は、次のコマンドを使用して、アプリケーションのコンポーネントのコンパイルとリンクを行います。

表 43. **クラス定義のコンパイルおよびリンクのコマンド**

コマンド	入力	出力
`cob2 -c -qdll,thread Classname.cbl`	*Classname*.cbl	*Classname*.o, *Classname*.java
`cob2 -bdll -o libClassname.so Classname.o /usr/lpp/java/IBM/J1.3/bin/classic/libjvm.x /usr/lpp/cobol/lib/igzcjava.x`	*Classname*.o	lib*Classname*.so
`javac Classname.java`	*Classname*.java	*Classname*.class

`cob2` および `javac` コマンドが正常に発行されると、プログラムの実行可能コンポーネントである、実行可能 DLL モジュール lib*Classname*.so およびクラス・ファイル *Classname*.class が生成されます。これらのコマンドによって生成されるファイルは、すべて現行作業ディレクトリーに置かれます。

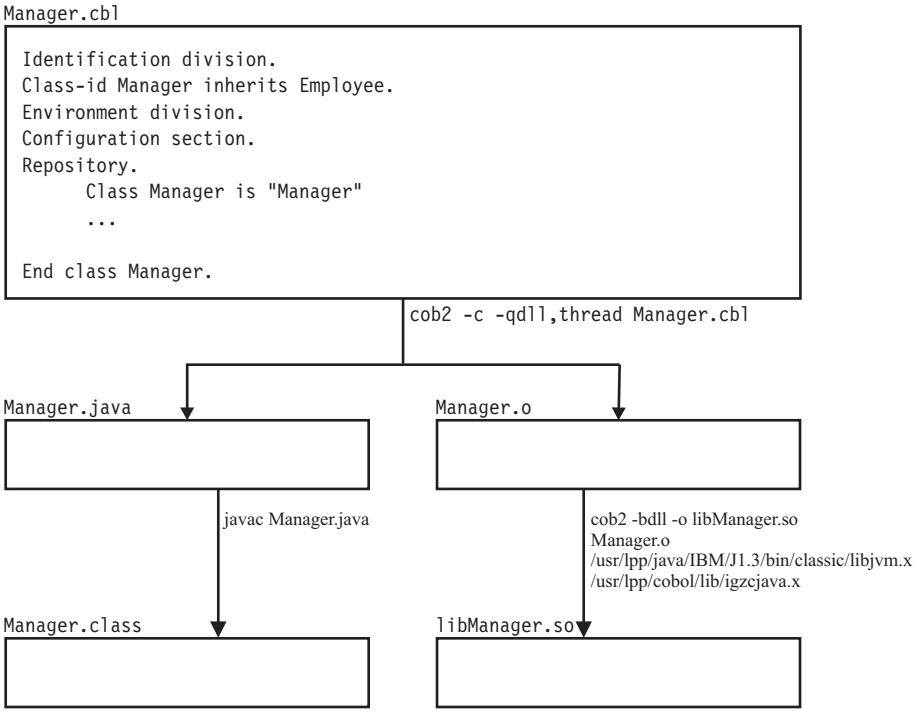
『例: z/OS UNIX のもとでの COBOL クラス定義のコンパイルとリンク』

関連タスク 319 ページの『第 15 章 UNIX のもとでのコンパイル』
619 ページの『クラス定義用の REPOSITORY 段落』

関連参照 324 ページの『cob2 の構文およびオプション』

例: z/OS UNIX のもとでの COBOL クラス定義のコンパイルとリンク

次の例は、z/OS UNIX コマンド・シェルを使用して Manager.cbl という COBOL クラス定義をコンパイルおよびリンクするときに使用するコマンドと生成されるファイルを示したものです。



クラス・ファイル Manager.class と DLL モジュール libManager.so は、アプリケーションの実行可能コンポーネントであり、現行作業ディレクトリーに生成されます。

UNIX のもとでのオブジェクト指向アプリケーションの実行

z/OS UNIX アプリケーションとしてオブジェクト指向 COBOL アプリケーションを実行することを推奨します。 Java プログラムまたは COBOL クラスの main ファクトリー・メソッドでアプリケーションが始まる場合は、そうしなければなりません。

COBOL クラスのための DLL が置かれているディレクトリーを LIBPATH 環境変数で指定します。また、それらの COBOL クラスと関連付けられた Java クラス・ファイルが置かれているディレクトリー・パスを次のように CLASSPATH 環境変数で指定します。

- パッケージの一部ではないクラスの場合は、.class ファイルが収められているディレクトリーでクラス・パスを終了します。
- パッケージの一部であるクラスの場合は、「root」パッケージ (完全なパッケージ名における最初のパッケージ) が収められているディレクトリーでクラス・パスを終了します。
- .class ファイルを含む .jar ファイルの場合は、.jar ファイルの名前でクラス・パスを終了します。

複数のパス・エントリーはコロンで区切ります。

関連タスク 333 ページの『main メソッドで始まるオブジェクト指向アプリケーションの実行』 333 ページの『COBOL プログラムで始まるオブジェクト指向アプリケーションの実行』 334 ページの『J2EE COBOL クライアントの実行』 489 ページの

ジの『第 23 章 UNIX のもとでの COBOL プログラムの実行』 490 ページの『環境変数の設定およびアクセス』 613 ページの『第 30 章 オブジェクト指向プログラムの作成』 659 ページの『OO アプリケーションの構造化』

main メソッドで始まるオブジェクト指向アプリケーションの実行

COBOL と Java の混合アプリケーションの最初のルーチンが、Java クラスの main メソッドまたは COBOL クラスの main ファクトリー・メソッドである場合は、java コマンドを使用し、main メソッドを含むクラスの名前を指定して、アプリケーションを実行します。

java コマンドは、Java 仮想マシン (JVM) を初期化します。 JVM の初期化をカスタマイズするには、以下の例で示すように、java コマンドのオプションを指定します。

表 44. **JVM をカスタマイズするための Java コマンド・オプション**

目的	オプション
システム・プロパティーを指定する	-D*name=value*
ガーベッジ・コレクションについての詳しいメッセージを JVM が生成するよう要求する	-verbose:gc
クラス・ロードについての詳しいメッセージを JVM が生成するよう要求する	-verbose:class
ネイティブ・メソッドおよび他の Java ネイティブ・インターフェース・アクティビティーについての詳しいメッセージを JVM が生成するよう要求する	-verbose:jni
Java の初期ヒープ・サイズを *value* バイトに設定する	-Xms*value*
Java の最大ヒープ・サイズを *value* バイトに設定する	-Xmx*value*

JVM がサポートするオプションの詳細については、java -h コマンドの出力または関連参照を参照してください。

関連参照

Persistent Reusable Java Virtual Machine User's Guide
WebSphere for z/OS: Applications (Java Naming and Directory Interface (JNDI))

COBOL プログラムで始まるオブジェクト指向アプリケーションの実行

COBOL と Java の混合アプリケーションの先頭のルーチンが COBOL プログラムである場合は、コマンド・プロンプトでプログラム名を指定してアプリケーションを実行します。COBOL プログラムのプロセスで JVM がまだ実行されていない場合は、COBOL のランタイムが JVM を自動的に初期化します。

JVM の初期化をカスタマイズするには、COBJVMINITOPTIONS 環境変数を設定してオプションを指定します。オプションを区切るには、ブランクを使用します。以下に例を示します。

```
export COBJVMINITOPTIONS="-Xms10000000 -Xmx20000000 -verbose:gc"
```

関連タスク
339 ページの『IBM SDK for z/OS、Java 2 Technology Edition の使用』
489 ページの『第 23 章 UNIX のもとでの COBOL プログラムの実行』
490 ページの『環境変数の設定およびアクセス』

関連参照
Persistent Reusable Java Virtual Machine User's Guide
WebSphere for z/OS: Applications (Java Naming and Directory Interface (JNDI))

J2EE COBOL クライアントの実行:

COBOL プログラムでオブジェクト指向構文を使用すると、Java 2 Platform, Enterprise Edition (J2EE) クライアントをインプリメントできます。例えば、WebSphere® for z/OS 環境で稼働する Enterprise Bean 上でメソッドを呼び出すことができます。

COBOL J2EE クライアントを実行する前に、Java システム・プロパティー java.naming.factory.initial を設定し、WebSphere のネーム・サービスにアクセスする必要があります。以下に例を示します。

```
export COBJVMINITOPTIONS
="-Djava.naming.factory.initial=com.ibm.websphere.naming.WsnInitialContextFactory"
```

676 ページの『例: COBOL で書かれた J2EE クライアント』

JCL または TSO/E でのオブジェクト指向アプリケーションのコンパイル、リンク、および実行

オブジェクト指向構文を使用するアプリケーションのコンパイル、リンク、および実行は、z/OS UNIX 環境で行うことが推奨されます。

ただし、限られた環境では、標準のバッチ JCL または TSO/E コマンドを使用して、オブジェクト指向アプリケーションをコンパイル、リンク、または実行することも可能です。そのためには、関連タスクにあるガイドラインに従う必要があります。例えば、COBOL のメインプログラムとサブプログラムで構成され、次のような機能を持つアプリケーションの場合は、この方法に従う必要があります。

- すべて Java でインプリメントされたオブジェクトにアクセスする。
- WebSphere サーバーで稼働する Enterprise Bean にアクセスする。

関連タスク 『JCL または TSO/E でのオブジェクト指向アプリケーションのコンパイル』 336 ページの『JCL または TSO/E でのオブジェクト指向アプリケーションの準備と実行』 329 ページの『UNIX のもとでの OO アプリケーションのコンパイル、リンク、実行』

JCL または TSO/E でのオブジェクト指向アプリケーションのコンパイル

バッチ JCL または TSO/E を使用してオブジェクト指向 COBOL プログラムまたはクラス定義をコンパイルした場合は、通常、生成されるオブジェクト・ファイルは、DD 名が SYSLIN または SYSPUNCH であるデータ・セットに書き込まれます。

コンパイラー・オプション RENT、DLL、THREAD、および DBCS を使用する必要があります。RENT と DBCS はデフォルトです。

COBOL プログラムまたはクラス定義が JNI 環境構造を使用して JNI の呼び出し可能サービスにアクセスする場合は、HFS から JNI という名前の PDS メンバーまたは PDSE メンバーに JNI.cpy ファイルをコピーし、SYSLIB DD ステートメントでそのライブラリーを指定して、COBOL ソースでは COPY JNI の形式で COPY ステートメントを使用します。

クラス定義を含む COBOL ソース・ファイルは、他のクラスまたはプログラム定義を含むことはできません。

COBOL クラス定義をコンパイルすると、COBOL クラス定義に対応するクラス定義を含む Java ソース・プログラムが、オブジェクト・ファイルに加えて生成されます。生成される Java ソース・ファイルを HFS 中のファイルに書き込むには、SYSJAVA DD 名を使用します。以下に例を示します。

```
//SYSJAVA   DD PATH='/u/userid/java/Classname.java',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=SIRWXU,
// FILEDATA=TEXT
```

この生成済みの Java クラス定義は、決して編集しないでください。COBOL クラス定義を変更する場合は、更新された COBOL クラス定義を再コンパイルして、オブジェクト・ファイルと Java クラス定義の両方を再生成しなければなりません。

Java クラス定義をコンパイルするには、z/OS UNIX シェルのコマンド・プロンプトから javac コマンドを使用するか、または BPXBATCH ユーティリティーを使用します。

337 ページの『例: JCL によるオブジェクト指向アプリケーションのコンパイル、リンク、実行』

関連タスク 282 ページの『JCL を使用したコンパイル』 294 ページの『TSO のもとでのコンパイル』 303 ページの『ソース・ライブラリーの指定 (SYSLIB)』 305 ページの『Java ソース出力ファイル (SYSJAVA) の作成』 663 ページの『JNI サービスへのアクセス』 329 ページの『UNIX のもとでのオブジェクト指向アプリケーションのコンパイル』 330 ページの『UNIX のもとでのオブジェクト指向アプリケーションの準備』

関連参照
357 ページの『DBCS』
359 ページの『DLL』
383 ページの『RENT』
396 ページの『THREAD』
803 ページの『付録 F. JNI.cpy』
UNIX システム・サービス・ユーザーズ・ガイド (BPXBATCH ユーティリティー)

JCL または TSO/E でのオブジェクト指向アプリケーションの準備と実行

オブジェクト指向アプリケーションは、z/OS z/OS UNIX 環境で実行することを推奨します。バッチ JCL または TSO/E からオブジェクト指向アプリケーションを実行するには、BPXBATCH ユーティリティーを使用する必要があります。

ただし、限られた環境においては、標準バッチ JCL (EXEC PGM=*COBPROG*) または TSO/E の CALL コマンドを使用して、オブジェクト指向アプリケーションを実行できます。そのためには、アプリケーションを準備する際に、以下の要件に従う必要があります。

- COBOL プログラムで開始するようにアプリケーションを構成します。 (アプリケーションが、Java プログラムまたは COBOL クラスの main ファクトリー・メソッドで始まる場合は、z/OS UNIX でアプリケーションを実行し、アプリケーション・コンポーネントを HFS に格納する必要があります。)
- **リンク・エディットの考慮事項:** COBOL プログラムのロード・モジュールを PDSE にリンクします。オブジェクト指向の構文を含んだ COBOL プログラムは、AMODE 31 を使用してリンク・エディットする必要があります。
- アプリケーションが使用する COBOL または Java のクラスに関連付けられたクラス・ファイルおよび DLL は、HFS に存在していなければなりません。UNIX のもとでのオブジェクト指向アプリケーションの準備に関する関連タスクで説明されているように、クラス・ファイルと DLL の名前を設定する必要があります。
- メインプログラムにオブジェクト・デックをバインドするときは、DLL サイド・ファイルの libjvm.x と igzcjava.x に対して、INCLUDE 制御ステートメントを指定します。以下に、その例を示します。

  ```
  INCLUDE '/usr/lpp/java/IBM/J1.3/bin/classic/libjvm.x'
  INCLUDE '/usr/lpp/cobol/lib/igzcjava.x'
  ```

- Java に必要な環境変数設定を含むファイルを作成します。例えば、ファイル /u/*userid*/javaenv は、PATH、LIBPATH、および CLASSPATH 環境変数を設定するために以下の 3 行を含んでいることがあります (LIBPATH 設定は、文書の長さに制限があるため、2 つの行の下に示されていますが、内部ブランクのない中断されていない 1 つの行に設定を指定する必要があります。)

  ```
  PATH=/bin:/usr/lpp/java/IBM/J1.3/bin
  LIBPATH=/lib:/usr/lib:/usr/lpp/java/IBM/J1.3/bin:
        /usr/lpp/java/IBM/J1.3/bin/classic:/u/userid/applications
  CLASSPATH=/u/userid/applications
  ```

  アプリケーションで使用する JVM の初期設定をカスタマイズするには、同じファイルで COBJVMINITOPTIONS 環境変数を設定します。例えば、WebSphere サーバーで動作する Enterprise Bean にアクセスするには、Java システム・プロパティー java.naming.factory.initial を設定する必要があります。詳細については、UNIX のもとでのオブジェクト指向アプリケーションの実行に関する関連タスクを参照してください。

COBOL プログラムで開始するオブジェクト指向アプリケーションを、標準バッチ JCL または TSO/E の CALL コマンドを使用して実行するときは、以下の指針に従ってください。

- _CEE_ENVFILE 環境変数を使用して、Java で必要な環境変数設定を含むファイルの場所を指定します。_CEE_ENVFILE を設定するには、ENVAR ランタイム・オプションを使用します。
- POSIX(ON) ランタイム・オプションを指定します。
- DD ステートメントを使用して、Java の標準入力、出力、およびエラーのストリームに対する HFS 中のファイルを指定します。
  - c=System.in.read(); などのステートメントからの入力に対しては JAVAIN DD を使用します。
  - System.out.println(*string*); などのステートメントからの出力に対しては JAVAOUT DD を使用します。
  - System.err.println(*string*); などのステートメントからの出力には、JAVAERR DD を使用します。
- STEPLIB DD ステートメントなどを使用して、SCEERUN2 と SCEERUN のロード・ライブラリーをシステム・ライブラリーの探索順序で使用できるようにします。

『例: JCL によるオブジェクト指向アプリケーションのコンパイル、リンク、実行』

関連タスク
330 ページの『UNIX のもとでのオブジェクト指向アプリケーションの準備』
332 ページの『UNIX のもとでのオブジェクト指向アプリケーションの実行』
659 ページの『OO アプリケーションの構造化』
UNIX システム・サービス・ユーザーズ・ガイド
(BPXBATCH ユーティリティー)
言語環境プログラム・プログラミング・ガイド
(バッチでのアプリケーションの実行)

関連参照
XL C/C++ プログラミング・ガイド (_CEE_ENVFILE)
言語環境プログラム・プログラミング・リファレンス (ENVAR)

例: JCL によるオブジェクト指向アプリケーションのコンパイル、リンク、実行

この例は、Java メソッドを呼び出す COBOL クライアントの、コンパイル、リンク、および実行に使用できる JCL を示しています。

この例は次のものを示しています。

- オブジェクト指向 COBOL プログラムをコンパイル、リンク、および実行するための JCL である TSTHELLO。
- COBOL プログラムが起動するメソッドを含んでいる Java クラス定義 HelloJ。
- Java が必要とする環境変数の設定値を含んでいる HFS ファイル ENV。

プログラム TSTHELLO に対する JCL

```
//TSTHELLO JOB ,
//  TIME=(1),MSGLEVEL=(1,1),MSGCLASS=H,CLASS=A,REGION=100M,
//  NOTIFY=&SYSUID,USER=&SYSUID
```

```
//*
// SET COBPRFX='IGY.V4R1M0'
// SET LIBPRFX='CEE'
//*
//COMPILE EXEC PGM=IGYCRCTL,
// PARM='SIZE(5000K)'
//SYSLIN   DD DSNAME=&&OBJECT(TSTHELLO),UNIT=VIO,DISP=(NEW,PASS),
//            SPACE=(CYL,(1,1,1))
//SYSPRINT DD SYSOUT=*
//STEPLIB  DD DSN=&COBPRFX..SIGYCOMP,DISP=SHR
//         DD DSN=&LIBPRFX..SCEERUN,DISP=SHR
//SYSUT1   DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT2   DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT3   DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT4   DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT5   DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT6   DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT7   DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSIN    DD *
       cbl dll,thread
       Identification division.
       Program-id. "TSTHELLO" recursive.
       Environment division.
       Configuration section.
       Repository.
           Class HelloJ is "HelloJ".
       Data Division.
       Procedure division.
           Display "COBOL program TSTHELLO entered"
           Invoke HelloJ "sayHello"
           Display "Returned from java sayHello to TSTHELLO"
           Goback.
       End program "TSTHELLO".
/*
//LKED EXEC PGM=IEWL,PARM='RENT,LIST,LET,DYNAM(DLL),CASE(MIXED)'
//SYSLIB   DD DSN=&LIBPRFX..SCEELKED,DISP=SHR
//         DD DSN=&LIBPRFX..SCEELKEX,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSTERM  DD SYSOUT=*
//SYSLMOD  DD DSN=&&GOSET(TSTHELLO),DISP=(MOD,PASS),UNIT=VIO,
//            SPACE=(CYL,(1,1,1)),DSNTYPE=LIBRARY
//SYSDEFSD DD DUMMY
//OBJMOD   DD DSN=&&OBJECT,DISP=(OLD,DELETE)
//SYSLIN   DD *
  INCLUDE OBJMOD(TSTHELLO)
  INCLUDE '/usr/lpp/java/IBM/J1.3/bin/classic/libjvm.x'
  INCLUDE '/usr/lpp/cobol/lib/igzcjava.x'
/*
//GO EXEC PGM=TSTHELLO,COND=(4,LT,LKED),
//         PARM='/ENVAR("_CEE_ENVFILE=/u/userid/ootest/tsthello/ENV")
//             POSIX(ON)'
//STEPLIB  DD DSN=*.LKED.SYSLMOD,DISP=SHR
//         DD DSN=&LIBPRFX..SCEERUN2,DISP=SHR
//         DD DSN=&LIBPRFX..SCEERUN,DISP=SHR
//SYSOUT   DD SYSOUT=*
//CEEDUMP  DD SYSOUT=*
//SYSUDUMP DD DUMMY
//JAVAOUT  DD PATH='/u/userid/ootest/tsthello/javaout',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=(SIRUSR,SIWUSR,SIRGRP)
```

クラス HelloJ の定義

```
class HelloJ {
   public static void sayHello() {
     System.out.println("Hello World, from Java!");
   }
}
```

HelloJ.java は、javac コマンドを使用してコンパイルされます。結果として生成される .class ファイルは、HFS ディレクトリー u/*userid*/ootest/tsthello に格納されます。このディレクトリーは、環境変数設定ファイルの CLASSPATH 環境変数で指定されています。

環境変数設定ファイル ENV

```
PATH=/bin:/usr/lpp/java/IBM/J1.3/bin:.
LIBPATH=/lib:/usr/lib:/usr/lpp/java/IBM/J1.3/bin:
      /usr/lpp/java/IBM/J1.3/bin/classic:/u/userid/ootest/tsthello
CLASSPATH=/u/userid/ootest/tsthello
```

(上に示した LIBPATH の設定は、本書に幅の制約があるため 2 行に渡っています。しかし、実際の設定はブランクを含まない連続した 1 行で指定する必要があります。)

環境変数設定ファイルは、u/*userid*/ootest/tsthello ディレクトリーにも存在しています。このディレクトリーは、JCL の _CEE_ENVFILE 環境変数で指定されています。

IBM SDK for z/OS、Java 2 Technology Edition の使用

IBM SDK for z/OS、Java 2 Technology Edition、V1.4 は、言語環境プログラムによって定義された XPLINK リンケージ規約に基づいています。

アプリケーションが、Java プログラムから、または COBOL クラスの main ファクトリー・メソッドから開始される場合、JVM を開始しアプリケーションを実行する java コマンドによって XPLINK 環境は自動的に開始されます。

COBOL クラスまたは Java クラスのメソッドを呼び出す COBOL プログラムからアプリケーションが開始される場合、XPLINK 環境が初期化されるように XPLINK(ON) ランタイム・オプションを指定する必要があります。ただし、XPLINK(ON) をデフォルト設定にすることはお勧めしません。XPLINK(ON) は、明確にこの設定を必要とするアプリケーションにのみ使用してください。

z/OS UNIX のもとでアプリケーションを実行する場合、次のように _CEE_RUNOPTS 環境変数を使用して XPLINK(ON) オプションを設定できます。

```
_CEE_RUNOPTS="XPLINK(ON)"
```

ただし、z/OS UNIX シェル・セッション全体で有効になるように _CEE_RUNOPTS="XPLINK(ON)" をエクスポートすることはお勧めしません。例えば、OO COBOL アプリケーションが App1Driver という名前の COBOL プログラムから開始されるとしましょう。 XPLINK オプションの影響を App1Driver アプリケーションの実行に限定する方法の 1 つは、App1Driver のコマンド行呼び出しで _CEE_RUNOPTS 変数を次のように設定することです。

```
_CEE_RUNOPTS="XPLINK(ON)" App1Driver
```

関連タスク

332 ページの『UNIX のもとでのオブジェクト指向アプリケーションの実行』
490 ページの『環境変数の設定およびアクセス』

関連参照

491 ページの『ランタイム環境変数』
言語環境プログラム・プログラミング・リファレンス (XPLINK)
XL C/C++ プログラミング・ガイド (_CEE_RUNOPTS)

第 17 章 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 45. **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	346 ページの『ARITH』	ARITH(COMPAT)	AR(C|E)
	349 ページの『CICS』	NOCICS	なし
	350 ページの『CODEPAGE』	CODEPAGE(01140)	CP(*ccsid*)
	353 ページの『CURRENCY』	NOCURRENCY	CURR|NOCURR
	357 ページの『DBCS』	DBCS	なし
	368 ページの『LIB』	LIB	なし
	373 ページの『NSYMBOL』	NSYMBOL(NATIONAL)	NS(DBCS|NAT)
	374 ページの『NUMBER』	NONUMBER	NUM|NONUM
	383 ページの『QUOTE/APOST』	QUOTE	Q|APOST
	386 ページの『SEQUENCE』	SEQUENCE	SEQ|NOSEQ
	388 ページの『SQL』	NOSQL	なし
	390 ページの『SQLCCSID』	SQLCCSID	SQLC|NOSQLC
	401 ページの『WORD』	NOWORD	WD|NOWD
	401 ページの『XMLPARSE』	XMLPARSE(XMLSS)	XP(X)|XP(C)
日付処理	355 ページの『DATEPROC』	NODATEPROC、または DATEPROC(FLAG,NOTRIG) (DATEPROC だけが指定された場合)	DP|NODP
	366 ページの『INTDATE』	INTDATE(ANSI)	なし
	404 ページの『YEARWINDOW』	YEARWINDOW(1900)	YW

表 45. **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
マップおよびリスト	367 ページの『LANGUAGE』	LANGUAGE(ENGLISH)	LANG(EN|UE|JA|JP)
	368 ページの『LINECOUNT』	LINECOUNT(60)	LC
	369 ページの『LIST』	NOLIST	なし
	370 ページの『MAP』	NOMAP	なし
	377 ページの『OFFSET』	NOOFFSET	OFF|NOOFF
	387 ページの『SOURCE』	SOURCE	S|NOS
	388 ページの『SPACE』	SPACE(1)	なし
	391 ページの『TERMINAL』	NOTERMINAL	TERM|NOTERM
	400 ページの『VBREF』	NOVBREF	なし
	402 ページの『XREF』	XREF(FULL)	X|NOX
オブジェクト・デックの生成	352 ページの『COMPILE』	NOCOMPILE(S)	C|NOC
	358 ページの『DECK』	NODECK	D|NOD
	372 ページの『NAME』	NONAME、または NAME(NOALIAS) (NAME だけが指定された場合)	なし
	376 ページの『OBJECT』	OBJECT	OBJ|NOOBJ
	380 ページの『PGMNAME』	PGMNAME(COMPAT)	PGMN(CO|LU|LM)
オブジェクト・コード制御	346 ページの『ADV』	ADV	なし
	347 ページの『AWO』	NOAWO	なし
	359 ページの『DLL』	NODLL	なし
	362 ページの『EXPORTALL』	NOEXPORTALL	EXP|NOEXP
	363 ページの『FASTSRT』	NOFASTSRT	FSRT|NOFSRT
	375 ページの『NUMPROC』	NUMPROC(NOPFD)	なし
	378 ページの『OPTIMIZE』	NOOPTIMIZE	OPT|NOOPT
	380 ページの『OUTDD』	OUTDD(SYSOUT)	OUT
	397 ページの『TRUNC』	TRUNC(STD)	なし
	405 ページの『ZWB』	ZWB	なし
仮想記憶域の使用量	348 ページの『BUFSIZE』	4096	BUF
	354 ページの『DATA』	DATA(31)	なし
	361 ページの『DYNAM』	NODYNAM	DYN|NODYN
	383 ページの『RENT』	RENT	なし
	385 ページの『RMODE』	AUTO	なし
	386 ページの『SIZE』	SIZE(MAX)	SZ
デバッグと診断	358 ページの『DIAGTRUNC』	NODIAGTRUNC	DTR|NODTR
	360 ページの『DUMP』	NODUMP	DU|NODU
	363 ページの『FLAG』	FLAG(I,I)	F|NOF
	364 ページの『FLAGSTD』	NOFLAGSTD	なし
	390 ページの『SSRANGE』	NOSSRANGE	SSR|NOSSR
	392 ページの『TEST』	NOTEST	なし

表 45. **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
その他	345 ページの『ADATA』	NOADATA	なし
	362 ページの『EXIT』	NOEXIT	EX(INX,LIBX,PRTX,ADX)
	371 ページの『MDECK』	NOMDECK	NOMD|MD|MD(C)|MD(NOC)
	377 ページの『OPTFILE』	なし	なし
	396 ページの『THREAD』	NOTHREAD	なし

インストール先デフォルト: コンパイラーがインストールされたときにセットアップされたデフォルト・オプションは、他のオプションでオーバーライドしない限り、プログラムでは有効になります。(インストール先によっては、特定のコンパイラー・オプションは、オーバーライドできないように固定としてセットアップされます。問題がある場合は、システム管理担当者に連絡してください。) 有効なデフォルトのコンパイラー・オプションを見つけるためには、オプションを指定しないで、テスト・コンパイルを実行してください。出力リストには、インストール先によって指定されたデフォルト・オプションがリストされます。

オーバーライド不可能なオプション: インストール先によっては、特定のコンパイラー・オプションは、オーバーライドできないようにセットアップされます。問題がある場合は、システム管理担当者に連絡してください。

パフォーマンスに関する考慮事項: ARITH、AWO、 DYNAM、FASTSRT、NUMPROC、OPTIMIZE、RENT、SQLCCSID、SSRANGE、TEST、THREAD、および TRUNC 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ことがあります。

関連タスク
281 ページの『第 14 章 z/OS のもとでのコンパイル』
294 ページの『TSO のもとでのコンパイル』
319 ページの『第 15 章 UNIX のもとでのコンパイル』
723 ページの『第 34 章 プログラムのチューニング』

関連参照
344 ページの『矛盾するコンパイラー・オプション』
407 ページの『第 18 章 コンパイラー指示ステートメント』
『標準 COBOL 85 に準拠するオプション設定』
735 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』

標準 COBOL 85 に準拠するオプション設定

標準 COBOL 85 に準拠するためには、コンパイラー・オプションとランタイム・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。

- ADV
- NOCICS
- NODATEPROC
- NODLL

- DYNAM
- NOEXPORTALL
- NOFASTSRT
- LIB
- NAME(ALIAS) または NAME(NOALIAS)
- NUMPROC(NOPFD) または NUMPROC(MIG)
- PGMNAME(COMPAT) または PGMNAME(LONGUPPER)
- QUOTE
- NOTHREAD
- TRUNC(STD)
- NOWORD
- ZWB

以下のランタイム・オプションが必要です。

- AIXBLD
- CBLQDA(ON)
- TRAP(ON)

関連参照
言語環境プログラム・プログラミング・リファレンス

矛盾するコンパイラー・オプション

Enterprise COBOL 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です。

矛盾するオプションが階層中の同じレベルに指定されている (例えば、PROCESS または CBL ステートメントに DECK と NODECK の両方が指定されている) 場合は、最後に指定したオプションが有効になります。

相互に排他的なコンパイラー・オプションを同じレベルに指定すると、コンパイラーはエラー・メッセージを生成し、オプションの一方を対立しない値に強制します。例えば、PROCESS ステートメントで OFFSET と LIST の両方を指定すると、指定した順序に関係なく、OFFSET が有効になり、LIST は無視されます。

しかし、高レベルの優先順位でコーディング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、JCL ステートメントでは OFFSET をコーディングし、PROCESS ステートメントでは LIST 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、LIST が有効になります。

表 46. **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 Ignored[1]	強制的にオンにされる [1]
CICS	NOLIB	LIB
	DYNAM	NODYNAM
	NORENT	RENT
DLL	DYNAM	NODYNAM
	NORENT	RENT
EXIT	DUMP	NODUMP
EXPORTALL	NODLL	DLL
	DYNAM	NODYNAM
	NORENT	RENT
MDECK	NOLIB	LIB
NSYMBOL(NATIONAL)	NODBCS	DBCS
OFFSET	LIST	NOLIST
SQL	NOLIB	LIB
TEST TEST(HOOK)	NOOBJECT	OBJECT
	OPT(STD) または OPT(FULL)	NOOPTIMIZE
THREAD	NORENT	RENT
WORD	FLAGSTD	NOFLAGSTD
1. 固定されているインストール先デフォルト・オプションと対立する場合を除きます。		

関連タスク
306 ページの『z/OS のもとでのコンパイラー・オプションの指定』
312 ページの『バッチ・コンパイルでのコンパイラー・オプションの指定』
320 ページの『UNIX のもとでのコンパイラー・オプションの指定』

関連参照
377 ページの『OPTFILE』

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

ADATA は、IBM Windows COBOL compiler を使用したリモート・コンパイルで必要です。 z/OS では、SYSADATA ファイルは、DD 名 SYSADATA に書き込まれます。 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。

ADATA は、PROCESS (CBL) ステートメントでは指定できません。指定できるのは、以下のいずれかの方法に限られます。

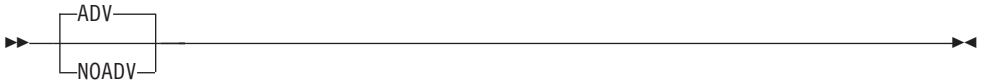
- JCL の PARM パラメーター
- cob2 コマンド・オプションとして
- インストール先デフォルト値として
- COBOPT 環境変数で

関連参照
809 ページの『付録 G. COBOL SYSADATA ファイルの内容』
319 ページの『UNIX のもとでの環境変数の設定』
324 ページの『cob2 の構文およびオプション』

ADV

ADV が意味を持つのは、ソース・コードで WRITE ... ADVANCING を使用する場合だけです。ADV が有効になっていると、コンパイラーは、印刷制御文字を入れるための 1 バイトをレコード長に追加します。

ADV オプションの構文



デフォルト: ADV

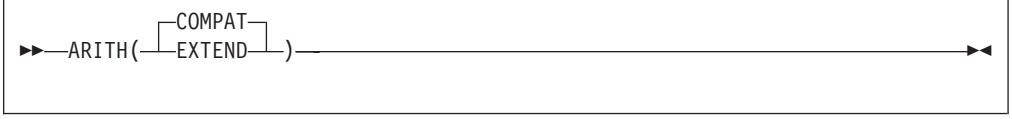
省略形: なし

既に印刷制御文字のための 1 バイトを組み込むようにレコード長を調整している場合は、NOADV を使用してください。

ARITH

ARITH は、整数にコーディングできる最大桁数、および固定小数点の中間結果で使用される桁数に影響を与えます。

ARITH オプションの構文



デフォルト: ARITH(COMPAT)

省略形: AR(C)、AR(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 および NUMVAL-C 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 および 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
753 ページの『付録 A. 中間結果および算術精度』

AWO

AWO を指定すると、暗黙の APPLY WRITE-ONLY 節が、プログラム内のこの節に適格なすべてのファイルに対して活動化されます。適格にするためには、ファイルを物理順次編成にして、ブロック化可変長レコードを指定しなければなりません。

AWO オプションの構文

デフォルト: NOAWO

省略形: なし

関連タスク
13 ページの『バッファーおよび装置スペースの最適化』

BUFSIZE

BUFSIZE は、コンパイラー作業データ・セットごとに、ある容量の主記憶域をバッファーに割り振るために使用します。通常、バッファー・サイズを大きくすると、コンパイラーのパフォーマンスが向上します。

BUFSIZE オプションの構文

デフォルト: 4096

省略形: BUF

nnnnn は 10 進数で、少なくとも 256 でなければなりません。

*nnn*K は、1KB 単位で 10 進数を指定します。1KB = 1024 バイトです。

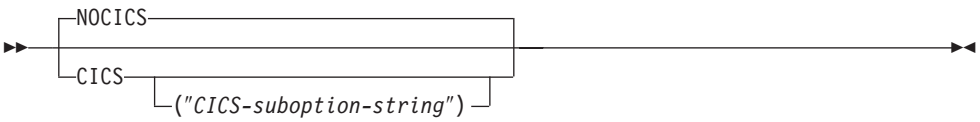
BUFSIZE と SIZE の両方を使用している場合、バッファーに割り振られる容量は、SIZE オプションによってコンパイルに使用可能にされる主記憶域の容量に含まれます。

BUFSIZE は、使用される装置のトラック容量を超えてはならず、また、データ管理サービスで許可される最大量を超えてはなりません。

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 ステートメントや EXEC DLI ステートメントが含まれていて、そのプログラムが分離型の CICS 変換プログラムで処理されていない場合は、CICS オプションを指定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。

CICS オプションを指定した場合、コンパイラーは、CICS Transaction Server バージョン 2 以上にアクセスする必要があります。

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。

引用符または単一引用符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。

長いサブオプション・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 ステートメントに置くことができます。それぞれの CICS サブオプションは、指定された順に連結されます。以下に例を示します。

```
//STEP1 EXEC IGYWC, . . .
// PARM.COBOL='CICS("string1")'
//COBOL.SYSIN DD *
       CBL CICS('string2')
       CBL CICS("string3")
       IDENTIFICATION DIVISION.
       PROGRAM-ID. DRIVER1.
       . . .
```

コンパイラーは、以下のサブオプション・ストリングを組み込みの 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 オプションの複数インスタンスが見つかった場合は、各オプションで最後に指定されたものが使用されます。コンパイラーは、連結 CICS サブオプション・ストリングの長さを 4K バイトに限定しています。

関連概念
460 ページの『組み込みの CICS 変換プログラム』

関連タスク 458 ページの『CICS オプションを使用したコンパイル』
460 ページの『CICS サブオプションの分離』
CICS Application Programming Guide (CICS 変換プログラム・オプションの指定)

関連参照 344 ページの『矛盾するコンパイラー・オプション』

CODEPAGE

CODEPAGE は、文字エンコードに依存する COBOL 操作をコンパイル時および実行時に処理するために、EBCDIC コード・ページのコード化文字セット ID (CCSID) を指定する場合に使用します。

CODEPAGE オプションの構文

デフォルト: CODEPAGE(1140)

省略形: CP(*ccsid*)

ccsid は、EBCDIC コード・ページに対する有効な CCSID を表す整数でなければなりません。

デフォルトの CCSID 1140 は CCSID 37 (EBCDIC Latin-1、USA) と同等ですが、さらにユーロ記号を含んでいます。

ccsid では次のエンコードを指定します。

- COBOL ソース・プログラム内における英数字、国別、および DBCS リテラルのエンコード
- 実行時の英数字および DBCS データ項目のコンテンツのデフォルト・エンコード
- XML エレメントおよび属性名を作成するために XML GENERATE ステートメントが処理する際の、DBCS ユーザー定義語のエンコード
- 文書の受け取りデータ項目が英数字である場合に XML GENERATE ステートメントが作成する XML 文書のデフォルト・エンコード
- 文書が XML PARSE ステートメントによって処理される際に、英数字データ項目の XML 文書で想定されるデフォルト・エンコード

CODEPAGE *ccsid* は、コード・ページに依存する操作をコンパイル時または実行時に実行し、デフォルト・コード・ページをオーバーライドする明示的な CCSID が指定されていない場合に使用されます。このような操作には次のものがあります。

- リテラル値の Unicode への変換
- 移動操作の一部としての英数字データと国別 (Unicode) データ間の変換、比較、または組み込み関数 DISPLAY-OF および NATIONAL-OF
- INVOKE ステートメントなどのオブジェクト指向言語、またはクラス定義およびメソッド定義
- XML 構文解析
- XML 生成
- 実行時の XML 生成の一部としての DBCS 名の処理
- SQLCCSID オプションが有効である場合の SQL ストリング・ホスト変数の処理
- EXEC SQL ステートメントのソース・コードの処理

ただし、COBOL ソース・プログラム内の次の項目のエンコードは、CODEPAGE コンパイラー・オプションの影響を受けません。

- USAGE NATIONAL を持つデータ項目

  このような項目は常に、UTF-16BE (ビッグ・エンディアン)、CCSID 1200 でエンコードされます。
- 基本 COBOL 文字セットの文字 (文字に関する下記の関連参照にある該当文字の表を参照)

  基本 COBOL 文字、デフォルト通貨記号 ($)、引用符 (")、および小文字ローマ字のエンコードは、EBCDIC コード・ページによって異なりますが、コンパイラーは常に、EBCDIC コード・ページ 1140 エンコードを使用してこれらの文字を解釈します。特に、デフォルト通貨記号は (CURRENCY コンパイラー・オプションまたは SPECIAL-NAMES 段落の CURRENCY SIGN 節によって変更されていない限り) 常に値が X'5B' の文字であり、引用符は常に値が X'7F' の文字になります。

たとえば次に示すように、一部の COBOL 操作では、明示的なエンコードの指定を使用することによって、CODEPAGE *ccsid* をオーバーライドすることができます。

- コード・ページを第 2 引数として指定する、DISPLAY-OF および NATIONAL-OF 組み込み関数
- WITH ENCODING 句を指定する XML PARSE ステートメント
- WITH ENCODING 句を指定する XML GENERATE ステートメント

さらに、CURRENCY コンパイラー・オプション、または SPECIAL-NAMES 段落の CURRENCY SIGN 節を使用して次のものをオーバーライドすることができます。

- ソース・プログラム内の数字編集データ項目の PICTURE 文字ストリングで使用されるデフォルト通貨記号
- 実行時に数字編集データ項目のコンテンツで使用される通貨記号

DBCS コード・ページ:

プログラムに次のいずれかの項目が含まれている場合には、 CODEPAGE オプションを使用して、*ccsid* を下表に示されている EBCDIC マルチバイト文字セット (MBCS) CCSID のいずれかに設定し、COBOL プログラムをコンパイルします。

- DBCS 文字から成るユーザー定義語
- DBCS (USAGE DISPLAY-1) データ項目
- DBCS リテラル

下表の CCSID はすべて、SBCS と DBCS コード化文字セットの組み合わせを示す混合コード・ページを指定します。また、これらは DB2 が混合データでサポートする CCSID です。

表 47. **EBCDIC マルチバイト・コード化文字セット ID**

各国語	MBCS CCSID	SBCS CCSID コンポーネント	DBCS CCSID コンポーネント
日本語 (カタカナ-漢字)	930	290	300
日本語 (カタカナ-漢字とユーロ)	1390	8482	16684
日本語 (カタカナ-漢字)	5026	290	4396
日本語 (ローマ字-漢字)	939	1027	300
日本語 (ローマ字-漢字とユーロ)	1399	5123	16684
日本語 (ローマ字-漢字)	5035	1027	4396
韓国語	933	833	834
韓国語	1364	13121	4930
中国語 (簡体字)	935	836	837
中国語 (簡体字)	1388	13124	4933
中国語 (繁体字)	937	28709	835

関連概念
474 ページの『COBOL および DB2 CCSID の決定』

関連タスク
72 ページの『通貨記号の使用』
561 ページの『第 28 章 XML 入力の処理』
593 ページの『第 29 章 XML 出力の生成』

関連参照
353 ページの『CURRENCY』
390 ページの『SQLCCSID』
文字 (*Enterprise COBOL 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: CINOC

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 (DECK、 LIST、OBJECT、OFFSET、 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
417 ページの『コーディング・エラーの検出』

関連参照
315 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: CURRINOCURR

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、QUOTE|APOST コンパイラー・オプションの設定に関係なく、単一引用符または二重引用符で区切ることができます。

関連タスク
72 ページの『通貨記号の使用』

DATA

DATA オプションは、動的データ域のストレージおよび他の動的ランタイム・ストレージが 16MB 境界より上から取得されるのか下から取得されるのかに影響を与えます。

DATA オプションの構文

デフォルト: DATA(31)

省略形: なし

再入可能プログラムの場合は、DATA コンパイラー・オプションおよび HEAP ランタイム・オプションによって、動的データ域のストレージ (WORKING-STORAGE や FD レコード域など) を 16MB 境界より下から獲得するか (DATA(24))、制限のないストレージから獲得するか (DATA(31)) が制御されます。 (DATA は LOCAL-STORAGE データの位置に影響を与えません。代わりに、STACK ランタイム・オプションが、プログラムの AMODE とともに、その位置を制御します。)

ランタイム・オプション HEAP(,,BELOW) を指定すると、 DATA コンパイラー・オプションの効果はなくなります。すべての動的データ域のストレージは、 16MB 境界より下から割り振られます。ただし、HEAP(,,ANYWHERE) が有効化されていると、動的データ域のストレージは、プログラムを DATA(24) コンパイラー・オプションを使用してコンパイルした場合には、境界より下から割り振られ、DATA(31) コンパイラー・オプションを使用してコンパイルした場合には、制限のないストレージから割り振られます。

31 ビット・アドレッシング・モードで実行されているプログラムで、24 ビット・アドレッシング・モードのプログラムにデータ引数を渡す場合は、 DATA(24) を指定します。そうすると、データは必ず呼び出されたプログラムからアドレス可能になります。

外部データおよび QSAM バッファー: DATA オプションは、ストレージおよびそのアドレス可能度に影響を与える他のコンパイラー・オプションおよびランタイム・オプションと相互作用します。詳細については、関連情報を参照してください。

関連概念
45 ページの『ストレージとそのアドレス可能度』

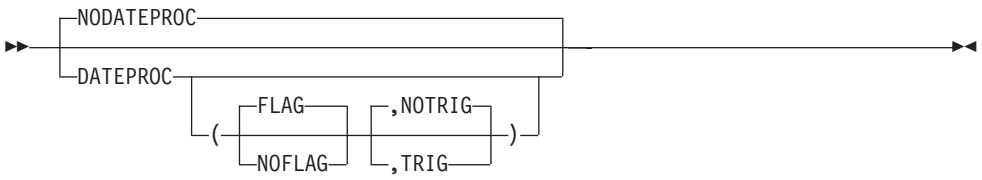
関連タスク
言語環境プログラム・プログラミング・ガイド (ランタイム・オプションの使用)

関連参照
192 ページの『QSAM ファイル用のバッファーの割り振り』

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,NOTRIG) (DATEPROC だけが指定された場合)

省略形: DPINODP

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。

DATEPROC(TRIG)

DATEPROC(TRIG) を指定すると、2000 年言語拡張が使用可能になり、コンパイラーがウィンドウ表示日付フィールドの操作に適用する自動ウィンドウ操作では、日付フィールドや、ウィンドウ表示日付フィールドに保管されたり比較されたりするその他の非日付フィールドにおいて、特定のトリガー値または限界値が重要になります。これらの特殊値は、テストの対象になったり、上限値または下限値として使ったりする無効日付を表します。

パフォーマンスの考慮: DATEPROC(TRIG) オプションを使用すると、ウィンドウ表示日付の比較処理が遅くなります。

DATEPROC(NOTRIG)

DATEPROC(NOTRIG) を指定すると、2000 年言語拡張が使用可能になり、コンパイラーがウィンドウ表示日付の操作に適用する自動ウィンドウ操作では、オペランド内に特定のトリガー値があっても認識されません。日付の年部分の値だけが自動ウィンドウ操作の対象になります。

パフォーマンスの考慮: DATEPROC(NOTRIG) オプションは、ウィンドウ表示日付フィールドにおける有効な日付値を想定するパフォーマンス・オプションです。

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。

使用上の注意: FLAG|NOFLAG サブオプションおよび TRIG|NOTRIG サブオプションは、任意の順序で指定できます。どちらのサブオプションも省略した場合のデフォルトは、現行の設定になります。 ただし、DATEPROC の後ろに左括弧をコーディングした場合には、少なくとも 1 つ以上のサブオプションをコーディングする必要があります。

関連参照

363 ページの『FLAG』
404 ページの『YEARWINDOW』

DBCS

DBCS を使用すると、コンパイラーは、X'0E' (SO) および X'0F'(SI) を英数字リテラルの 2 バイト部分のシフト・コードとして認識するようになります。

DBCS オプションの構文

デフォルト: DBCS

省略形: なし

DBCS が有効であると、リテラルはカテゴリー英数字のままで、リテラルの 2 バイト部分が構文検査されます。

関連参照

344 ページの『矛盾するコンパイラー・オプション』

DECK

DECK は、オブジェクト・コードを 80 桁のレコード形式で作成する場合に使用します。DECK オプションを使用する場合は、コンパイル用の JCL で SYSPUNCH を必ず定義するようにしてください。

DECK オプションの構文

デフォルト: NODECK

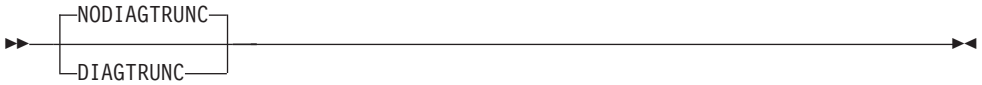
省略形: DINOD

関連タスク
304 ページの『オブジェクト・コードの作成 (SYSLIN または SYSPUNCH)』

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、診断が出されます。

TRUNC(BIN) オプションを指定した場合は、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は行われません。

関連概念
54 ページの『数値データの形式』
120 ページの『参照修飾子』

関連参照
397 ページの『TRUNC』

DLL

DLL は、ダイナミック・リンク・ライブラリー (DLL) サポートで使用可能なオブジェクト・モジュールを生成するようコンパイラーに指示する場合に使用します。DLL を使用可能にする必要があるのは、プログラムが DLL の一部である場合、プログラムが DLL を参照する場合、あるいはプログラムに INVOKE ステートメントやクラス定義などのオブジェクト指向 COBOL 構文が含まれている場合です。

DLL オプションの構文

デフォルト: NODLL

省略形: なし

リンク・エディットの考慮事項: DLL オプションを指定してコンパイルされた COBOL プログラムは、 RENT および AMODE(31) リンク・エディット・オプションを使用してリンク・エディットする必要があります。

NODLL は、DLL として使用できないオブジェクト・モジュールを生成するようにコンパイラーに指示します。

関連タスク
504 ページの『動的呼び出しの作成』

関連参照
344 ページの『矛盾するコンパイラー・オプション』

DUMP

DUMP は、コンパイル時に内部コンパイラー・エラーに関するシステム・ダンプを作成する場合に使用します。

DUMP オプションの構文

デフォルト: NODUMP

省略形: DUINODU

通常は不使用: DUMP オプションは、IBM 担当員から依頼があったときにのみ使用してください。

ダンプは、コンパイラーのレジスターのリストとストレージ・ダンプで構成され、主として診断を行う担当者がコンパイラーのエラーを判別するために使用するものです。

DUMP オプションを使用する場合は、コンパイル時に DD ステートメントを組み込んで、SYSABEND、SYSUDUMP、または SYSMDUMP を定義してください。

DUMP を指定すると、コンパイラーは、異常終了処理の前に診断メッセージを出しません。その代わりに、ユーザー異常終了コードは IGY*ppnnnn* で出されます。一般に、メッセージ IGY*ppnnnn* はコンパイル時のユーザー異常終了コード *nnnn* に対応しています。ただし、IGY*pp*5*nnn* メッセージと IGY*pp*1*nnn* メッセージはどちらも、ユーザー異常終了 1*nnn* を生成します。NODUMP オプションを使用して再コンパイルすれば、そのメッセージが実際は 5*nnn* と 1*nnn* のどちらなのかを区別することができます。

次のものを含め、正常な終了処理を行いたい場合には、NODUMP を使用してください。

- コンパイル中でそれまでに作成された診断メッセージ。
- エラーの記述。
- 現在実行中のコンパイラー・フェーズの名前。
- エラー検出時に処理されていた COBOL ステートメントの行番号。(OPTIMIZE を使用してコンパイルした場合は、行番号が正しく示されないことがあります。エラーによっては、プログラムの最後の行が示される場合があります。)
- 汎用レジスターの内容。

DUMP および OPTIMIZE コンパイラー・オプションを一緒に使用すると、コンパイラーは、次の最適化プログラム・メッセージの代わりにシステム・ダンプを作成します。

```
"IGYOP3124-W  This statement may cause a program exception at
              execution time."
```

この状況はコンパイラー・エラーではありません。 NODUMP オプションを使用すると、コンパイラーはメッセージ IGYOP3124-W を出して、処理を継続することができます。

関連タスク
言語環境プログラム・デバッグのガイド (異常終了コードの理解)

関連参照
344 ページの『矛盾するコンパイラー・オプション』

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、常にターゲット・プログラムは実行時にロードされます。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: DYNINODYN

制約事項: 以下の場合に DYNAM コンパイラー・オプションを使用してはなりません。

- CICS 変換プログラムまたは CICS コンパイラー・オプションによって処理される COBOL プログラム
- EXEC SQL ステートメントが含まれており、CICS または DB2 呼び出し接続機能 (CAF) のもとで実行される COBOL プログラム

COBOL プログラムがダイナミック・リンク・ライブラリー (DLL) としてリンクされているプログラムを呼び出す場合は、DYNAM オプションを使用してはなりません。代わりに、NODYNAM および DLL オプションを使用してそのプログラムをコンパイルするようにしてください。

関連タスク 509 ページの『静的呼び出しと動的呼び出しの両方の作成』
479 ページの『DYNAM または NODYNAM コンパイラー・オプションの選択』

関連参照
344 ページの『矛盾するコンパイラー・オプション』

EXIT

EXIT コンパイラー・オプションの詳細については、以下の最初の関連参照を参照してください。

関連参照
787 ページの『付録 E. EXIT コンパイラー・オプション』
344 ページの『矛盾するコンパイラー・オプション』

EXPORTALL

EXPORTALL を使用すると、オブジェクト・デックをリンク・エディットして DLL を形成するときに、PROGRAM-ID 名および各代替入り口点名をそれぞれのプログラム定義から自動的にエクスポートするようコンパイラーに指示することができます。

EXPORTALL オプションの構文

デフォルト: NOEXPORTALL

省略形: EXPINOEXP

これらの記号が DLL からエクスポートされた場合、エクスポートされたプログラム名と入り口点名は、同じ DLL にリンクされたプログラムからだけでなく、アプリケーション内のルート・ロード・モジュールまたはその他の DLL ロード・モジュールのプログラムから呼び出すことができるようになります。

EXPORTALL オプションを指定する場合は、RENT リンカー・オプションも併せて指定する必要があります。

NOEXPORTALL は、記号をエクスポートしないようにコンパイラーに指示します。この場合、プログラムには、この COBOL プログラム定義と一緒に同じロード・モジュールにリンク・エディットされた他のルーチンからしかアクセスすることができません。

関連参照
344 ページの『矛盾するコンパイラー・オプション』

FASTSRT

FASTSRT は、IBM DFSORT またはそれと同等のものが COBOL の代わりに入出力を実行することを許可します。

FASTSRT オプションの構文

デフォルト: NOFASTSRT

省略形: FSRTINOFSRT

関連タスク
253 ページの『FASTSRT を使用してのソートのパフォーマンスの向上』

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: FINOF

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* にコーディングされる重大度は、*x* にコーディングされる重大度より低くなってはいけません。FLAG(*x*,*y*) を使用するには、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。以下に例を示します。

```
    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。(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。
- *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

315 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 を使用して、準拠していると見なされ、プログラムに組み込まれた 標準 COBOL 85 エレメントに関する通知メッセージを取得するように、標準 COBOL 85 のレベルまたはサブセットを指定します。

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
- 廃止された言語エレメント
- 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よう、標準 COBOL 85 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 は、廃止された言語エレメントに「廃止」のフラグを付けることを表します。

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、メッセージ・コードとテキストとともに、フラグ付き節、ステートメントまたはヘッダーが出てきた行番号と桁が示されています。最下部には、フラグ付けされた項目の合計とそれらのタイプがまとめられています。

```
  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
```

関連参照
344 ページの『矛盾するコンパイラー・オプション』

INTDATE

INTDATE(ANSI) は、コンパイラーに、日付組み込み関数で使用する整数日付形式に標準 COBOL 85 の開始日を使用するように指示します。日付 1 は、1601 年 1 月 1 日です。INTDATE(LILIAN) は、コンパイラーに、日付組み込み関数で使用する整数日付形式に言語環境プログラムのリリアン開始日を使用するように指示します。日付 1 は、1582 年 10 月 15 日です。

INTDATE オプションの構文

デフォルト: INTDATE(ANSI)

省略形: なし

INTDATE(LILIAN) を使用すると、日付組み込み関数は、言語環境プログラムの日付呼び出し可能サービスと互換性のある結果を戻します。

使用上の注意: INTDATE(LILIAN) が有効なときは、CEECBLDY は使用不能になります。これは、組み込み関数または呼び出し可能サービスを使用して ANSI 整数を意味のある日付に変換する方法がないためです。INTDATE(LILIAN) が有効になっている呼び出しのターゲットとして、CEECBLDY を指定した CALL *literal* ステートメントをコーディングすると、コンパイラーはこれを診断し、呼び出しターゲットを CEEDAYS に変換します。

関連タスク
67 ページの『データ呼び出し可能サービスの使用』

LANGUAGE

LANGUAGE オプションは、コンパイラー出力の印刷に使用する言語を選択する場合に使用します。選択された言語で印刷される情報には次のものがあります。診断メッセージ、ソース・リストのページ・ヘッダーとスケール・ヘッダー、FIPS メッセージ・ヘッダー、メッセージ要約ヘッダー、コンパイル要約、および特定のコンパイラー・オプション (MAP、XREF、VBREF、および FLAGSTD) を選択した結果として生じるヘッダーと表記が含まれます。

LANGUAGE オプションの構文

デフォルト: LANGUAGE(ENGLISH)

省略形: LANG(ENIUEIJAIJP)

name は、コンパイラー出力メッセージに使用する言語を指定します。LANGUAGE オプションに使用できる可能な値を、以下の表に示します。

表 48. **LANGUAGE コンパイラー・オプションの値**

Name (名前)	省略形 [1]	出力言語
ENGLISH	EN	英大/小文字混合 (デフォルト)
JAPANESE	JA、JP	日本語 (日本語文字セットを使用)
UENGLISH[2]	UE	英大文字

1. インストール先のシステム・プログラマーが、ここに説明されている以外の言語を提供した場合は、その言語名の少なくとも最初の 2 文字を指定しなければなりません。
2. UENGLISH 以外の言語を指定するには、該当する言語の機能をインストールしなければなりません。

コンパイル時に (CBL または PROCESS ステートメントを使用して) LANGUAGE オプションが変更された場合は、一部の初期テキストは、コンパイラーの開始時に有効であった言語を使用して印刷されます。

NATLANG: NATLANG ランタイム・オプションを使用すると、エラー・メッセージ、月名、および曜日名を含む、ランタイム環境で使用される各国語を制御することができます。LANGUAGE コンパイラー・オプションと NATLANG ランタイム・オプションは互いに独立して働きます。それらのどちらかに優先権を与えずに同時に使用することができます。

LIB

プログラムで COPY、BASIS、または REPLACE ステートメントを使用する場合は、LIB コンパイラー・オプションを有効にする必要があります。

LIB オプションの構文

デフォルト: NOLIB

省略形: なし

COPY ステートメントと BASIS ステートメントの場合は、さらに、コピーされたコードをコンパイラーが獲得するライブラリー (複数も可) を定義する必要があります。ライブラリーは、環境に応じて、DD ステートメント、ALLOCATE コマンド、または環境変数を使用して定義してください。JCL を使用する場合は、SYSUT5 を割り振るための DD ステートメントも含める必要があります。

関連参照
407 ページの『第 18 章 コンパイラー指示ステートメント』
344 ページの『矛盾するコンパイラー・オプション』

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

次の項目も出力リストに書き込まれます。

- グローバル・テーブル
- リテラル・プール
- WORKING-STORAGE および LOCAL-STORAGE に関する情報
- プログラムの WORKING-STORAGE と LOCAL-STORAGE のサイズ、およびプログラムが NORENT オプションを使用してコンパイルされた場合には、オブジェクト・コード内でのその位置

出力が生成されるのは、以下の場合です。

- COMPILE オプションを指定している、または NOCOMPILE(*x*) オプションが有効であり、エラー・レベル *x* 以上が発生していない。
- OFFSET オプションを指定していない。

アセンブラー・リスト出力を制限したい場合は、PROCEDURE DIVISION で *CONTROL (または *CBL) LIST か、NOLIST ステートメントを使用します。 *CONTROL NOLIST 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 LIST ステートメントによって出力が通常の LIST 形式に戻されない限り、リストには含められません。

関連タスク
422 ページの『リストの入手』

関連参照 344 ページの『矛盾するコンパイラー・オプション』
*CONTROL (*CBL) ステートメント (*Enterprise COBOL 言語解説書*)

MAP

MAP を使用すると、DATA DIVISION に定義した項目のリストを作成することができます。

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
- グローバル・テーブル
- リテラル・プール
- 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 と LOCAL-STORAGE のサイズ、およびプログラムが NORENT オプションを使用してコンパイルされた場合には、オブジェクト・コード内でのその位置

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。以下に例を示します。

```
*CONTROL NOMAP            *CBL NOMAP
    01  A                     01  A
    02  B                     02  B
*CONTROL MAP              *CBL MAP
```

MAP オプションを選択すると、組み込み MAP 報告書もソース・コード・リストに印刷することができます。圧縮 MAP 情報は、DATA DIVISION の FILE SECTION、LOCAL-STORAGE SECTION、および LINKAGE SECTION のデータ名定義の右側に印刷されます。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。

427 ページの『例: MAP 出力』

関連概念

411 ページの『第 19 章 デバッグ』

関連タスク

422 ページの『リストの入手』

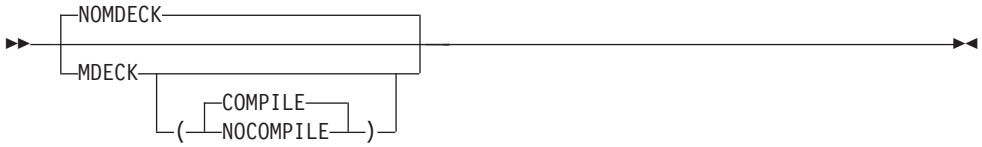
関連参照
*CONTROL (*CBL) ステートメント (*Enterprise COBOL 言語解説書*)

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、または EXEC SQL INCLUDE ステートメントの拡張) の出力がファイルに書き込まれることを指定します。

Enterprise COBOL が z/OS UNIX のもとで稼働している場合、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。 TSO またはバッチで稼働している Enterprise COBOL の場合、MDECK 出力は、SYSMDECK DD ステートメントで定義されたデータ・セットに書き込まれます。このステートメントでは、RECFM F または FB であり、LRECL が 80 バイトの MVS データ・セットを指定している必要があります。

MDECK オプションの構文



デフォルト: NOMDECK

省略形: NOMD、MD、MD(C)、MD(NOC)

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、COMPILEINOCOMPILE、 DECKINODECK、および OBJECTINOOBJECT コンパイラー・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、ライブラリー処理が完了し、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、DECK、および OBJECT コンパイラー・オプションの設定値にかかわらず、構文検査や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。

オプション指定:

MDECK を PROCESS または CBL ステートメントに指定することはできません。このオプションを指定できるのは、以下を使用する場合のみです。

- JCL の PARM パラメーター

- cob2 コマンド・オプション
- インストール先のデフォルト
- COBOPT 環境変数

MDECK 出力ファイルの内容:

MDECK オプションを、CICS コンパイラー・オプション (組み込みの CICS 変換プログラム) または SQL コンパイラー・オプション (DB2 coprocessor) と一緒に使用すると、一般に、COBOL ソース・プログラム内の EXEC CICS または EXEC SQL ステートメントが、MDECK 出力にそのまま組み込まれます。EXEC SQL INCLUDE ステートメントは、COPY ステートメントと同様に MDECK 出力内で拡張されます。

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。

関連タスク
296 ページの『アセンブラー・プログラムからコンパイラーを開始する』
306 ページの『ライブラリー処理出力ファイル (SYSMDECK) の定義』

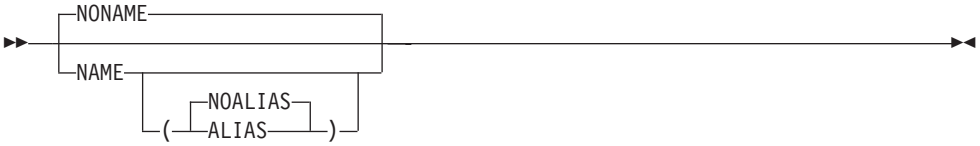
関連参照
344 ページの『矛盾するコンパイラー・オプション』
407 ページの『第 18 章 コンパイラー指示ステートメント』

NAME

NAME は、各オブジェクト・モジュールについてのリンク・エディット NAME カードを生成する場合に使用します。NAME は、バッチ・コンパイルを行う際に、各ロード・モジュールの名前を生成する目的でも使用できます。

NAME を指定すると、作成される各オブジェクト・モジュールに NAME カードが追加されます。ロード・モジュール名は、PROGRAM-ID ステートメントからモジュール名を形成する際の規則を使用して作成されます。

NAME オプションの構文



デフォルト: NONAME、または NAME(NOALIAS) (NAME だけが指定された場合)

省略形: なし

NAME(ALIAS) を指定し、プログラムに ENTRY ステートメントが含まれている場合には、 ENTRY ステートメントごとにリンク・エディット ALIAS カードが生成されます。

NAME または NAME(ALIAS) オプションは、言語環境プログラムのプリリンカーでプリリンクされるプログラムをコンパイルする場合には使用できません。

関連参照
PROGRAM-ID 段落 (*Enterprise COBOL 言語解説書*)

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、前のリリースの IBM COBOL との互換性を提供します。NSYMBOL(NATIONAL) オプションにより、前述の言語エレメントの処理がこの点に関して 標準 COBOL 2002 に準拠するようになります。

NSYMBOL(NATIONAL) は、Unicode データや Java とのインターオペラビリティーのためのオブジェクト指向構文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。

関連参照
344 ページの『矛盾するコンパイラー・オプション』

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: NUMINONUM

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。

バッチ・コンパイルを行っており、LIB と NUMBER が有効な場合は、バッチ・コンパイルのすべてのプログラムが 1 つの入力ファイルとして扱われます。入力ファイル全体のシーケンス番号は昇順でなければなりません。

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。

NUMPROC

数値内部 10 進数およびゾーン 10 進数データで非優先符号を使用することがある場合には常に NUMPROC(NOPFD) を使用してください。

NUMPROC オプションの構文

デフォルト: NUMPROC(NOPFD)

省略形: なし

コンパイラーは、任意の有効符号構成 (X'A'、X'B'、X'C'、X'D'、X'E'、または X'F') を受け入れます。ほとんどの場合の推奨オプションは NUMPROC(NOPFD) です。

NUMPROC(PFD) の場合、数値内部 10 進数およびゾーン 10 進数データの処理のパフォーマンスが向上します。 このオプションは、プログラム・データが以下の IBM システム標準と正確に一致する場合にのみ 使用してください。

ゾーン 10 進数、符号なし: 符号バイトの高位 4 ビットに X'F' が入ります。

ゾーン 10 進数、符号付きオーバーパンチ: 符号バイトの高位 4 バイトに、X'C' (数値が正または 0 の場合)、および X'D' (数値がそれ以外の場合) が入ります。

ゾーン 10 進数、分離符号: 分離符号は、文字「+」(数値が正または 0 の場合)、および「-」(数値がそれ以外の場合) を含みます。

内部 10 進数、符号なし: 下位バイトの下位 4 ビットには、X'F' が入ります。

内部 10 進数、符号付き: 下位バイトの下位 4 ビットには、その数値が正または 0 であれば X'C' が入り、そうでなければ X'D' が入ります。

COBOL 算術ステートメントが作成するデータは、上記の IBM システム標準に適合します。しかし、REDEFINES を使用したり、グループ移動を行うと、データが変更されて、この標準に適合しなくなることがあります。NUMPROC(PFD) を使用する場合は、グループ移動を行うのではなく、INITIALIZE ステートメントを使用してデータ・フィールドを初期化しなければなりません。

NUMPROC(PFD) を使用すると、数値データのクラス・テストに影響を与えることがあります。PL/I または FORTRAN で書かれたプログラムを COBOL プログラムで呼び出す場合は、NUMPROC(NOPFD) または NUMPROC(MIG) を使用しなければなりません。

サイン表記は、NUMPROC オプションばかりでなく、インストール時のオプション NUMCLS の影響も受けます。

NUMPROC(MIG) は、OS/VS COBOL プログラムを Enterprise COBOL に移行する際の援助手段として使用してください。 NUMPROC(MIG) が有効であると、次の処理が行われます。

- MOVE ステートメントと算術演算の出力の場合のみ、優先符号が作成されます。
- 入力では明示符号修復は行われません。
- 変換の際に暗黙符号修復が行われることがあります。
- 数値比較は、論理比較ではなく、10 進比較によって行われます。

関連タスク
61 ページの『非互換データの検査 (数値のクラス・テスト)』

関連参照
60 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』

OBJECT

OBJECT を使用すると、生成されたオブジェクト・コードをディスクまたはテープに保管し、リンケージ・エディターまたはバインダーへの入力として後で使用することができます。

OBJECT オプションの構文

デフォルト: OBJECT

省略形: OBJ|NOOBJ

OBJECT を指定する場合は、コンパイル用の JCL の中に SYSLIN DD ステートメントを組み込んでください。

DECK と OBJECT の相違点は、データ・セットの経路指定だけです。

- DECK 出力は、SYSPUNCH の DD 名に関連するデータ・セットに送られます。
- OBJECT 出力は、SYSLIN の DD 名に関連するデータ・セットに送られます。

ご使用のシステムの指針に従ってオプションを使用してください。

関連参照
344 ページの『矛盾するコンパイラー・オプション』

OFFSET

OFFSET は、PROCEDURE DIVISION の圧縮リストを作成する場合に使用します。

OFFSET オプションの構文

デフォルト: NOOFFSET

省略形: OFFINOOFF

OFFSET を使用した場合、PROCEDURE DIVISION の圧縮リストには、行番号、ステートメント参照、および各ステートメントに対して生成された最初の命令の位置が入れられます。さらに、リストは以下のものも示します。

- グローバル・テーブル
- リテラル・プール
- プログラムの WORKING-STORAGE のサイズ、およびオブジェクト・コードにおけるその位置 (NORENT オプションを指定してプログラムをコンパイルする場合)

関連参照
344 ページの『矛盾するコンパイラー・オプション』

OPTFILE

OPTFILE は、データ・セットでの COBOL コンパイラー・オプションの指定を有効化するために使用します。コンパイラー・オプション・データ・セットを使用することによって、JCL PARM ストリングに指定されたオプションの 100 文字制限を回避します。

OPTFILE オプションの構文

デフォルト: なし

省略形: なし

OPTFILE は、コンパイラー呼び出しオプションとして、または COBOL ソース・プログラムの PROCESS または CBL ステートメントで、指定できます。 OPTFILE は、インストール先デフォルトとしては指定できません。

OPTFILE は、z/OS UNIX 環境で cob2 コマンドを使用してコンパイルした場合には、無視されます。 (その環境では、COBOPT 環境変数が、OPTFILE に同等の機能を提供します。)

OPTFILE が有効化されている場合には、コンパイラー・オプションは、SYSOPTF DD ステートメントで指定したデータ・セットから読み取られます。 SYSOPTF データ・セットには、RECFM F または FB、および 80 バイトの LRECL が存在している必要があります。 SYSOPTF データ・セットのフォーマットの詳細については、コンパイラー・オプション・データ・セットの定義に関する下記の関連タスクを参照してください。

SYSOPTF データ・セットのオプションの優先順位は、OPTFILE オプションを指定した場所で決まります。例えば、呼び出し PARM ストリングで OPTFILE を指定した場合、 PARM ストリングで後から指定したオプションによって、競合する SYSOPTF データ・セットで指定されたすべてのオプションが置き換えられます。

(概念的には、PARM ストリングの OPTFILE は、SYSOPTF データ・セットにあるオプションで置き換えられます。それから、コンパイラー・オプションおよび競合するコンパイラー・オプションの優先順位に関する規則が適用されます。)

アセンブラー・プログラム内から COBOL コンパイラーを始動した場合、SYSOPTF でコンパイラー・オプション・データ・セットを指定する代わりに、代替 DD 名リストを使用して、使用する DD 名を指定することができます。

関連タスク

296 ページの『アセンブラー・プログラムからコンパイラーを開始する』
302 ページの『コンパイラー・オプション・データ・セットの定義 (SYSOPTF)』
306 ページの『z/OS のもとでのコンパイラー・オプションの指定』
319 ページの『第 15 章 UNIX のもとでのコンパイル』

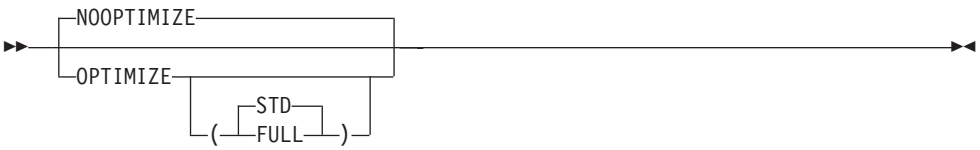
関連参照

344 ページの『矛盾するコンパイラー・オプション』

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。

OPTIMIZE オプションの構文



デフォルト: NOOPTIMIZE

省略形: OPTINOOPT

サブオプションを付けないで 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。

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)。プログラムがこの手法を使用しているかどうかを調べるには、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てください。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。

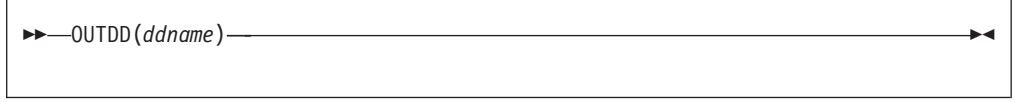
関連概念
732 ページの『最適化』

関連参照 344 ページの『矛盾するコンパイラー・オプション』
392 ページの『TEST』

OUTDD

OUTDD は、システム論理出力装置に向けられる DISPLAY 出力を特定の DD 名に送りたいことを指定する場合に使用します。 OUTDD で指名する DD 名で、階層ファイル・システム内のファイルを指定することができます。この DD 名が割り当てられていない場合における動作のデータの表示については、関連タスクを参照してください。

OUTDD オプションの構文



デフォルト: OUTDD(SYSOUT)

省略形: OUT

MSGFILE ランタイム・オプションを使用すると、RPTOPTS および RPTSTG ランタイム・オプションによって生成されるすべての実行時診断および報告書が書き込まれるファイルの DD 名を指定することができます。 IBM 提供のデフォルト値は MSGFILE(SYSOUT) です。OUTDD コンパイラー・オプションと MSGFILE ランタイム・オプションで同じ DD 名が指定された場合は、システム論理出力装置に送られる DISPLAY 出力とエラー・メッセージ情報は同じ宛先に経路指定されます。

制約事項: OUTDD オプションは、CICS では無効です。

関連タスク
42 ページの『システム論理出力装置上でのデータの表示』
454 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
言語環境プログラム・プログラミング・リファレンス (MSGFILE)

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(COMPAT)

省略形: PGMN(LMILUICO)

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。

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の呼び出し
  - 別個にコンパイルされたプログラムへの静的呼び出し
  - 静的 SET *procedure-pointer* TO ENTRY *literal* ステートメント
  - 静的 SET *function-pointer* TO ENTRY *literal* ステートメント
  - ネストされたプログラムの CANCEL

PGMNAME(COMPAT)

PGMNAME(COMPAT) を使用すると、次のように、プログラムは、COBOL コンパイラーの旧バージョンと互換性のある方法で処理されます。

- 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、またはハイフンでなければならず (ただし、プログラム名がリテラル形式で入力されており、最外部のプログラムにある場合を除く)、リテラルには拡張文字の @、#、および $ を含めることができる。
- 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。

外部プログラム名はコンパイラーによって次のように処理されます。

- 大文字に変換される。
- 8 文字に切り捨てられる。
- ハイフンはゼロ (0) に変換される。
- 先頭文字が英字でない場合は、次のように変換される。
  - 1 から 9 は A から I に変換される。
  - その他はすべて J に変換される。

PGMNAME(LONGUPPER)

PGMNAME(LONGUPPER) 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、またはハイフンでなければならない。
- 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。

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、またはハイフンでなければならない。
- 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。

外部プログラム名はコンパイラーによって次のように処理されます。

- 大文字に変換される。
- ハイフンはゼロ (0) に変換される。
- 先頭文字が英字でない場合は、次のように変換される。
  - 1 から 9 は A から I に変換される。
  - その他はすべて J に変換される。

ネストされたプログラムの名前はコンパイラーによって大文字に変換されますが、それ以外はそのまま処理され、切り捨ても変換も行われません。

PGMNAME(LONGMIXED)

PGMNAME(LONGMIXED) 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(LONGMIXED) 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。

(PGMNAME オプションの影響を受ける、上記にリストされたコンテキストの中で) プログラム名に使用されるリテラルには、X'41'-X'FE' の範囲の文字を含めることができます。

使用上の注意

- 以下のエレメントは、PGMNAME オプションの影響を受けません。
  - クラス名およびメソッド名。
  - システム名 (SELECT .. . ASSIGN の中の割り当て名、および COPY ステートメントの中のテキスト名またはライブラリー名)。
  - 動的呼び出し。動的呼び出しはターゲット・プログラム名で解決されます。このとき、ターゲット・プログラム名は 8 文字に切り詰められ、大文字に変換されて、埋め込まれたハイフンや先行桁は変換されます。
  - ネストされていないプログラムの CANCEL。ネーム・レゾリューションは動的呼び出しのメカニズムと同じものを使用します。
- PGMNAME オプションは、ネストされたプログラムの呼び出し、および呼び出し側と一緒にリンクされるプログラムへの静的呼び出しに影響を与えます。

- **リンク・エディットの考慮事項:** PGMNAME(LONGUPPER) または PGMNAME(LONGMIXED) オプションを使用してコンパイルされた COBOL プログラムは、AMODE 31 でリンク・エディットする必要があります。
- プログラム名が 8 バイト以下ですべて大文字でない限り、PGMNAME(LONGMIXED) または PGMNAME(LONGUPPER) オプションでコンパイルされた COBOL プログラムへの動的呼び出しは許可されません。さらに、プログラムの名前はそれが入っているモジュールの名前に対して固有でなければなりません。
- PGMNAME(LONGMIXED) によってサポートされる拡張文字セットを使用するときは、名前の解決に使用されるメカニズムに応じて、リンケージ・エディター、バインダー、プリリンカー、またはシステム規則のうちの該当するものに適合する名前を使用してください。

  コンマや括弧などの文字は使用しないでください。これらの文字は、リンケージ・エディターやバインダーの制御ステートメントの構文で使用されます。

QUOTE/APOST

QUOTE は、形象定数 [ALL] QUOTE または [ALL] QUOTES が 1 つ以上の引用符 (") 文字を表すようにする場合に使用します。APOST は、形象定数 [ALL] QUOTE または [ALL] QUOTES が 1 つ以上の引用符 (') 文字を表すようにする場合に使用します。

QUOTE/APOST オプションの構文

デフォルト: QUOTE

省略形: QIAPOST

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、単一引用符または二重引用符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。

RENT

RENT としてコンパイルされたプログラムは再入可能オブジェクト・プログラムとして生成されます。NORENT としてコンパイルされたプログラムは再入不可オブジェクト・プログラムとして生成されます。再入可能プログラムまたは再入不可プログラムは、メインプログラムまたはサブプログラムとして呼び出すことができます。

RENT オプションの構文

デフォルト: RENT

省略形: なし

DATA および RMODE 設定: RENT オプションは、ストレージおよびアドレス可能度に影響を与える他のコンパイラー・オプションと相互作用します。再入可能プログラムを拡張アドレス方式で実行する場合は、DATA(24|31) オプションを使用して、動的データ域を制限のないストレージで割り振るのか 16MB より下から獲得されたストレージで割り振るのかを制御することができます。プログラムが 16MB より上の仮想記憶アドレスで拡張アドレス方式で実行される場合は、プログラムを RENT または RMODE(ANY) でコンパイルしなければなりません。

RENT は、生成されたオブジェクト・プログラムの RMODE (常駐モード) にも影響を与えます。Enterprise COBOL プログラムはすべて AMODE ANY です。

DATA: DATA オプションの設定は、NORENT を指定してコンパイルされたプログラムに影響を与えません。

再入可能にする必要がある Enterprise COBOL プログラムについては、プログラムの再入可能化に関する下記の関連タスクを参照してください。

リンク・エディットについての考慮事項: ロード・モジュール内のすべてのプログラムが RENT を指定してコンパイルされる場合は、RENT リンケージ・エディター・オプションまたはバインダー・オプションを使用して、そのロード・モジュールをリンク・エディットすることをお勧めします。(逐次に再使用可能な非 COBOL プログラムがロード・モジュールに含まれている場合は、RENT ではなく、REUS リンケージ・エディター・オプションまたはバインダー・オプションを使用してください。)

ロード・モジュール内のいずれかのプログラムが NORENT でコンパイルされた場合には、そのロード・モジュールは RENT または REUS リンク・エディット属性を使用してリンク・エディットしてはなりません。CANCEL ステートメントがその後の CALL においてプログラムの最新コピーを保証するようにするためには、NOREUS リンケージ・エディターまたはバインダー・オプションを使用する必要があります。

関連概念
45 ページの『ストレージとそのアドレス可能度』

関連タスク
518 ページの『プログラムを再入可能にする』
DB2 アプリケーション・プログラミングおよび SQL ガイド
(再入可能コードの使用)

関連参照
344 ページの『矛盾するコンパイラー・オプション』

RMODE

RMODE オプションの設定は、生成されたオブジェクト・プログラムの RMODE (常駐モード) にも影響を与えます。

RMODE オプションの構文

デフォルト: AUTO

省略形: なし

RMODE(AUTO) オプションでコンパイルされたプログラムは、NORENT が指定された場合は RMODE 24 になり、RENT が指定された場合は RMODE ANY になります。RMODE(AUTO) は、VS COBOL II などの古いコンパイラーと互換性があります。これらのコンパイラーは、NORENT でコンパイルされたプログラムに対しては RMODE 24 を生成し、RENT でコンパイルされたプログラムに対しては RMODE ANY を生成していました。

RMODE(24) オプションでコンパイルされたプログラムは、NORENT または RENT のいずれが指定されたかに関係なく、RMODE 24 になります。

RMODE(ANY) オプションでコンパイルされたプログラムは、NORENT または RENT のいずれが指定されたかに関係なく、RMODE ANY になります。

DATA および RENT: RMODE オプションは、ストレージおよびアドレス可能度に影響を与える他のコンパイラー・オプションおよびランタイム・オプションにも相互作用します。モードの異なるプログラム間でのデータの受け渡しに関する情報については、関連概念を参照してください。

リンク・エディットに関する考慮事項: COBOL が生成するオブジェクト・コードに属性 RMODE 24 がある場合には、RMODE 24 でオブジェクト・コードをリンク・エディットする必要があります。 COBOL が生成するオブジェクト・コードに属性 RMODE ANY がある場合には、RMODE ANY または RMODE 24 でオブジェクト・コードをリンク・エディットすることができます。

関連概念
45 ページの『ストレージとそのアドレス可能度』

関連参照
192 ページの『QSAM ファイル用のバッファーの割り振り』
344 ページの『矛盾するコンパイラー・オプション』

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 EBCDIC 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: SEQINOSEQ

COPY ステートメントを使用する場合、SEQUENCE が有効なときは、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のシーケンス・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、EBCDIC 照合シーケンスではなく、数字に従って行われます。

バッチ・コンパイルを行っている場合は、LIB と SEQUENCE が有効であると、バッチ・コンパイルのすべてのプログラムが 1 つの入力ファイルとして扱われます。入力ファイル全体のシーケンス番号は昇順でなければなりません。

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
418 ページの『行シーケンス問題の検出』

SIZE

SIZE は、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ために使用します。

SIZE オプションの構文

デフォルト: SIZE(MAX)

省略形: SZ

nnnnn には、少なくとも 851968 以上の 10 進数を指定します。

*nnn*K は、1KB 単位で 10 進数を指定します。1KB = 1024 バイトです。最小許容値は、832K です。

MAX は、ユーザー領域で使用可能な最大のストレージ・ブロックを要求します。

コンパイラーがユーザー領域に特定量の未使用ストレージを使用可能な状態で残す必要がある場合には、SIZE(MAX) を使用しないでください。例えば、CICS または SQL コンパイラー・オプションを使用している場合、SIZE(4000K) といった値を指定します。 (この値は大抵のプログラムで機能するはずです。) 31 ビット・モードでコンパイルする場合、SIZE(MAX) を指定したときは、コンパイラーは以下の順序でストレージを使用します。

- 16MB より上 - ユーザー領域のすべてのストレージ
- 16MB より下 - 以下のためのストレージ:
  - 作業ファイル・バッファー
  - 16MB 境界より下にロードしなければならないコンパイラー・モジュール

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: SINOS

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。

427 ページの『例: MAP 出力』

関連参照
*CONTROL (*CBL) ステートメント (*Enterprise COBOL 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

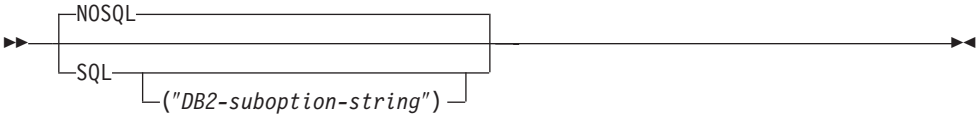
SPACE が意味を持つのは、SOURCE コンパイラー・オプションが有効な場合だけです。

関連参照
387 ページの『SOURCE』

SQL

SQL コンパイラー・オプションを使用すると、DB2 coprocessor 機能を使用可能にし、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、それが 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

SQL オプションを使用すると、DB2 コプロセッサーは、データベース要求モジュール (DBRM) を DD 名 DBRMLIB に書き込みます。DB2 がコンパイルするマシンで使用可能になっている必要があります

NOSQL オプションを指定した場合は、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。

長いサブオプション・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 ステートメントに置くことができます。以下に例を示します。

```
//STEP1 EXEC IGYWC, . . .
// PARM.COBOL='SQL("string1")'
//COBOL.SYSIN DD *
       CBL SQL("string2")
       CBL SQL('string3')
       IDENTIFICATION DIVISION.
       PROGRAM-ID. DRIVER1.
       . . .
```

それぞれの DB2 サブオプションは、指定された順に連結されます。そのため、上の例では、コンパイラーは次のサブオプション・ストリングを DB2 coprocessor 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 DB2 オプションの複数インスタンスが見つかった場合は、各オプションで最後に指定されたものが使用されます。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 4K バイトに限定しています。

関連概念

467 ページの『DB2 coprocessor』
474 ページの『COBOL および DB2 CCSID の決定』

関連タスク

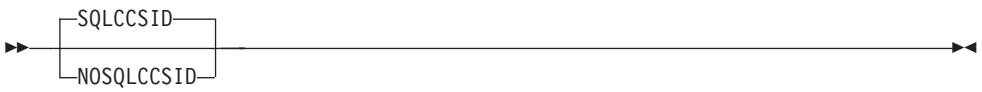
472 ページの『SQL オプションを使用したコンパイル』
473 ページの『DB2 サブオプションの分離』

関連参照
344 ページの『矛盾するコンパイラー・オプション』

SQLCCSID

SQLCCSID コンパイラー・オプションは、CODEPAGE コンパイラー・オプションが COBOL プログラムの SQL ステートメントの処理に影響を与えるかどうかを制御するために使用します。

SQLCCSID オプションの構文



デフォルト: SQLCCSID

省略形: SQLCINOSQLC

SQLCCSID オプションは、組み込みの DB2 コプロセッサー(SQL コンパイラー・オプション) を使用した場合にのみ有効です。

関連概念
467 ページの『DB2 coprocessor』
474 ページの『COBOL および DB2 CCSID の決定』

関連タスク 475 ページの『SQLCCSID または NOSQLCCSID オプションを使用したプログラミング』

関連参照 475 ページの『SQL ステートメントのストリング・ホスト変数のコード・ページ決定』 350 ページの『CODEPAGE』 388 ページの『SQL』

SSRANGE

SSRANGE を使用すると、添え字 (ALL 添え字を含む) または指標が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することができます。それぞれの添え字または指標は、個別に妥当性を検査されるわけではありません。むしろ、テーブルの領域外の区域を参照しないようにするために、有効アドレスが検査されます。

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。

SSRANGE オプションの構文

デフォルト: NOSSRANGE

省略形: SSRINOSSR

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。
- 長さの値 (指定されている場合) が 1 以上である。
- 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。

SSRANGE がコンパイル時に有効であると、範囲検査コードが生成されます。CHECK(OFF) ランタイム・オプションを指定すれば、範囲検査を抑制することができます。そうすれば、範囲検査コードはオブジェクト・コードで休止状態になります。オプションとして、範囲検査コードを使用し、予期しないエラーを再コンパイルせずに解決するときに、役立てることもできます。

範囲外条件が検出されると、エラー・メッセージが生成され、プログラムは終了します。

覚え書き: 範囲検査が行われるのは、プログラムを SSRANGE オプションを指定してコンパイルし、かつ CHECK(ON) オプションを指定して実行した場合のみです。

関連概念
120 ページの『参照修飾子』

関連タスク
418 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 SYSTERM DD 名に送ることができます。

TERMINAL オプションの構文

デフォルト: NOTERMINAL

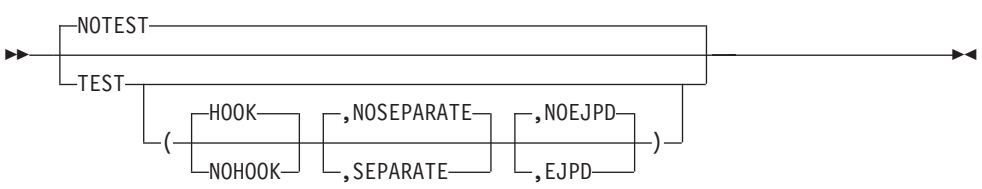
省略形: TERMINOTERM

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 を使用すると、デバッグ・ツール がバッチ・デバッグおよび対話式デバッグを実行できるようにするオブジェクト・コードが生成されます。 TEST を使用すれば、言語環境プログラムによって生成された定様式ダンプにあるシンボリック変数の組み込みも可能になります。

TEST オプションの構文



デフォルト・オプション: NOTEST

サブオプションのデフォルトは、HOOK、NOSEPARATE、NOEJPD です。

省略形: SEPINOSEP

TEST サブオプションは、どのような順序でも指定できます。また、どのようなサブオプションの組み合わせ (1 つでも、2 つでも、すべてでも) でも指定することができます。ただし、TEST の後ろに左括弧をコーディングした場合には、少なくとも 1 つ以上のサブオプションをコーディングする必要があります。

使用できるデバッグ・サポートの量は、以下で説明するように、どの TEST サブオプションを使用するかによって決まります。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、および定様式ダンプにシンボリック変数を組み込む必要がない場合には、NOTEST を使用します。

フック・サブオプション (コンパイルされたフック vs 動的フック)

HOOK　コンパイルされたフックは、すべてのステートメント、ラベル、パス点、お

よびすべてのプログラム入り口点と出口点 (最外部プログラムと含まれているプログラムの両方) で生成されます。さらに、DATEPROC オプションが有効であると、すべての日付処理ステートメントでフックが生成されます。

パス点とは、ロジック・フローが必ずしも順次でなくてもよいか、あるいはロジック・フローを変更できるプログラム内の任意の場所です。パス点の例としては、IF-THEN-ELSE 構成、PERFORM ループ、ON SIZE ERROR 句、および CALL ステートメントなどがあります。

NOHOOK コンパイルされたフックは生成されません。 TEST(NOHOOK) を使用すれば、デバッグ・ツールの動的デバッグ機能 (SET DYNDEBUG ON) を使用して、プログラムを対話式にデバッグできます。

シンボリック・デバッグ情報サブオプション

シンボリック・デバッグを有効化するために必要な情報は、TEST オプションが有効化されている場合には常に生成されます。

SEPARATE
デバッグの能力を維持しながらモジュールのサイズを制御するには、SEPARATE サブオプションを指定します。記号情報は、オブジェクト・モジュールではなく SYSDEBUG ファイルに書き込まれます。デバッグ能力を維持しながらモジュール・サイズを制御する方法については、後述のセクションを参照してください。

NOSEPARATE
オブジェクト・モジュールにシンボリック・デバッグ情報を組み込むには、NOSEPARATE サブオプションを使用します。

JUMPTO および GOTO 有効化サブオプション

EJPD および NOEJPD サブオプションは、実動デバッグ・セッションでの、デバッグ・ツール・コマンド JUMPTO および GOTO の有効化を制御します。 TEST(NOHOOK) および OPTIMIZE コンパイラー・オプションが指定されている場合にのみ、これらのサブオプションは有効になります。

EJPD TEST(NOHOOK,. . .,EJPD) および OPTIMIZE が指定された場合:

- JUMPTO および GOTO が有効化されます。
- プログラムの最適化の程度は低減されます。最適化は、ステートメント内で実行されますが、ほとんどの最適化はステートメント境界を超えません。

NOEJPD TEST(NOHOOK,. . .,NOEJPD および OPTIMIZE が指定された場合:

- JUMPTO および GOTO は有効化されません。
- 通常程度のプログラムの最適化が実行されます。

デバッグ能力を維持したモジュール・サイズの制御:

TEST オプションを指定すると、コンパイラーはデバッグ・ツールがデータ名や段落名などの解決に使用するデバッグ情報テーブルを生成します。この情報は、ストレージを大量に使用する場合があります。この情報をオブジェクト・プログラムにコンパイルするのか、または別の SYSDEBUG データ・セットに書き込むのかを選択することができます。

- ロード・モジュールを小さくする場合は、SEPARATE サブオプションを使用して、デバッグ・ツールセッションで使用するデバッグ・ファイルを別に保持します。
- 別のデバッグ・ファイルを管理する必要性をなくすには、NOSEPARATE サブオプションを使用してコンパイルします。ただし、このサブオプションによって、ロード・モジュールが大きくなることに注意してください。

JCL または TSO から COBOL コンパイラーを呼び出し、TEST (. . .,SEPARATE,. . .) を指定すると、シンボリック・デバッグ情報テーブルは、SYSDEBUG DD ステートメントで指定したデータ・セットに書き込まれます。このステートメントのコーディング方法および SYSDEBUG データ・セットの詳細については、デバッグ・データ・セットの定義方法および論理レコード長とブロック・サイズについての下記の関連情報を参照してください。

z/OS UNIX シェルから COBOL コンパイラーを呼び出すときに、TEST(. . .,SEPARATE,. . .) を指定すると、シンボリック・デバッグ情報テーブルは、現行ディレクトリー内の *file*.dbg に書き込まれます。*file* は COBOL ソース・ファイルの名前です。

パフォーマンスとデバッグ能力:

以下のように、得られるデバッグ能力の程度およびプログラムのパフォーマンスを制御できます。

- デバッグに一部制限がかかりますが、パフォーマンスを最高にするには、OPTIMIZE および TEST(NOHOOK,. . .,NOEJPD) を使用してコンパイルします。

  デバッグ・ツール の動的デバッグ機能 (SET DYNDEBUG ON) を使用すると、プログラムにコンパイルされたデバッグ・フックがない場合でも、プログラムを対話式にデバッグできます。

  TEST(NOHOOK,. . .,NOEJPD) では、プログラムを効率化するために、 OPTIMIZE (OPT(STD) または OPT(FULL)) も使用してコンパイルできますが、デバッグにいくつかの制限がかかります。
  - デバッグ・ツールのコマンド JUMPTO および GOTO がサポートされません。
  - DESCRIBE ATTRIBUTES コマンドを除くデバッグ・ツール・コマンドは、OPT(FULL) オプションによってプログラムから廃棄されたデータ項目を参照することはできません。
  - デバッグ・ツールのコマンド AT CALL *entry-name* がサポートされません。
- 上記の実働デバッグ・シナリオよりはプログラムのパフォーマンスが低下しますが、デバッグ・ツールのコマンド JUMPTO および GOTO を有効化するには、OPTIMIZE および TEST(NOHOOK,. . .,EJPD) を指定します。

  OPT(FULL) によって破棄される項目の参照および AT CALL コマンドに関する上記の制約は、この組み合わせのオプションを使用した場合にも適用されます。
- パフォーマンスが中程度になりますが、デバッグに対する制約を減らすには、NOOPT と TEST(NOHOOK) を指定します。

  この組み合わせは、最適化されたコードほど速く動作しませんが、デバッグ能力は向上します。すべてのデバッグ・ツールのコマンドがサポートされます (AT CALL *entry-name* を除く)。

- パフォーマンスはもっとも遅くなりますが、デバッグ能力を最大にするには、NOOPT と TEST(HOOK) を指定します。

  TEST(HOOK) を指定すると、コンパイラーはすべてのステートメントにコンパイルされたフックを設けるので、コードの実行は遅くなりますが、すべてのデバッグ・ツールのコマンドがサポートされます。

言語環境プログラム:

サブオプションのいずれかを指定した TEST オプションによって、以下の 2 つの機能をダンプに追加して、言語環境プログラムからの定様式ダンプを改善することができます。

- 単なるオフセットではなく、失敗したステートメントを示す行番号。
- プログラム変数の値

NOTEST を使用すれば、ダンプでは、プログラム変数も失敗したステートメントの行番号も示されません。

Enterprise COBOL は、言語環境プログラムが提供するダンプ・サービスを使用して、言語環境プログラム準拠のその他のメンバー言語によって生成されるダンプの内容および形式と一致するダンプを生成します。

処理されない条件に対して言語環境プログラムがダンプを作成するかどうかは、ランタイム・オプション TERMTHDACT の設定値によって決まります。
TERMTHDACT(DUMP) を指定した場合は、重大度 2 以上の条件が処理されないと、ダンプが生成されます。

SEPARATE サブオプションおよび言語環境プログラム:

TEST(. . .,SEPARATE,. . .) を使用してコンパイルされたプログラムの場合、言語環境プログラムは、(DD 名 SYSDEBUG に書き込まれる) 別のデバッグ・データ・セットの名前をオブジェクト・プログラムから取得します。別のデバッグ・データ・セットの名前を変更するには、言語環境プログラムの COBOL デバッグ・ファイル出口を使用します。

関連タスク
305 ページの『デバッグ・データ・セットの定義 (SYSDEBUG)』
言語環境プログラム・デバッグのガイド (TERMTHDACT
を使用した言語環境プログラムのダンプの生成)
Debug Tool User's Guide (TEST ランタイム・オプションを使用した Debug Tool の使用)
言語環境プログラム・カスタマイズ (COBOL デバッグ・ファイル名の修正)

関連参照 301 ページの『論理レコード長とブロック・サイズ』
326 ページの『cob2 入出力ファイル』
344 ページの『矛盾するコンパイラー・オプション』
378 ページの『OPTIMIZE』
言語環境プログラム・プログラミング・リファレンス (TEST | NOTEST)

THREAD

THREAD は、COBOL プログラムで、複数の POSIX スレッドまたは PL/I タスクを持つ 言語環境プログラム・エンクレーブでの実行が有効化されることを示します。

THREAD オプションの構文

デフォルト: NOTHREAD

省略形: なし

THREAD オプションを指定してコンパイルされたプログラムは、スレッド化されていないアプリケーションでも使用することができます。ただし、スレッド化されたアプリケーションで COBOL プログラムを実行する場合は、言語環境プログラム・のエンクレーブ内のすべての COBOL プログラムが THREAD オプションを指定してコンパイルされている必要があります。

NOTHREAD を指定した場合、その COBOL プログラムは、複数の POSIX スレッドまたは PL/I タスクを含むエンクレーブで実行することができません。

Enterprise COBOL より前のコンパイラーでコンパイルされたプログラムは、NOTHREAD を指定してコンパイルされたものとして処理されます。

THREAD オプションが有効化されている場合には、次の項目はサポートされません。これらの言語エレメントが検出された場合は、エラーとして診断されます。

- ALTER ステートメント
- DEBUG-ITEM 特殊レジスター
- プロシージャー名が指定されていない GO TO ステートメント
- PROGRAM-ID 節の INITIAL 句
- ネストされたプログラム
- RERUN
- 分割モジュール
- SORT または MERGE ステートメント
- STOP リテラル・ステートメント
- USE FOR DEBUGGING ステートメント

また、スレッド化されている場合とスレッド化されていない場合では、一部の言語構成要素のセマンティクスが異なります。

スレッド化されたアプリケーションの場合、プログラミングおよび環境に関するさまざまな制約が適用されますが、スレッド化されていないアプリケーションでのプ

ログラムの使用についてはあまり制約がありません。例えば、THREAD オプションを指定してコンパイルされたプログラムは、実行時に複数の POSIX スレッドまたは PL/I タスクがアプリケーションに含まれていない場合は、 CICS 環境および IMS 環境で実行することができ、AMODE 24 で実行することも、マルチスレッド化をサポートしない他のプログラムとの間で呼び出しを行うこともできます。

THREAD オプションを指定してコンパイルされたプログラムは、言語環境プログラムの事前初期設定ルーチン CEEPIPI を呼び出すことにより作成された再使用可能環境でサポートされます。しかし、IGZERRE または ILBOSTP0 を呼び出すことにより作成された再使用可能環境や、RTEREUS ランタイム・オプションを使用して作成された再使用可能環境は、THREAD オプションを指定してコンパイルされたプログラムをサポートしません。

パフォーマンスの考慮: THREAD オプションを使用する場合は、自動的に生成される逐次化ロジックのオーバーヘッドが原因で、実行時のパフォーマンスが多少低下することがあります。

関連タスク
549 ページの『第 27 章 マルチスレッド化のための COBOL プログラムの準備』

関連参照
344 ページの『矛盾するコンパイラー・オプション』

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。
- Enterprise COBOL のもとで TRUNC(OPT) オプションを指定してコンパイルされたプログラムの結果が、同じプログラムを OS/VS COBOL のもとで NOTRUNC を指定してコンパイルした場合の結果と異なる場合があります。非ゼロの高位桁を実際に失わなければ、この違いは現れません。

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 (IMS、DB2、C/C++、 FORTRAN、および PL/I 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。

TRUNC の例 1

```
01  BIN-VAR     PIC S99 USAGE BINARY.
. . .
        MOVE 123451 to BIN-VAR
```

次の表に、MOVE 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	Display
送り出し側	123451	00|01|E2|3B	123451
受け取り側 TRUNC(STD)	51	00|33	51
受け取り側 TRUNC(OPT)	-7621	E2|3B	2J
受け取り側 TRUNC(BIN)	-7621	E2|3B	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からです。

TRUNC の例 2

```
01  BIN-VAR     PIC 9(6)  USAGE BINARY
. . .
        MOVE 1234567891 to BIN-VAR
```

次の表に、MOVE 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	Display
送り出し側	1234567891	49|96|02|D3	1234567891
受け取り側 TRUNC(STD)	567891	00|08|AA|53	567891
受け取り側 TRUNC(OPT)	567891	53|AA|08|00	567891
受け取り側 TRUNC(BIN)	1234567891	49|96|02|D3	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
54 ページの『数値データの形式』

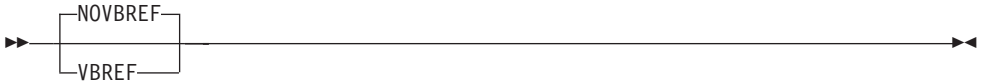
関連タスク
458 ページの『CICS オプションを使用したコンパイル』

関連参照
VALUE 節 (*Enterprise COBOL 言語解説書*)

VBREF

VBREF は、ソース・プログラムの中で使用されるすべての動詞、およびこれらの動詞が使用されている行番号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動詞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

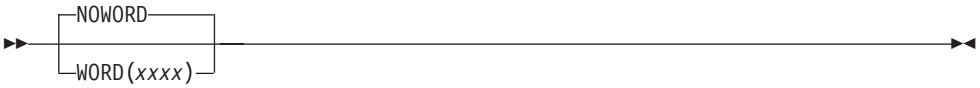
省略形: なし

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。

WORD

WORD(*xxxx*) は、コンパイル時に代替予約語テーブルを使用することを指定するために使用します。

WORD オプションの構文



デフォルト: NOWORD

省略形: WDINOWD

xxxx には、コンパイルで使用する予約語テーブル (IGYC*xxxx*) の名前の後半の文字を指定します。 IGYC は名前の最初の部分の標準 4 文字で、*xxxx* は 1 から 4 文字の長さにすることができます。

代替予約語テーブルを使用して、IBM 提供のデフォルトの予約語テーブルに対する変更を行います。システム・プログラマーが、設置場所に対して 1 つ以上の代替予約語テーブルを作成している可能性があります。代替予約語テーブルの名前については、システム・プログラマーにお問い合わせください。

Enterprise COBOL には、CICS アプリケーション専用の代替予約語テーブル (IGYCCICS) が用意されています。エラー・メッセージを使用して、CICS でサポートされない COBOL のワードにフラグを付けるようにセットアップされています。コンパイル時にこの CICS 予約語テーブルを使用したい場合は、コンパイラー・オプション WORD(CICS) を指定してください。

関連タスク
458 ページの『CICS オプションを使用したコンパイル』

関連参照
344 ページの『矛盾するコンパイラー・オプション』
463 ページの『CICS 予約語テーブル』

XMLPARSE

XMLPARSE は、XML 処理で使用するパーサー (したがってプログラムで使用可能になる XML 処理機能) を選択するために使用します。

XMLPARSE オプションの構文

デフォルト: XMLSS

省略形: XP(X)、XP(C)

XMLPARSE(XMLSS) オプションを指定すると、XML PARSE ステートメントは z/OS XML System Services パーサーを使用して処理されます。次の XML 構文解析機能は、XMLPARSE(XMLSS) オプションを指定した場合にのみ使用可能になります。

- 拡張名前空間処理 (特殊レジスター XML-NAMESPACE、XML-NNAMESPACE、XML-NAMESPACE-PREFIX、および XML-NNAMESPACE-PREFIX)
- 文書フラグメントの Unicode UTF-16 への自動変換を選択するための、XML PARSE ステートメントの RETURNING NATIONAL 句
- 入力文書のエンコードを指定するための、XML PARSE ステートメントの ENCODING 句
- UTF-8 でエンコードされた XML 文書の直接構文解析
- XML のバッファーごとの XML 文書の構文解析
- XML 構文解析の System z™ Application Assist Processors (zAAPs) へのオフロード

XMLPARSE(COMPAT) オプションを指定した場合、XML PARSE ステートメントは、COBOL ランタイムの組み込みコンポーネントである XML パーサーを使用して処理されます。 XML PARSE ステートメントの結果および操作上の動作は、Enterprise COBOL バージョン 3 互換です。 XMLPARSE(COMPAT) を指定した場合、Enterprise COBOL は、上述の XMLPARSE(XMLSS) の拡張機能をサポートしません。 XML PARSE ステートメントの RETURNING NATIONAL および ENCODING 句の構文は許容されません。

関連タスク

561 ページの『第 28 章 XML 入力の処理』

関連参照

XML PARSE ステートメント (*Enterprise COBOL 言語解説書*)

z/OS XML System Services User's Guide and Reference

XREF

XREF は、ソート済みの相互参照リストを入手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: XINOX

XREF、XREF(FULL)、または XREF(SHORT) を選択できます。サブオプションを何も指定しないで XREF を指定すると、XREF(FULL) が有効です。

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、およびそれらが定義されている行番号を表示するリストにセクションが含まれています。外部プログラム名が識別されます。

また、関連コピーブックを取得したデータ・セットまたは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。

EBCDIC データ名とプロシージャー名は英数字順にリストされます。DBCS データ名とプロシージャー名は、プログラムでの物理的順序に基づいてリストされ、DBCSXREF インストール・オプションが DBCS 配列プログラムで選択された場合を除いて、EBCDIC データ名とプロシージャー名の前に表示されます。DBCSXREF オプションが選択された場合は、DBCS データ名とプロシージャー名は、DBCS プログラムで指定されたとおりに順序付けられます。

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、DBCS 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
411 ページの『第 19 章 デバッグ』

関連タスク
422 ページの『リストの入手』

関連参照
言語環境プログラム・デバッグのガイド (COBOL コンパイラー・オプション)

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 進数

  これ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

  これは、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時点の年号より 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。

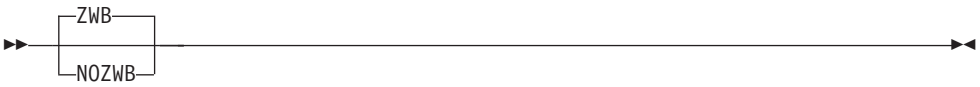
– 現在の年号がコンパイル単位の世紀ウィンドウ内にある。

例えば、現在の年号が 2007 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)、比較の際にその項目を使用しても、 ZWB の影響を受けることはありません。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。同じ COBOL ソース・プログラムでも、このオプションの設定によって結果が異なることがあります。

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 18 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のに役立つステートメントがいくつかあります。

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

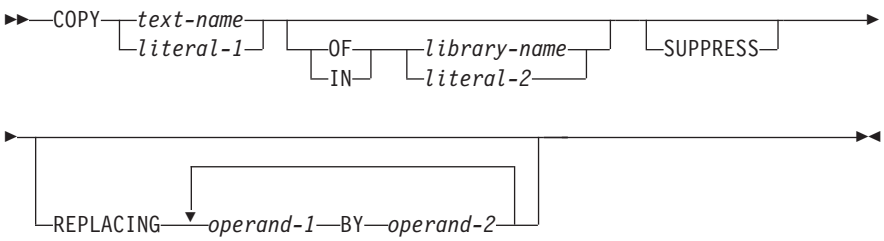
この拡張ソース・プログラム・ライブラリー・ステートメントは、コンパイルのソースとして、完全な COBOL プログラムを提供します。形式化および処理の規則については、COPY ステートメントの *text-name* の説明を参照してください。

***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。名前の *CONTROL と *CBL は同義語です。

COPY ステートメント

COPY ステートメントの構文



このライブラリー・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。ユーザー定義語を *text-name* または *library-name* と同じにすることができます。*text-name* および *library-name* の固有性は、システム依存名の形式化および変換の規則が適用された後で決められます。*library-name* を省略した場合は、SYSLIB と見なされます。

JCL によってコンパイルするとき:

text-name、*library-name*、および *literal* は、次のように処理されます。

- 名前 (1 文字から 30 文字までの長さ) は 8 文字に切り捨てられます。*text-name* および *library-name* の最初の 8 文字だけが、識別名として使用されます。これらの 8 文字は、1 つの COBOL ライブラリー内では固有でなければなりません。
- 名前は大文字に変換されます。
- 先頭文字でも最後の文字でもないハイフンはゼロ (0) に変換され、警告メッセージが出されます。

- 先頭文字が数値である場合、文字 1 から 9 は A から I へ変換され、ゼロ (0) は J へ変換され、警告メッセージが生成されます。

以下に例を示します。

```
COPY INVOICES1Q
COPY "Company-#Employees" IN Personellib
```

IN/OF 句の中の *library-name* は、コピー元の区分データ・セットを識別する DD 名です。次の例に示すように、DD ステートメントを使用して、*library-name* を定義します。

```
//COPYLIB DD DSNAME=ABC.COB,VOLUME=SER=111111,
//           DISP=SHR,UNIT=3380
```

複数のコピー・ライブラリーを指定するには、JCL または JCL と IN/OF 句の組み合わせのいずれかを使用してください。JCL だけを使用する場合は、SYSLIB に対する DD ステートメント上でデータ・セットを連結します。あるいは、複数の DD ステートメントを定義し、COPY ステートメントに IN/OF 句を組み込みます。

コピー・ライブラリーの最大ブロック・サイズは、データ・セットが常駐している装置によって決まります。

z/OS UNIX シェルでコンパイルするとき:

cob2 コマンドを使用してコンパイルすると、コピーブックは HFS から取り込まれます。*text-name*、*library-name*、および *literal* は、次のように処理されます。

- ユーザー定義語は大文字に変換されます。リテラルは変換されません。UNIX は大/小文字の区別をするので、ファイル名が小文字であるか大/小文字混合である場合は、それをリテラルとして指定します。
- *text-name* がリテラルのとき、*library-name* が省略されていると、*text-name* は直接的に使用されます。すなわち、ファイル名、相対パス名、または絶対パス名 (先頭文字が / の場合) として使用されます。以下に、その例を示します。

  ```
  COPY "MyInc"
  COPY "x/MyInc"
  COPY "/u/user1/MyInc"
  ```

- *text-name* がユーザー定義語で、その名前の環境変数が定義されていると、その環境変数の値が、コピーブックを含むファイルの名前として使用されます。

  その名前の環境変数が定義されていない場合には、コピーブックは、以下の名前として、この順に探索されます。

  1. *text-name*.cpy
  2. *text-name*.CPY
  3. *text-name*.cbl
  4. *text-name*.CBL
  5. *text-name*.cob
  6. *text-name*.COB
  7. *text-name*

- *library-name* がリテラルである場合は、それはコピー・ファイル *text-name* までの実際のパス (相対または絶対) として扱われます。
- *library-name* がユーザー定義語であると、それは環境変数として扱われます。その環境変数の値がパスとして使用されます。環境変数が設定されていないと、エラーとなります。
- *library-name* と *text-name* の両方が指定された場合、コンパイラーは、*library-name* と *text-name* の間にパス区切り文字 (*/*) を入れて 2 つの値を連結することによって、コピーブックのパス名を形成します。例えば、COPY MYCOPY OF MYLIB が次のように設定されているとします。

  ```
  export MYCOPY=mystuff/today.cpy
  export MYLIB=/u/user1
  ```

  この設定の結果、次のようになります。

  ```
  /u/user1/mystuff/today.cpy
  ```

library-name が環境変数で、この環境変数の値がコピーブックのコピー元のパスを識別する場合は、次の例に示すように、export コマンドを使用して *library-name* を定義します。

```
export COPYLIB=/u/mystuff/copybooks
```

環境変数の名前は大文字でなければなりません。複数のコピー・ライブラリーを指定するには、コロン (**:**) で区切った複数のパス名を、環境変数に設定してください。

library-name が省略されたときに、*text-name* が絶対パス名でないと、コピーブックは、次の順序で探索されます。

1. 現行ディレクトリー
2. -I cob2 オプションで指定されたパス
3. SYSLIB 環境変数で指定されたパス

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
コンパイラーはこのステートメントをコメントとして処理します。

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
このステートメントは、最外部の 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どのコンパイラー・オプションが使用されるのかを示します。

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。

SERVICE LABEL ステートメント

このステートメントは、制御の流れを示すために CICS 変換プログラムによって生成され、CEE3SRP の呼び出しのために再開点で使用する必要があります。一般的に使用されるものではありません。

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

7 ページの『ソース・リストのヘッダーの変更』
306 ページの『z/OS のもとでのコンパイラー・オプションの指定』
320 ページの『UNIX のもとでのコンパイラー・オプションの指定』
319 ページの『UNIX のもとでの環境変数の設定』
743 ページの『反復コーディングの除去』

関連参照

324 ページの『cob2 の構文およびオプション』
COPY ステートメント (*Enterprise COBOL 言語解説書*)

第 19 章 デバッグ

アプリケーションのプログラム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。

プログラムの問題が容易に検出されず、利用できるデバッガーがない場合、プログラムのストレージ・ダンプを分析する必要があります。

対話式デバッグの場合は、デバッグ・ツールを使用できます。デバッグ・ツールは、次のような生産性拡張機能を提供します。

- 対話式デバッグ (フルスクリーン・モードまたは行モード) またはバッチ・モードのデバッグ

  対話式フルスクリーン・モード・セッション中に、デバッグ・ツールのフルスクリーン・サービスと 3270 装置上のセッション・パネル・ウィンドウを使用して、実行中のプログラムをデバッグすることができます。
- COBOL 類似コマンド

  サポートされる高水準言語ごとに、ブレークポイントで取るべきアクションを指定するコマンドが、そのプログラム言語に類似した構文で提供されます
- 混合言語デバッグ

  異なる言語で作成されたプログラムが入っているアプリケーションをデバッグすることができます。デバッグ・ツールが、実行中のプログラムまたはサブプログラムの言語を自動的に判別します。
- COBOL-CICS デバッグ

  デバッグ・ツールは、対話式モードとバッチ・モードの両方で CICS アプリケーションのデバッグをサポートします。
- リモート・デバッグのサポート

  ワークステーション・ユーザーは、z/OS に常駐するプログラムをデバッグするために、Rational® Developer for System z を使用できます。

関連タスク

412 ページの『ソース言語によるデバッグ』
416 ページの『コンパイラー・オプションを使用したデバッグ』
422 ページの『デバッガーの使用』
422 ページの『リストの入手』
Debug Tool User's Guide

関連参照

Debug Tool Reference and Messages

言語環境プログラム・デバッグのガイド (システム・
ダンプのフォーマットと分析、サンプル COBOL プログラムのデバッグ)

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
『プログラム・ロジックのトレース』
413 ページの『入出力エラーの検出および処理』
414 ページの『データの妥当性検査』
414 ページの『初期化されていないデータの検出』
414 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 (*Enterprise COBOL 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。

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
24 ページの『範囲終了符号』

関連参照
DISPLAY ステートメント (*Enterprise COBOL 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
268 ページの『ERROR 宣言のコーディング』
269 ページの『ファイル状況キーの使用』

関連参照
状況キー (*Enterprise COBOL 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
102 ページの『条件式のコーディング』
157 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 (*Enterprise COBOL 言語解説書*)

初期化されていないデータの検出

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 (*Enterprise COBOL 言語解説書*)
SET ステートメント (*Enterprise COBOL 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・ルーチンが実行される回数、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:

- THREAD オプションを指定してコンパイルするプログラムでは、USE FOR DEBUGGING 宣言を使用できません。
- USE FOR DEBUGGING 宣言は、WITH DEBUGGING MODE 節が指定されていると、TEST(HOOK) 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。

『例: USE FOR DEBUGGING』

関連参照
SOURCE-COMPUTER 段落 (*Enterprise COBOL 言語解説書*)
デバッグ行 (*Enterprise COBOL 言語解説書*)
デバッグ・セクション (*Enterprise COBOL 言語解説書*)
DEBUGGING 宣言 (*Enterprise COBOL 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データ・セットに書き込みます。 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。COPY または BASIS ステートメントからのテキスト名およびライブラリー名、およびコピーブックを取得する関連データ・セットまたはファイル (XREF)
- DATA DIVISION 内のデータ項目 (MAP)
- 動詞参照 (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
『コーディング・エラーの検出』
418 ページの『行シーケンス問題の検出』
418 ページの『有効範囲の検査』
419 ページの『診断するエラーのレベルの選択』
421 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』
421 ページの『データ項目のリスト』
422 ページの『リストの入手』

関連参照 341 ページの『第 17 章 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。

TSO フォアグラウンドでコンパイルする場合、 TERM コンパイラー・オプションを使用し、ユーザーのデータ・セットを SYSTERM データ・セットとして定義することにより、メッセージを画面に送信することができます。

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
352 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
386 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: 所要のエレメントの有効アドレスが指定されたテーブルの最大境界内にあるかどうか。
- 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ることになります。

- 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。

参照されたデータが入っているデータ項目の範囲外にあるアドレスが生成された場合は、エラー・メッセージが生成され、プログラムは実行を停止します。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
390 ページの『SSRANGE』
735 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 49. **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。

420 ページの『例: 組み込みメッセージ』

関連タスク
315 ページの『コンパイル・エラー・メッセージのリストの生成』

関連参照

317 ページの『コンパイル・エラー・メッセージの重大度コード』
315 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』
363 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-|--+  Map and Cross Reference
...
  090671**             /
  090672**             ****************************************************************
  090673**             ***          I N I T I A L I Z E   P A R A G R A P H          **
  090674**             ***  Open files. Accept date, time and format header lines.    **
  090675**             ***   Load location-table.                                     **
  090676**             ****************************************************************
  090677**                100-initialize-paragraph.
  090678**                  move spaces to ws-transaction-record                                IMP 331
  090679**                  move spaces to ws-commuter-record                                   IMP 307
  090680**                  move zeroes to commuter-zipcode                                     IMP 318
  090681**                  move zeroes to commuter-home-phone                                  IMP 319
  090682**                  move zeroes to commuter-work-phone                                  IMP 320
  090683**                  move zeroes to commuter-update-date                                 IMP 324
  090684**                  open input update-transaction-file                                  204
==090684==> IGYPS2052-S An error was found in the definition of file "LOCATION-FILE".  The
                        reference to this file was discarded.
  090685**                       location-file                                                  193
  090686**                       i-o commuter-file                                              181
  090687**                       output print-file                                              217
  090688**                    if commuter-file-status not = "00" and not = "97"                 241
  090689**    1                 display "100-OPEN"
  090690**    1                 move 100 to comp-code                                          231
  090691**    1                 perform 500-vsam-error                                         91069
  090692**    1                 perform 900-abnormal-termination                               91114
  090693**                    end-if
  090694**                    accept ws-date from date                                        UND
==090694==> IGYPS2121-S "WS-DATE" was not defined as a data-name.  The statement was discarded.
  090695**                    move corr ws-date to header-date                                UND 455
==090695==> IGYPS2121-S "WS-DATE" was not defined as a data-name.  The statement was discarded.
  090696**                    accept ws-time from time                                        UND
==090696==> IGYPS2121-S "WS-TIME" was not defined as a data-name.  The statement was discarded.
  090697**                    move corr ws-time to header-time                                UND 449
==090697==> IGYPS2121-S "WS-TIME" was not defined as a data-name.  The statement was discarded.
  090698**                    read location-file                                              193
==090698==> IGYPS2053-S An error was found in the definition of file "LOCATION-FILE".  This
                        input/output statement was discarded.
  090699**                      at end
  090700**    1                     set location-eof to true                                  256
  090701**                    end-read
...
LineID  Message code  Message text
          IGYSC0090-W   1700 sequence errors were found in this program.
          IGYSC3002-I   A severe error was found in the program.  The "OPTIMIZE" compiler option was cancelled.
     160  IGYDS1089-S   "ASSIGNN" was invalid.  Scanning was resumed at the next area "A" item, level-number, or
                        the start of the next clause.
     193  IGYGR1207-S   The "ASSIGN" clause was missing or invalid in the "SELECT" entry for file "LOCATION-FILE".
                        The file definition was discarded.
     269  IGYDS1066-S   "REDEFINES" object "WS-DATE" was not the immediately preceding level-1 data item.
                        The "REDEFINES" clause was discarded.
   90602  IGYPS2052-S   An error was found in the definition of file "LOCATION-FILE".  The reference to this file
                        was discarded. Same message on line:  90684
   90694  IGYPS2121-S   "WS-DATE" was not defined as a data-name.  The statement was discarded.
                        Same message on line:  90695
   90696  IGYPS2121-S   "WS-TIME" was not defined as a data-name.  The statement was discarded.
                        Same message on line:  90697
   90698  IGYPS2053-S   An error was found in the definition of file "LOCATION-FILE".  This input/output statement
                        was discarded. Same message on line:  90709
Messages    Total    Informational    Warning    Error    Severe    Terminating
Printed:       13            1              1                11
* Statistics for COBOL program IGYTCARA:
*    Source records = 1735
```

```
*    Data Division statements = 287
*    Procedure Division statements = 471
End of compilation 1,  program IGYTCARA,  highest severity 12.
Return code 12
```

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データ・セットまたはファイルへの、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下の関連参照を参照してください。

445 ページの『例: XREF 出力: データ名相互参照』
446 ページの『例: XREF 出力: プログラム名相互参照』
446 ページの『例: XREF 出力: COPY/BASIS 相互参照』
447 ページの『例: 組み込み相互参照』

関連タスク
422 ページの『リストの入手』

関連参照
402 ページの『XREF』

データ項目のリスト

MAP コンパイラー・オプションを使用すると、DATA DIVISION の項目と暗黙的に宣言されたすべての項目のリストを作成することができます。システム・ダンプのデータ項目の内容を突き止める場合に、MAP 出力を使用してください。

MAP オプションを使用すると、圧縮された MAP 情報を含んでいる組み込み MAP 要約が、COBOL ソース・データ宣言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 MAPINOMAP (または *CBL MAPINOMAP) 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。以下に例を示します。

```
*CONTROL NOMAP
    01  A
    02  B
*CONTROL MAP
```

427 ページの『例: MAP 出力』

関連タスク
『リストの入手』

関連参照
370 ページの『MAP』

デバッガーの使用

デバッグ・ツールを使用して、Enterprise COBOL プログラムをデバッグすることができます。 TEST コンパイラー・オプションを使用すれば、デバッガーによって実行可能プログラムをステップスルーできるように、COBOL プログラムを準備することができます。

リモート・デバッグの場合、Rational Developer for System z のデバッグ・パースペクティブにより、z/OS または UNIX のもとで実行されるデバッグ・ツール・エンジンが提供するデバッグ情報にアクセスするためのクライアント・グラフィカル・ユーザー・インターフェースが提供されます。

TEST のサブオプション SEPARATE を指定すると、デバッグ・ツール用の記号情報テーブルをオブジェクト・モジュールとは別個のデータ・セットに生成することができます。また、TEST(NOHOOK,. . .) オプションでコンパイルすれば、コンパイル・フックではなくオーバーレイ・フックを使用して COBOL プログラムのデバッグを可能にすることもできます (*実動デバッグ*)。 (コンパイル・フックを使用すると、ランタイム TEST オプションがオフのときでも、パフォーマンスはある程度低下します。)

ほとんどのデバッグ機能を取得するには、NOOPTIMIZE および TEST(HOOK,. . .) コンパイラー・オプションを指定します。

最適パフォーマンスと対比した最大デバッグ機能を得るために使用するコンパイラー・オプションの詳細については、TEST 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
305 ページの『デバッグ・データ・セットの定義 (SYSDEBUG)』
Debug Tool User's Guide (デバッグのためのプログラムの準備)

関連参照
392 ページの『TEST』

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 50. **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。	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]	NOSOURCE、NOXREF、NOVBREF、NOMAP、NOOFFSET、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	387 ページの『SOURCE』
ストレージ・ダンプ内で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)	370 ページの『MAP』 [2]
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、動詞が 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データ・セットまたは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・データ・セットまたは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データ・セットまたはファイル 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	402 ページの『XREF』 [2,3]
プログラム内の障害のある動詞を見つける。または、プログラムの実行時に移動されたデータ項目のストレージ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	369 ページの『LIST』 [2,4]
PROCEDURE DIVISION セクションを移動または追加した後でも有効な論理パスがまだあるか検査する。	圧縮 PROCEDURE DIVISION リスト	圧縮された動詞リスト、グローバル・テーブル、WORKING-STORAGE 情報、およびリテラル	377 ページの『OFFSET』

表 50. **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
特定の動詞のインスタンスを見つける。	アルファベット順の動詞	使用されたそれぞれの動詞、各動詞が使用された回数、各動詞が使用された行番号	400 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. ソースに *CONTROL LIST および *CONTROL NOLIST (または *CBL LIST および *CBL NOLIST) ステートメントを入れることによって、生成されるオブジェクト・コードの選択的リストを制御することができます。 *CONTROL ステートメントは PROCESS (または CBL) ステートメントと異なることに注意してください。

   出力が生成されるのは、以下の場合です。

   - COMPILE オプションを指定している (または NOCOMPILE(*x*) オプションが有効であり、エラー・レベル *x* 以上が発生していない)。
   - OFFSET オプションを指定していない。OFFSET と LIST は互いに排他的なオプションで、OFFSET の方が優先されます。

425 ページの『例: 短縮リスト』
427 ページの『例: SOURCE および NUMBER 出力』
427 ページの『例: MAP 出力』
429 ページの『例: 組み込みマップ要約』
432 ページの『例: ネストされたプログラム・マップ』
445 ページの『例: XREF 出力: データ名相互参照』
446 ページの『例: XREF 出力: プログラム名相互参照』
446 ページの『例: XREF 出力: COPY/BASIS 相互参照』
447 ページの『例: 組み込み相互参照』
448 ページの『例: OFFSET コンパイラー出力』
449 ページの『例: VBREF コンパイラー出力』

関連タスク

315 ページの『コンパイル・エラー・メッセージのリストの生成』
432 ページの『LIST 出力の読み取り』
言語環境プログラム・デバッグのガイド (COBOL プログラムのデバッグ)

関連参照

315 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。

```
Invocation parameters:      (1)
 OPTFILE
PROCESS(CBL) statements:    (2)
 CBL NODECK
 CBL NOADV,NODYN,NONAME,NONUMBER,QUOTE,SEQ,DUMP
 CBL NOSOURCE, NOXREF, NOVBREF, NOMAP, NOOFFSET, NOLIST
Options from SYSOPTF:       (3)
 C,NODU,FLAG(I),X,MAP,NOLIST,RENT,OPT,SSR
 TEST(NOHOOK,SEP)  TRUNC(OPT)
Options in effect:          (4)
 NOADATA
 NOADV
   QUOTE
   ARITH(COMPAT)
 NOAWO
   BUFSIZE(4096)
 NOCICS
   CODEPAGE(1140)
   COMPILE
 NOCURRENCY
   DATA(31)
 NODATEPROC
   DBCS
 NODECK
 NODIAGTRUNC
 NODLL
   DUMP
 NODYNAM
 NOEXIT
 NOEXPORTALL
 NOFASTSRT
   FLAG(I)
 NOFLAGSTD
   INTDATE(ANSI)
   LANGUAGE(EN)
 NOLIB
   LINECOUNT(60)
 NOLIST
 NOMAP
 NOMDECK
 NONAME
   NSYMBOL(NATIONAL)
 NONUMBER
   NUMPROC(NOPFD)
   OBJECT
 NOOFFSET
   OPTIMIZE(STD)
   OUTDD(SYSOUT)
   PGMNAME(COMPAT)
   RENT
   RMODE(AUTO)
   SEQUENCE
   SIZE(MAX)
 NOSOURCE
   SPACE(1)
 NOSQL
   SQLCCSID
   SSRANGE
 NOTERM
   TEST(NOHOOK,SEPARATE,NOEJPD)
 NOTHREAD
   TRUNC(OPT)
 NOVBREF
 NOWORD
   XMLPARSE(XMLSS)
 NOXREF
   YEARWINDOW(1900)
   ZWB

LineID  Message code  Message text   (5)
        IGYDS0139-W   Diagnostic messages were issued during processing of compiler options.
                      These messages are located at the beginning of the listing.
```

```
       IGYSC0090-W   3 sequence errors were found in this program.
   160 IGYDS1089-S   "ASSIGNN" was invalid.  Scanning was resumed at the next area "A" item,
                     level-number,or the start of the next clause.
   193 IGYGR1207-S   The "ASSIGN" clause was missing or invalid in the "SELECT" entry for file
                     "LOCATION-FILE".  The file definition was discarded.
   269 IGYDS1066-S   "REDEFINES" object "WS-DATE" was not the immediately preceding level-1 data item.
                     The "REDEFINES" clause was discarded.
   901 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:   1008
Messages    Total    Informational    Warning    Error    Severe    Terminating   (6)
Printed:    14                        3                   11
  * Statistics for COBOL program IGYTCARA:    (7)
*    Source records = 1735
*    Data Division statements = 287
*    Procedure Division statements = 471
End of compilation 1,  program IGYTCARA,  highest severity 12.  (8)
Return code 12
```

(1) コンパイラー呼び出し時にコンパイラーに渡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、オプションが渡されていない場合には表示されません。

OPTFILE
SYSOPTF データ・セットからオプションを要求します。

(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。

NOOFFSET
PROCEDURE DIVISION の圧縮されたリストを抑制します。

NOMAP DATA DIVISION で定義された項目のマップ・レポートを抑制します。

(3) SYSOPTF データ・セットから取得したオプション (OPTFILE コンパイラー・オプションが指定されたため)。

NOLIST ソース・コードのアセンブラー言語拡張を抑制します。

TEST(NOHOOK,SEP)
プログラムは、デバッグ・ツールまたは定様式ダンプで使用するためにコンパイルされました。

(4) このコンパイルの開始時のオプションの状況。

(5) 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、リストの先頭に示されます。

(6) 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。

(7) プログラム IGYTCARA のプログラム統計。

(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。リストに示された注釈番号は、後続の説明の番号と対応しています。

```
                (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 IGYTCARA - Beginning"
               087900     perform 050-create-vsam-master-file.                              90633
               088150     display "perform 050-create-vsam-master finished".
088151**        088125     perform 100-initialize-paragraph                                  90677
               088200     display "perform 100-initialize-paragraph finished"
               088300     read update-transaction-file into ws-transaction-record           204 331
               088400         at end
             1 088500       set transaction-eof to true                                   254
               088600     end-read
               088700     display "READ completed"
               088800     perform until transaction-eof                                     254
             1 088900       display "inside perform until loop"
             1 089000       perform 200-edit-update-transaction                           90733
             1 089100       display "After perform 200-edit   "
             1 089200       if no-errors                                                  365
             2 089300         perform 300-update-commuter-record                          90842
             2 089400         display "After perform 300-update "
             1 089650       else
089651**     2 089600         perform 400-print-transaction-errors                        90995
             2 089700         display "After perform 400-errors "
             1 089800       end-if
             1 089900       perform 410-re-initialize-fields                              91056
             1 090000       display "After perform 410-reinitialize"
             1 090100       read update-transaction-file into ws-transaction-record       204 331
             1 090200           at end
             2 090300         set transaction-eof to true                                 254
             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                          S = Spanned file
    E = EXTERNAL                      O = Has OCCURS clause               U = Undefined format file
    F = Fixed-length file             OG= Group has own length definition V = Variable-length file
    FB= Fixed-length blocked file     R = REDEFINES                       VB= Variable-length blocked file

(2)     (3)  (4)                      (5)        (6)   (7)         (8)          (9)           (10)
Source   Hierarchy and                Base       Hex-Displacement  Asmblr Data                Data Def
LineID   Data Name                    Locator    Blk   Structure   Definition   Data Type     Attributes
     4  PROGRAM-ID IGYTCARA----------------------------------------------------------------------------*
   181   FD COMMUTER-FILE                                                      VSAM           F
   183   1  COMMUTER-RECORD           BLF=00000  000              DS 0CL80     Group
   184     2  COMMUTER-KEY           BLF=00000  000   0 000 000  DS 16C       Display
   185     2  FILLER                 BLF=00000  010   0 000 010  DS 64C       Display
   187   FD COMMUTER-FILE-MST                                                  VSAM           F
   189   1  COMMUTER-RECORD-MST       BLF=00001  000              DS 0CL80     Group
```

```
190     2  COMMUTER-KEY-MST            BLF=00001  000   0 000 000   DS 16C       Display
191     2  FILLER                      BLF=00001  010   0 000 010   DS 64C       Display
193   FD LOCATION-FILE                                                QSAM           FB
198   1  LOCATION-RECORD               BLF=00002  000               DS 0CL80     Group
199     2  LOC-CODE                    BLF=00002  000   0 000 000   DS 2C        Display
200     2  LOC-DESCRIPTION             BLF=00002  002   0 000 002   DS 20C       Display
201     2  FILLER                      BLF=00002  016   0 000 016   DS 58C       Display
204   FD UPDATE-TRANSACTION-FILE                                      QSAM           FB
209   1  UPDATE-TRANSACTION-RECORD     BLF=00003  000               DS 80C       Display
217   FD PRINT-FILE                                                   QSAM           FB
222   1  PRINT-RECORD                  BLF=00004  000               DS 121C      Display
229   1  WORKING-STORAGE-FOR-IGYCARA   BLW=00000  000               DS 1C        Display
231   77 COMP-CODE                     BLW=00000  008               DS 2C        Binary
232   77 WS-TYPE                       BLW=00000  010               DS 3C        Display
235   1  I-F-STATUS-AREA               BLW=00000  018               DS 0CL2      Group
236     2  I-F-FILE-STATUS             BLW=00000  018   0 000 000   DS 2C        Display
237     88 I-O-SUCCESSFUL
240   1  STATUS-AREA                   BLW=00000  020               DS 0CL8      Group
241     2  COMMUTER-FILE-STATUS        BLW=00000  020   0 000 000   DS 2C        Display
242     88 I-O-OKAY
243     2  COMMUTER-VSAM-STATUS        BLW=00000  022   0 000 002   DS 0CL6      Group
244       3  VSAM-R15-RETURN-CODE      BLW=00000  022   0 000 002   DS 2C        Binary
245   77 UNUSED-DATA-ITEM              BLW=XXXXX  022               DS 10C       Display  (11)
```

(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) このデータ項目に使用されるベース・ロケーター。

(6) ベース・ロケーター値の先頭からの 16 進変位。

(7) 収容構造の先頭からの 16 進変位。

(8) データが定義されている方法を示す、疑似アセンブラー・コード。構造に可変長フィールドが含まれている場合、構造の最大長が示されます。

(9) データ型および使用法。

(10) 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。

(11) UNUSED-DATA-ITEM は PROCEDURE DIVISION で参照されませんでした。OPTIMIZE(FULL) が指定されていたため、UNUSED-DATA-ITEM は削除され、その結果、ベース・ロケーターは XXXXX に設定されました。

429 ページの『例: 組み込みマップ要約』
432 ページの『例: ネストされたプログラム・マップ』

関連参照

429 ページの『MAP 出力で使用される用語』
431 ページの『LIST および MAP 出力で使用される記号』

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.    IGYTCARA.
  . . .
000177    Data division.
000178    File section.
000179
000180
000181    FD  COMMUTER-FILE
000182        record 80 characters.                                    (1)       (2) (3)       (4)
  . . .
000222      01 print-record                  pic x(121).               BLF=00004+000           121C
  . . .
000228    Working-storage section.
000229      01 Working-storage-for-IGYCARA    pic x.                    BLW=00000+000           1C
000230
000231      77 comp-code                     pic S9999 comp.           BLW=00000+008           2C
000232      77 ws-type                       pic x(3)   value spaces.  BLW=00000+010           3C
000233
000234
000235      01 i-f-status-area.                                         BLW=00000+018           0CL2
000236        05 i-f-file-status             pic x(2).                 BLW=00000+018,0000000   2C
000237          88 i-o-successful            value zeroes.
000238
000239
000240      01 status-area.                                             BLW=00000+020           0CL8
000241        05 commuter-file-status        pic x(2).                 BLW=00000+020,0000000   2C
000242          88 i-o-okay                  value zeroes.
000243        05 commuter-vsam-status.                                  BLW=00000+022,0000002   0CL6
000244          10 vsam-r15-return-code      pic 9(2) comp.            BLW=00000+022,0000002   2C
000245          10 vsam-function-code        pic 9(1) comp.            BLW=00000+024,0000004   2C
000246          10 vsam-feedback-code        pic 9(3) comp.            BLW=00000+026,0000006   2C
000247
000248      77 update-file-status            pic xx.                   BLW=00000+028           2C
000249      77 loccode-file-status           pic xx.                   BLW=00000+030           2C
000250      77 updprint-file-status          pic xx.                   BLW=00000+038           2C
000251
000252      01 flags.                                                  BLW=00000+040           0CL3
000253        05 transaction-eof-flag        pic x value space.        BLW=00000+040,0000000   1C
000254            88 transaction-eof         value "Y".
000255        05 location-eof-flag           pic x value space.        BLW=00000+041,0000001   1C
000256            88 location-eof            value "Y".
000257        05 transaction-match-flag      pic x.                    BLW=00000+042,0000002   1C
  . . .
000876    procedure division.
000877      000-do-main-logic.
000878        display "PROGRAM IGYTCARA - Beginning"
000879        perform 050-create-vsam-master-file.
```

(1) このデータ項目に使用されるベース・ロケーター。

(2) ベース・ロケーター値の先頭からの 16 進変位。

(3) 収容構造の先頭からの 16 進変位。

(4) データが定義されている方法を示す、疑似アセンブラー・コード。

関連参照

431 ページの『LIST および MAP 出力で使用される記号』

MAP 出力で使用される用語

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を説明しています。

表 51. **MAP 出力で使用される用語**

用語	定義	説明
ALPHABETIC	DS *n*C	英字データ項目 (PICTURE A)

表 51. **MAP 出力で使用される用語** *(続き)*

用語	定義	説明
ALPHA-EDIT	DS *n*C	英字編集データ項目
AN-EDIT	DS *n*C	英数字編集データ項目
BINARY	DS 1H[2]、1F[2]、2F[2]、2C、 4C、または 8C	バイナリー・データ項目 (USAGE BINARY、COMPUTATIONAL、または COMPUTATIONAL-5)
COMP-1	DS 4C	単精度内部浮動小数点データ項目 (USAGE COMPUTATIONAL-1)
COMP-2	DS 8C	倍精度内部浮動小数点データ項目 (USAGE COMPUTATIONAL-2)
DBCS	DS *n*C	DBCS データ項目 (USAGE DISPLAY-1)
DBCS-EDIT	DS *n*C	DBCS 編集済みデータ項目 (USAGE DISPLAY-1)
DISP-FLOAT	DS *n*C	表示浮動小数点データ項目 (USAGE DISPLAY)
DISPLAY	DS *n*C	英数字データ項目 (PICTURE X)
DISP-NUM	DS *n*C	ゾーン 10 進数データ項目 (USAGE DISPLAY)
DISP-NUM-EDIT	DS *n*C	数字編集データ項目 (USAGE DISPLAY)
FD		ファイル定義
FUNCTION-PTR	DS *n*C	関数ポインター (USAGE FUNCTION-POINTER)
GROUP	DS 0CL*n*[1]	固定長英数字グループ・データ項目
GRP-VARLEN	DS 0CL*n*[1]	可変長英数字グループ・データ項目
INDEX	DS *n*C	指標データ項目 (USAGE INDEX)
INDEX-NAME	DS *n*C	索引名
NATIONAL	DS *n*C	カテゴリー国別データ項目 (USAGE NATIONAL)
NAT-EDIT	DS *n*C	国別編集データ項目 (USAGE NATIONAL)
NAT-FLOAT	DS *n*C	国別浮動小数点データ項目 (USAGE NATIONAL)
NAT-GROUP	DS 0CL*n*[1]	国別グループ (GROUP-USAGE NATIONAL)
NAT-GRP-VARLEN	DS 0CL*n*[1]	国別可変長グループ (GROUP-USAGE NATIONAL)
NAT-NUM	DS *n*C	国別 10 進数データ項目 (USAGE NATIONAL)
NAT-NUM-EDIT	DS *n*C	国別数字編集データ項目 (USAGE NATIONAL)
OBJECT-REF	DS *n*C	オブジェクト参照データ項目 (USAGE OBJECT REFERENCE)
PACKED-DEC	DS *n*P	内部 10 進データ項目 (USAGE PACKED-DECIMAL または COMPUTATIONAL-3)
POINTER	DS *n*C	ポインター・データ項目 (USAGE POINTER)
PROCEDURE-PTR	DS *n*C	プロシージャー・ポインター (USAGE PROCEDURE-POINTER)
SD		ソート・ファイル定義
VSAM, QSAM, LINESEQ		ファイル処理方式
1 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

1. *n* は、固定長グループのバイト単位のサイズ、および可変長グループのバイト単位の最大サイズです。
2. SYNCHRONIZED 節が表示されると、これらのフィールドが使用されます。

LIST および MAP 出力で使用される記号

次の表は、LIST または MAP オプションによって作成されるリストで使用される記号を説明しています。

表 52. **LIST および MAP 出力で使用される記号**

記号	定義
APBdisp=n[1]	すべての添え字パラメーター・ブロックの変位
AVN=n[1]	ALTER ステートメントの変数名セル
BL=n[1]	特殊レジスターのベース・ロケーター
BLA=n[1]	英数字一時記憶域 [4] のベース・ロケーター
BLF=n[1]	ファイルのベース・ロケーター
BLK=n[1]	LOCAL-STORAGE のベース・ロケーター
BLL=n[1]	LINKAGE SECTION のベース・ロケーター
BLM=n[1]	ファクトリー・データのベース・ロケーター
BLO=n[1]	オブジェクト・インスタンス・データのベース・ロケーター
BLS=n[1]	ソート項目のベース・ロケーター
BLT=n[1]	XML-TEXT および XML-NTEXT のベース・ロケーター
BLV=n[1]	位置が変更できるデータのベース・ロケーター
BLW=n[1]	WORKING-STORAGE のベース・ロケーター
BLX=n[1]	外部データのベース・ロケーター
CBL=n[1]	定数グローバル・テーブル (CGT) のベース・ロケーター
CLLE=@=n[1]	TGT におけるロード・リスト入り口アドレス
CLO=n[1]	クラス・オブジェクト・セル
DOV=n[1]	DSA オーバーフロー・セル
EVALUATE=n[1]	ブール・セルを評価する
FCB=n[1]	ファイル制御ブロック (FCB) のアドレス
GN=n(hhhhh)[2,]	生成されたプロシージャー名と、16 進数のそのオフセット
IDX=n[1]	指標名のベース・ロケーター
IDX=n[1]	指標セル番号
ILS=n[1]	LOCAL-STORAGE テーブルまたはインスタンス変数の指標セル
ODOSAVE=n[1]	ODO 保管セル番号
OPT=nnnn[3]	最適化プログラムの一時記憶セル
PBL=n[1]	プロシージャー・コードのベース・ロケーター
PFM=n[1]	PERFORM n 回セル
PGMLIT AT + nnnn[3]	リテラル・プールの先頭からのプログラム・リテラルの変位
PSV=n[1]	保管セル番号を実行する
PVN=n[1]	PERFORM ステートメントの変数名セル
RBKST=n[1]	レジスター逆保管セル
SFCB=n[1]	外部ファイルの 2 次ファイル制御ブロック
SYSLIT AT + nnnn[3]	システム・リテラル・プールの先頭からのシステム・リテラルの変位
TGT FDMP TEST INFO.AREA + nnnn[3]	FDUMP/TEST 情報域

表 52. **LIST および MAP 出力で使用される記号** *(続き)*

記号	定義
TGTFIXD + nnnn[3]	タスク・グローバル・テーブル (TGT) の固定部分の先頭からのオフセット
TOV=n[1]	TGT オーバーフロー・セル番号
TS1=aaaa	サブプール 1 の一時記憶セル番号
TS2=aaaa	サブプール 2 の一時記憶セル番号
TS3=aaaa	サブプール 3 の一時記憶セル番号
TS4=aaaa	サブプール 4 の一時記憶セル番号
V (ルーチン名)	外部ルーチンのアセンブラー VCON
VLC=n[1]	可変長名セル番号 (ODO)
VNI=n[1]	変数名初期化
WHEN=n[1]	WHEN セル番号を評価する

1. n は記入項目の番号です。ベース・ロケーターの場合は XXXXX になることもあります。これは、OPTIMIZE(FULL) 処理によって削除されたデータ項目を表します。
2. (hhhhh) は 16 進数のプログラム・オフセットです。
3. nnnn は、記入項目の先頭からの 10 進数のオフセットです。
4. 英数字一時記憶域は、英数字組み込み関数および英数字 EVALUATE ステートメントのサブジェクトを処理する際に使用される一時データ値です。

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
  Program Attribute codes (rightmost column) have the following meanings:
          C = COMMON
          I = INITIAL (1)
          U = PROCEDURE DIVISION USING...                          (5)
  Source Nesting                                                  Program
  LineID Level   Program Name from PROGRAM-ID paragraph           Attributes
      2     0   NESTMAIN. . . . . . . . . . . . . . . . . . . . . U
    120     1 (4) SUBPRO1 . . . . . . . . . . . . . . . . . . . . I,C,U
 (2)199     2       NESTED1 . . . . . . . . . . . . . . . . . . . I,C,U
    253     1     SUBPRO2 . . . . . . . . . . . . . . . . . . . . U
    335     2       NESTED2 . . . . . . . . . . . . . . . . . . . C,U
           (3)
```

(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。

LIST 出力の読み取り

LIST コンパイラー出力の各部分は、プログラムのデバッグに有用である場合があります。

LIST コンパイラー・オプションは、次の 7 つの出力を作成します。

- プログラムの初期化コードのアセンブラー・リスト (プログラム・シグニチャー情報バイト)。このリストからは、次のようなプログラム特性を検査することができます。
  - 有効なコンパイラー・オプション
  - 存在するデータ項目のタイプ
  - PROCEDURE DIVISION で使用されている動詞
- プログラムのソース・コードのアセンブラー・リスト

  異常終了発生時に実行されていた命令のストレージ内のアドレスから、その命令に対応する COBOL 動詞を見つけることができます。障害のある命令のアドレスが見つかったら、アセンブラー・リストに進み、その命令が生成される対象となった動詞を調べてください。
- オブジェクト・モジュールでのコンパイラー生成テーブルの位置
- タスク・グローバル・テーブル (TGT) のマップ (プログラム・グローバル・テーブル (PGT) および定数グローバル・テーブル (CGT) に関する情報を含む)

  TGT を使用すると、プログラムが実行されている環境に関する情報を見つけることができます。
- WORKING-STORAGE および制御ブロックの位置とサイズに関する情報

  LIST 出力の WORKING-STORAGE 部分を使用すると、WORKING-STORAGE で定義されているデータ項目の位置がわかります (RENT オプションを指定してコンパイルしたプログラムの場合は、WORKING STORAGE の先頭位置は示されません。)
- 動的保管域 (DSA) のマップ

  DSA (スタック・フレーム とも呼ばれる) のマップには、別個のコンパイル済みプロシージャーに入るたびに獲得されたストレージの内容に関する情報が入れられます。
- 動的ストレージの使用についてのリテラルとコードの位置に関する情報

LIST 出力を理解するために、アセンブラー言語でプログラミングができる必要はありません。アセンブラー・コードの大部分を伴うコメントは、コードが実行する機能を概念的に理解するのに役立ちます。

434 ページの『例: プログラム初期化コード』
441 ページの『例: ソース・コードから生成されるアセンブラー・コード』
443 ページの『例: TGT メモリー・マップ』
444 ページの『例: DSA メモリー・マップ』
444 ページの『例: WORKING-STORAGE の位置とサイズ』

関連参照

435 ページの『シグニチャー情報バイト: コンパイラー・オプション』
437 ページの『シグニチャー情報バイト: DATA DIVISION』
438 ページの『シグニチャー情報バイト: ENVIRONMENT DIVISION』
438 ページの『シグニチャー情報バイト: PROCEDURE DIVISION 動詞』
440 ページの『シグニチャー情報バイト: 多数の PROCEDURE DIVISION 項目』
言語環境プログラム・プログラミング・ガイド (スタック・ストレージの概要)

例: プログラム初期化コード

プログラム初期化コードのリストは、COBOL ソース・プログラムの特性に関する情報を提供します。プログラムの特性を検査するには、プログラム・シグニチャー情報バイトを解釈してください。

```
 (1)     (2)                  (3)                        (4)
000000          IMIN         DS    0H                   PROGRAM:IMIN
                             USING *,15
000000  47F0 F028            B     40(,15)              BYPASS CONSTANTS. BRANCH TO @STM
000004  00                   DC    AL1(0)               ZERO NAME LENGTH FOR DUMPS
000005  C3C5C5               DC    CL3'CEE'             CEE EYE CATCHER                        (5)
000008  00000110             DC    X'00000110'          STACK FRAME SIZE
00000C  00000014             DC    A(@PPA1-IMIN)        OFFSET TO PPA1 FROM PRIMARY ENTRY
000010  47F0 F001            B     1(,15)               RESERVED
000014                @PPA1  DS    0H                   PPA1 STARTS HERE
000014  98                   DC    X'98'                OFFSET TO LENGTH OF NAME FROM PPA1
000015  CE                   DC    X'CE'                CEL SIGNATURE
000016  AC                   DC    X'AC'                CEL FLAGS: '10101100'B
000017  00                   DC    X'00'                MEMBER FLAGS FOR COBOL
000018  000000B6             DC    A(@PPA2)             ADDRESS OF PPA2
00001C  00000000             DC    F'0'                 OFFSET TO THE BDI (NONE)
000020  00000000             DC    F'0'                 ADDRESS OF ENTRY POINT DESCRIPTORS
000024  0000                 DC    X'0000'              RESERVED
000026  00                   DC    X'00'                DSA FPR 8-15 SAVE AREA OFFSET/16
000027  00                   DC    X'00'                DSA FPR 8-15 SAVE AREA BIT MASK
000028                @STM   DS    0H                   STM STARTS HERE
000028  90EC D00C            STM   14,12,12(13)      @STM: SAVE CALLER'S REGISTERS
00002C  4110 F038            LA    1,56(,15)            GET ADDRESS OF PARMLIST INTO R1
000030  98EF F04C            LM    14,15,76(15)         LOAD ADDRESSES FROM @BRVAL
000034  07FF                 BR    15                   DO ANY NECESSARY INITIALIZATION
000036  0000                 DC    AL2'0'               AVAILABLE HALF-WORD
000038                @MAINENT DS    0H                   PRIMARY ENTRY POINT ADDRESS
000038  00000000             DC    A(IMIN)         @PARMS: 1) PRIMARY ENTRY POINT ADDRESS
00003C  00000000             DC    AL4'0'               2) Available
000040  000003C0             DC    A(DAB)               3) DAB ADDRESS                         (6)
000044  000000AE             DC    A(@EPNAM)            4) ENTRY POINT NAME ADDRESS
000048  00000000             DC    A(IMIN)              5) CURRENT ENTRY POINT ADDRESS
00004C  00000272             DC    A(START)        @BRVAL: 6) PROCEDURE CODE ADDRESS
000050  00000000             DC    V(IGZCBSO)           7) INITIALIZATION ROUTINE
000054  000000CA             DC    A(@CEEPARM)          8) ADDRESS OF PARM LIST FOR CEEINT
000058  00104001             DC    X'00104001'          DSA WORD 0 CONSTANT
00005C  00000000             DC    AL4'0'               AVAILABLE WORD
000060  00000000             DC    AL4'0'               AVAILABLE WORD
000064  00000000             DC    AL4'0'               AVAILABLE WORD
000068  F2F0F0F7             DC    CL4'2007'     @TIMEVRS: YEAR OF COMPILATION                 (7)
00006C  F0F9F3F0             DC    CL4'0930'            MONTH/DAY OF COMPILATION               (8)
000070  F1F0F4F8             DC    CL4'1048'            HOURS/MINUTES OF COMPILATION           (9)
000074  F1F6                 DC    CL2'16'              SECONDS FOR COMPILATION DATE
000076  F0F4F0F1F0F0         DC    CL6'040100'          VERSION/RELEASE/MOD LEVEL OF PROD      (10)
00007C  0474                 DC    X'0474'              UNSIGNED BINARY CODE PAGE CCSID VALUE (11)
00007E  0000                 DC    AL2'0'               AVAILABLE HALF-WORD
000080  0000                 DC    X'0000'              INFO. BYTES 28-29                      (12)
000082  076C                 DC    X'076C'              SIGNED BINARY YEARWINDOW OPTION VALUE
000084  A0487C4C2000         DC    X'A0487C4C2000'      INFO. BYTES 1-6
00008A  000000080000         DC    X'000000080000'      INFO. BYTES 7-12
000090  000000000800         DC    X'000000000800'      INFO. BYTES 13-18                      (12)
000096  0000000000           DC    X'0000000000'        INFO. BYTES 19-23
00009B  00                   DC    X'00'                COBOL SIGNATURE LEVEL
00009C  00000001             DC    X'00000001'          # DATA DIVISION STATEMENTS            (13)
0000A0  00000003             DC    X'00000003'          # PROCEDURE DIVISION STATEMENTS        (14)
0000A4  000080               DC    X'000080'            INFO. BYTES 24-26                      (12)
0000A7  00                   DC    X'00'                INFO. BYTE 27
0000A8  40404040             DC    C'    '              USER LEVEL INFO (LVLINFO)              (15)
0000AC  0004                 DC    X'0004'              LENGTH OF PROGRAM NAME
0000AE                @EPNAM DS    0H                   ENTRY POINT NAME
0000AE  C9D4C9D540404040     DC    C'IMIN    '          PROGRAM NAME                           (16)
0000B6                @PPA2  DS    0H                   PPA2 STARTS HERE
0000B6  05                   DC    X'05'                CEL MEMBER IDENTIFIER
0000B7  00                   DC    X'00'                CEL MEMBER SUB-IDENTIFIER
0000B8  00                   DC    X'00'                CEL MEMBER DEFINED BYTE
0000B9  01                   DC    X'01'                CONTROL LEVEL OF PROLOG
0000BA  00000000             DC    V(CEESTART)          VCON FOR LOAD MODULE
0000BE  00000000             DC    F'0'                 OFFSET TO THE CDI (NONE)
0000C2  FFFFFFB2             DC    A(@TIMEVRS-@PPA2)    OFFSET TO TIMESTAMP/VERSION INFO
0000C6  00000000             DC    A(IMIN)              ADDRESS OF CU PRIMARY ENTRY POINT
0000CA                @CEEPARM DS    0H                   PARM LIST FOR CEEINT
0000CA  00000038             DC    A(@MAINENT)          POINTER TO PRIMARY ENTRY PT ADDR
0000CE  00000008             DC    A(@PARMCEE-@CEEPARM) OFFSET TO PARAMETERS FOR CEEINT
0000D2                @PARMCEE DS    0H                   PARAMETERS FOR CEEINT
0000D2  00000006             DC    F'6'                 1) NUMBER OF ENTRIES IN PARM LIST
0000D6  00000038             DC    A(@MAINENT)          2) POINTER TO PRIMARY ENTRY PT ADDR
0000DA  00000000             DC    V(CEESTART)          3) ADDRESS OF CEESTART
0000DE  00000000             DC    V(CEEBETBL)          4) ADDRESS OF CEEBETBL
0000E2  00000005             DC    F'5'                 5) CEL MEMBER IDENTIFIER
0000E6  00000000             DC    F'0'                 6) FOR CEL MEMBER USE
. . .
```

(1) COBOL プログラムの先頭からのオフセット。

(2) アセンブラー命令の 16 進表記。

(3) COBOL プログラムに対して生成される疑似アセンブラー・コード。

(4) アセンブラー・コードを説明するコメント。

(5) COBOL コンパイラーが言語環境プログラム対応であることを示す目印。

(6) タスク・グローバル・テーブル (TGT) のアドレスか、またはプログラムが再入可能な場合は、動的アクセス・ブロック (DAB) のアドレス。

(7) プログラムがコンパイルされたときの 4 桁の年。

(8) プログラムがコンパイルされたときの月および日。

(9) プログラムがコンパイルされたときの時刻。

(10) このプログラムをコンパイルするために使用された COBOL コンパイラーのバージョン、リリース、および修正レベル (それぞれが 2 桁で表される)。

(11) コード・ページの CCSID 値 (CODEPAGE コンパイラー・オプションより取得)。

(12) プログラム・シグニチャー情報バイト。これらは、以下のプログラム・エレメントに関する情報を提供します。

- コンパイラー・オプション
- DATA DIVISION
- ENVIRONMENT DIVISION (環境部)
- PROCEDURE DIVISION (手続き部)

(13) DATA DIVISION の中のステートメントの数。

(14) PROCEDURE DIVISION の中のステートメントの数。

(15) 4 バイトのユーザー制御レベル情報フィールド。このフィールドの値は LVLINFO によって制御されます。

(16) IDENTIFICATION DIVISION で使用されるプログラム名。

関連参照

『シグニチャー情報バイト: コンパイラー・オプション』
437 ページの『シグニチャー情報バイト: DATA DIVISION』
438 ページの『シグニチャー情報バイト: ENVIRONMENT DIVISION』
438 ページの『シグニチャー情報バイト: PROCEDURE DIVISION 動詞』
440 ページの『シグニチャー情報バイト: 多数の PROCEDURE DIVISION 項目』

シグニチャー情報バイト: コンパイラー・オプション

次の表は、LIST コンパイラー・オプションを使用したときに提供されるプログラム初期化コードのリストの一部である、プログラム・シグニチャー情報を示しています。

表 53. **コンパイラー・オプションのシグニチャー情報バイト**

バイト	ビット	オン	オフ
1	0	ADV	NOADV
	1	APOST	QUOTE
	2	DATA(31)	DATA(24)
	3	DECK	NODECK
	4	DUMP	NODUMP
	5	DYNAM	NODYNAM
	6	FASTSRT	NOFASTSRT
	7	予約済み	
2	0	LIB	NOLIB
	1	LIST	NOLIST
	2	MAP	NOMAP
	3	NUM	NONUM
	4	OBJ	NOOBJ
	5	OFFSET	NOOFFSET
	6	OPTIMIZE	NOOPTIMIZE
	7	OUTDD オプションで指定された DD 名が使用される。	OUTDD(SYSOUT) が有効
3	0	NUMPROC(PFD)	NUMPROC(NOPFD)
	1	RENT	NORENT
	2	予約済み	
	3	SEQUENCE	NOSEQUENCE
	4	SIZE(MAX)	SIZE(*value*)
	5	SOURCE	NOSOURCE
	6	SSRANGE	NOSSRANGE
	7	TERM	NOTERM
4	0	TEST	NOTEST
	1	TRUNC(STD)	TRUNC(OPT)
	2	WORD が指定された。	NOWORD
	3	VBREF	NOVBREF
	4	XREF	NOXREF
	5	ZWB	NOZWB
	6	NAME	NONAME
	7	予約済み	
5	0	NUMPROC(MIG)	
	1	NUMCLS(ALT)	NUMCLS(PRIM)
	2	DBCS	NODBCS
	3	AWO	NOAWO
	4	TRUNC(BIN)	TRUNC(BIN) ではない
	6	CURRENCY	NOCURRENCY
	7	コンパイル単位はクラス	コンパイル単位はプログラム

表 53. **コンパイラー・オプションのシグニチャー情報バイト** *(続き)*

バイト	ビット	オン	オフ
26	0	RMODE(ANY)	RMODE(24)
	1–3	TEST(HOOK)	TEST(NOHOOK)
	4	OPT(FULL)	OPT(STD) または NOOPT
	5	INTDATE(LILIAN)	INTDATE(ANSI)
	6	TEST(SEPARATE)	TEST(SEPARATE) ではない
	7	予約	
27	0	PGMNAME(LONGUPPER)	PGMNAME(LONGUPPER) ではない
	1	PGMNAME(LONGMIXED)	PGMNAME(LONGMIXED) ではない
	2	DLL	NODLL
	3	EXPORTALL	NOEXPORTALL
	4	DATEPROC	NODATEPROC
	5	ARITH(EXTEND)	ARITH(COMPAT)
	6	THREAD	NOTHREAD
	7	TEST(EJPD)	TEST(NOEJPD)
28	0	SQL	NOSQL
	1	CICS	NOCICS
	2	MDECK	NOMDECK
	3	SQLCCSID	NOSQLCCSID
	4	OPTFILE が有効である。	OPTFILE が有効でない。
	5	XMLPARSE(XMLSS)	XMLPARSE(COMPAT)

シグニチャー情報バイト: DATA DIVISION

次の表は、LIST コンパイラー・オプションを使用したときに提供されるプログラム初期化コードのリストの一部である、プログラム・シグニチャー情報を示しています。

表 54. **DATA DIVISION のシグニチャー情報バイト**

バイト	ビット	項目
6	0	QSAM ファイル記述子
	1	VSAM 順次ファイル記述子
	2	VSAM 索引付きファイル記述子
	3	VSAM 相対ファイル記述子
	4	ファイル記述子内の CODE-SET 節 (ASCII ファイル)
	5	スパン・レコード
	6	PIC G または PIC N (DBCS データ項目)
	7	データ記述記入項目の OCCURS DEPENDING ON 節

表 54. **DATA DIVISION のシグニチャー情報バイト** *(続き)*

バイト	ビット	項目
7	0	データ記述記入項目の SYNCHRONIZED 節
	1	データ記述記入項目の JUSTIFIED 節
	2	USAGE IS POINTER 項目
	3	複合 OCCURS DEPENDING ON 節
	4	DATA DIVISION の外部浮動小数点項目
	5	DATA DIVISION の内部浮動小数点項目
	6	行順次ファイル
	7	USAGE IS PROCEDURE-POINTER 項目または FUNCTION-POINTER 項目

関連参照
369 ページの『LIST』

シグニチャー情報バイト: ENVIRONMENT DIVISION

次の表は、LIST コンパイラー・オプションを使用したときに提供されるプログラム初期化コードのリストの一部である、プログラム・シグニチャー情報を示しています。

表 55. **ENVIRONMENT DIVISION のシグニチャー情報バイト**

バイト	ビット	項目
8	0	FILE-CONTROL 段落の FILE STATUS 節
	1	INPUT-OUTPUT SECTION の I-O-CONTROL 段落の RERUN 節
	2	SPECIAL-NAMES 段落で定義された UPSI スイッチ

シグニチャー情報バイト: PROCEDURE DIVISION 動詞

次の表は、LIST コンパイラー・オプションを使用したときに提供されるプログラム初期化コードのリストの一部である、プログラム・シグニチャー情報を示しています。

表 56. **PROCEDURE DIVISION 動詞のシグニチャー情報バイト**

バイト	ビット	項目
9	0	ACCEPT
	1	ADD
	2	ALTER
	3	CALL
	4	CANCEL
	6	CLOSE
10	0	COMPUTE
	2	DELETE
	4	DISPLAY
	5	DIVIDE

表 56. **PROCEDURE DIVISION 動詞のシグニチャー情報バイト** *(続き)*

バイト	ビット	項目
11	1	END-PERFORM
	2	ENTER
	3	ENTRY
	4	EXIT
	5	EXEC
	6	GO TO
	7	IF
12	0	INITIALIZE
	1	INVOKE
	2	INSPECT
	3	MERGE
	4	MOVE
	5	MULTIPLY
	6	OPEN
	7	PERFORM
13	0	READ
	2	RELEASE
	3	RETURN
	4	REWRITE
	5	SEARCH
	7	SET
14	0	SORT
	1	START
	2	STOP
	3	STRING
	4	SUBTRACT
	7	UNSTRING
15	0	USE
	1	WRITE
	2	CONTINUE
	3	END-ADD
	4	END-CALL
	5	END-COMPUTE
	6	END-DELETE
	7	END-DIVIDE

表 56. **PROCEDURE DIVISION 動詞のシグニチャー情報バイト** *(続き)*

バイト	ビット	項目
16	0	END-EVALUATE
	1	END-IF
	2	END-MULTIPLY
	3	END-READ
	4	END-RETURN
	5	END-REWRITE
	6	END-SEARCH
	7	END-START
17	0	END-STRING
	1	END-SUBTRACT
	2	END-UNSTRING
	3	END-WRITE
	4	GOBACK
	5	EVALUATE
	7	SERVICE
18	0	END-INVOKE
	1	END-EXEC
	2	XML
	3	END-XML

戻りコードの検査: コンパイラーから 4 より大きな戻りコードが戻された場合は、情報バイトで示される一部の動詞がプログラムから破棄された可能性があることを意味します。

シグニチャー情報バイト: 多数の PROCEDURE DIVISION 項目

次の表は、LIST コンパイラー・オプションを使用したときに提供されるプログラム初期化コードのリストの一部である、プログラム・シグニチャー情報を示しています。

表 57. **PROCEDURE DIVISION 項目のシグニチャー情報バイト**

バイト	ビット	項目
21	0	16 進数リテラル
	1	更新 GO TO
	2	I-O ERROR 宣言
	3	LABEL 宣言
	4	DEBUGGING 宣言
	5	プログラムのセグメンテーション
	6	OPEN . . . EXTEND
	7	EXIT PROGRAM

表 57. **PROCEDURE DIVISION 項目のシグニチャー情報バイト** *(続き)*

バイト	ビット	項目
22	0	CALL リテラル
	1	CALL ID
	2	CALL . . . ON OVERFLOW
	3	CALL . . . LENGTH OF
	4	CALL . . . ADDRESS OF
	5	CLOSE . . . REEL/UNIT
	6	使用されている指数
	7	使用されている浮動小数点項目
23	0	COPY
	1	BASIS
	2	プログラムの DBCS 名
	3	プログラムのシフトアウトとシフトイン
	4 から 7	ASM2 モジュール IGYBINIT への入り口での最高レベルのエラー重大度
24	0	DBCS リテラル
	1	REPLACE
	2	参照変更が使用された
	3	ネストされたプログラム
	4	INITIAL
	5	COMMON
	6	SELECT . . . OPTIONAL
	7	EXTERNAL
25	0	GLOBAL
	1	RECORD IS VARYING
	2	LABEL 宣言で使用される ACCEPT FROM SYSIPT
	3	LABEL 宣言で使用される DISPLAY UPON SYSLST
	4	LABEL 宣言で使用される DISPLAY UPON SYSPCH
	5	組み込み関数が使用された
29	0	プログラムにおける Java ベースのオブジェクト指向構文
	1	プログラムで使用される FUNCTION RANDOM
	2	プログラムで使用される NATIONAL データ

関連参照

369 ページの『LIST』

例: ソース・コードから生成されるアセンブラー・コード

次の例は、LIST コンパイラー・オプションを使用したときに、ソース・コードから生成されるアセンブラー・コードのリストを示しています。このリストを使用して、失敗した命令に対応する COBOL 動詞を見つけることができます。

```
データ妥当性検査および更新プログラム     IGYTCARA Date 12/30/2007 Time 10:48:16
000433  MOVE
000435  READ
000436  SET   (1)

    (2)          (3)              (5)                          (6)
        000F26  92E8 A00A             MVI  10(10),X'E8'          LOCATION-EOF-FLAG
        000F2A                GN=13   EQU  *
        000F2A  47F0 B426             BC   15,1062(0,11)         GN=75(000EFA)
        000F2E                GN=74   EQU  *
000439  IF
        000F2E  95E8 A00A             CLI  10(10),X'E8'          LOCATION-EOF-FLAG
        000F32  4780 B490             BC   8,1168(0,11)          GN=14(000F64)
000440  DISPLAY
        000F36  5820 D05C             L    2,92(0,13)            TGTFIXD+92
        000F3A  58F0 202C             L    15,44(0,2)            V(IGZCDSP )
        000F3E  4110 97FF             LA   1,2047(0,9)           PGMLIT AT +1999
        000F42  05EF                  BALR 14,15
000443  CALL
        000F44  4130 A012             LA   3,18(0,10)            COMP-CODE
        000F48  5030 D21C             ST   3,540(0,13)           TS2=4
        000F4C  9680 D21C             OI   540(13),X'80'         TS2=4
        000F50  4110 D21C             LA   1,540(0,13)           TS2=4
        000F54  58F0 9000             L    15,0(0,9)             V(ILBOABN0)
        000F58  05EF                  BALR 14,15
        000F5A  50F0 D078             ST   15,120(0,13)          TGTFIXD+120
        000F5E  BF38 D089             ICM  3,8,137(13)           TGTFIXD+137
        000F62  0430                  SPM  3,0
        000F64        (4)     GN=14   EQU  *
        000F64  5820 D154             L    2,340(0,13)           VN=3
        000F68  07F2                  BCR  15,2
```

(1) ソース行番号と、COBOL 動詞、段落名、またはセクション名。

行 000436 の SET は COBOL 動詞です。名前の前のアスタリスク (*) は、その名前が段落名またはセクション名であることを示しています。

(2) モジュール内でのオブジェクト・コード命令の相対位置 (16 進表記)。

(3) オブジェクト・コード命令 (16 進表記)。

16 進数字の最初の 2 桁または 4 桁が命令で、残りの桁は命令オペランドです。2 つのオペランドのある命令もあります。

(4) コード・シーケンス用のコンパイラー生成名 (GN)。

(5) アセンブラー言語に非常に類似した形式のオブジェクト・コード命令。

(6) オブジェクト・コード命令に関するコメント。

- マシン命令で使用される 1 つまたは 2 つのオペランドは、右側に表示されます。複数の構造で定義される (このようにして、ソース・プログラム内で修飾によって固有にされた) データ名には、その直後にアスタリスクが置かれます。
- オペランドとして出てくる生成ラベルの相対位置は、括弧内に表示されます。

関連参照

431 ページの『LIST および MAP 出力で使用される記号』

例: TGT メモリー・マップ

次の例は、タスク・グローバル・テーブル (TGT) の LIST 出力を示しています。この出力には、プログラムが実行される環境に関する情報が伴います。

```
データ妥当性検査および更新プログラム     IGYTCARA Date 12/30/2007 Time 10:48:16
                    *** TGT MEMORY MAP ***
                    (1)       (2)
                    TGTLOC

                    000000  RESERVED - 72 BYTES
                    000048  TGT IDENTIFIER
                    00004C  RESERVED - 4 BYTES
                    000050  TGT LEVEL INDICATOR
                    000051  RESERVED - 3 BYTES
                    000054  32 BIT SWITCH
                    000058  POINTER TO RUNCOM
                    00005C  POINTER TO COBVEC
                    000060  POINTER TO PROGRAM DYNAMIC BLOCK TABLE
                    000064  NUMBER OF FCB'S
                    000068  WORKING-STORAGE LENGTH
                    00006C  RESERVED - 4 BYTES
                    000070  ADDRESS OF IGZESMG WORK AREA
                    000074  ADDRESS OF 1ST GETMAIN BLOCK (SPACE MGR)
                    000078  RESERVED - 2 BYTES
                    00007A  RESERVED - 2 BYTES
                    00007C  RESERVED - 2 BYTES
                    00007E  MERGE FILE NUMBER
                    000080  ADDRESS OF CEL COMMON ANCHOR AREA
                    000084  LENGTH OF TGT
                    000088  RESERVED - 1 SINGLE BYTE FIELD
                    000089  PROGRAM MASK USED BY THIS PROGRAM
                    00008A  RESERVED - 2 SINGLE BYTE FIELDS
                    00008C  NUMBER OF SECONDARY FCB CELLS
                    000090  LENGTH OF THE ALTER VN(VNI) VECTOR
                    000094  COUNT OF NESTED PROGRAMS IN COMPILE UNIT
                    000098  DDNAME FOR DISPLAY OUTPUT
                    0000A0  RESERVED - 8 BYTES
                    0000A8  POINTER TO COM-REG SPECIAL REGISTER
                    0000AC  RESERVED - 52 BYTES
                    0000E0  ALTERNATE COLLATING SEQUENCE TABLE PTR.
                    0000E4  ADDRESS OF SORT G.N. ADDRESS BLOCK
                    0000E8  ADDRESS OF PGT
                    0000EC  RESERVED - 4 BYTES
                    0000F0  POINTER TO 1ST IPCB
                    0000F4  ADDRESS OF THE CLLE FOR THIS PROGRAM
                    0000F8  POINTER TO ABEND INFORMATION TABLE
                    0000FC  POINTER TO TEST INFO FIELDS IN THE TGT
                    000100  ADDRESS OF START OF COBOL PROGRAM
                    000104  POINTER TO ALTER VNI'S IN CGT
                    000108  POINTER TO ALTER VN'S IN TGT
                    00010C  POINTER TO FIRST PBL IN THE PGT
                    000110  POINTER TO FIRST FCB CELL
                    000114  WORKING-STORAGE ADDRESS
                    000118  POINTER TO FIRST SECONDARY FCB CELL
                    00011C  POINTER TO STATIC CLASS INFO BLOCK 1
                    000120  POINTER TO STATIC CLASS INFO BLOCK 2

                    *** VARIABLE PORTION OF TGT ***

                    000124  BASE LOCATORS FOR SPECIAL REGISTERS
                    00012C  BASE LOCATORS FOR WORKING-STORAGE   (3)
                    000134  BASE LOCATORS FOR LINKAGE-SECTION
                    000138  BASE LOCATORS FOR FILES
                    00014C  CLLE ADDR. CELLS FOR CALL LIT. SUB-PGMS.
```

```
                000170  INDEX CELLS
                000194  FCB CELLS
                0001A8  INTERNAL PROGRAM CONTROL BLOCKS
```

(1) TGT の先頭からの TGT フィールドの 16 進オフセット。

(2) TGT フィールドの内容の説明。

(3) COBOL データ域のベース・ロケーターの TGT フィールド。

例: DSA メモリー・マップ

次の例は、動的保管域 (DSA) の LIST 出力を示しています。 DSA には、別個にコンパイルされたプロシージャーに入ったときに獲得されたストレージの内容に関する情報が含まれています。

```
データ妥当性検査および更新プログラム    IGYTCARA Date 12/30/2007 Time 10:48:16
          *** DSA MEMORY MAP ***
          (1)     (2)
          DSALOC

          000000  REGISTER SAVE AREA
          00004C  STACK NAB (NEXT AVAILABLE BYTE)
          000058  ADDRESS OF INLINE-CODE PRIMARY DSA
          00005C  ADDRESS OF TGT
          000060  ADDRESS OF CAA
          000084  SWITCHES
          000088  CURRENT INT. PROGRAM OR METHOD NUMBER
          00008C  ADDRESS OF CALL STATEMENT PROGRAM NAME
          000090  CALC ROUTINE REGISTER SAVE AREA
          0000C4  ADDRESS OF FILE MUTEX USE COUNT CELLS
          0000C8  PROCEDURE DIVISION RETURNING VALUE

          *** VARIABLE PORTION OF DSA ***

          0000D0  BACKSTORE CELLS FOR SYMBOLIC REGISTERS
          000158  BASE LOCATORS FOR ALPHANUMERIC TEMPS
          00015C  VARIABLE-LENGTH CELLS
          000170  ODO SAVE CELLS
          00017C  VARIABLE NAME (VN) CELLS FOR PERFORM
          0001EC  PERFORM SAVE CELLS
          000320  TEMPORARY STORAGE-1
          000330  TEMPORARY STORAGE-2
          000500  ALL PARAMETER BLOCK
          000564  ALPHANUMERIC TEMPORARY STORAGE
```

(1) DSA の先頭からの DSA フィールドの 16 進オフセット。

(2) DSA フィールドの内容の説明。

例: WORKING-STORAGE の位置とサイズ

次の例は、RENT オプションでコンパイルされたプログラムの WORKING-STORAGE に関する LIST 出力を示しています。

```
     (1)                                (2)
WRK-STOR WILL BE ALLOCATED FOR 000015B0 BYTES
```

(1) WORKING-STORAGE の識別。

(2) WORKING-STORAGE の長さ (16 進表記)。

関連概念

45 ページの『ストレージとそのアドレス可能度』

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

      264    ABEND-ITEM1
      265    ABEND-ITEM2
      347    ADD-CODE . . . . . . . . . . .   1126 1192
      381    ADDRESS-ERROR. . . . . . . . .   M1156
      280    AREA-CODE. . . . . . . . . . .   1266 1291 1354 1375
      382    CITY-ERROR . . . . . . . . . .   M1159

       (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
      943    050-CREATE-STL-MASTER-FILE . .   P879
      995    100-INITIALIZE-PARAGRAPH . . .   P881
     1471    1100-PRINT-I-F-HEADINGS. . . .   P926
     1511    1200-PRINT-I-F-DATA. . . . . .   P928
     1573    1210-GET-MILES-TIME. . . . . .   P1540
     1666    1220-STORE-MILES-TIME. . . . .   P1541
     1682    1230-PRINT-SUB-I-F-DATA. . . .   P1562
     1706    1240-COMPUTE-SUMMARY . . . . .   P1563
     1052    200-EDIT-UPDATE-TRANSACTION. .   P890
     1154    210-EDIT-THE-REST. . . . . . .   P1145
     1189    300-UPDATE-COMMUTER-RECORD . .   P893
     1237    310-FORMAT-COMMUTER-RECORD . .   P1194 P1209
     1258    320-PRINT-COMMUTER-RECORD. . .   P1195 P1206 P1212 P1222
     1318    330-PRINT-REPORT . . . . . . .   P1208 P1232 P1286 P1310 P1370
     1342    400-PRINT-TRANSACTION-ERRORS .   P896
```

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。

『例: XREF 出力: プログラム名相互参照』
『例: XREF 出力: COPY/BASIS 相互参照』
447 ページの『例: 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、z/OS 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データ・セット名への、COPY または BASIS ステートメント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

   (1)          (1)          (2)                    (3)         (4)
  Text-name     Library      File name              Concat      ISPF
  (Member)      (DDNAME)     (Data set name)        Level       Created

  ACTIONS       OTHERLIB     USERID.COBOL.COPY        0         1992/07/11
  ACTIONS       SYSLIB       USERID.COBOL.COPY        0         1992/07/11
  CUSTOMER      ALTDDXXY     USERID.COBOL.LIB3        0         2007/06/01
  CUSTOMER      SYSLIB       USERID.COBOL.LIB2PDSE    1         2007/06/07
  HOUSE         ALTDDXXY     USERID.COBOL.LIB2        1         2007/06/07
  HOUSE         SYSLIB       USERID.COBOL.LIB2PDSE    1
  IMOTOR        SYSLIB       USERID.COBOL.LIB4X       3         2007/06/07
  ISOVERFY      SYSLIB       USERID.COBOL.COPY        0
  NSMAP         SYSLIB       USERID.COBOL.LIB3        2
```

(1) Text-name (テキスト名) および Library (ライブラリー: Library-name (ライブラリー名) の省略形) は、ソースのステートメント COPY *text-name* OF *library-name* (例: Copy ACTIONS Of OTHERLIB) からのものです。

(2) COPY メンバーがコピーされたデータ・セットの名前。

(3) 連結レベルの省略形。指定されたデータ・セットが指定された DD 名の連結で、最初のデータ・セットから何レベルの深さにあるのかを示します。

たとえば、上の例の 4 つのデータ・セットは DD 名 SYSLIB に連結されています。

```
DDNAME       DSNAME                      (concatenation level)

SYSLIB  DD   DSN=USERID.COBOL.COPY,            0
        DD   DSN=USERID.COBOL.LIB2PDSE,        1
        DD   DSN=USERID.COBOL.LIB3,            2
        DD   DSN=USERID.COBOL.LIB4X            3
```

たとえば、上のリストで表示されているメンバー NSMAP は、SYSLIB 連結の最初のデータ・セットから 2 レベル下にある、データ・セット USERID.COBOL.LIB3 で検出されたということになります。

(4) PDS または PDSE が ISPF で STATS ON を使用して編集された場合、作成日が表示されます。

z/OS UNIX シェルでコンパイルした場合、相互参照は以下の抜粋のように表示されます。

```
         COPY/BASIS cross-reference of text-names, library names, and file names

    (5)                    (5)                (6)
 Text-name                Library-name        File name

'/copydir/copyM.cbl'     SYSLIB              /u/JSMITH/cobol//copydir/copyM.cbl
'/copyA.cpy'             SYSLIB              /u/JSMITH/cobol//copyA.cpy
'cobol/copyA.cpy'        ALTDD2              /u/JSMITH/cobol/copyA.cpy
'copy/stuff.cpy'         ALTDD2              /u/JSMITH/copy/stuff.cpy
'copydir/copyM.cbl'      SYSLIB              /u/JSMITH/cobol/copydir/copyM.cbl
'copydir/copyM.cbl'      SYSLIB (default)    /u/JSMITH/cobol/copydir/copyM.cbl
'stuff.cpy'              ALTDD               /u/JSMITH/copy/stuff.cpy
"copyA.cpy"        (7)   SYSLIB (default)    /u/JSMITH/cobol/copyA.cpy
"reallyXXVeryLongLon>    SYSLIB (default)  (8)<JSMITH/cobol/reallyXXVeryLongLongName.cpy
OTHERDD                  ALTDD2              /u/JSMITH//copy/other.cob
. . .

Note: Some names were truncated.   > = truncated on right   < = truncated on left
```

(5) ソースの COPY ステートメントからのものです。たとえば、上の相互参照の 3 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 'cobol/copyA.cpy' Of ALTDD2
```

(6) COPY メンバーをコピーした元のファイルの完全修飾パス

(7) 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。

(8) 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。

例: 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-vsam-master-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: OFFSET コンパイラー出力

次の例は、圧縮された動詞のリスト、グローバル・テーブル、WORKING-STORAGE 情報、およびリテラルが含まれているコンパイラー・リストを示しています。このリストは、OFFSET コンパイラー・オプションからの出力です。

```
データ妥当性検査および更新プログラム     IGYTCARA Date 12/30/2007 Time 10:48:16
  . . .
  (1)      (2)     (3)
LINE #  HEXLOC  VERB       LINE #  HEXLOC  VERB       LINE #  HEXLOC  VERB
000880 0026F0 DISPLAY      000881 002702 PERFORM      000933 002702 OPEN
000934 002722 IF           000935 00272C DISPLAY      000936 002736 PERFORM
001389 002736 DISPLAY      001390 002740 DISPLAY      001391 00274A DISPLAY
001392 002754 DISPLAY      001393 00275E DISPLAY      001394 002768 DISPLAY
001395 002772 DISPLAY      000937 00277C PERFORM      001434 00277C DISPLAY
001435 002786 STOP         000939 0027A2 MOVE         000940 0027AC WRITE
000941 0027D6 IF           000942 0027E0 DISPLAY      000943 0027EA PERFORM
001389 0027EA DISPLAY      001390 0027F4 DISPLAY      001391 0027FE DISPLAY
001392 002808 DISPLAY      001393 002812 DISPLAY      001394 00281C DISPLAY
001395 002826 DISPLAY      000944 002830 DISPLAY      000945 00283A PERFORM
001403 00283A DISPLAY      001404 002844 DISPLAY      001405 00284E DISPLAY
001406 002858 DISPLAY      001407 002862 CALL         000947 002888 CLOSE
```

(1) 行番号。ユーザーの行番号またはコンパイラー生成の行番号がリストされます。

(2) この動詞用に生成されたコードの、プログラムの先頭からのオフセット (16 進表記)。

動詞は発生順にリストされ、使用されるたびに 1 度ずつリストされます。

(3) 使用される動詞。

関連参照

377 ページの『OFFSET』

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動詞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 . . . . . .  101  101
2        ADD. . . . . . . .  129  130
1        CALL . . . . . . .  140
5        CLOSE. . . . . . .  90  94  97  152  153
20       COMPUTE. . . . . .150  164  164  165  166  166  166  166  167  168  168  169  169  170  171  171
                             171  172  172  173
2        CONTINUE . . . . .106  107
2        DELETE . . . . . .96  119
47       DISPLAY. . . . . .88  90  91  92  92  93  94  94  94  95  96  96  97  99  99  100  100  100  100
                             103  109  117  117  118  119  138  139  139  139  139  139  140  140  140
                             140  143  148  148  149  149  149  152  152  152  153  162
2        EVALUATE . . . . .116  155
47       IF . . . . . . . .88  90  93  94  94  95  96  96  97  99  100  103  105  105  107  107  107  109
                             110  111  111  112  113  113  113  113  114  114  115  115  116  118  119  124
                             124  126  127  129  132  133  134  135  136  148  149  152  152
183      MOVE . . . . . . .90  93  95  98  98  98  98  98  99  100  101  101  102  104  105  105  106  106
                             107  107  108  108  108  108  108  108  109  110  111  112  113  113  113  114
                             114  114  115  115  116  116  117  117  117  118  118  118  119  119  120  121
                             121  121  121  121  121  121  121  121  121  122  122  122  122  122  123  123
                             123  123  123  123  123  124  124  124  125  125  125  125  125  125  125  126
                             126  126  126  126  127  127  127  127  128  128  129  129  130  130  130  130
                             131  131  131  131  131  132  132  132  132  132  132  133  133  133  133  133
                             134  134  134  134  134  135  135  135  135  135  135  136  136  137  137  137
                             137  137  138  138  138  138  141  141  142  142  144  144  144  144  145  145
                             145  145  146  149  150  150  150  151  151  155  156  156  157  157  158  158
                             159  159  160  160  161  161  162  162  162  168  168  168  169  169  170  171
                             171  172  172  173  173
5        OPEN . . . . . . .93  95  99  144  148
62       PERFORM. . . . . .88  88  88  88  89  89  89  91  91  91  91  93  93  94  94  95  95  95  95  96
                             96  96  97  97  97  100  100  101  102  104  109  109  111  116  116  117  117
                             117  118  118  118  118  119  119  119  120  120  124  125  127  128  133  134
                             135  136  136  137  150  151  151  153  153
8        READ . . . . . . .88  89  96  101  102  108  149  151
1        REWRITE. . . . . .118
4        SEARCH . . . . . .106  106  141  142
46       SET. . . . . . . .88  89  101  103  104  105  106  108  108  136  141  142  149  150  151  152  154
                             155  156  156  156  156  157  157  157  157  158  158  158  158  159  159  159
                             159  160  160  160  160  161  161  161  161  162  162  164  164
2        STOP . . . . . . .92  143
4        STRING . . . . . .123  126  132  134
33       WRITE. . . . . . .94  116  129  129  129  129  129  130  130  130  130  145  146  146  146  146  147
                             147  151  165  165  166  166  167  174  174  174  174  174  174  174  175  175
```

(1) その動詞がプログラムで使用されている回数。

(2) 動詞。

(3) その動詞が使用されている行番号。

第 3 部 特定の環境に合わせた COBOL プログラムの目標

第 20 章 COBOL プログラムの開発 (CICS の場合)

CICS 用に作成された COBOL プログラムは、CICS Transaction Server のもとで実行することができます。 CICS サービスを使用する CICS COBOL アプリケーション・プログラムは、CICS のコマンド・レベル・インターフェースを使用する必要があります。

CICS コンパイラー・オプションが指定されていると、Enterprise COBOL コンパイラーは、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。 COBOL for OS/390 & VM バージョン 2 リリース 2 より前のコンパイラーの場合、EXEC CICS コマンドを COBOL コードに変換するには、分離型の変換ステップが必要です。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。

組み込みの CICS 変換プログラムを使用するには、CICS Transaction Server バージョン 2 またはそれ以降が必要です。

プログラムをコンパイルし、リンク・エディットした後、CICS テーブルの更新などその他のステップを実行する必要があります。これらのステップを実行しないと、CICS のもとで COBOL プログラムを実行することはできません。しかし、これらの CICS トピックは、本書の説明範囲を超えています。CICS の詳細については、関連参照を参照してください。

実行時エラーの処理方法を判別するには、CBLPSHPOP ランタイム・オプションを設定します。CICS の HANDLE と CBLPSHPOP については、関連タスクを参照してください。

関連概念
460 ページの『組み込みの CICS 変換プログラム』

関連タスク
454 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』
458 ページの『CICS オプションを使用したコンパイル』
462 ページの『分離型の CICS 変換プログラムの使用』
464 ページの『CICS HANDLE を使用したエラー処理』
言語環境プログラム・プログラミング・ガイド (CICS 環境での条件処理: CBLPSHPOP ランタイム・オプションの使用)
CICS Application Programming Guide

関連参照
349 ページの『CICS』

CICS のもとで実行する COBOL プログラムのコーディング

CICS のもと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。

制約事項: COBOL クラス定義およびメソッド (オブジェクト指向 COBOL) を CICS 環境で実行することはできません。また、CICS のもとで実行するプログラムをコーディングするときは、次のコードを使用しないでください。

- ENVIRONMENT DIVISION での FILE-CONTROL 記入項目 (FILE-CONTROL 記入項目が SORT ステートメントで使用される場合を除く)。
- DATA DIVISION の FILE SECTION (FILE SECTION が SORT ステートメントで使用される場合を除く)
- 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する
- USE 宣言部分 (USE FOR DEBUGGING を除く) を使用する
- 次の COBOL 言語ステートメントを使用する
  - ACCEPT 形式 1: データ転送 (システムの日付と時刻を取得するには、形式 2 の ACCEPT を使用できます)
  - CLOSE
  - DELETE
  - DISPLAY UPON CONSOLE
  - DISPLAY UPON SYSPUNCH
  - MERGE
  - OPEN
  - READ
  - RERUN
  - REWRITE
  - START
  - STOP *literal*
  - WRITE

分離型の CICS 変換プログラムを使用する予定である場合、EXEC コマンドを含んでいる REPLACE ステートメントはいずれも、プログラムの PROCEDURE DIVISION ヘッダーの後に配置する必要があります。そうしないとコマンドは変換されません。

ファイルの入出力のコーディング: ほとんど入出力処理に CICS コマンドを使用する必要があります。したがって、ファイルを記述したり、OPEN、CLOSE、READ、

START、REWRITE、WRITE、または DELETE ステートメントをコーディングすることはありません。かわりに、CICS コマンドを使用してデータの検索、更新、挿入、および削除を行います。

16 MB 境界より上で実行されるように COBOL プログラムをコーディングする: Enterprise COBOL では、16 MB 境界より上で実行されるように COBOL プログラムをコーディングするとき、以下の制約事項が適用されます。

- IMS/ESA® バージョン 6 (またはそれ以降) を DBCTL なしで使用しているときは、DL/I CALL ステートメントがサポートされるのは、呼び出しで渡されるすべてのデータが 16MB 境界より下に常駐している場合だけです。このため、DATA(24) コンパイラー・オプションを指定する必要があります。しかし、IMS/ESA バージョン 6 (またはそれ以降) を DBCTL とともに使用している場合は、DATA(31) コンパイラー・オプションを使用して、16MB より上に常駐するデータを渡すことができます。

  DL/I CALL ステートメントではなく EXEC DLI を使用する場合は、IMS 製品のレベルに関係なく、DATA(31) を指定することができます。
- 受け取りプログラムが AMODE 31 でリンク・エディットされている場合、渡されるアドレスは、31 ビットまたは 24 ビットの長さで、左端バイトが 0 に設定されていなければなりません。
- 受け取りプログラムが AMODE 24 でリンク・エディットされている場合、渡されるアドレスは 24 ビットの長さでなければなりません。

データ項目の内容の表示: システム論理出力装置 (SYSOUT、SYSLIST、SYSLST) への DISPLAY は CICS のもとでサポートされます。 DISPLAY 出力は、言語環境プログラムのメッセージ・ファイル (一時データ・キュー CESE) に書きこまれます。ただし、DISPLAY . . . UPON CONSOLE と DISPLAY . . . UPON SYSPUNCH はサポートされません。

関連概念
460 ページの『組み込みの CICS 変換プログラム』

関連タスク
260 ページの『CICS のもとでのソート』
『CICS のもとでのシステム日付の取得』
456 ページの『COBOL プログラムとの間の呼び出し』
458 ページの『ECI 呼び出しの成否の判断』
462 ページの『分離型の CICS 変換プログラムの使用』

関連参照
261 ページの『CICS SORT アプリケーションの制約事項』

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 環境で使用すると、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 環境で使用すると、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
40 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 (*Enterprise COBOL 言語解説書*)

COBOL プログラムとの間の呼び出し

CALL を使用して、VS COBOL II、COBOL for MVS & VM、COBOL for OS/390 & VM、および Enterprise COBOL のプログラムとの間で呼び出しを行うことができます。しかし、これらのプログラムから OS/VS COBOL プログラムを呼び出したり、OS/VS COBOL からこれらのプログラムを呼び出す場合には、CALL ステートメントを使用することはできません。代わりに、EXEC CICS LINK を使用してください。

分離型の CICS 変換プログラムまたは組み込みの CICS 変換プログラムで処理された、個別にコンパイルされた COBOL プログラムを呼び出す場合は、DFHEIBLK および DFHCOMMAREA を、CALL ステートメントの 1 番目と 2 番目のパラメーターとして渡す必要があります。

分離型の CICS 変換プログラムまたは組み込みの CICS 変換プログラムによって処理される呼び出し先プログラムは、言語に関して CICS がサポートする任意の関数を含むことができます。

CICS のもとで実行する場合、COBOL 動的呼び出しを使用できます。 COBOL プログラムが、分離型の CICS 変換プログラムまたは組み込みの CICS 変換プログラムによって処理された場合、または EXEC SQL ステートメントを含んでいる場合、NODYNAM コンパイラー・オプションが必要です。この場合、CALL *identifier* を NODYNAM コンパイラー・オプションと一緒に使用すると、プログラムを動的に呼び出すことができます。 COBOL プログラムが EXEC SQL ステートメントを含んでおらず、また分離型の CICS 変換プログラムや組み込みの CICS 変換プログラムによって処理されていない場合、NODYNAM コンパイラー・オプションを指定してコンパ

イルする必要はありません。この場合、DYNAM コンパイラー・オプションと一緒に CALL *literal* を使用するか、あるいは CALL *identifier* を使用して、動的にプログラムを呼び出すことができます。

動的に呼び出されるプログラムは、CICS 自動インストールを使用していない場合は、CICS プログラム処理テーブル (PPT) に定義されていなければなりません。CICS 環境では、 COBOL プログラムは CICS PROGRAM 定義に RELOAD=YES オプションをコーディングするサブプログラムに対する動的呼び出しをサポートしません。 RELOAD=YES で定義されたプログラムに対する動的呼び出しは、ストレージ不足を生じさせる可能性があります。 COBOL から動的に呼び出すプログラムには、RELOAD=NO オプションを使用してください。

他の高水準言語との言語間通信 (ILC) がサポートされます。 ILC がサポートされていない場合は、代わりに CICS の LINK、XCTL、および RETURN を使用できます。

次の表に、COBOL プログラムとアセンブラー言語プログラム間の呼び出し関係を示しています。この表では、「*言語環境プログラム・プログラミング・ガイド*」で解説されているインターフェースに準拠しているアセンブラー言語プログラムを*言語環境プログラム準拠* アセンブラー・プログラムと呼んでいます。このインターフェースに準拠していないアセンブラー言語プログラムは、*非 LE 準拠* アセンブラー・プログラムと呼んでいます。

表 58. **CICS のもとでの COBOL およびアセンブラー間の呼び出し**

COBOL プログラムとアセンブラー・プログラム間の呼び出し	**言語環境プログラム準拠アセンブラー・プログラム**	**非言語環境プログラム準拠アセンブラー・プログラム**
Enterprise COBOL プログラムからアセンブラー・プログラムを呼び出せるか	はい	はい
アセンブラー・プログラムから Enterprise COBOL を呼び出せるか	はい (ただし、アセンブラー・プログラムがメインプログラムでない場合)	いいえ

ネストされたプログラムのコーディング: 組み込みの CICS 変換プログラムを使用してコンパイルする場合、この変換プログラムにより、GLOBAL 節と一緒に DFHEIBLK および DFHCOMMAREA 制御ブロックが最外部プログラムに生成されます。したがって、ネストされたプログラムをコーディングするときは、ネストされたプログラムの呼び出しに対する引数として、これらの制御ブロックを渡す必要がありません。

ネストされたプログラムをコーディングし、分離型の CICS 変換プログラムを使用する予定である場合、EXEC コマンドまたは EXEC インターフェース・ブロック (EIB) への参照を含んでいるネストされたプログラムへのパラメーターとして、DFHEIBLK および DFHCOMMAREA を渡してください。制御階層内で、このようなネストされたプログラムとその最上位のプログラムとの間にあるプログラムにも、同じパラメーターを渡す必要があります。

関連概念

460 ページの『組み込みの CICS 変換プログラム』

関連タスク
462 ページの『分離型の CICS 変換プログラムの使用』
479 ページの『DYNAM または NODYNAM コンパイラー・オプションの選択』
275 ページの『プログラム呼び出し時のエラーの処理』
Language Environment Writing ILC Applications (CICS のもとでの ILC)
CICS External Interfaces Guide
言語環境プログラム・プログラミング・ガイド

関連参照
361 ページの『DYNAM』

ECI 呼び出しの成否の判断

外部 CICS インターフェース (ECI) の呼び出しの後で、RETURN-CODE 特殊レジスターの内容は予期できない値に設定されます。このため、外部 CICS インターフェースを正常に使用した後、COBOL プログラムが正常に終了した場合でも、ジョブ・ステップが未定義の戻りコードで終了する可能性があります。

終了時に意味のある戻りコードが戻されるようにするには、プログラムを終了する前に RETURN-CODE 特殊レジスターを設定してください。 CICS への最後の呼び出しに関する状況をジョブの戻りコードに反映させるには、外部 CICS インターフェースへの最後の呼び出しで戻された応答コードに基づいて RETURN-CODE 特殊レジスターを設定します。

関連タスク
CICS External Interfaces Guide

CICS オプションを使用したコンパイル

CICS コンパイラー・オプションを使用すると、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することができます。

NOCICS オプションを指定場合、コンパイラーはソース・プログラムで検出した CICS ステートメントをすべて診断し破棄します。これまで分離型の CICS 変換プログラムを使用していた場合は、NOCICS を指定する必要があります。

CICS オプションは、コンパイラー・オプション・ソース (すなわち、コンパイラー呼び出し、PROCESS または CBL ステートメント、あるいはインストール・デフォルト) のいずれにでも指定することができます。CICS オプションが COBOL インストール・デフォルトである場合、CICS サブオプションを指定することはできません。ただし、組み込みの CICS 変換プログラムで行った変更は非 CICS アプリケーションには適切でないので、CICS オプションをインストール・デフォルトとして使用することは推奨しません。

Enterprise COBOL の規則に従い、CBL ステートメントや PROCESS ステートメントはすべて、コメント行より前に置く必要があります。

COBOL コンパイラーは、CICS コンパイラー・オプションで提供された CICS サブオプション・ストリングを、組み込みの CICS 変換プログラムが使用できるようにします。このストリングの内容は、組み込みの CICS 変換プログラムに対してのみ有効です。

組み込みの CICS 変換プログラムを使用するときは、以下のオプションを指定してコンパイルする必要があります。

表 59. **組み込みの CICS 変換プログラムに必要なコンパイラー・オプション**

コンパイラー・オプション	説明
CICS	NOLIB、DYNAM、または NORENT を指定すると、コンパイラーは、LIB、NODYNAM、および RENT を強制的にオンにします。
LIB	CICS とともに、このオプションを有効にする必要があります。
NODYNAM	CICS とともに、このオプションを有効にする必要があります。
RENT	CICS とともに、このオプションを有効にする必要があります。
SIZE(xxx)	xxx は、組み込みの CICS 変換プログラムのための十分なストレージをユーザー領域に残すようなサイズ値 (MAX ではない) に設定する必要があります。

さらに、CICS のもとでサポートされない言語エレメントにコンパイラーがフラグを立てるように、コンパイラー・オプション WORD(CICS) を使用することをお勧めします。

組み込みの CICS 変換プログラムを使用してプログラムをコンパイルするには、COBOL に付いている標準 JCL プロシージャー・ステートメントを使用できます。上記のコンパイラー・オプションを指定することに加えて、2 つの方法で JCL を変更する必要があります。

- COBOL ステップに STEPLIB オーバーライドを指定します。
- 組み込みの CICS 変換プログラム・サービスがリンク・リストに含まれていない場合、それらのサービスを含んでいるデータ・セットを追加します。

CICS Transaction Server V3R2 用のデータ・セットのデフォルト名は CICSTS32.CICS.SDFHLOAD ですが、インストール先によってはこの名前が変更されている場合があります。例えば、以下の行が JCL に含まれている場合があります。

```
//STEPLIB  DD  DSN=CICSTS32.CICS.SDFHLOAD,DISP=SHR
```

COBOL コンパイラー・リストには、組み込みの CICS 変換プログラムが生成するエラー診断 (CICS ステートメントの構文エラーなど) が含まれます。このリストは入力ソースを反映するものであり、組み込みの CICS 変換プログラムが生成する COBOL ステートメントは含まれていません。

一連のプログラムのコンパイル: CICS オプションを使用して、一連の COBOL プログラムが含まれているソース・ファイルをコンパイルする場合、オプションの優先順位 (高い順) は、次のとおりです。

- コンパイル単位を開始する CBL カードまたは PROCESS カードに指定されたオプション
- コンパイラーを開始するときに指定されたオプション

- CICS デフォルト・オプション

関連概念
『組み込みの CICS 変換プログラム』

関連タスク
454 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』
『CICS サブオプションの分離』
CICS Application Programming Guide

関連参照
349 ページの『CICS』
344 ページの『矛盾するコンパイラー・オプション』

CICS サブオプションの分離

CICS サブオプションの指定を複数の CBL ステートメントに分割することができます。 CICS サブオプションは累積的です。コンパイラーは、複数のソースからの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、JCL ファイルに以下のコードが含まれているとします。

```
//STEP1 EXEC IGYWC, . . .
//PARM.COBOL="CICS("FLAG(I)")"
//COBOL.SYSIN DD *
  CBL CICS("DEBUG")
  CBL CICS("LINKAGE")
  IDENTIFICATION DIVISION.
  PROGRAM-ID. COBOL1.
```

コンパイル時に、コンパイラーは次の CICS サブオプション・ストリングを組み込みの CICS 変換プログラムに渡します。

```
"FLAG(I) DEBUG LINKAGE"
```

連結ストリングは、グループの前後に引用符または単一引用符を付けてシングル・スペースで区切ります。コンパイラーが同じ CICS 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で最後に指定されたサブオプションが有効になります。コンパイラーでは、連結された CICS サブオプション・ストリングの長さは 4KB に制限されます。

関連参照
349 ページの『CICS』

組み込みの CICS 変換プログラム

CICS コンパイラー・オプションを使用して COBOL プログラムをコンパイルすると、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、コンパイラーは、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。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。

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。

- デバッグ・ツールを使用して、分離型の CICS 変換プログラムによって提供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 ステートメントや EXEC DLI 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データ・セットが不要です。
- 出力リストは 2 つではなく 1 つだけ作成されます。
- EXEC CICS が含まれているネストされたプログラムの使用が簡単です。最外部のプログラムの GLOBAL 属性を使用して DFHCOMMAREA および DFHEIBLK が生成されます。ネストされたプログラムの呼び出し時にこれらを引数として指定する必要はなく、ネストされたプログラムの PROCEDURE DIVISION ヘッダーの USING 句で指定する必要もありません。
- EXEC CICS ステートメントが含まれているネストされたプログラムを個別のファイルに保管し、COPY ステートメントを使用して、それらのネストされたプログラムを組み込むことができます。
- 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でコンパイルすることができます。
- CICS 制御ブロックの 2 進数フィールドは BINARY ではなく COMP-5 の形式で生成されるようになったので、TRUNC コンパイラー・オプションの設定との依存関係がありません。CICS プログラムの TRUNC については、アプリケーション・ロジックおよびユーザー定義の 2 進数フィールドの使用に関する要件に従っている限り、任意の設定値を使用することができます。

関連概念
CICS Application Programming Guide (組み込みの CICS 変換プログラム)

関連タスク 454 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』 458 ページの『CICS オプションを使用したコンパイル』

関連参照
349 ページの『CICS』
397 ページの『TRUNC』

分離型の CICS 変換プログラムの使用

CICS のもとで COBOL プログラムを実行するために、分離型の CICS 変換プログラムを使用して CICS コマンドを COBOL ステートメントに変換してから、プログラムをコンパイルし、リンクして実行可能モジュールを作成することができます。しかし、分離型の変換プログラムよりも、Enterprise COBOLの組み込みの CICS 変換プログラムを使用することをお勧めします。

CICS ステートメントを個別に変換するには、COBOL3 変換プログラム・オプションを使用します。このオプションを使用すると、次の行が挿入されます。

```
CBL RENT,NODYNAM,LIB
```

CICS 変換プログラム・オプション NOCBLCARD を使用すれば、CBL ステートメントの挿入を抑止することができます。

CICS には、変換プログラム・オプション ANSI85 があり、これは以下の言語機能 (標準 COBOL 85 で導入) をサポートします。

- リテラルの間にあるブランク行
- 任意の文字を含むシーケンス番号
- すべての COBOL ワードでサポートされる小文字
- REPLACE ステートメント
- バッチ・コンパイル
- ネストされたプログラム
- 参照変更
- GLOBAL 変数
- コンマ、セミコロン、およびスペースの交換可能性
- シンボリック文字定義

分離型の CICS 変換プログラムを使用したときは、プログラムのコンパイル時に以下のコンパイラー・オプションを使用してください。

表 60. **分離型の CICS 変換プログラムに必要なコンパイラー・オプション**

必要なコンパイラー・オプション	条件
RENT	
NODYNAM	プログラムが CICS 変換プログラムによって変換される。
LIB	プログラムに COPY または BASIS ステートメントが含まれている。

さらに、CICS のもとでサポートされない言語エレメントにコンパイラーがフラグを立てるように、コンパイラー・オプション WORD(CICS) を使用することをお勧めします。

TRUNC コンパイラー・オプションに関する以下の推奨は、2 進数データ項目の予期値に基づいています。

表 61. **分離型の CICS 変換プログラムに推奨される TRUNC コンパイラー・オプション**

推奨コンパイラー・オプション	条件
TRUNC(OPT)	すべての 2 進数データ項目が 2 進数データ項目についての PICTURE および USAGE 節に準拠している。
TRUNC(BIN)	すべての 2 進数データ項目が 2 進数データ項目についての PICTURE および USAGE 節に準拠しているわけではない。

例えば、分離型の CICS 変換プログラムを使用する場合、データ項目を、8 桁より大きな値を受け取る可能性のある PIC S9(8) BINARY として定義するときは、TRUNC(BIN) コンパイラー・オプションを使用するか、項目を USAGE COMP-5 に変更するか、PICTURE 節を変更してください。

また、効果のない以下のオプションの使用を避けることもできます。

- ADV
- FASTSRT
- OUTDD

コンパイラーの入力データ・セットは、変換の結果として受け取ったデータ・セットであり、デフォルトでは SYSPUNCH です。

関連概念
460 ページの『組み込みの CICS 変換プログラム』

関連タスク
458 ページの『CICS オプションを使用したコンパイル』

CICS 予約語テーブル

COBOL は、CICS アプリケーション・プログラム用に代替予約語テーブル (IGYCCICS) を用意しています。コンパイラー・オプション WORD(CICS) が指定されると、CICS のもとでサポートされない COBOL ワードにフラグが立てられ、エラー・メッセージが出されます。

IBM 提供のデフォルトの予約語テーブルにより制約を受ける COBOL ワード以外に、IBM 提供の CICS 予約語テーブルも次の COBOL ワードに制約を与えます。

- CLOSE
- DELETE
- FD
- **FILE**
- **FILE-CONTROL**
- **INPUT-OUTPUT**
- I-O-CONTROL
- MERGE
- OPEN

- READ
- RERUN
- REWRITE
- **SD**
- **SORT**
- START
- WRITE

CICS のもとで SORT ステートメントを使用したい場合は (COBOL は CICS のもとで SORT ステートメントのインターフェースをサポートします)、制限付きとマークされたワードのリストから上記の**太字**のワードを除去するよう CICS 予約語テーブルを変更する必要があります。

関連タスク
458 ページの『CICS オプションを使用したコンパイル』
260 ページの『CICS のもとでのソート』

関連参照
401 ページの『WORD』

CICS HANDLE を使用したエラー処理

CBLPSHPOP ランタイム・オプションの設定は、プログラムが CALL ステートメントを使用して COBOL サブプログラムを呼び出すときの HANDLE 指定の状態に影響を与えます。

CBLPSHPOP が ON の場合は、CALL ステートメントを使用して COBOL サブプログラム (ネストされたプログラムではないもの) が呼び出されると、以下のことが起こります。

1. プログラム初期化の一部として、実行時に、呼び出し側プログラムの HANDLE 指定が延期されます (EXEC CICS PUSH HANDLE を使用して)。
2. 呼び出されたプログラムが独自の HANDLE コマンドを出すまで、HANDLE のデフォルトのアクションが適用されます。
3. プログラム終了の一部として、実行時に、呼び出し側プログラムの HANDLE 指定が回復されます (EXEC CICS POP HANDLE を使用して)。

CICS HANDLE CONDITION または CICS HANDLE AID コマンドを使用する場合は、CICS HANDLE コマンドで指定された LABEL が、CICS HANDLE ラベルへの分岐を引き起こす CICS コマンドと同じ PROCEDURE DIVISION に入っていなければなりません。LABEL オプションを指定した CICS HANDLE コマンドでは、COBOL CALL ステートメントを使用して呼び出された別のプログラムによって引き起こされた条件、援助機能、または異常終了を処理することはできません。LABEL オプションを指定した CICS HANDLE コマンドを使用してプログラム間分岐を実行しようとすると、トランザクションが異常終了します。

ネストされたプログラム内で条件、援助機能、または異常終了が発生する場合、その条件、援助機能、または異常終了の LABEL は、同じネストされたプログラム内に存在しなければなりません。そうでない場合は、予測不能な結果を招きます。

パフォーマンスの考慮事項: CBLPSHPOP が OFF であると、実行時に、COBOL サブプログラムへの CALL に対して CICS PUSH または POP は実行されません。サブプログラムがいずれの EXEC CICS 条件処理コマンドも使用しない場合は、CBLPSHPOP(OFF) を実行することで、PUSH HANDLE コマンドおよび POP HANDLE コマンドのオーバーヘッドを除去できます。その結果、CBLPSHPOP(ON) を指定して実行した場合と比較して、パフォーマンスが向上する場合があります。

VS COBOL II ランタイムから言語環境プログラム・ランタイムにアプリケーションを移行する場合は、関連参照で CBLPSHPOP オプションについての追加考慮事項を参照してください。

『例: CICS HANDLE を使用したエラー処理』

関連タスク

740 ページの『CICS、IMS、または VSAM での効率的な実行』

関連参照

Enterprise COBOL コンパイラーおよびランタイム 移行ガイド (CICS HANDLE コマンドおよび CBLPSHPOP ランタイム・オプション)

Enterprise COBOL Version 3 Performance Tuning

例: CICS HANDLE を使用したエラー処理

次の例は、COBOL プログラムでの CICS HANDLE の使用を示しています。

プログラム A には CICS HANDLE CONDITION コマンドがありますが、プログラム B には CICS HANDLE コマンドはありません。プログラム A はプログラム B を呼び出します。プログラム A はネストされたプログラム A1 も呼び出します。条件は 3 つのシナリオのいずれかで取り扱われます。

(1) CBLPSHPOP(ON): プログラム B の CICS READ コマンドによってある条件が引き起こされても、その条件はプログラム A では処理されません (実行時に CICS PUSH HANDLE が実行されたため、HANDLE 指定が中断されたことが原因です)この条件はトランザクション異常終了となります。

(2) CBLPSHPOP(OFF): プログラム B の CICS READ コマンドによってある条件が引き起こされても、その条件はプログラム A では処理されません (実行時に、LABEL オプションを指定した CICS HANDLE コマンドを使用して、プログラム間分岐の試みを診断します)。この条件はトランザクション異常終了となります。

(3) ネストされたプログラム A1 の CICS READ コマンドによって条件が引き起こされた場合、制御のフローはラベル ERR-1 に移り、予測できない結果が生じます。

```
*************************************************************
*  Program A                                                *
*************************************************************
 ID DIVISION.
 PROGRAM-ID. A.
```

```
 . . .
 PROCEDURE DIVISION.
     EXEC CICS HANDLE CONDITION
               ERROR(ERR-1)
               END-EXEC.
     CALL 'B' USING DFHEIBLK DFHCOMMAREA.
     CALL 'A1'.
     . . .
 THE-END.
     EXEC CICS RETURN END-EXEC.
 ERR-1.
 . . .
* Nested program A1.
 ID DIVISION.
 PROGRAM-ID. A1.
 PROCEDURE DIVISION.
     EXEC CICS READ              (3)
               FILE('LEDGER')
               INTO(RECORD)
               RIDFLD(ACCTNO)
               END-EXEC.
 END PROGRAM A1.
 END PROGRAM A.
*
************************************************************
*  Program B                                               *
************************************************************
 ID DIVISION.
 PROGRAM-ID. B.
 . . .
 PROCEDURE DIVISION.
     EXEC CICS READ              (1)  (2)
               FILE('MASTER')
               INTO(RECORD)
               RIDFLD(ACCTNO)
               END-EXEC.
     . . .
 END PROGRAM B.
```

第 21 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 独立型プリコンパイラーを使用するか、あるいは SQL コンパイラー・オプションを指定してコンパイルし、DB2 coprocessor を使用します。

関連概念
『DB2 coprocessor』
474 ページの『COBOL および DB2 CCSID の決定』

関連タスク
468 ページの『SQL ステートメントのコーディング』
472 ページの『SQL オプションを使用したコンパイル』
479 ページの『DYNAM または NODYNAM コンパイラー・オプションの選択』

関連参照
476 ページの『DB2 プリコンパイラーと DB2 coprocessor の動作方法の相違』

DB2 coprocessor

DB2 coprocessor (DB2 では *SQL ステートメント coprocessor* と呼ばれる) を使用すると、別個のプリコンパイル・ステップを使用しなくても、組み込み SQL ステートメントが含まれたソース・プログラムをコンパイラーが使用するようになります。

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 coprocessor とインターフェースします。この coprocessor 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。

別個のプリコンパイル・ステップの使用も引き続きサポートされますが、coprocessor の使用をお勧めします。

- coprocessor を使用すると、デバッグ・ツールによる対話式デバッグは向上します。リスト中の SQL ステートメント (生成された COBOL ソースではない) が表示されるからです。
- COBOL コンパイラー・リストには、DB2 coprocessor が生成するエラー診断 (SQL ステートメントの構文エラーなど) が含まれます。

- プリコンパイル・ステップ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 coprocessor を使用するときには適用されません。 DB2 coprocessor を使用した場合は次のようになります。
  - ネストされたプログラムのいずれでも SQL ステートメントを使用することができます (プリコンパイラーの場合、SQL ステートメントの使用は最外部のプログラムに制限されます)。
  - コピーブックで SQL ステートメントを使用することができます。
  - REPLACE ステートメントは SQL ステートメントに影響を及ぼします。

DB2 coprocessor を使用してコンパイルすると、オブジェクト・モジュールやリストのような通常の COBOL コンパイラー出力と一緒に、DB2 データベース要求モジュール (DBRM) が生成されます。DBRM は、COBOL コンパイル・ステップに関して JCL 内の DBRMLIB DD ステートメントで指定されたデータ・セットに書き込みます。 DBRM データ・セットには、DB2 バインド・プロセスへの入力として、プログラム内の SQL ステートメントおよびホスト変数に関する情報が入ります。

DB2 coprocessor を使用するプログラムをコンパイルするには、SQL コンパイラー・オプションを指定する必要があります。

関連概念
474 ページの『COBOL および DB2 CCSID の決定』

関連タスク
472 ページの『SQL オプションを使用したコンパイル』

関連参照
476 ページの『DB2 プリコンパイラーと DB2 coprocessor の動作方法の相違』
388 ページの『SQL』

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。

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 THREAD コンパイラー・オプションを使用するプログラムの場合には、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 に宣言する。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。

制約事項: オブジェクト指向クラスまたはメソッドで SQL ステートメントを使用することはできません。

関連タスク
『DB2 coprocessor を用いた SQL INCLUDE の使用』
『SQL ステートメントでの文字データの使用』
470 ページの『SQL ステートメントでの国別 10 進数データの使用』
471 ページの『SQL ステートメントでの国別グループ項目の使用』
471 ページの『SQL ステートメントでのバイナリー項目の使用』
472 ページの『SQL ステートメントの成否の判断』
DB2 アプリケーション・プログラミングおよび SQL ガイド (COBOL アプリケーションでの SQL ステートメントのコーディング)

関連参照 475 ページの『SQL ステートメントのストリング・ホスト変数のコード・ページ決定』 *DB2 SQL リファレンス*

DB2 coprocessor 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* の処理は、REPLACING 句を持たない COPY *literal-1* ステートメント内のリテラルと同じ規則に従います。

SQL INCLUDE ステートメントのライブラリー探索順序は、ライブラリー名を指定しない COBOL COPY ステートメントを解決するためにコンパイラーが使用するのと同じ SYSLIB 連結です。

関連参照 407 ページの『第 18 章 コンパイラー指示ステートメント』
476 ページの『DB2 プリコンパイラーと DB2 coprocessor の動作方法の相違』
COPY ステートメント (*Enterprise COBOL 言語解説書*)

SQL ステートメントでの文字データの使用

EXEC SQL ステートメントで使用する文字データのホスト変数を記述するのに、以下の USAGE 節のいずれかをコーディングできます。すなわち、1 バイトまたは UTF-8 データの場合は USAGE DISPLAY、DBCS データの場合は USAGE DISPLAY-1、または UTF-16 データの場合は USAGE NATIONAL です。

独立型 DB2 プリコンパイラーを使用する場合、USAGE NATIONAL で宣言されたホスト変数に関して、EXEC SQL DECLARE ステートメントでコード・ページ (CCSID) を指定する必要があります。 USAGE DISPLAY または DISPLAY-1 で宣言されたホスト変数のコード・ページを指定しなければならないのは、COBOL CODEPAGE コンパイラー・オプションで有効な CCSID が、文字および図形データ用に DB2 が使用する CCSID と一致しない場合のみです。

次のコードを見てください。統合 DB2 coprocessor を使用する場合には (下記の関連概念に詳述されているように、SQLCCSID コンパイラー・オプションを使用して)、強調表示された 2 つのステートメントは不要です。コード・ページ情報は暗黙的に処理されるからです。

```
CBL CODEPAGE(1140) NSYMBOL(NATIONAL)
. . .
WORKING-STORAGE SECTION.
    EXEC SQL INCLUDE SQLCA END-EXEC.
01  INT1 PIC S9(4) USAGE COMP.
01  C1140.
    49 C1140-LEN  PIC S9(4) USAGE COMP.
    49 C1140-TEXT PIC X(50).
    EXEC SQL DECLARE :C1140 VARIABLE CCSID 1140 END-EXEC.
01  G1200.
    49 G1200-LEN  PIC S9(4) USAGE COMP.
    49 G1200-TEXT PIC N(50) USAGE NATIONAL.
    EXEC SQL DECLARE :G1200 VARIABLE CCSID 1200 END-EXEC.
    . . .
    EXEC SQL FETCH C1 INTO :INT1, :C1140, :G1200 END-EXEC.
```

EXEC SQL DECLARE *variable-name* VARIABLE CCSID *nnnn* END-EXEC を指定した場合は、その指定によって暗黙の CCSID がオーバーライドされます。例えば、次のようなコードを使用すると、DB2 は、C1208-TEXT を、COBOL の CODEPAGE コンパイラー・オプションに対して有効な CCSID としてではなく、UTF-8 (CCSID 1208) でエンコードされているものとして扱います。

```
01 C1208.
    49 C1208-LEN  PIC S9(4) USAGE COMP.
    49 C1208-TEXT PIC X(50).
    EXEC SQL DECLARE :C1208 VARIABLE CCSID 1208 END-EXEC.
```

NSYMBOL コンパイラー・オプションは、EXEC SQL ステートメントの内部の文字リテラルには影響を及ぼしません。EXEC SQL ステートメントの文字リテラルは、文字定数についての SQL の規則に従います。

関連概念
474 ページの『COBOL および DB2 CCSID の決定』

関連タスク
DB2 アプリケーション・プログラミングおよび SQL ガイド (COBOL アプリケーションでの SQL ステートメントのコーディング)

関連参照
476 ページの『DB2 プリコンパイラーと DB2 coprocessor の動作方法の相違』
350 ページの『CODEPAGE』
DB2 SQL リファレンス

SQL ステートメントでの国別 10 進数データの使用

統合 DB2 coprocessor または DB2 プリコンパイラーのどちらかを使用すると、国別 10 進数ホスト変数を EXEC SQL ステートメントで使用できます。どちらの場合にも CCSID を EXEC SQL DECLARE ステートメントで指定する必要はありません。CCSID 1200 が自動的に使用されます。

EXEC SQL ステートメントで指定する国別 10 進数ホスト変数はいずれも次のような特性を持っている必要があります。

- 符号付きでなければなりません。
- SIGN LEADING SEPARATE 節で指定する必要があります。
- 暗黙的または明示的に USAGE NATIONAL が有効である必要があります。

関連概念
54 ページの『数値データの形式』

関連タスク
142 ページの『国別数値データ項目の定義』

関連参照
476 ページの『DB2 プリコンパイラーと DB2 coprocessor の動作方法の相違』

SQL ステートメントでの国別グループ項目の使用

国別グループ項目を EXEC SQL ステートメントでホスト変数として使用することができます。国別グループ項目は、基本項目としてではなく、グループ・セマンティクスで (すなわち、そのグループ項目に従属するホスト変数セットの省略表現として) 扱われます。

国別グループ内のすべての従属項目は USAGE NATIONAL を持っている必要があるので、国別グループ項目では可変長ストリングを記述できません。

関連タスク
144 ページの『国別グループの使用』

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 として宣言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 として宣言する場合は、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
54 ページの『数値データの形式』

関連参照
397 ページの『TRUNC』

SQL ステートメントの成否の判断

DB2 は、SQL ステートメントの実行を終了すると、戻りコードを、SQLCA 構造体に入れて (例外が 1 つあります) 上で送り、操作が成功したか失敗したかを示します。プログラムは戻りコードをテストし、必要なアクションを取らなければなりません。

例外が起こるのは、プログラムが DSN のもとで、TSO バッチ・モード・モジュール IKJEFT01 の代替入り口点の 1 つ (IKJEFT1A または IKJEFT1B) から実行された場合です。この場合、戻りコードはレジスター 15 に入れて渡されます。

SQL ステートメントの実行後、RETURN-CODE 特殊レジスターの内容が有効でなくなる場合があります。このため、SQL ステートメントが正常に実行され、COBOL プログラムが正常に終了した場合でも、ジョブ・ステップが未定義の戻りコードで終了することがあります。終了時に意味のある戻りコードが渡されるようにするには、プログラムを終了する前に、RETURN-CODE 特殊レジスターを設定します。

関連タスク
DB2 アプリケーション・プログラミングおよび SQL ガイド (COBOL アプリケーションでの SQL ステートメントのコーディング)

SQL オプションを使用したコンパイル

SQL コンパイラー・オプションを使用すると、DB2 coprocessor が使用可能になり、DB2 サブオプションを指定できるようになります。

SQL オプションは、コンパイラー・オプション・ソース (すなわち、コンパイラー呼び出し、PROCESS または CBL ステートメント、あるいはインストール・デフォルト) のいずれにでも指定することができます。SQL オプションが COBOL インストール・デフォルトであるときは、DB2 サブオプションを指定することはできません。ただし、DB2 プロダクトのインストール・デフォルトをカスタマイズすることによって、デフォルトの DB2 サブオプションを指定することができます。

SQL コンパイラー・オプションで指定された DB2 サブオプション・ストリングは、DB2 coprocessor で使用可能になります。ストリングの内容を見るのは、DB2 coprocessor だけです。

DB2 coprocessor を使用するには、下記の表に示されているオプションを指定してコンパイルする必要があります。 また、コンパイルを行うマシンで DB2 が使用可能でなければなりません。.

表 62. **DB2 coprocessor に必要なコンパイラー・オプション**

コンパイラー・オプション	説明
SQL	一緒に NOLIB を使用すると、LIB が強制的にオンにされます。
LIB	SQL と一緒に指定する必要があります。
SIZE(*xxx*)	*xxx* は、DB2 coprocessor サービスのための十分なストレージをユーザー領域に残すようなサイズ値 (MAX ではない) です。

標準の JCL プロシージャー・ステートメントを使用して、プログラムを DB2 coprocessor でコンパイルすることができます。上記のコンパイラー・オプションを指定する以外に、以下の項目を JCL に指定する必要があります。

- 生成済みデータベース要求モジュール (DBRM) の位置を指定した DBRMLIB DD ステートメント。
- DB2 coprocessor サービスが含まれたデータ・セットを追加する COBOL ステップ用の STEPLIB オーバーライド (これらのサービスが LNKLST に含まれている場合を除く)。通常、このデータ・セットは DSN910.SDSNLOAD ですが、インストール先で名前を変更している場合もあります。

例えば、以下の行が JCL に含まれる場合があります。

```
//DBRMLIB  DD  DSN=PAYROLL.MONTHLY.DBRMLIB.DATA(MASTER),DISP=SHR
//STEPLIB  DD  DSN=DSN910.SDSNLOAD,DISP=SHR
```

バッチでのコンパイル: SQL オプションを使用して、一連の COBOL プログラム (バッチ・コンパイル・シーケンス) を含んでいるソース・ファイルをコンパイルする場合、SQL オプションはバッチ・シーケンスの最初のプログラムに有効でなければなりません。SQL オプションが CBL または PROCESS ステートメントに指定された場合、CBL または PROCESS ステートメントは、バッチ・シーケンスの最初のプログラムより前に置かれなければなりません。

関連概念
467 ページの『DB2 coprocessor』
474 ページの『COBOL および DB2 CCSID の決定』

関連タスク
『DB2 サブオプションの分離』
479 ページの『DYNAM または NODYNAM コンパイラー・オプションの選択』

関連参照
361 ページの『DYNAM』
388 ページの『SQL』
DB2 コマンド解説書

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・ファイルに以下のコードが含まれているとします。

```
//STEP1 EXEC IGYWC, . . .
//  PARM.COBOL='SQL("string1")'
//COBOL.SYSIN    DD *
    CBL SQL("string2")
    CBL SQL("string3")
      IDENTIFICATION DIVISION.
    PROGRAM-ID. DRIVER1.
```

コンパイル時、コンパイラーは次のサブオプション・ストリングを DB2 coprocessor 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。

COBOL および DB2 CCSID の決定

すべての DB2 ストリング・データ (BLOB、BINARY、および VARBINARY データを除く) には関連するコード化スキームとコード化文字セット ID (CCSID) があります。このことは、固定長および可変長の文字ストリング、固定長および可変長の図形文字ストリング、CLOB ホスト変数、ならびに DBCLOB ホスト変数にもいえます。

組み込み DB2 coprocessor を使用すると、SQL ステートメント処理で使用されるストリング・ホスト変数に関連付けられるコード・ページ CCSID の決定は、COBOL SQLCCSID オプションの設定、使用されるプログラミング手法、および各種 DB2 構成オプションによって異なります。

SQL および SQLCCSID COBOL コンパイラー・オプションを使用すると、CODEPAGE コンパイラー・オプションで指定されたり、ホスト変数の COBOL データ・タイプから決定されたりする CCSID 値 *nnnnn* は自動的に COBOL から DB2 に伝えられます。 DB2 はホスト変数に COBOL CCSID を関連付け、そうしなければ DB2 の外部メカニズムおよびデフォルトによって暗黙に指定される CCSID をオーバーライドします。この関連付けられた CCSID は、ホスト変数を参照する SQL ステートメントの処理に使用されます。

SQL および NOSQLCCSID コンパイラー・オプションを使用すると、CODEPAGE コンパイラー・オプションで指定される CCSID 値 *nnnnn* は COBOL プログラム内の COBOL ステートメントの処理にのみ使用されます。この CCSID は SQL ステートメントの処理には使用されません。代わって、DB2 は、SQL ステートメントの処理で、DB2 の外部メカニズムおよびデフォルトによって指定される CCSID に従ってホスト変数のデータ値がエンコードされるものと見なします。

関連概念
467 ページの『DB2 coprocessor』

関連タスク 475 ページの『SQLCCSID または NOSQLCCSID オプションを使用したプログラミング』

関連参照 475 ページの『SQL ステートメントのストリング・ホスト変数のコード・ページ決定』 350 ページの『CODEPAGE』 388 ページの『SQL』 390 ページの『SQLCCSID』

SQL ステートメントのストリング・ホスト変数のコード・ページ決定

統合 DB2 coprocessor (SQL コンパイラー・オプション) を使用すると、SQL ステートメントのストリング・ホスト変数を処理するコード・ページは、優先度の高い順から、次に示すように決定されます。

- USAGE NATIONAL を持つホスト変数は、常に DB2 によって CCSID 1200 (Unicode UTF-16) を使用して処理されます。次に例を示します。

  ```
  01  hostvariable pic n(10) usage national.
  ```

- 明示的な FOR BIT DATA 宣言を持つ英数字ホスト変数は、DB2 によって CCSID 66535 に設定されます。これは、変数がエンコードされた文字を表さないことを示します。次に例を示します。

  ```
  EXEC SQL DECLARE hostvariable VARIABLE FOR BIT DATA END-EXEC
  ```

- BLOB、BINARY、または VARBINARY ホスト変数には CCSID 関連がありません。これらのストリング・タイプはエンコードされた文字を表しません。
- SQLDA で明示的な CCSID オーバーライドを指定したホスト変数は、その CCSID で処理されます。
- 明示的な CCSID を使用して宣言で指定したホスト変数は、その CCSID で処理されます。次に例を示します。

  ```
  EXEC SQL DECLARE hostvariable VARIABLE CCSID nnnnn END-EXEC
  ```

- 英数字ホスト変数は、SQLCCSID コンパイラー・オプションが有効であると、CODEPAGE コンパイラー・オプションからの CCSID *nnnnn* で処理されます。
- DBCS ホスト変数は、SQLCCSID オプションが有効であると、マップされた値 *mmmmm* で処理されます。これは、CODEPAGE(*nnnnn*) コンパイラー・オプションからの混合 (MBCS) CCSID *nnnnn* の純粋 DBCS CCSID コンポーネントです。
- 英数字または DBCS ホスト変数は、NOSQLCCSID オプションが有効であると、DB2 ENCODING バインド・オプション (指定された場合) からの CCSID か、DB2 インストール・パネル DSNTIPF を通じて DSNHDECP で設定された APPLICATION ENCODING からの CCSID で処理されます。

関連参照
350 ページの『CODEPAGE』
390 ページの『SQLCCSID』

SQLCCSID または NOSQLCCSID オプションを使用したプログラミング

一般に、統合 DB2 coprocessor を使用する新しいアプリケーションの場合は、SQLCCSID オプションが推奨されます。また、既存のアプリケーションの場合は長期的な方向性として、このオプションが推奨されます。既存のプリコンパイラー・ベースのアプリケーションを、統合 DB2 coprocessor を使用するように移行するためのメカニズムとしては、NOSQLCCSID オプションが推奨されます。

以下の特性のいずれかを備えた COBOL-DB2 アプリケーションの場合は、SQLCCSID オプションが推奨されます。

- COBOL Unicode サポートを使用する

- CCSID エンコードに間接に依存する他の COBOL 構文 (Java の相互運用性のための XML サポートやオブジェクト指向構文など) を使用する
- DB2 が前提とするデフォルトの CCSID とは異なる CCSID でエンコードされる文字データを処理する

DB2 プリコンパイラーの動作との最高の互換性を必要とするアプリケーションの場合は、NOSQLCCSID オプションが推奨されます。

COBOL 英数字データ項目を、FOR BIT DATA サブタイプで定義される DB2 ストリング・データとやり取りするホスト変数として使用する場合は、次のいずれかを行う必要があります。

- NOSQLCCSID コンパイラー・オプションを使用する
- それらのホスト変数に対して、明示的な FOR BIT DATA 宣言を指定する。例えば、次のように指定します。

```
EXEC SQL DECLARE hostvariable VARIABLE FOR BIT DATA END-EXEC
```

使用上の注意

- DB2 DCLGEN コマンドを使用してテーブルの COBOL 宣言を生成する場合は、必要なら、FOR BIT DATA 宣言を自動的に作成させることもできます。これを行うには、DCLGEN コマンドの DCLBIT(YES) オプションを指定します。
- **パフォーマンスに関する考慮事項:** SQLCCSID コンパイラー・オプションを使用すると、SQL 処理にかなりのパフォーマンス・オーバーヘッドがかかることがあります。これは、SQLCCSID が有効であると、デフォルトの DB2 CCSID 関連メカニズムが、ホスト変数単位で機能するメカニズムによってオーバーライドされるからです。

関連概念
467 ページの『DB2 coprocessor』

関連参照
390 ページの『SQLCCSID』

DB2 プリコンパイラーと DB2 coprocessor の動作方法の相違

以下のセクションでは、独立型 COBOL DB2 プリコンパイラーと統合 COBOL DB2 coprocessor の動作の相違を列挙します。

EXEC SQL INCLUDE ステートメントの最後のピリオド

プリコンパイラー: DB2 プリコンパイラーでは、各 EXEC SQL INCLUDE ステートメントをピリオドで終了する必要がありません。ピリオドが指定されていると、プリコンパイラーはそれをステートメントの一部として処理します。ピリオドが指定されていないと、プリコンパイラーはあたかもピリオドが指定されているかのようにステートメントを受け入れます。

Coprocessor: DB2 coprocessor はそれぞれの EXEC SQL INCLUDE ステートメントを COPY ステートメントのように扱います。DB2 coprocessor では、ピリオドでステートメントが終了していることが必要です。以下に例を示します。

```
IF A = B THEN
    EXEC SQL INCLUDE some_code_here END-EXEC.
             ELSE
    . . .
           END-IF
```

IF ステートメントがピリオドで終了していないことに注意してください。

EXEC SQL INCLUDE とネストされた COPY REPLACING

プリコンパイラー: DB2 プリコンパイラーでは、EXEC SQL INCLUDE ステートメントが、REPLACING 句を使用する COPY ステートメントを含むコピーブックを参照できます。

Coprocessor: DB2 coprocessor では、EXEC SQL INCLUDE ステートメントは、REPLACING 句を使用する COPY ステートメントを含むコピーブックを参照できません。 coprocessor は各 EXEC SQL INCLUDE ステートメントを COPY ステートメントと同様に処理します。ネストされた COPY ステートメントに REPLACING 句を指定することはできません。

EXEC SQL と REPLACE または COPY REPLACING

プリコンパイラー: DB2 プリコンパイラーを使用して、COBOL REPLACE ステートメントおよび COPY ステートメントの REPLACING 句は、 EXEC SQL ステートメントで作成された拡張ソースに作用します。 REPLACE および REPLACING の COBOL 規則が使用されます。

コプロセッサー: DB2 コプロセッサーを使用して、REPLACE および COPY . . . REPLACING ステートメントは、EXEC SQL ステートメントなど、元のソース・プログラムに作用します。

次の例のように、異なる動作が起こる可能性があります。

```
REPLACE == ABC == By == XYZ ==.
01  G.
    02  ABC PIC X(10).
    . . .
    EXEC SQL SELECT * INTO :G.ABC FROM TABLE1 END-EXEC
```

プリコンパイラーでは、G.ABC への参照は、拡張ソースで ABC of G として表示され、XYZ of G で置き換えられます。コプロセッサーでは、ABC が元のソース・ストリング G.ABC で、区切り文字で区切られていないので、置換は行われません。

END-EXEC ステートメントの後のソース・コード

プリコンパイラー: DB2 プリコンパイラーは、同じ行で END-EXEC ステートメントより後にあるコードをすべて無視します。

Coprocessor: DB2 coprocessor は、同じ行で END-EXEC ステートメントより後にあるコードを処理します。

ホスト変数の複数定義

プリコンパイラー: DB2 プリコンパイラーでは、ホスト変数の参照が固有である必要はありません。有効な DB2 データ・タイプに最初にマップした定義が使用されます。

Coprocessor: DB2 coprocessor では、各ホスト変数参照が固有でなければなりません。 coprocessor は、ホスト変数に対する固有でない参照を診断します。 ホスト変数参照を完全修飾して、それが固有なものとなるようにする必要があります。

EXEC SQL ステートメントの継続行

プリコンパイラー: DB2 プリコンパイラーでは、EXEC SQL ステートメントの開始桁が 12 から 72 の間でなければなりません。ステートメントの継続行は、8 から 72 桁目の間であればどこからでも開始できます。

Coprocessor: DB2 coprocessor では、継続行も含め EXEC SQL ステートメントのすべての行を 12 から 72 桁目の間にコーディングする必要があります。

ビット・データ・ホスト変数

プリコンパイラー: DB2 プリコンパイラーでは、COBOL の英数字データ項目を、サブタイプ FOR BIT DATA を持つ DB2 文字データを保持するホスト変数として使用することができます。そのホスト変数を FOR BIT DATA として宣言する明示の EXEC SQL DECLARE VARIABLE ステートメントは必要ありません。

Coprocessor: DB2 coprocessor では、COBOL の英数字データ項目を、サブタイプ FOR BIT DATA を持つ DB2 文字データを保持するホスト変数として使用できますが、その場合は、COBOL プログラム内でそのホスト変数に対して明示の EXEC SQL DECLARE VARIABLE ステートメントが指定されていなければなりません。以下に例を示します。

```
EXEC SQL DECLARE :HV1 VARIABLE FOR BIT DATA END-EXEC.
```

EXEC SQL DECLARE . . . FOR BIT DATA ステートメントを追加する代わりに、NOSQLCCSID コンパイラー・オプションを使用することもできます。詳細については、下記のコード・ページ決定に関する関連参照を参照してください。

SQL-INIT-FLAG

プリコンパイラー: DB2 プリコンパイラーでは、プログラムが複数回呼び出されたときに異なるアドレスに存在する可能性のあるホスト変数を渡す場合は、呼び出し先プログラムが SQL-INIT-FLAG をリセットする必要があります。このフラグをリセットすることにより、次回に SQL ステートメントが実行されたとき、ストレージを初期化するよう DB2 に指示することができます。このフラグをリセットするには、呼び出し先プログラムの PROCEDURE DIVISION にステートメント MOVE ZERO TO SQL-INIT-FLAG を挿入します。ただし、当該ホスト変数を使用する実行可能な SQL ステートメントより前に挿入する必要があります。

Coprocessor: DB2 coprocessor では、呼び出し先プログラムが SQL-INIT-FLAG をリセットする必要はありません。プログラムの移植を容易にするため、プログラム内

で SQL-INIT-FLAG が自動的に定義されます。ただし、SQL-INIT-FLAG を変更するステートメント (MOVE ZERO TO SQL-INIT-FLAG など) は、プログラム内の SQL 処理に影響を及ぼしません。

関連概念
467 ページの『DB2 coprocessor』

関連参照 475 ページの『SQL ステートメントのストリング・ホスト変数のコード・ページ決定』 390 ページの『SQLCCSID』

DYNAM または NODYNAM コンパイラー・オプションの選択

EXEC SQL ステートメントを含んでいる COBOL プログラムの場合、コンパイラー・オプションの DYNAM を選ぶか NODYNAM を選ぶかは、稼働環境によって異なります。

次に、稼働環境とそれに対応する処置を示します。

- TSO または IMS: DYNAM または NODYNAM のいずれかのコンパイラー・オプションを使用できます。

  IMS と DB2 は、言語インターフェース・モジュールの共通別名 DSNHLI を共用していることに注意してください。ライブラリーは次のように連結する必要があります。

  – DYNAM オプションを指定して IMS を使用する場合、IMS ライブラリーを最初に連結してください。
  – DB2 のもとでのみアプリケーションを実行する場合、DB2 ライブラリーを最初に連結してください。
- CICS または DB2 呼び出し接続機能 (CAF): NODYNAM コンパイラー・オプションを使用する必要があります。

  ストアード・プロシージャーでは CAF を使用するので、COBOL ストアード・プロシージャーも NODYNAM オプションを指定してコンパイルする必要があります。

関連タスク
472 ページの『SQL オプションを使用したコンパイル』
DB2 アプリケーション・プログラミングおよび SQL ガイド (呼び出し接続機能のプログラミング)

関連参照
361 ページの『DYNAM』

第 22 章 COBOL プログラムの開発 (IMS の場合)

COBOL プログラムのコーディングは、IMS のもとで実行する場合とほとんど同じですが、以下の推奨事項と制約事項に留意してください。

COBOL では、IMS メッセージ処理プログラム (MPP) は、IMS 以外の入力ステートメントまたは出力ステートメント (READ、WRITE、REWRITE 、OPEN、および CLOSE など) を使用しません。

Enterprise COBOL では、次のインターフェースを使用して IMS 機能を呼び出すことができます。

- CBLTDLI 呼び出し
- 言語環境プログラム呼び出し可能サービス CEETDLI

CEETDLI への呼び出しは、CBLTDLI への呼び出しと同様にコーディングされます。CEETDLI は基本的には CBLTDLI と同じことを行います。

IMS Java 従属領域でもオブジェクト指向 COBOL プログラムを実行できます。また、単一のアプリケーションで、オブジェクト指向 COBOL と Java 言語を併用できます。

関連タスク 『IMS のもとで実行するための COBOL プログラムのコンパイルおよびリンク』 482 ページの『IMS のもとでのオブジェクト指向 COBOL と Java の使用』 483 ページの『IMS Java アプリケーションからの COBOL メソッドの呼び出し』 484 ページの『COBOL で開始する COBOL と Java の混合アプリケーションの作成』 484 ページの『混合言語 IMS アプリケーションの作成』

IMS のもとで実行するための COBOL プログラムのコンパイルおよびリンク

IMS 環境で最高のパフォーマンスを得るためには、RENT コンパイラー・オプションを使用してください。このオプションを使用すると、COBOL で再入可能コードが生成されます。その後、アプリケーション・プログラムを*プリロード・モード* (プログラムが常にストレージにある) または*非プリロード・モード*のいずれでも実行することができます。別のオプションで再コンパイルする必要はありません。

IMS を使用すると、COBOL プログラムをプリロードできます。プリロードによってパフォーマンスが向上します。これは、プログラムが既にストレージにあると (必要が生じるたびにライブラリーから取り出すよりも) プログラムに対する後続の要求をより速く処理できるためです。

IMS プログラムについては、RENT コンパイラー・オプションを使用することをお勧めします。プリロードして実行するプログラム、またはプリロードおよび非プリロードの両方で実行するプログラムには、RENT コンパイラー・オプションを使用する必要があります。さらに、COBOL プログラムを含むロード・モジュールをプリロ

ードするときは、そのロード・モジュール内のすべての COBOL プログラムが RENT オプションを指定してコンパイルされていなければなりません。

RENT オプションを指定してコンパイルされたプログラムは、z/OS リンク・パック域に入れることができます。そこでは、IMS 従属領域間でプログラムを共用することができます。

16MB 境界より上で実行するには、RENT または NORENT RMODE(ANY) を指定してアプリケーション・プログラムをコンパイルする必要があります。IMS アプリケーション・プログラムのデータは 16MB 境界より上に常駐させることができ、IMS サービスを使用するプログラムでは DATA(31) RENT または RMODE(ANY) NORENT を指定することができます。

IMS のもとで COBOL プログラムを正しく実行するために推奨されるリンク・エディット属性は、次のとおりです。

- RENT コンパイラー・オプションを指定してコンパイルされた COBOL プログラムのみを含むロード・モジュールをリンクするには、RENT としてリンクする。
- COBOL RENT プログラムとその他のプログラムが混在するロード・モジュールをリンクするために、他のプログラムについて推奨されるリンク・エディット属性を使用する。

関連概念
45 ページの『ストレージとそのアドレス可能度』

関連タスク
479 ページの『DYNAM または NODYNAM コンパイラー・オプションの選択』
言語環境プログラム・プログラミング・ガイド (IMS での条件処理)

関連参照
354 ページの『DATA』
383 ページの『RENT』
Enterprise COBOL コンパイラーおよびランタイム 移行ガイド (IMS の考慮事項)

IMS のもとでのオブジェクト指向 COBOL と Java の使用

IMS Java 従属領域で稼働するアプリケーションで、オブジェクト指向 COBOL と Java を併用できます。

例えば、次のようなことが可能です。

- IMS Java アプリケーションから COBOL のメソッドを呼び出します。アプリケーションのメッセージ処理部分を Java で作成し、 COBOL のメソッドを呼び出して IMS データベースにアクセスします。
- COBOL クラスの main メソッドで開始して Java のルーチンを呼び出す、COBOL と Java の混合アプリケーションを作成します。

このようなアプリケーションは、Java メッセージ処理 (JMP) 従属領域または Java バッチ処理 (JBP) 従属領域で実行する必要があります。メッセージ・キューから読み取るプログラムは、言語に関係なく、JMP 従属領域で実行する必要があります。

関連タスク 648 ページの『ファクトリー・セクションの定義』 613 ページの『第 30 章 オブジェクト指向プログラムの作成』 663 ページの『第 31 章 Java メソッドとの通信』 329 ページの『第 16 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行』 *IMS Java 手引きおよび解説書*

IMS Java アプリケーションからの COBOL メソッドの呼び出し

Enterprise COBOL のオブジェクト指向言語機能を利用すれば、IMS Java プログラムから呼び出すことのできる COBOL メソッドを記述できます。

COBOL クラスを定義し、Enterprise COBOL コンパイラーでコンパイルすると、コンパイラーによって、ネイティブ・メソッドを含んだ Java クラス定義およびそれらのネイティブ・メソッドを実装するオブジェクト・コードが生成されます。これらを使用すれば、ほかのクラスを使用する場合とまったく同じように、このクラスのインスタンスを作成し、IMS Java 従属領域で稼働する IMS Java プログラムからクラスのメソッドを呼び出すことができます。

例えば、適切な DL/I 呼び出しを使用して IMS データベースにアクセスする COBOL クラスを定義できます。このクラスの実装を IMS Java プログラムから利用できるようにするためには、以下のステップを実行します。

1. Enterprise COBOL コンパイラーで COBOL のクラスをコンパイルし、クラス定義を含む Java ソース・ファイル (.java) と、ネイティブ・メソッドのインプリメンテーションを含むオブジェクト・モジュール (.o) を生成します。
2. 生成された Java ソース・ファイルを Java コンパイラーでコンパイルして、クラス・ファイル (.class) を生成します。
3. オブジェクト・コードをリンクして、HFS (.so) のダイナミック・リンク・ライブラリー (DLL) を生成しますCOBOL DLL を含む HFS ディレクトリーは、IMS 領域プロシージャーの ENVIRON= パラメーターで示されている IMS.PROCLIB メンバーで指定されているように、LIBPATH でリストされなければなりません。
4. マスター JVM オプション・メンバーの共有可能アプリケーション・クラス・パス (IMS 領域プロシージャーの JVMOPMAS= パラメーターで指定されている IMS.PROCLIB メンバー内の ibm.jvm.sharable.application.class.path) を更新し、JVM が Java クラス・ファイルにアクセスできるようにします。

混合言語アプリケーションの初期ルーチンを Java で作成する場合は、IMS Java の IMSApplication クラスから派生するクラスをインプリメントする必要があります。

Java プログラムからプロシージャー型 COBOL プログラムを直接呼び出すことはできません。既存の COBOL IMS コードを再利用するには、次のいずれかの技法を使用します。

- COBOL のコードを COBOL クラスのメソッドとして再構築します。
- 既存のプロシージャー型コードに対するラッパーとして機能する COBOL のクラス定義とメソッドを記述します。ラッパーのコードでは、COBOL の `CALL` ステートメントを使用して、プロシージャー型 COBOL プログラムにアクセスできます。

関連タスク 329 ページの『第 16 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行』 659 ページの『OO アプリケーションの構造化』 658 ページの『プロシージャー指向 COBOL プログラムのラッピング』 *IMS Java 手引きおよび解説書*

COBOL で開始する COBOL と Java の混合アプリケーションの作成

IMS Java 従属領域で稼働するアプリケーションは、クラスの main メソッドで開始する必要があります。main ファクトリー・メソッドを持つ COBOL クラス定義は、この要件を満たしています。したがって、COBOL と Java IMS の混合アプリケーションの先頭のルーチンとして、このメソッドを使用できます。

Enterprise COBOL は、main メソッドを備えた Java クラスを生成します。IMS Java 従属領域は、IMS Java IMSApplication サブクラスの main メソッドに対して領域が行うのと同じ方法で、このメソッドを検索し、インスタンス化し、呼び出すことができます。アプリケーション全体を COBOL でコーディングできますが、通常は、Java ルーチンを呼び出すためにこの種のアプリケーションを作成します。COBOL ランタイム・サポートは、IMS Java 従属領域の JVM 内で稼働しているときは、この JVM を自動的に検出し、JVM を使用して Java クラスのメソッドを呼び出します。

ただし、COBOL アプリケーションは IMSApplication クラスから派生しないので、IMS Java クラスを使用して、メッセージを処理したり、トランザクションを同期したりすることはできません。代わりに、メッセージの処理 (GU および GN) やトランザクションの同期 (CHKP) のためには、COBOL で DL/I 呼び出しを使用する必要があります。

関連タスク
659 ページの『OO アプリケーションの構造化』
IMS Java 手引きおよび解説書
Persistent Reusable Java Virtual Machine User's Guide

混合言語 IMS アプリケーションの作成

混合言語 IMS アプリケーションを記述する場合には、STOP RUN ステートメントの効果に注意して、メッセージの処理方法およびトランザクションの同期方法を理解し、データベースにアクセスし、アプリケーション・インターフェース・ブロック (AIB) を使用する必要があります。

関連タスク
485 ページの『STOP RUN ステートメントの使用』
485 ページの『メッセージの処理とトランザクションの同期』
485 ページの『データベースへのアクセス』
486 ページの『アプリケーション・インターフェース・ブロックの使用』

STOP RUN ステートメントの使用

アプリケーションの COBOL の部分で STOP RUN ステートメントを使用すると、COBOL と Java のすべてのルーチン (JVM を含む) が終了します。

制御は即座に IMS に返されます。プログラムとトランザクションは、停止状態のままになります。

メッセージの処理とトランザクションの同期

IMS メッセージ処理アプリケーションは、COBOL と Java の両方の言語で記述されたアプリケーション・コンポーネントにロジックを分散させるのではなく、COBOL または Java のいずれかで、すべてのメッセージ処理とトランザクション同期を実行する必要があります。

COBOL コンポーネントでは、メッセージ処理用 (GU と GN) およびトランザクション同期用 (CHKP) の DL/I サービスに対して、CALL ステートメントを使用します。Java コンポーネントは、IMS Java クラスを使用してこれらの機能を実行します。IMSFieldMessage から派生したクラスのオブジェクト・インスタンスを使用して、アプリケーションの COBOL コンポーネントと Java コンポーネントの間のすべての IMS のメッセージ通信を行うことができます。

関連タスク
IMS Java 手引きおよび解説書
IMS アプリケーション・プログラミング: トランザクション・マネージャー

データベースへのアクセス

IMS データベースへのアクセスには、Java または COBOL のどちらかだけを使用することも、両方の言語を併用することもできます。

制約事項: IMS Java 従属領域で稼働する COBOL ルーチンでは、現在、DB2 データベース・アクセス用の EXEC SQL ステートメントはサポートされていません。

推奨: Java と COBOL の両方から同じデータベース・プログラム連絡ブロック (PCB) にアクセスしないでください。アプリケーションの Java 部分と COBOL 部分は、同じデータベース位置を共用します。アプリケーションのある部分での呼び出しでデータベース位置が変化すると、アプリケーションの別の部分でのデータベース位置に影響があります。 (アプリケーションの影響を受ける部分が同じ言語で記述されていても、異なる言語で記述されていても、この問題は発生します。)

混合アプリケーションの Java コンポーネントが SQL SELECT 節を作成し、Java Database Connectivity (JDBC) を使用して、IMS データベースを照会して結果を取り出すものとします。IMS Java クラス・ライブラリーは、IMS に対する適切な要求を構成し、データベースの正しい位置を確立します。その後、セグメント検索引数 (SSA) を作成し、同じデータベース PCB に対する GU (Get Unique) 要求を IMS に対して発行する COBOL メソッドを呼び出すと、おそらく、その PCB に対するデータベース中の位置がその要求によって変化します。この場合、最初の SQL SELECT 節を使用してさらにレコードを取り出そうとする後続の JDBC 要求は、データベース位置が変化しているため誤りになります。複数の言語から同じ PCB にアクセス

する必要がある場合は、言語間呼び出しの後で、データベースのレコードにさらにアクセスする前に、データベース位置を確立し直してください。

関連タスク
IMS Java 手引きおよび解説書

アプリケーション・インターフェース・ブロックの使用

IMS Java 従属領域で実行される COBOL アプリケーションは通常、AIB インターフェースを使用する必要があります。これは、IMS Java 従属領域が、PCB アドレスをアプリケーションに提供しないからです。

AIB インターフェースを使用するには、PCB 名 (PSBGEN の一部として定義されていなければならない) を AIB のリソース名フィールドに設定することで、呼び出しに対して要求された PCB を指定します。 (AIB では、プログラム仕様ブロック (PSB) 定義のすべての PCB に名前がある必要があります。) PCB アドレスを直接指定することはなく、アプリケーションは PCB リスト内での PCB の相対的な位置を知る必要はありません。呼び出しが完了すると、 AIB は、アプリケーションが渡した PCB 名に対応する PCB アドレスを戻します。

または、リソース名としてサブ関数 FIND と PCB 名を使用して、IMS INQY 呼び出しを行うことによって、PCB アドレスを取得することができます。この呼び出しでは、PCB のアドレスが返され、このアドレスを COBOL プログラムに渡すことができます。 (この方法でも、PCB 名を PSBGEN の一部として定義する必要がありますが、アプリケーションで AIB インターフェースを使用する必要はなくなります。)

『例: アプリケーション・インターフェース・ブロックの使用』

関連タスク
IMS Java 手引きおよび解説書

例: アプリケーション・インターフェース・ブロックの使用:

次の例は、COBOL アプリケーションで AIB インターフェースを使用するにはどうすればよいかを示しています。

```
Local-storage section.
   copy AIB.
   . . .
Linkage section.
01 IOPCB.
   05 logtterm     pic x(08).
   05             pic x(02).
   05 tpstat       pic x(02).
   05 iodate       pic x(04).
   05 iotime       pic x(04).
   05             pic x(02).
   05 seqnum       pic x(02).
   05 mod          pic x(08).
Procedure division.
   Move spaces to input-area
   Move spaces to AIB
   Move "DFSAIB" to AIBRID
   Move length of AIB to AIBRLEN
   Move "IOPCB" to AIBRSNM1
   Move length of input-area to AIBOALEN
```

```
    Call "CEETDLI" using GU, AIB, input-area
    Set address of IOPCB to AIBRESA1
    If tpstat = spaces
* . . process input message
```

第 23 章 UNIX のもとでの COBOL プログラムの実行

z/OS UNIX 環境で COBOL プログラムを実行するには、Enterprise COBOL または COBOL for OS/390 & VM を使用して COBOL プログラムをコンパイルします。これらのプログラムは再入可能でなければなりません。したがって、コンパイラーおよびリンカー・オプション RENT を使用してください。

HFS から実行する予定であれば、リンカー・オプション AMODE 31 を使用してください。z/OS UNIX アプリケーション内から呼び出す AMODE 24 プログラムは、MVS PDS または PDSE に常駐させる必要があります。

z/OS UNIX のもとでの実行には、以下の制約事項が適用されます。

- SORT および MERGE ステートメントはサポートされません。
- 事前初期設定用として旧バージョンの COBOL インターフェース (ランタイム・オプション RTEREUS と関数 IGZERRE および ILBOSTP0) を使用した場合、再使用可能な環境を確立することはできません。
- NOTHREAD オプションを指定してコンパイルした COBOL プログラムを複数のスレッドで実行することはできません。 2 番目のスレッドで COBOL アプリケーションを開始すると、COBOL 実行時に、ソフトウェア条件を受け取ります。NOTHREAD を指定してコンパイルした COBOL プログラムは、初期プロセス・スレッド (IPT) か、C または PL/I のルーチンから作成する 1 つの非 IPT 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 COBOL プログラムが THREAD オプションを指定してコンパイルされている場合は、複数のスレッドで COBOL プログラムを実行できます。

デバッグ・ツール を使用すれば、リモート・デバッグ・モードで (例えば、Rational Developer for System z のデバッグ・パースペクティブ を使用)、あるいは VTAM® 端末を使用してフルスクリーン・モード (MFI) で、z/OS UNIX プログラムをデバッグできます。

関連タスク 319 ページの『第 15 章 UNIX のもとでのコンパイル』 332 ページの『UNIX のもとでのオブジェクト指向アプリケーションの実行』 『UNIX 環境での実行』 490 ページの『環境変数の設定およびアクセス』 493 ページの『UNIX/POSIX API の呼び出し』 495 ページの『メインプログラム・パラメーターへのアクセス』 *言語環境プログラム・プログラミング・ガイド*

関連参照
383 ページの『RENT』

UNIX 環境での実行

COBOL プログラムは、z/OS UNIX シェルまたはシェル以外から、z/OS UNIX 実行環境のいずれかで実行できます。

- z/OS UNIX シェルからは、OMVS シェル (OMVS) または ISPF シェル (ISHELL) でプログラムを実行できます。

  シェル・プロンプトでプログラム名を入力してください。プログラムは、現行ディレクトリーまたは検索パスに入っている必要があります。

  プログラムを開始する前に環境変数 _CEE_RUNOPTS を設定することによってのみ、ランタイム・オプションを指定することができます。

  カタログ式 MVS データ・セットに常駐するプログラムは、tso ユーティリティーを使用して、シェルから実行することができます。以下に例を示します。

  ```
  tso "call 'my.loadlib(myprog)'"
  ```

  ISPF シェルは、stdout および stderr のみを、端末ではなく HFS ファイルに送信します。
- シェル以外からは、TSO/E またはバッチでプログラムを実行できます。

  HFS ファイルに常駐する z/OS UNIX COBOL プログラムを TSO/E プロンプトから呼び出すには、BPXBATCH ユーティリティーを使用するか、REXX exec で spawn() syscall を使用してください。

  HFS ファイルに常駐する z/OS UNIX COBOL プログラムを JCL EXEC ステートメントを使用して呼び出すには、BPXBATCH ユーティリティーを使用してください。

関連タスク
332 ページの『UNIX のもとでのオブジェクト指向アプリケーションの実行』
『環境変数の設定およびアクセス』
493 ページの『UNIX/POSIX API の呼び出し』
495 ページの『メインプログラム・パラメーターへのアクセス』
185 ページの『QSAM ファイルの定義および割り振り』
233 ページの『行順次ファイルの定義および割り振り』
223 ページの『VSAM ファイルの割り振り』
41 ページの『画面上またはファイル内での値の表示 (DISPLAY)』
言語環境プログラム・プログラミング・ガイド (z/OS UNIX C/C++ アプリケーション・プログラムの実行: POSIX 対応プログラムの実行)

関連参照
392 ページの『TEST』
UNIX システム・サービス・ユーザーズ・ガイド (BPXBATCH ユーティリティー)
言語環境プログラム・プログラミング・リファレンス

環境変数の設定およびアクセス

z/OS UNIX COBOL プログラムの環境変数は、export および set を使用してシェルから、またはプログラムから、設定できます。

プログラムの実行を開始する前にシェルから環境変数を設定およびリセットするのが一般的な手順ですが、実行中にプログラムから環境変数を設定、リセット、およびアクセスすることができます。

BPXBATCH を使用してプログラムを実行している場合は、STDENV DD ステートメントを使用して環境変数を設定することができます。

環境変数を設定されていない状態にリセットするには、z/OS UNIX シェル・コマンド unset を使用してください。COBOL プログラムから環境変数をリセットするには、setenv() 関数を呼び出してください。

すべての環境変数の値を調べるには、export コマンドをパラメーターを付けずに使用します。COBOL プログラムから環境変数の値にアクセスするには、getenv() 関数を呼び出してください。

492 ページの『例: 環境変数の設定とアクセス』

関連タスク
489 ページの『UNIX 環境での実行』
『実行に影響を与える環境変数の設定』
495 ページの『メインプログラム・パラメーターへのアクセス』
332 ページの『UNIX のもとでのオブジェクト指向アプリケーションの実行』
319 ページの『UNIX のもとでの環境変数の設定』

関連参照
『ランタイム環境変数』
言語環境プログラム・プログラミング・リファレンス
MVS プログラム管理: ユーザーズ・ガイドおよび解説書

実行に影響を与える環境変数の設定

シェルから UNIX COBOL プログラムの環境変数を設定するには、export または set コマンドを使用してください。プログラム内から環境変数を設定するには、POSIX 関数 setenv() または putenv() を呼び出します。

例えば、環境変数 MYFILE を設定するには、次のように指定します。

```
export MYFILE=/usr/mystuff/notes.txt
```

492 ページの『例: 環境変数の設定とアクセス』

関連タスク
493 ページの『UNIX/POSIX API の呼び出し』
319 ページの『UNIX のもとでの環境変数の設定』

関連参照
『ランタイム環境変数』

ランタイム環境変数

COBOL プログラムでは、いくつかのランタイム変数に関心があります。

それらのランタイム環境変数を次に示します。

_CEE_ENVFILE
環境変数を読み取るファイルの指定。

_CEE_RUNOPTS
ランタイム・オプションを指定します。

CLASSPATH
オブジェクト指向アプリケーションに必要な Java .class ファイルのディレクトリー・パスを指定します。

COBJVMINITOPTIONS
COBOL が JVM を初期化するときに使用される Java 仮想マシン (JVM) オプションを指定します。

_IGZ_SYSOUT
DISPLAY 出力の宛先を指定します。使用できる値は stdout と stderr だけです。

LIBPATH
ダイナミック・リンク・ライブラリーのディレクトリー・パスを指定します。

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。

STEPLIB
LNKLST に含まれていないプログラムの場所を指定します。

関連タスク
42 ページの『システム論理出力装置上でのデータの表示』

関連参照
XL C/C++ プログラミング・ガイド (_CEE_ENVFILE)
言語環境プログラム・プログラミング・リファレンス

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.
```

UNIX/POSIX API の呼び出し

標準の UNIX/POSIX 関数は、CALL *literal* ステートメントを使用して、z/OS UNIX プログラムおよび従来の z/OS COBOL プログラムから呼び出すことができます。これらの関数は、言語環境プログラム の一部です。

これらは 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をコンパイルするときは、コンパイラー・オプション NODYNAM および PGMNAME(LONGMIXED) を使用する必要があります。

fork()、exec()、および spawn() 関数は、COBOL プログラムから、または COBOL プログラムと同じプロセスに含まれている非 COBOL プログラムから呼び出すことができます。ただし、以下の制約事項に留意しなければなりません。

- fork を発行したときに開かれていた COBOL の順次ファイル、索引付きファイル、または相対ファイルに、fork されたプロセスからアクセスすることはできません。このようなアクセス (CLOSE、READ、WRITE、REWRITE、DELETE、または START) を試みた場合、ファイル状況コード 92 が返されます。 fork を出したときにオープンされた行順次ファイルにはアクセスできます。
- 以下の状況の中から当てはまるものがあるプロセスでは、fork() 関数を使用できません。
  - COBOL の SORT または MERGE が実行されている。
  - 宣言が実行されている。
  - プロセスに、複数の 言語環境プログラム・エンクレーブ (COBOL 実行単位) が含まれている。
  - プロセスで、いずれかの COBOL 再使用可能環境インターフェースが使用されている。
  - プロセスが OS/VS COBOL または VS COBOL II プログラムを実行したことがある。
- 1 つの例外を除いて、DD 割り振りは親プロセスから子プロセスへ継承されません。例外はローカル spawn で、この場合は、親プロセスと同じアドレス・スペー

ス内に子プロセスが作成されます。ローカル spawn は、spawn() 関数を呼び出す前に、環境変数 _BPX_ SHAREAS=YES を設定することによって要求します。

exec() 関数と spawn() 関数は、新規 UNIX プロセス内で新規の 言語環境プログラム・エンクレーブを開始します。したがって、exec() または spawn() 関数のターゲット・プログラムはメインプログラムであり、プロセス内のすべての COBOL プログラムは、すべてのファイルをクローズした初期状態で開始します。

SIGYSAMP データ・セットには、一部の POSIX ルーチンを呼び出すサンプル・コードが用意されています。

表 63. **POSIX 関数呼び出しを使用したサンプル**

目的	サンプル	使用される関数
一部のファイルおよびディレクトリー・ルーチンの使用法を示す	IGYTFL1	• getcwd() • mkdir() • rmdir() • access()
iconv ルーチンを使用してデータを変換する方法を示す	IGYTCNV	• iconv_open() • iconv() • iconv_close()
exec() ルーチンを使用して、他のプロセス関連ルーチンと一緒に新規プログラムを実行する方法を示す	IGYTEXC、IGYTEXC1	• fork() • getpid() • getppid() • execl() • perror() • wait()
errno 値の入手方法を示す	IGYTERNO、IGYTGETE	• perror() • fopen()
プロセス間通信メッセージ・ルーチンの使用法を示す	IGYTMSQ、IGYTMSQ2	• ftok() • msgget() • msgsnd() • perror() • fopen() • fclose() • msgrcv() • msgctl() • perror()

関連タスク

489 ページの『UNIX 環境での実行』
490 ページの『環境変数の設定およびアクセス』
495 ページの『メインプログラム・パラメーターへのアクセス』
言語環境プログラム・プログラミング・ガイド

関連参照
XL C/C++ ランタイム・ライブラリー・リファレンス
UNIX システム・サービス・プログラミング: アセンブラー呼び出し可能サービス 解説書

メインプログラム・パラメーターへのアクセス

z/OS UNIX シェル・コマンド行から、あるいは exec() または spawn() 関数を使用して COBOL プログラムを実行する場合、パラメーター・リストは参照によって渡される 3 つのパラメーターから構成されます。標準の COBOL コーディングを使用してこれらのパラメーターにアクセスすることができます。

引数カウント
2 番目および 3 番目のパラメーターで渡されるそれぞれの配列のエレメントの数が含まれている 2 進数のフルワード整数。

引数の長さのリスト
ポインターの配列。配列内の *n* 番目の記入項目は、引数リスト内の *n* 番目の記入項目の長さを含むフルワードの 2 進整数のアドレスです。

引数リスト
ポインターの配列。配列内の *n* 番目の記入項目は、spawn() 関数、exec() 関数、またはコマンド呼び出しで引数として渡される *n* 番目の文字ストリングのアドレスです。各文字ストリングは、ヌル終了文字ストリングです。

この配列は空になることはありません。最初の引数は、開始するプロセスと関連付けられたファイルの名前を表す文字ストリングです。

『例: メインプログラム・パラメーターへのアクセス』

関連タスク
489 ページの『UNIX 環境での実行』
490 ページの『環境変数の設定およびアクセス』
493 ページの『UNIX/POSIX API の呼び出し』

例: メインプログラム・パラメーターへのアクセス

次の例は、参照によって渡される 3 つのパラメーターを示しています。

```
 Identification division.
 Program-id. "EXECED".
*****************************************************************
* This sample program displays arguments received via exec()   *
* function of z/OS UNIX                                        *
*****************************************************************
 Data division.
 Working-storage section.
 01 curr-arg-count pic 9(9) binary value zero.
 Linkage section.
 01 arg-count pic 9(9) binary.                      (1)
 01 arg-length-list.                                (2)
     05 arg-length-addr pointer occurs 1 to 99999
           depending on curr-arg-count.
 01 arg-list.                                       (3)
     05 arg-addr pointer occurs 1 to 99999
           depending on curr-arg-count.
 01 arg-length pic 9(9) binary.
```

```
 01 arg pic X(65536).
 Procedure division using arg-count arg-length-list arg-list.
*****************************************************************
* Display number of arguments received                          *
*****************************************************************
     Display "Number of arguments received: " arg-count
*****************************************************************
* Display each argument passed to this program                  *
*****************************************************************
     Perform arg-count times
       Add 1 to curr-arg-count
* *******************************************************
* * Set address of arg-length to address of current     *
* * argument length and display                         *
* *******************************************************
       Set Address of arg-length
         to arg-length-addr(curr-arg-count)
       Display
         "Length of Arg " curr-arg-count " = " arg-length
* *******************************************************
* * Set address of arg to address of current argument   *
* * and display                                         *
* *******************************************************
       Set Address of arg to arg-addr(curr-arg-count)
       Display "Arg " curr-arg-count " = " arg (1:arg-length)
     End-Perform
     Display "Display of arguments complete."
     Goback.
```

(1) このカウントには、2 番目および 3 番目のパラメーターで渡される配列のエレメントの数が含まれています。

(2) この配列には、引数リスト内の *n* 番目の記入項目の長さへのポインターが含まれています。

(3) この配列には、spawn() 関数、exec() 関数、またはコマンド呼び出しで引数として渡される *n* 番目の文字ストリングへのポインターが含まれています。

第 4 部 複雑なアプリケーションの構造化

第 24 章 サブプログラムの使用

多くのアプリケーションは、一緒にリンクされた別々にコンパイルされた複数のプログラムから構成されます。*実行単位* (言語環境プログラム ・エンクレーブ と同義の COBOL 用語) には 1 つ以上のオブジェクト・プログラムが含まれており、他の 言語環境プログラム のメンバー言語で書かれたオブジェクト・プログラムが含まれることもあります。

言語環境プログラムには言語間サポートが用意されており、これを使用すると、Enterprise COBOL プログラムは、言語環境プログラム の要件に適合するプログラムとの間で呼び出しを行うことができます。

名前の接頭部に関する注意事項: IBM 製品が使用している接頭部で始まるプログラム名は使用しないでください。名前が以下のいずれかの接頭部のプログラムを使用した場合、 `CALL` ステートメントは意図されたプログラムではなく、IBM ライブラリーまたはコンパイラー・ルーチンを呼び出してしまう可能性があります。

- AFB
- AFH
- CBC
- CEE
- EDC
- IBM
- IFY
- IGY
- IGZ
- ILB

関連概念
500 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
500 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』
502 ページの『別のプログラムへの制御権移動』
515 ページの『再帰呼び出しの実行』
516 ページの『オブジェクト指向プログラムとの間での呼び出し』
516 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』
518 ページの『プログラムを再入可能にする』
556 ページの『マルチスレッド化による COBOL 制限の処理』
Language Environment Writing ILC Applications

関連参照
言語環境プログラム・プログラミング・ガイド (規則の登録)

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
『メインプログラムまたはサブプログラムの終了と再入』
502 ページの『別のプログラムへの制御権移動』
515 ページの『再帰呼び出しの実行』

関連参照
言語環境プログラム・プログラミング・ガイド

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。

メインプログラムまたはサブプログラムで 3 つの終了ステートメントのいずれを使用することもできますが、次の表に示すように、その影響がそれぞれ異なります。

表 64. **終了ステートメントの影響**

終了ステートメント	メインプログラム	サブプログラム
EXIT PROGRAM	アクションはとられません。	実行単位を終了せずに呼び出し側プログラムに戻ります。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。 スレッド化された環境では、そのプログラムがスレッド内の最初の (最も古い) プログラムでない限り、スレッドは終了しません。
STOP RUN	呼び出し側プログラムに戻ります。[1] (オペレーティング・システムに戻る可能性があり、アプリケーションは終了します。) STOP RUN は実行単位を終了して、実行単位内で動的に呼び出されたプログラムと、それらとリンク・エディットされたすべてのプログラムを削除します。(メインプログラムは削除しません。) スレッド化された環境では、言語環境プログラム・のエンクレーブ全体 (そのエンクレーブ内で実行中のすべてのスレッドを含む) が終了します。	メインプログラムを呼び出したプログラムに直接戻ります。[1] (オペレーティング・システムに戻る可能性があり、アプリケーションは終了します。) STOP RUN は実行単位を終了して、実行単位内で動的に呼び出されたプログラムと、それらとリンク・エディットされたすべてのプログラムを削除します。(メインプログラムは削除しません。) スレッド化された環境では、言語環境プログラム・のエンクレーブ全体 (そのエンクレーブ内で実行中のすべてのスレッドを含む) が終了します。
GOBACK	呼び出し側プログラムに戻ります。[1] (オペレーティング・システムに戻る可能性があり、アプリケーションは終了します。) GOBACK は実行単位を終了して、実行単位内で動的に呼び出されたプログラムと、それらとリンク・エディットされたすべてのプログラムを削除します。(メインプログラムは削除しません。) スレッド化された環境では、スレッドが終了します。[2]	呼び出し側プログラムに戻ります。 スレッド化された環境では、そのプログラムがスレッド内の最初のプログラムであると、スレッドが終了します。[2]

1. メインプログラムが、言語環境プログラムのリンケージ規約に従っていない別の言語で書かれたプログラムによって呼び出された場合は、この呼び出し側プログラムに戻ります。
2. スレッドがエンクレーブ内の最初の実行スレッドである場合は、エンクレーブが終了します。

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。サブプログラムが実行単位の中で次に呼び出されると、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。
- INITIAL 属性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

17 ページの『WORKING-STORAGE と LOCAL-STORAGE の比較』

言語環境プログラム・プログラミング・ガイド (言語環境プログラム終了: スレッド終了)

関連タスク

511 ページの『ネストされた COBOL プログラムの呼び出し』
515 ページの『再帰呼び出しの実行』

別のプログラムへの制御権移動

別のプログラムへの制御権移動を行うのに、幾つかの異なる方法 (静的呼び出し、動的呼び出し、ネストされたプログラムの呼び出し、およびダイナミック・リンク・ライブラリー (DLL) の呼び出し) を使用できます。

すべての環境 (CICS を含む) において、Enterprise COBOL プログラム相互間での呼び出しだけでなく、Enterprise COBOL プログラムとその他の古いバージョンのコンパイラーでコンパイルされたプログラムとの間で、静的呼び出しおよび動的呼び出しを行うことができます。

OS/VS COBOL を Enterprise COBOL と併用する場合、非 CICS と CICS との間では、サポートに違いがあります。

非 CICS 環境の場合

Enterprise COBOL プログラムとその他の COBOL プログラムとの間で、静的呼び出しおよび動的呼び出しを行うことができます。

例外: UNIX 環境では、VS COBOL II または OS/VS COBOL プログラムを呼び出せません。

CICS 環境の場合

CICS 環境では、OS/VS COBOL プログラムを呼び出せません。 OS/VS COBOL プログラムとその他の COBOL プログラムとの間で制御権を移動するには、EXEC CICS LINK を使用する必要があります。

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て、アプリケーションを作成することができます。また、データ項目を不注意で変更しないようにするためには、PERFORM プロシージャーの代わりに、ネストさ

れたプログラムを使用することができます。ネストされたプログラムは、 CALL *literal* または CALL *identifier* ステートメントを使用して呼び出します。

ダイナミック・リンク・ライブラリー (DLL) の呼び出しは、COBOL の動的 CALL の代替方法として使用することができ、オブジェクト指向 COBOL アプリケーション、UNIX プログラム、および C/C++ と相互運用するアプリケーションでの使用に適しています。

z/OS では、2 つのロード・モジュールをリンクすると、論理的には単一のプログラムになり、そのプログラムには 1 次入り口点と代替入り口点があり、その入り口点にはそれぞれ独自の名前があります。サブプログラムを動的に呼び出すのに使われる個々の名前は、システムに認識されていなければなりません。このような個々の名前は、そのサブプログラムを含むロード・モジュールの NAME または ALIAS として、リンケージ・エディターまたはバインダーの制御ステートメントで指定する必要があります。

関連概念 507 ページの『AMODE 切り替え』 509 ページの『静的呼び出しと動的呼び出しのパフォーマンスについての考慮事項』 512 ページの『ネストされたプログラム』

関連タスク
『静的呼び出しの作成』
504 ページの『動的呼び出しの作成』
509 ページの『静的呼び出しと動的呼び出しの両方の作成』
511 ページの『ネストされた COBOL プログラムの呼び出し』

静的呼び出しの作成

NODYNAM および NODLL コンパイラー・オプションを使用してコンパイルされたプログラム内で CALL *literal* ステートメントを使用すると、静的呼び出しが行われます。これらのオプションが使用されていると、CALL *literal* の呼び出しはすべて、静的呼び出しとして処理されます。

静的呼び出しステートメントを使用すると、COBOL プログラムと呼び出されるプログラムはすべて同一のロード・モジュールの一部になります。呼び出されるプログラムに制御権が移動すると、そのプログラムは既にストレージにあるので、そのプログラムに対する分岐が行われます。それ以後 CALL ステートメントを実行すると、呼び出されるプログラムは最後に使われた状態で使えます。ただし、呼び出されるプログラムに INITIAL 属性がある場合は除きます。その場合、呼び出し先プログラムおよびその中に直接的または間接的に含まれているそれぞれのプログラムは、実行単位内で呼び出し先プログラムが呼び出されるたびに、その初期状態にされます。

代替入り口点を指定すると、静的 CALL ステートメントはいずれかの代替入り口点を使用して、呼び出されるサブプログラムに入ることができます。

510 ページの『例: 静的および動的 CALL ステートメント』

関連概念 509 ページの『静的呼び出しと動的呼び出しのパフォーマンスについての考慮事項』

関連タスク
『動的呼び出しの作成』
509 ページの『静的呼び出しと動的呼び出しの両方の作成』
516 ページの『オブジェクト指向プログラムとの間での呼び出し』

関連参照
359 ページの『DLL』
361 ページの『DYNAM』
CALL ステートメント (*Enterprise COBOL 言語解説書*)

動的呼び出しの作成

DYNAM および NODLL コンパイラー・オプションを使用してコンパイルしたプログラムで CALL *literal* ステートメントを使用した場合、または NODLL コンパイラー・オプションを使用してコンパイルしたプログラムで CALL *identifier* ステートメントを使用した場合には、動的呼び出しが発生します。

これらの形式の CALL ステートメントでは、呼び出される COBOL サブプログラムは、メインプログラムとはリンク・エディットされません。代わりに、それは独立したロード・モジュールにリンク・エディットされ、実行時にロードされます。ロードされるのは必要とされたとき (すなわち、呼び出されたとき) のみです。PROGRAM-ID 段落または ENTRY ステートメント内のプログラム名は、プログラムを含んでいるロード・モジュールの、対応するロード・モジュール名またはロード・モジュール別名と同一でなければなりません。

動的 CALL ステートメントで呼び出される各サブプログラムは、別のロード・モジュール (システム・リンク・ライブラリーまたはユーザー指定の専用ライブラリーのメンバーである) の一部である場合があります。その場合、サブプログラムは MVS ロード・ライブラリーに置かれていなければならず、階層ファイル・システムに常駐させることはできません。動的 CALL ステートメントが、ストレージに常駐していないサブプログラムを呼び出すと、そのサブプログラムは、2 次ストレージから、メインプログラムを含む領域または区画にロードされ、そのサブプログラムへの分岐が実行されます。

実行単位内でサブプログラムへの最初の動的呼び出しが行われると、そのサブプログラムの新規コピーが獲得されます。それ以後この同じサブプログラムを (元の呼び出し側または同じ実行単位内の他のサブプログラムによって) 呼び出すと、サブプログラムの同一コピーに最後に使われた状態で分岐されます (このサブプログラムが INITIAL 属性を処理する場合を除きます)。したがって、次の項目のどちらかを再初期化する必要があります。

- 更新済みの GO TO ステートメント
- データ項目

同じ COBOL プログラムを別の実行単位で呼び出す場合、それぞれの実行単位ごとに WORKING-STORAGE の別個のコピーが割り振られます。

制約事項: 次のものを動的呼び出しすることはできません。

- COBOL DLL プログラム

- PGMNAME(LONGMIXED) オプションを指定してコンパイルされた COBOL プログラム (プログラム名が 8 バイト以下ですべて大文字の場合は可能)。
- PGMNAME(LONGUPPER) オプションを指定してコンパイルされた COBOL プログラム (プログラム名が 8 バイト以下の場合は可能)。
- 同じ COBOL プログラム内の複数の入り口点 (その間で CANCEL ステートメントが実行された場合は可能)

510 ページの『例: 静的および動的 CALL ステートメント』

関連概念 『サブプログラムに関連した動的呼び出しの使用時期』 509 ページの『静的呼び出しと動的呼び出しのパフォーマンスについての考慮事項』

関連タスク
『サブプログラムの取り消し』
503 ページの『静的呼び出しの作成』
509 ページの『静的呼び出しと動的呼び出しの両方の作成』

関連参照
359 ページの『DLL』
361 ページの『DYNAM』
ENTRY ステートメント (*Enterprise COBOL 言語解説書*)
CALL ステートメント (*Enterprise COBOL 言語解説書*)
言語環境プログラム・プログラミング・リファレンス

サブプログラムの取り消し

サブプログラムに対して CANCEL ステートメントを発行すると、そのサブプログラムが占有していたストレージが解放されます。そのサブプログラムの後続の呼び出しは、最初の呼び出しと同じように機能します。当初の呼び出し元以外のプログラムから、サブプログラムの取り消しを行うことができます。

呼び出されるサブプログラムに複数の入り口点がある場合は、その間で CANCEL ステートメントが実行されない限り、別個の入り口点をそのサブプログラムへの動的 CALL ステートメントに指定することはできません。

含まれているプログラムが動的に呼び出され、それに対して CANCEL ステートメントが処理されると、そのプログラムは最初に使われた状態になります。しかし、そのプログラムは初期呼び出しでロードされず、プログラムが取り消されてもストレージは解放されません。

510 ページの『例: 静的および動的 CALL ステートメント』

関連概念 509 ページの『静的呼び出しと動的呼び出しのパフォーマンスについての考慮事項』

サブプログラムに関連した動的呼び出しの使用時期

サブプログラムに関連して動的呼び出しを使用するかどうかの判断は、ロード・モジュールの場所、サブプログラムの呼び出しの頻度、サブプログラムのサイズ、保

守の容易性、未使用状態のサブプログラムを呼び出す必要があるかどうか、AMODE 切り替えが必要かどうか、およびプログラム名がいつ認識されるか、といった要因によって左右されます。

動的に呼び出したいロード・モジュールは、階層ファイル・システムではなく MVS ロード・ライブラリーに入っている必要があります。

ほんの少しの条件でサブプログラムが呼び出される場合には、動的呼び出しを使用して、必要時にのみサブプログラムを取り込むことができます。

サブプログラムが非常に大きいか、あるいはその数が多い場合、静的呼び出しを使用すると、あまりにも多くの主記憶域を必要とする場合があります。使用する合計ストレージをより少なくするためには、2 つのサブプログラムを両方とも静的に呼び出すよりも、あるサブプログラムを呼び出してから取り消し、次に別のサブプログラムを呼び出してそれを取り消す必要があります。

保守を容易にすることに関心がある場合、動的呼び出しは役立ちます。動的に呼び出されたサブプログラムが変更されても、アプリケーションを再度リンク・エディットする必要はありません。

INITIAL 属性を使用してサブプログラムが呼び出されるときに必ず未使用状態にすることができない場合は、動的 CALL と CANCEL ステートメントの組み合わせを使用して未使用状態を設定できます。最初に COBOL プログラムが呼び出したサブプログラムを取り消した場合、次に呼び出したとき、サブプログラムはその未使用の状態に再初期化されます。

CANCEL ステートメントを使用して、非 COBOL プログラムによって動的にロードして分岐したサブプログラムを明示的に取り消しても、そのサブプログラムのストレージを解放したり、そのサブプログラムを削除するための処置は取られません。

31 ビットのアドレッシング・モードで実行する必要がある Enterprise COBOL と同じ実行単位内に OS/VS COBOL プログラムまたはその他の AMODE 24 プログラムがある場合。 COBOL 動的呼び出し処理には、 AMODE 31 プログラムを呼び出す AMODE 24 プログラムの AMODE 切り替え (逆の場合も同じ) が含まれます。この暗黙の AMODE 切り替えを実行するためには、言語環境プログラムのランタイム・オプション ALL31(OFF) を使用する必要があります。 ALL31(ON) が設定されると、AMODE 切り替えは実行されません。

AMODE 切り替えが実行されると、制御権は呼び出し側から言語環境プログラムのライブラリー・ルーチンに渡されます。切り替えの実行後、制御権は呼び出されたプログラムに移動します。ライブラリー・ルーチンの保管域は、呼び出し側プログラムの保管域と呼び出されたプログラムの保管域の間に位置付けられます。

呼び出されるプログラム名が実行時まで分からない場合、CALL *identifier* の形式を使用してください。ここで、*identifier* は、実行時に呼び出し先プログラムの名前が入れられるデータ項目です。例えば、プログラム内での条件付き処理に応じて呼び出されるプログラムが異なるような場合には、CALL *identifier* を使用できます。NODYNAM コンパイラー・オプションを使用する場合でも、CALL *identifier* は常に動的です。

510 ページの『例: 静的および動的 CALL ステートメント』

関連概念 『AMODE 切り替え』 509 ページの『静的呼び出しと動的呼び出しのパフォーマンスについての考慮事項』

関連タスク
504 ページの『動的呼び出しの作成』

関連参照
361 ページの『DYNAM』
CALL ステートメント (*Enterprise COBOL 言語解説書*)
言語環境プログラム・プログラミング・リファレンス

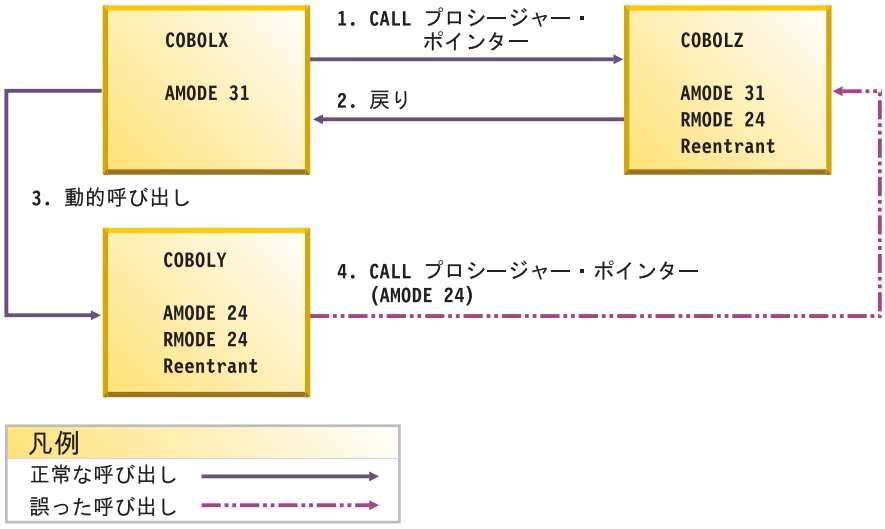
AMODE 切り替え

複数の COBOL サブプログラムを持つアプリケーションを使用する場合、その COBOL サブプログラムの一部を AMODE 31 に設定し、その他を AMODE 24 に設定することができます。

アプリケーションが COBOL プログラムでのみ構成され、静的呼び出しと動的呼び出しだけを使用する場合、各 COBOL サブプログラムは、常に適切な AMODE になります。例えば、AMODE 31 COBOL プログラムから AMODE 24 COBOL プログラムへの動的呼び出しを行っている場合、AMODE は自動的に切り替えられます。

ただし、プロシージャー・ポインター、関数ポインター、または COBOL サブプログラムを呼び出す他の言語を使用する場合は、COBOL プログラムがエンクレーブで複数回呼び出されたときに、呼び出されるたびに同じ AMODE になることを確認する必要があります。この場合、AMODE は自動的に切り替えられません。

次のシナリオは、プロシージャー・ポインターが COBOL サブプログラムの呼び出しに使用される場合に、AMODE の問題が生じる可能性を示しています。このシナリオは、COBOL プログラム COBOLY が呼び出されるときに毎回同じ AMODE にならないため、サポートされません。

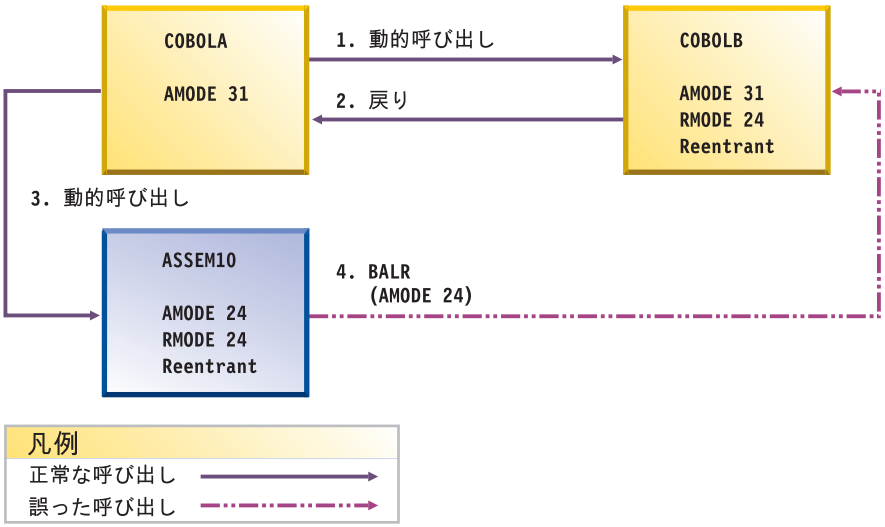


1. COBOLX は AMODE 31 です。これは SET ステートメントを使用して、プロシージャー・ポインターを COBOLZ に設定します。 COBOLZ は再入可能ロード・

モジュールであり、AMODE 31 と RMODE 24 です。COBOLX は、プロシージャー・ポインターを使用して COBOLZ を呼び出します。 COBOLZ は AMODE 31 になります。

2. COBOLZ は COBOLX に戻ります。
3. COBOLX は、COBOLY を動的に呼び出して、プロシージャー・ポインターを COBOLZ に渡します。 COBOLY は再入可能ロード・モジュールであり、AMODE 24 と RMODE 24 です。 COBOLY は AMODE 24 になります。
4. COBOLY はプロシージャー・ポインターを使用して COBOLZ を呼び出します。この呼び出しによって COBOLZ は AMODE 24 になりますが、これは COBOLZ が最初に呼び出されたときの AMODE と同じではありません。

次のシナリオは、COBOL とアセンブラー言語の混合を使用しています。このシナリオは、COBOL プログラム COBOLB が呼び出されるときに毎回同じ AMODE にならないため、サポートされません。



1. COBOLA は AMODE 31 です。COBOLA は COBOLB を動的に呼び出します。COBOLB は再入可能ロード・モジュールであり、AMODE 31 と RMODE 24 です。COBOLB は AMODE 31 になります。
2. COBOLB は COBOLA に戻ります。
3. COBOLA はアセンブラー言語である ASSEM10 を動的に呼び出します。ASSEM10 は再入可能ロード・モジュールであり、AMODE 24 と RMODE 24 です。 ASSEM10 は AMODE 24 になります。
4. ASSEM10 は COBOLB をロードします。ASSEM10 は COBOLB に対して BALR 命令を実行します。 COBOLB は AMODE 24 になりますが、COBOLB が最初に呼び出されたときの AMODE と同じではありません。

関連概念

45 ページの『ストレージとそのアドレス可能度』
505 ページの『サブプログラムに関連した動的呼び出しの使用時期』

関連タスク

504 ページの『動的呼び出しの作成』

関連参照
言語環境プログラム・プログラミング・リファレンス (ALL31)

静的呼び出しと動的呼び出しのパフォーマンスについての考慮事項

静的に呼び出されるプログラムは、呼び出し側プログラムと同じロード・モジュールにリンク・エディットされるので、静的呼び出しは動的呼び出しより高速です。動的呼び出しのサービスがアプリケーションで必要ない場合は、静的呼び出しが推奨されるメソッドです。

静的に呼び出されたプログラムは CANCEL を使用して削除できないので、静的呼び出しを使用すると、より多くの主記憶域を使用することになる場合があります。ストレージを重要視する場合には、動的呼び出しの使用を考えてください。呼び出しで使用するストレージは、次の要因によって異なります。

- サブプログラムが 2、3 回しか呼び出されないかどうか。サブプログラムが呼び出されるかどうかに関係なく、静的に呼び出されるプログラムはストレージにロードされます。動的に呼び出されるプログラムは、サブプログラムが呼び出される場合に限りロードされます。
- 後で CANCEL ステートメントを使用して、動的に呼び出されたサブプログラムを取り消すかどうか。

  静的に呼び出されたプログラムは削除できませんが、動的に呼び出されたプログラムは削除することができます。動的呼び出しを行った後で CANCEL ステートメントを使用して、動的に呼び出されたプログラムを (呼び出し後ではなく、アプリケーションで不要になった後で) 削除すると、必要なストレージは静的呼び出しを使用する場合よりも少なくて済みます。

関連概念
505 ページの『サブプログラムに関連した動的呼び出しの使用時期』

関連タスク
503 ページの『静的呼び出しの作成』
504 ページの『動的呼び出しの作成』

静的呼び出しと動的呼び出しの両方の作成

NODYNAM コンパイラー・オプションを使用してコンパイルされたプログラムでは、静的と動的の両方の CALL ステートメントを同一のプログラムに使用することができます。

この場合、CALL *literal* ステートメントを使用すると、呼び出されるサブプログラムはメインプログラムとリンク・エディットされ、1 つのロード・モジュールになります。 CALL *identifier* ステートメントを実行すると、別個のロード・モジュールが動的に呼び出されます。

1 つのプログラム内から同一のサブプログラムに対して動的 CALL ステートメントと静的 CALL ステートメントが出されると、そのサブプログラムの 2 つ目のコピーがストレージにロードされます。この構造では、サブプログラムが最後に使われた状態のままになるとは限らないので、結果が予測できなくなることがあります。

関連参照
361 ページの『DYNAM』

例: 静的および動的 CALL ステートメント

この例は、静的呼び出しおよび動的呼び出しのコーディング方法を示しています。

この例は次の 3 つの部分から構成されます。

- 静的呼び出しを使用してサブプログラムを呼び出すコード
- 動的呼び出しを使用して同じサブプログラムを呼び出すコード
- この 2 種類の呼び出しによって呼び出されるサブプログラム

次の例は、静的呼び出しをコーディングする方法を示しています。

```
PROCESS NODYNAM NODLL
IDENTIFICATION DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  RECORD-2                  PIC X.              (6)
01  RECORD-1.                                     (2)
    05  PAY                   PICTURE S9(5)V99.
    05  HOURLY-RATE           PICTURE S9V99.
    05  HOURS                 PICTURE S99V9.
. . .
PROCEDURE DIVISION.
    CALL "SUBPROG" USING RECORD-1.                (1)
    CALL "PAYMASTR" USING RECORD-1 RECORD-2.      (5)
    STOP RUN.
```

次の例は、動的呼び出しをコーディングする方法を示しています。

```
DATA DIVISION.
WORKING-STORAGE SECTION.
77  PGM-NAME                  PICTURE X(8).
01  RECORD-2                  PIC x.              (6)
01  RECORD-1.                                     (2)
    05  PAY                   PICTURE S9(5)V99.
    05  HOURLY-RATE           PICTURE S9V99.
    05  HOURS                 PICTURE S99V9.
. . .
PROCEDURE DIVISION.
. . .
    MOVE "SUBPROG" TO PGM-NAME.
    CALL PGM-NAME USING RECORD-1.                 (1)
    CANCEL PGM-NAME.
    MOVE "PAYMASTR" TO PGM-NAME.                  (4)
    CALL PGM-NAME USING RECORD-1 RECORD-2.        (5)
    STOP RUN.
```

次の例は、呼び出されるサブプログラムです。これは、前の 2 つの呼び出し側プログラムのそれぞれによって呼び出されます。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SUBPROG.
DATA DIVISION.
LINKAGE SECTION.
01  PAYREC.                                       (2)
    10 PAY            PICTURE S9(5)V99.
    10 HOURLY-RATE    PICTURE S9V99.
    10 HOURS          PICTURE S99V9.
77  PAY-CODE          PICTURE 9.                  (6)
PROCEDURE DIVISION USING PAYREC.                  (1)
```

```
. . .
    EXIT PROGRAM.                                  (3)
    ENTRY "PAYMASTR" USING PAYREC PAY-CODE.        (5)
    . . .
    GOBACK. (7)
```

(1) 処理は呼び出し側プログラムで開始されます。最初の CALL ステートメントが実行されると、SUBPROG (呼び出されるプログラム) の PROCEDURE DIVISION の先頭のステートメントに制御権が移動します。

各 CALL ステートメントでは、最初の USING オプションのオペランドが RECORD-1 として識別されます。

(2) SUBPROG が制御権を受け取ると、RECORD-1 内の値が SUBPROG で使用可能になります。ただし、SUBPROG では、この値は PAYREC として参照されます。

PAYREC および PAY-CODE 内の PICTURE 文字ストリングの文字数は、記述は同一ではありませんが、RECORD-1 および RECORD-2 の文字数と同じです。

(3) SUBPROG 中の処理が EXIT PROGRAM ステートメントに達すると、呼び出し側プログラムに制御権が戻されます。このプログラムでの処理は、2 番目の CALL ステートメントが発行されるまで継続されます。

(4) 動的に呼び出されるプログラムの例では、2 番目の CALL ステートメントは SUBPROG 内の別の入り口点を参照するため、2 番目の CALL ステートメントの前に CANCEL ステートメントが出されます。

(5) 呼び出し側プログラム中の 2 番目の CALL ステートメントによって制御権は再び SUBPROG に移動しますが、今回の処理は SUBPROG 中の ENTRY ステートメントの次のステートメントから開始されます。

(6) RECORD-1 内の値が再び PAYREC で使用可能になります。さらに、SUBPROG は対応する USING オペランド PAY-CODE を使用して、RECORD-2 中の値も使えるようになります。

静的にリンクされたプログラムから 2 度目に制御権が移動すると、SUBPROG は最後に使われた状態で使えるようになります (したがって、最初の実行時に SUBPROG ストレージ内の任意の値が変更されていると、その変更された値は依然として有効です)。しかし、動的にリンクされたプログラムから制御権が移動すると、CANCEL ステートメントが実行されているため、SUBPROG は初期状態で使用可能になります。

(7) 処理が GOBACK ステートメントに達すると、呼び出し側プログラム中の 2 番目の CALL ステートメントの直後のステートメントに制御権が戻されます。

呼び出し先プログラムと 2 つのいずれかの呼び出し側プログラムの特定の実行において、最初の CALL と 2 番目の CALL の間で RECORD-1 内の値が変更された場合は、2 番目の CALL ステートメントの実行時に渡される値は、元の値ではなく、変更された値になります。元の値を使用する場合は、その値を保管しなければなりません。

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。

含まれている 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、含まれている 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) 含まれている 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、含まれている 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際には、以下の指針に従ってください。

- 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 含まれている プログラムの名前を固有にしてください。含まれている 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。含まれている 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。含まれている プログラムが CONFIGURATION SECTION を持つことはできません。
- それぞれの 含まれている 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、含まれている プログラムと収容しているプログラムを終了させます。

ネストされたプログラムを含むプログラムをコンパイルするときは、THREAD オプションを使用できません。

関連概念
『ネストされたプログラム』

関連参照
514 ページの『名前の有効範囲』

ネストされたプログラム

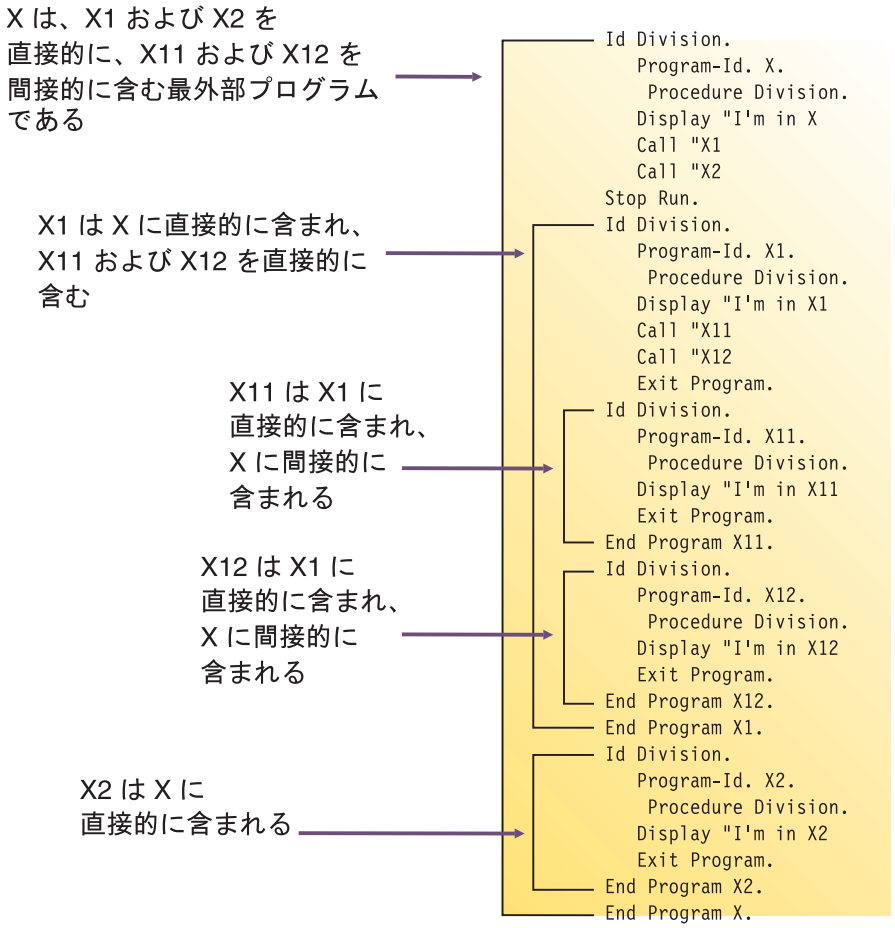
COBOL プログラムには、他の COBOL プログラムをネスト すること、つまり、他の COBOL プログラムを含めることができます。ネストされたプログラム自体にも他のプログラムを含められます。ネストされたプログラムは、プログラムに直接的に含めることも、間接的に含めることもできます。

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。



『例: ネストされたプログラムの構造』

関連タスク

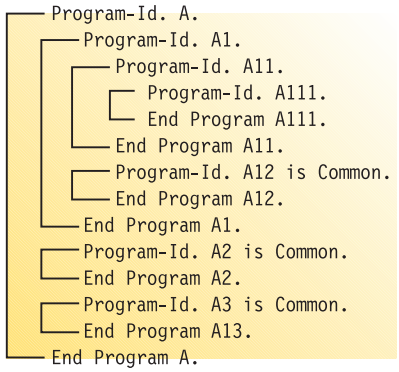
511 ページの『ネストされた COBOL プログラムの呼び出し』

関連参照

514 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、一部の含まれているプログラムが COMMON として識別された、ネストされた構造を示しています。



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。

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、条件が通知されます。この条件を未処理のままにしておくと、実行単位が終了します。

関連タスク
6 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 (*Enterprise COBOL 言語解説書*)

オブジェクト指向プログラムとの間での呼び出し

オブジェクト指向 (OO) プログラムを含むアプリケーションを作成する場合、OO COBOL プログラムは DLL プログラムなので、1 つ以上のダイナミック・リンク・ライブラリー (DLL) に置くことができますただし、各クラス定義は個別の DLL に置かれている必要があります。

COBOL DLL プログラムとの間での呼び出しでは、DLL リンケージを使用するか、または静的呼び出しのいずれかでなければなりません。COBOL DLL プログラムとの間での COBOL 動的呼び出しはサポートされていません。

COBOL の非 DLL プログラムから COBOL DLL プログラムを呼び出す必要がある場合は、DLL リンケージ・メカニズムに従っていることを保証する別の方法を使用することができます。

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・データ項目および関数ポインター・データ項目の設定には、形式 6 の SET ステートメントのみを使用することができます。

*プロシージャー・ポインター*とは、USAGE IS PROCEDURE-POINTER 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。例えば、例外条件が発生したときに、ユーザー作成エラー処理ルーチンに制御権を渡すためには、まずそのルーチンの入り口アドレスを CEEHDLR (条件管理 言語環境プログラム呼び出し可能サービス) に渡して、そのルーチンを登録させなければなりません。
- 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 RETURNING ステートメント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関数ポインターとして使用したりプロシージャー・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)。

SET ステートメントは、コンパイラー・オプション DYNAM|NODYNAM および DLL|NODLL の設定に応じて、プログラムと同じロード・モジュール内の入り口点、個別のロード・モジュール、または DLL からエクスポートされた入り口点を参照するように、プロシージャー・ポインターを設定します。したがって、これらのポインター・データ項目を使用する場合は、以下の点を考慮する必要があります。

- NODYNAM オプションおよび NODLL オプションを指定してプログラムをコンパイルするときに、ポインター項目をリテラル値 (入り口点の実際の名前) に設定する場合、その値は、同じロード・モジュール内の入り口点を参照している必要があります。そうしないと、参照は解決できません。
- NODLL オプションを指定してプログラムをコンパイルするときに、ポインター項目を実行時に入り口点の名前が入る ID に設定する場合、またはポインター項目をリテラルに設定し DYNAM オプションを指定してコンパイルする場合、ポインター項目は (リテラルであっても変数であっても) 別個のロード・モジュール内の入り口点を指している必要があります。入り口点は、1 次入り口点か、リンケージ・エディターまたはバインダーの ALIAS ステートメントで指定した代替入り口点になります。
- NODYNAM オプションおよび DLL オプションを指定してコンパイルするときに、ポインター項目をリテラル値 (入り口点の実際の名前) に設定する場合、その値は、同じロード・モジュール内の入り口点、または DLL モジュールからエクスポートされた入り口点名を参照している必要があります。後者の場合は、プログラム・ロード・モジュールのリンク・エディットに、ターゲット DLL モジュールの DLL サイド・ファイルを含めなければなりません。
- NODYNAM オプションおよび DLL オプションを指定してコンパイルするときに、ポインター項目を ID (実行時に入り口点名が入るデータ項目) に設定する場合、その ID 値は、DLL モジュールからエクスポートされた入り口点名を参照している必要があります。この場合、DLL モジュール名は、エクスポートされた入り口点の名前と一致しなければなりません。

ポインター項目を、動的に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、その後、動的に呼び出されたモジュールをプログラムで取り消すと、ポインター項目は未定義になります。その後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。

関連タスク 『使用するポインター・タイプの決定』 518 ページの『代替入り口点の呼び出し』 544 ページの『DLL でのプロシージャー・ポインターまたは関数ポインターの使用』

関連参照
359 ページの『DLL』
361 ページの『DYNAM』
CANCEL ステートメント (*Enterprise COBOL 言語解説書*)
形式 6: プロシージャー・ポインターおよび関数ポインターのデータ項目用の SET (*Enterprise COBOL 言語解説書*)
ENTRY ステートメント (*Enterprise COBOL 言語解説書*)

使用するポインター・タイプの決定

プロシージャー・ポインターを使用して、他の COBOL プログラムを呼び出したり、言語環境プログラム呼び出し可能サービスを呼び出したりします。関数ポインターを使用して、C/C++プログラムや Java Native Interface が提供するサービスと通信します。

COBOL-COBOL 間呼び出しの場合、プロシージャー・ポインターは関数ポインターよりも一層効率的であり、言語環境プログラム条件処理サービスの呼び出しに必要です。

C で書かれた多くの呼び出し可能サービスは、関数ポインターを戻します。以下に示すように COBOL 関数ポインターを使用して、COBOL プログラムからそのような C 関数ポインターを呼び出すことができます。

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. DEMO.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
*
 WORKING-STORAGE SECTION.
 01 FP USAGE FUNCTION-POINTER.
*
 PROCEDURE DIVISION.
     CALL "c-function" RETURNING FP.
     CALL FP.
```

関連タスク 544 ページの『DLL でのプロシージャー・ポインターまたは関数ポインターの使用』 663 ページの『JNI サービスへのアクセス』

代替入り口点の呼び出し

代替入り口点への静的呼び出しは制限なしで作動します。

代替入り口点への動的呼び出しには次のエレメントが必要です。

- 明示的に指定された NAME または ALIAS リンケージ・エディターまたはバインダー制御ステートメント、またはそれらを自動生成する NAME コンパイラー・オプションの使用。
- 異なる入り口点での同じモジュールに対する動的呼び出しに対する、介在的 CANCEL。CANCEL を使用すると、プログラムは、新しい入り口点で呼び出されたとき初期状態で呼び出されます。

呼び出し先プログラムで ENTRY ラベルを使用することにより、プログラムが実行を開始する別の入り口点を指定できます。ただし、この方法は、構造化プログラムではお勧めしません。

510 ページの『例: 静的および動的 CALL ステートメント』

関連参照
372 ページの『NAME』
CANCEL ステートメント (*Enterprise COBOL 言語解説書*)
ENTRY ステートメント (*Enterprise COBOL 言語解説書*)
MVS プログラム管理: ユーザーズ・ガイドおよび解説書

プログラムを再入可能にする

複数のユーザーが同時にアプリケーション・プログラムを実行する場合 (例えば、異なるアドレス・スペースのユーザーがリンク・パック域にある 1 つのプログラムにアクセスする場合)、RENT オプションを使用してコンパイルすることによって、プログラムを*再入可能*にする必要があります。

プログラマーとしては、変数の複数コピーについて心配する必要はありません。コンパイラーが、オブジェクト・モジュールの中で必要な再入可能制御コードを作成します。

以下の Enterprise COBOL プログラムは再入可能でなければなりません。

- CICS で使用されるプログラム
- IMS でプリロードされるプログラム
- DB2 ストアード・プロシージャーとして使用されるプログラム
- z/OS UNIX 環境で実行されるプログラム
- DLL サポートで使用可能なプログラム
- オブジェクト指向構文を使用するプログラム

再入可能プログラムの場合は、DATA コンパイラー・オプションと HEAP および ALL31 ランタイム・オプションを使用して、WORKING-STORAGE などの動的データ域を 16MB 境界より下のストレージから獲得するか、または上のストレージから獲得するかを制御します。

関連概念
45 ページの『ストレージとそのアドレス可能度』

関連タスク 538 ページの『DLL を作成するためのプログラムのコンパイル』 329 ページの『第 16 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行』

関連参照
383 ページの『RENT』
354 ページの『DATA』
言語環境プログラム・プログラミング・リファレンス (ALL31、HEAP)

第 25 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、共通データへのアクセス権限も必要です。

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
『データの受け渡し』
525 ページの『LINKAGE SECTION のコーディング』
526 ページの『引数を受け渡すための PROCEDURE DIVISION のコーディング』
530 ページの『戻りコード情報の引き渡し』
531 ページの『CALL . . . RETURNING の指定』
532 ページの『EXTERNAL 節によるデータの共用』
532 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)』
668 ページの『Java とのデータ共用』

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。

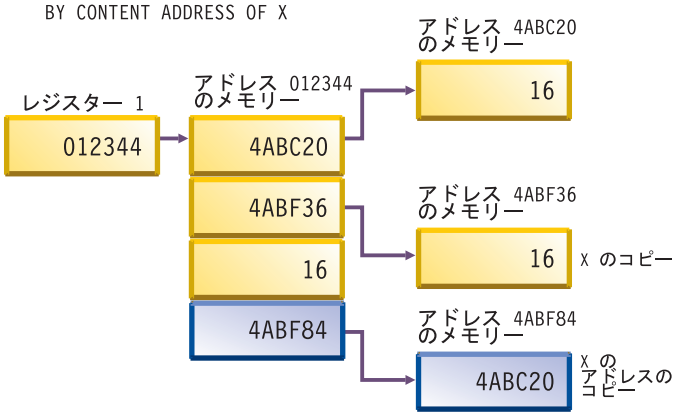
BY REFERENCE
サブプログラムは、データのコピーを処理するのではなく、呼び出し側プログラムのストレージ内のデータ項目を参照して処理します。 BY REFERENCE は、パラメーターに関して 3 つの方法のどれも指定されておらず、暗黙指定されてもいない場合の、パラメーターの想定引き渡しメカニズムです。

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。

次の図に、BY REFERENCE、BY CONTENT、および BY VALUE によって渡される値の違いを示します。

```
MOVE 16 TO X.
CALL ABC USING
    BY REFERENCE X
    BY CONTENT X
    BY VALUE X
    BY CONTENT ADDRESS OF X
```



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 65. **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。
CALL . . . BY REFERENCE *file-name*	データ制御ブロック (DCB) をアセンブラー・プログラムに渡す。	ファイル名は QSAM 順次ファイルを参照しなければなりません。[1]
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 *identifier*	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。	

表 65. **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 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。	

1. CALL オペランドとしてのファイル名は、COBOL への IBM 拡張部分として許可されます。拡張部分の使用は一般的に、コンパイラーの特定の内部インプリメンテーションによって異なります。制御ブロック・フィールドの設定は、今後のリリースで変更される場合があります。制御ブロックに対して行われる変更は、各自の責任であって、IBM がサポートするものではありません。

関連概念
45 ページの『ストレージとそのアドレス可能度』

関連タスク
『呼び出し側プログラムの中での引数の記述』
524 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』
525 ページの『OMITTED 引数に関するテスト』
531 ページの『CALL . . . RETURNING の指定』
532 ページの『EXTERNAL 節によるデータの共用』
532 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)』
668 ページの『Java とのデータ共用』

関連参照
CALL ステートメント (*Enterprise COBOL 言語解説書*)
USING 句 (*Enterprise COBOL 言語解説書*)
INVOKE ステートメント (*Enterprise COBOL 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、最高位の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。

16MB 境界より上のストレージで割り振られたパラメーターを AMODE 24 サブプログラムに渡してはなりません。RENT オプションが有効な場合は DATA(24) オプションを使用し、NORENT オプションが有効な場合は RMODE(24) オプションを使用してください。

関連概念
45 ページの『ストレージとそのアドレス可能度』

関連タスク
525 ページの『LINKAGE SECTION のコーディング』
526 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 (*Enterprise COBOL 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。

上記規則のいずれかに違反していると、呼び出し側プログラムが OPTIMIZE コンパイラー・オプションを指定してコンパイルされた場合、予期しない結果が起こる可能性があります。

関連タスク
525 ページの『LINKAGE SECTION のコーディング』

関連参照
USING 句 (*Enterprise COBOL 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。以下に例を示します。

```
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
```

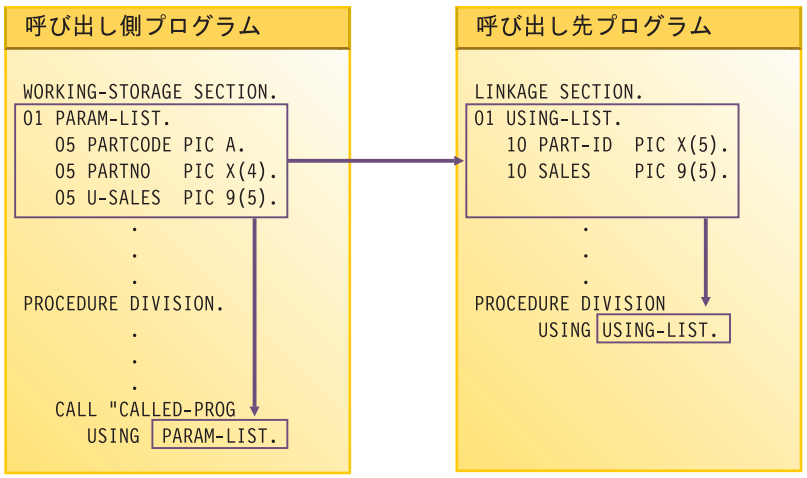
関連参照
CALL ステートメント (*Enterprise COBOL 言語解説書*)
USING 句 (*Enterprise COBOL 言語解説書*)

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 (*Enterprise COBOL 言語解説書*)
USING 句 (*Enterprise COBOL 言語解説書*)
CALL ステートメント (*Enterprise COBOL 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
525 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 (*Enterprise COBOL 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' と合わせてストリング処理動詞を使用する場合、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
117 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 (*Enterprise COBOL 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 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。

『例: チェーン・リストを処理するためのポインターの使用』

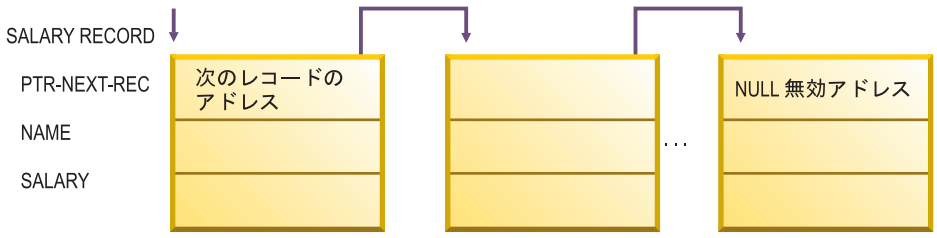
関連タスク
525 ページの『LINKAGE SECTION のコーディング』
526 ページの『引数を受け渡すための PROCEDURE DIVISION のコーディング』

関連参照
SET ステートメント (*Enterprise COBOL 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の論理は、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
```

次のコードは、チェーン・リストを処理するこの例で使用される、呼び出し側プログラム LISTS の概要です。

```
 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 CUMULATE 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 節を使用して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

528 ページの『ポインターによるチェーン・リストの処理』

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。(メソッドは RETURN-CODE 特殊レジスターに情報を戻しませんが、プログラムへの呼び出しの後でこのレジスターを検査できます。)

また、メソッドの PROCEDURE DIVISION ヘッダーで RETURNING 句を使用して、起動プログラムまたはメソッドに情報を戻すこともできます。 PROCEDURE DIVISION . . . RETURNING で CALL . . . RETURNING を指定しても、RETURN-CODE レジスターは設定されません。

RETURN-CODE 特殊レジスターの理解

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レジスター 15 に保管されます。

制御が呼び出しから COBOL プログラムまたはメソッドに戻されると、レジスター 15 の内容は、呼び出し側プログラムまたはメソッドの RETURN-CODE 特殊レジスターに保管されます。COBOL プログラムからオペレーティング・システムに制御が戻されると、特殊レジスターの内容はユーザー戻りコードとして戻されます。

非 COBOL プログラムから COBOL プログラムに制御が戻る場合には、このような RETURN-CODE 特殊レジスターの処理を考慮する必要が生じることがあります。非 COBOL プログラムが、戻りコードを戻すためにレジスター 15 を使用しない場合、COBOL プログラムの RETURN-CODE 特殊レジスターが無効値で更新されることがあります。 Enterprise COBOL プログラムがオペレーティング・システムに戻る前に、この特殊レジスターを意味のある値に設定しない限り、無効な戻りコードがシステムに渡されます。

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。

INVOKE ステートメントを使用して RETURN-CODE 特殊レジスターを設定することはできません。

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
```

CEEPIPI: 言語環境プログラムの事前初期設定サービス (CEEPIPI) を使用して呼び出されるプログラムで PROCEDURE DIVISION RETURNING を指定すると結果は不定となります。

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やメソッド (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 節は、グループ項目または EXTERNAL である従属項目には指定できません。

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、そのプログラムまたはメソッド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。

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またはメソッド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。

『例: 外部ファイルの使用』

関連タスク
14 ページの『入出力操作でのデータの使用』

関連参照
EXTERNAL 節 (*Enterprise COBOL 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。

Name (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.
```

```
 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.
```

第 26 章 DLL または DLL アプリケーションの作成

ダイナミック・リンク・ライブラリー (DLL) または DLL アプリケーションの作成は、通常の COBOL アプリケーションの作成と似ています。すなわち、ソース・コードを書き、コンパイルし、リンクします。

DLL または DLL アプリケーションを作成する際には、以下の特別な考慮事項が適用されます。

- ロード・モジュールまたはアプリケーションの各部分を互いに関連付ける方法、または他の DLL と関連付ける方法を決定する。
- どんなリンクまたは呼び出しのメカニズムを使用するかを決定する。

DLL ロード・モジュールを作成するか、または別個の DLL を参照するロード・モジュールを作成するかによって、若干異なるコンパイラーとリンケージ・エディターまたはバインダー・オプションを使用する必要があります。

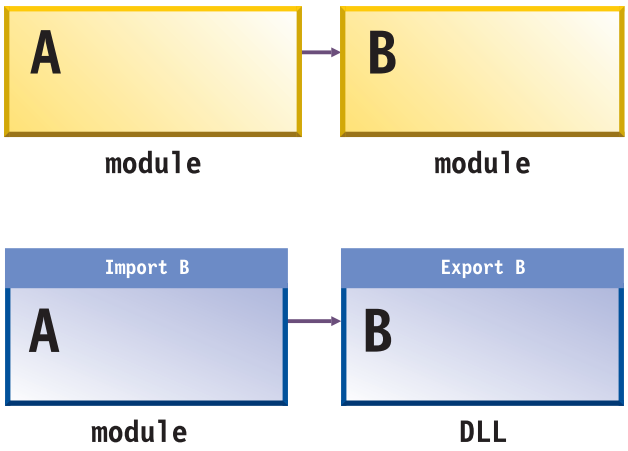
関連概念
『ダイナミック・リンク・ライブラリー (DLL)』

関連タスク 322 ページの『UNIX のもとでの DLL の作成』 538 ページの『DLL を作成するためのプログラムのコンパイル』 539 ページの『DLL のリンク』 542 ページの『DLL での CALL ID の使用』 543 ページの『DLL リンケージと動的呼び出しの併用』 547 ページの『C/C++ プログラムでの COBOL DLL の使用』 548 ページの『OO COBOL アプリケーションでの DLL の使用』 544 ページの『DLL でのプロシージャー・ポインターまたは関数ポインターの使用』

ダイナミック・リンク・ライブラリー (DLL)

DLL は、ロード・モジュールまたはプログラム・オブジェクトであり、他の別個のロード・モジュールからアクセスすることができます。

DLL は、プログラム、関数、または変数の定義を DLL、DLL アプリケーション、または非 DLL へエクスポート するという点で、従来のロード・モジュールとは異なります。このため、ターゲット・ルーチンを、参照ルーチンと同じロード・モジュールにリンクする必要はありません。アプリケーションが初めて別個の DLL を参照すると、システムは自動的にその DLL をメモリーにロードします。すなわち、DLL 内でプログラムを呼び出すことは、動的 `CALL` を使用してロード・モジュールを呼び出すことに似ています。



DLL アプリケーションは、プログラム、関数、または変数のインポートされた定義を参照するアプリケーションです。

z/OS DLL のいくつかの機能は COBOL の動的 CALL ステートメントで提供される機能とオーバーラップしますが、DLL の方が、次のように、通常の z/OS ロード・モジュールおよび動的呼び出しよりも利点があります。

- DLL は COBOL、および C/C++ 間で共通なので、複数のプログラム言語を使用するアプリケーションの相互協調処理が向上します。再入可能な COBOL および C/C++ DLL も、スムーズに相互協調処理することができます。
- 長いプログラム名を持つ、別個の DLL モジュールに入っているプログラムへの呼び出しを行うことができます。 (動的呼び出し解決は、プログラム名を 8 文字に切り捨てます。) COBOL オプションの PGMNAME(LONGUPPER) または PGMNAME(LONGMIXED) と COBOL DLL サポートを使用すると、最高 160 文字までの名前を持つロード・モジュール間で呼び出しを行うことができます。

DLL は、z/OS プログラム管理バインダーによって提供される機能に基づいて、IBM z/OS 言語環境プログラムによってサポートされます。 DLL サポートは、z/OS のもとでバッチ環境で実行中のアプリケーションで、あるいは TSO、CICS、UNIX、または IMS 環境で使用可能です。

関連参照
380 ページの『PGMNAME』
MVS プログラム管理: ユーザーズ・ガイドおよび解説書 (DLL のためのバインダー・サポート)

DLL を作成するためのプログラムのコンパイル

DLL オプションを使用して COBOL プログラムをコンパイルすると、そのプログラムは DLL サポートに使用可能になります。 DLL サポートを使用するアプリケーションは再入可能でなければなりません。このため、RENT コンパイラー・オプションを使用してコンパイルし、それらを RENT バインダー・オプションとリンクしなければなりません。

DLL サポートのあるアプリケーションでは、プログラムまたはクラスの場所に応じて以下のコンパイラー・オプションを使用してください。

表 66. **DLL アプリケーションのコンパイラー・オプション**

プログラムまたはクラスの場所	使用するオプション
ルート・ロード・モジュール	DLL、RENT、NOEXPORTALL
他のロード・モジュールによって使用される DLL ロード・モジュール	DLL、RENT、EXPORTALL

DLL ロード・モジュールに、DLL モジュール内からだけ使用されるプログラムが含まれている場合は、NOEXPORTALL を使用してコンパイルすることによって、これらのルーチンを隠すことができます。

540 ページの『例: プロシージャー型 DLL アプリケーションのサンプル JCL』

関連タスク

322 ページの『UNIX のもとでの DLL の作成』
『DLL のリンク』
541 ページの『特定の DLL のプリリンク』
537 ページの『第 26 章 DLL または DLL アプリケーションの作成』

関連参照

359 ページの『DLL』
362 ページの『EXPORTALL』
383 ページの『RENT』

DLL のリンク

DLL 可能オブジェクト・モジュールを別個の DLL ロード・モジュールにリンクするか、またはそれらを静的にリンクすることができます。リンク時に、アプリケーションを 1 つのモジュールとしてパッケージするか、いくつかの DLL モジュールとしてパッケージするかを決定することができます。

DLL アプリケーションをリンクする場合は、z/OS バインダーの DLL サポートをお勧めします。バインダーは COBOL コンパイラーからの出力を直接受け取ることができ、このため、プリリンク・ステップは必要ありません。ただし、DLL を PDS ロード・ライブラリーに常駐させる必要がある場合は、標準のリンケージ編集の前に、言語環境プログラムのプリリンカーを使用する必要があります。

バインダー・ベースの DLL は、PDS ではなく、PDSE または HFS ファイルに常駐させなければなりません。

バインダーを使用して DLL アプリケーションをリンクする場合は、次のオプションを使用します。

表 67. **DLL アプリケーションのバインダー・オプション**

コードのタイプ	リンクに使用するバインダー・パラメーター
DLL アプリケーション	DYNAM(DLL)、RENT
大/小文字混合のエクスポート済みプログラム名を使用するアプリケーション クラス定義または INVOKE ステートメント	CASE(MIXED)

SYSDEFSD DD ステートメントを指定して、バインダーが DLL 定義サイド・ファイルを作成するデータ・セットを指示してください。このサイド・ファイルには、DLL によってエクスポートされた各記号についての IMPORT 制御ステートメントが入っています。バインダー SYSLIN 入力 (DLL コードを参照するバインディング・コード) には、リンクされるモジュールから参照される DLL のための DLL 定義サイド・ファイルを含めなければなりません。

DLL リンケージに使用させたくないプログラムがモジュールに含まれている場合には、定義サイド・ファイルを編集してそれらのプログラムを除去することができます。

『例: プロシージャー型 DLL アプリケーションのサンプル JCL』

関連タスク
322 ページの『UNIX のもとでの DLL の作成』
537 ページの『第 26 章 DLL または DLL アプリケーションの作成』
538 ページの『DLL を作成するためのプログラムのコンパイル』
541 ページの『特定の DLL のプリリンク』

関連参照
MVS プログラム管理: ユーザーズ・ガイドおよび解説書 (DLL のためのバインダー・サポート)

例: プロシージャー型 DLL アプリケーションのサンプル JCL

以下の例は、DLL サブプログラムを呼び出すメインプログラムから構成されるアプリケーションの作成方法を示しています。

最初のステップでは、サブプログラム DemoDLLSubprogram を含む DLL ロード・モジュールを作成します。 2 番目のステップでは、プログラム MainProgram を含むメイン・ロード・モジュールを作成します。3 番目のステップでアプリケーションを実行します。

```
//DLLSAMP  JOB ,
//  TIME=(1),MSGLEVEL=(1,1),MSGCLASS=H,CLASS=A,
//  NOTIFY=&SYSUID,USER=&SYSUID
// SET LEPFX='SYS1'
//*-------------------------------------------------------------------
//* Compile COBOL subprogram, bind to form a DLL.
//*-------------------------------------------------------------------
//STEP1 EXEC IGYWCL,REGION=80M,GOPGM=DEMODLL,
//       PARM.COBOL='RENT,PGMN(LM),DLL,EXPORTALL',
//       PARM.LKED='RENT,LIST,XREF,LET,MAP,DYNAM(DLL),CASE(MIXED)'
//COBOL.SYSIN    DD *
       Identification division.
       Program-id. "DemoDLLSubprogram".
       Procedure division.
           Display "Hello from DemoDLLSubprogram!".
       End program "DemoDLLSubprogram".
/*
//LKED.SYSDEFSD DD DSN=&&SIDEDECK,UNIT=SYSDA,DISP=(NEW,PASS),
//             SPACE=(TRK,(1,1))
//LKED.SYSLMOD   DD DSN=&&GOSET(&GOPGM),DSNTYPE=LIBRARY,DISP=(MOD,PASS)
//LKED.SYSIN     DD DUMMY
```

```
//*--------------------------------------------------------------------
//* Compile and bind COBOL main program
//*--------------------------------------------------------------------
//STEP2 EXEC IGYWCL,REGION=80M,GOPGM=MAINPGM,
//       PARM.COBOL='RENT,PGMNAME(LM),DLL',
//       PARM.LKED='RENT,LIST,XREF,LET,MAP,DYNAM(DLL),CASE(MIXED)'
//COBOL.SYSIN    DD *
       Identification division.
       Program-id. "MainProgram".
       Procedure division.
           Call "DemoDLLSubprogram"
           Stop Run.
       End program "MainProgram".
/*
//LKED.SYSIN    DD DSN=&&SIDEDECK,DISP=(OLD,DELETE)
//*--------------------------------------------------------------------
//* Execute the main program, calling the subprogram DLL.
//*--------------------------------------------------------------------
//STEP3 EXEC PGM=MAINPGM,REGION=80M
//STEPLIB DD DSN=&&GOSET,DISP=(OLD,DELETE)
//        DD DSN=&LEPFX..SCEERUN,DISP=SHR
//SYSOUT  DD SYSOUT=*
//CEEDUMP DD SYSOUT=*
```

特定の DLL のプリリンク

DLL が PDSE ファイルまたは HFS ファイルではなく、PDS ロード・ライブラリーに常駐しなければならない場合は、標準のリンケージ編集を行う前に、言語環境プログラムのプリリンカーを使用する必要があります。

DLL ソースをコンパイルした後、オブジェクト・モジュールをプリリンクして 1 つのオブジェクト・モジュールにします。

1. プリリンク・ステップに対する SYSDEFSD DD ステートメントを指定して、プリリンカーが DLL 定義サイド・ファイルを作成するデータ・セットを指示します。このサイド・ファイルには、DLL によってエクスポートされた各記号についての IMPORT プリリンカー制御ステートメントが入れられます。プリリンカーは、このサイド・ファイルを使用して、新規の DLL を参照する他のモジュールをプリリンクします。
2. DLLNAME(*xxx*) プリリンカー・オプションを指定して、サイド・ファイルに IMPORT 制御ステートメントを作成する際に使用するプリリンカーの DLL ロード・モジュール名を指示します。あるいは、プリリンカーが、NAME プリリンカー制御ステートメントから、またはプリリンク・ステップに対する SYSMOD DD ステートメントの PDS メンバー名から、DLL ロード・モジュール名を入手することができます。
3. 新しい DLL が他の DLL を参照する場合は、このプリリンク・ステップへの入力であるオブジェクト・デックと一緒に、これらの DLL の定義サイド・ファイルを含めてください。これらのサイド・ファイルは、プリリンカーに、現行モジュールの記号参照を他の DLL からエクスポートされた記号に解決するように指示します。

通常のようにリンケージ・エディターまたはバインダーを使用して、プリリンカーによって作成されるオブジェクト・モジュールから DLL ロード・モジュールを作成します。リンケージ・エディターまたはバインダーの RENT オプションを指定してください。

関連タスク
538 ページの『DLL を作成するためのプログラムのコンパイル』
539 ページの『DLL のリンク』

DLL での CALL ID の使用

DLL オプションを指定してコンパイルされた COBOL プログラムの場合、CALL *identifier* および CALL *literal* ステートメントを使用して DLL を呼び出すことができます。ただし、CALL *identifier* の場合には、追加の考慮事項があります。

identifier の内容または *literal* の場合は、次のいずれかのプログラムの名前を使用してください。

- CALL *identifier* ステートメントを含んでいるプログラムからの呼び出しに適格な、同じコンパイル単位内のネストされたプログラム。
- 別個のバインドされた DLL モジュール内のプログラム。ターゲット・プログラム名は DLL からエクスポートされていなければなりません。また、DLL モジュール名は、ターゲット・プログラムのエクスポートされた名前と一致していなければなりません。

ネストされていない場合、ランタイム環境は、CALL ステートメントを含むプログラムの PGMNAME コンパイラー・オプションの設定に従って、*identifier* 内のプログラム名を解釈します。また、ターゲット DLL からエクスポートされたプログラム名は、ターゲット・プログラムのコンパイル時に使用された PGMNAME オプションの設定に従って解釈されます。

階層ファイル・システム (HFS) におけるターゲット DLL の探索には、大/小文字の区別があります。ターゲット DLL が PDS または PDSE メンバーである場合、DLL メンバー名は 8 文字以下でなければなりません。PDS または PDSE メンバーとして DLL を探索するために、ランタイム環境は名前を自動的に大文字に変換します。

ランタイム環境が上記のいずれかの CALL ステートメントを解決できない場合、制御権は CALL ステートメントの ON EXCEPTION または ON OVERFLOW 句に移動します。この状況で CALL ステートメントにこれらの句の 1 つが指定されていないと、言語環境プログラムは重大度 3 の条件を発生させます。

関連タスク
543 ページの『DLL リンケージと動的呼び出しの併用』
538 ページの『DLL を作成するためのプログラムのコンパイル』
539 ページの『DLL のリンク』

関連参照
359 ページの『DLL』
380 ページの『PGMNAME』
CALL ステートメント (*Enterprise COBOL 言語解説書*)
543 ページの『HFS での DLL の探索順序』

HFS での DLL の探索順序

階層ファイル・システム (HFS) を使用する場合は、CALL ステートメントの DLL 参照を解決するための探索順序は、言語環境プログラムの POSIX ランタイム・オプションの設定によって異なります。

POSIX ランタイム・オプションが ON の場合の探索順序は、次のとおりです。

1. ランタイム環境は、HFS に DLL がないかどうか探します。LIBPATH 環境変数が設定されていると、リストされたそれぞれのディレクトリーが探索されます。これが設定されていない場合は、現行ディレクトリーだけが探索されます。HFS での DLL の探索には、大/小文字の区別があります。
2. HFS で DLL が見つからないと、ランタイム環境は、呼び出し側の MVS ロード・ライブラリー探索順序から DLL をロードすることを試みます。この場合、DLL 名は 8 文字以下でなければなりません。この探索では、ランタイム環境は自動的に DLL 名を大文字に変換します。

POSIX ランタイム・オプションが OFF に設定されていると、探索順序は逆になります。

1. ランタイム環境は、呼び出し側のロード・ライブラリー用の探索順序から DLL をロードすることを試みます。
2. このロード・ライブラリーから DLL をロードできない場合、ランタイム環境は HFS から DLL をロードすることを試みます。

関連タスク
542 ページの『DLL での CALL ID の使用』

関連参照
言語環境プログラム・プログラミング・リファレンス (POSIX)

DLL リンケージと動的呼び出しの併用

複数の別個にバインドされたモジュールとして構築されるアプリケーション (つまり、言語環境プログラム・エンクレーブ) の場合は、モジュール相互間で 1 つのリンケージ形式 (つまり、動的呼び出しリンケージまたは DLL リンケージのいずれか) のみを使用しなければなりません。

DLL リンケージ とは、DLL および NODYNAM 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呼び出しを指します (呼び出しは、別個のモジュール内のエクスポートされた名前に解決されます)。DLL リンケージが、別個のモジュール内に定義されているメソッドの呼び出しを指すこともあります。

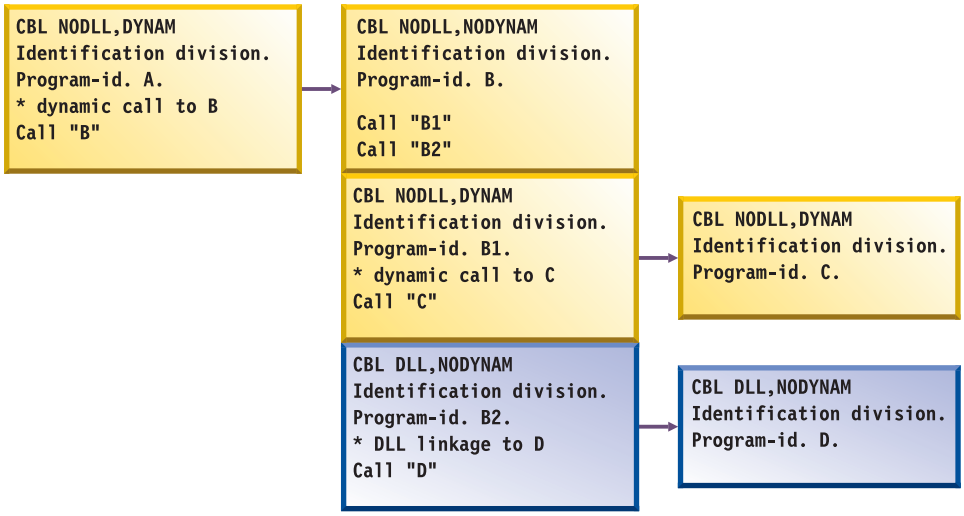
しかし、アプリケーションによってはもっと多くの柔軟性が必要になります。そのような場合は、プログラムが次のようにコンパイルされていれば、言語環境プログラム・エンクレーブ内で DLL リンケージと COBOL 動的呼び出しリンケージの両方を使用することができます。

プログラム A	プログラム B	コンパイル・オプション
動的呼び出しを含む	動的呼び出しのターゲット	NODLL

プログラム A	プログラム B	コンパイル・オプション
DLL リンケージを使用する	ターゲット・プログラムまたはメソッドを含む	DLL

プログラムに、別個にコンパイルされたプログラムに対する CALL ステートメントが含まれている場合、一方のプログラムを DLL コンパイラー・オプションでコンパイルし、他方を NODLL でコンパイルすると、呼び出しがサポートされるのは、2 つのプログラムを一緒に同じモジュールでバインドした場合だけです。

次の図では、別個にバインドされたモジュールをいくつか示します。これらのモジュールには、動的呼び出しと DLL リンケージが混在しています。



DLL アプリケーションのコンポーネントはすべて同じ AMODE でなければなりません。COBOL 動的呼び出しによって通常提供される自動 AMODE 切り替えは、DLL リンケージでは利用できません。

DLL リンケージを使用して呼び出されるプログラムを取り消すことはできません。

関連概念
537 ページの『ダイナミック・リンク・ライブラリー (DLL)』

関連タスク 538 ページの『DLL を作成するためのプログラムのコンパイル』 539 ページの『DLL のリンク』 『DLL でのプロシージャー・ポインターまたは関数ポインターの使用』 545 ページの『非 DLL からの DLL の呼び出し』

関連参照
359 ページの『DLL』
362 ページの『EXPORTALL』

DLL でのプロシージャー・ポインターまたは関数ポインターの使用

DLL と非 DLL の両方を含む実行単位では、プロシージャー・ポインターおよび関数ポインター・データ項目の使用には注意してください。

NODLL オプションを指定してコンパイルするプログラムで SET *procedure-pointer-1* TO ENTRY *entry-name* ステートメントまたは SET *function-pointer-1* TO ENTRY *entry-name* ステートメントを使用するときは、DLL オプションを指定してコンパイルされるプログラムへのポインターを渡さないでください。しかし、このステートメントを DLL オプションでコンパイルされたプログラムで使用するときは、別個にバインドされた DLL モジュールに含まれているプログラムへのポインターを渡すことができます。

NODYNAM および DLL オプションでコンパイルするとき、*entry-name* が ID である場合、その ID 値は、DLL モジュールからエクスポートされる入り口点名を参照しなければなりません。DLL モジュール名は、エクスポートされる入り口点の名前と一致していなければなりません。この場合は、次の点にも注意してください。

- ID に含まれているプログラム名は、CALL ステートメントを含むプログラムの PGMNAME(COMPAT|LONGUPPER|LONGMIXED) コンパイラー・オプションの設定に従って解釈されます。
- ターゲット DLL からエクスポートされるプログラム名は、ターゲット・プログラムのコンパイル時に使用された PGMNAME オプションの設定に従って解釈されます。
- HFS におけるターゲット DLL の探索には、大/小文字の区別があります。
- ターゲット DLL が PDS または PDSE メンバーである場合、DLL メンバー名は 8 文字以下でなければなりません。PDS または PDSE メンバーとして DLL を探索するために、名前は自動的に大文字に変換されます。

関連タスク

542 ページの『DLL での CALL ID の使用』
516 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』
538 ページの『DLL を作成するためのプログラムのコンパイル』
539 ページの『DLL のリンク』

関連参照

359 ページの『DLL』
362 ページの『EXPORTALL』

非 DLL からの DLL の呼び出し

NODLL オプションを使用してコンパイルされた COBOL プログラムから DLL を呼び出すことができますが、制限があります。

以下の方法を使用して、DLL リンケージに確実に従うようにすることができます。

- COBOL の非 DLL プログラムから呼び出したい COBOL DLL プログラムを、メインプログラムを含むロード・モジュールに入れます。COBOL の非 DLL プログラムから静的呼び出しを使用して、COBOL DLL プログラムを呼び出します。

  メインプログラムを含むロード・モジュールに含まれる COBOL DLL プログラムからは、他の DLL 内の COBOL DLL プログラムを呼び出せます。
- COBOL DLL プログラムを DLL に入れ、それを COBOL 非 DLL プログラムから CALL *function-pointer* を使用して呼び出します (*function-pointer* には、ターゲ

ット・プログラムの関数記述子を設定します)。 DLL 内におけるプログラムの関数記述子のアドレスを入手するには、dllload および dllqueryfn を使用する C ルーチンを呼び出します。

『例: 非 DLL からの DLL の呼び出し』

関連タスク
516 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』

例: 非 DLL からの DLL の呼び出し

次の例では、DLL に含まれていない COBOL プログラム (COBOL1) から、DLL 内の COBOL プログラム (DLL OOC05R の中のプログラム ooc05R) を呼び出す方法が示されています。

```
CBL NODYNAM
       IDENTIFICATION DIVISION.
       PROGRAM-ID. 'COBOL1'.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
       DATA DIVISION.
       FILE SECTION.
       WORKING-STORAGE SECTION.
       01 DLL-INFO.
           03 DLL-LOADMOD-NAME PIC X(12).
           03 DLL-PROGRAM-NAME PIC X(160).
           03 DLL-PROGRAM-HANDLE FUNCTION-POINTER.
       77 DLL-RC               PIC S9(9) BINARY.
       77 DLL-STATUS           PIC X(1) VALUE 'N'.
           88 DLL-LOADED       VALUE 'Y'.
           88 DLL-NOT-LOADED   VALUE 'N'.

       PROCEDURE DIVISION.

           IF DLL-NOT-LOADED
           THEN
      *      Move the names in. They must be null terminated.
             MOVE Z'OOC05R' TO DLL-LOADMOD-NAME
             MOVE Z'ooc05r' TO DLL-PROGRAM-NAME

      *      Call the C routine to load the DLL and to get the
      *      function descriptor address.
             CALL 'A1CCDLGT' USING BY REFERENCE DLL-INFO
                                   BY REFERENCE DLL-RC
             IF DLL-RC = 0
             THEN
               SET DLL-LOADED TO TRUE
             ELSE
               DISPLAY 'A1CCLDGT failed with rc = '
                 DLL-RC
               MOVE 16 TO RETURN-CODE
               STOP RUN
             END-IF
           END-IF

      *    Use the function pointer on the call statement to call the
      *    program in the DLL.
      *    Call the program in the DLL.
           CALL DLL-PROGRAM-HANDLE

           GOBACK.
```

```
#include <stdio.h>
#include <dll.h>
#pragma linkage (A1CCDLGT,COBOL)

typedef struct dll_lm {
  char         dll_loadmod_name[(12]);
  char         dll_func_name[(160]);
  void         (*fptr) (void); /* function pointer */
  } dll_lm;

void A1CCDLGT (dll_lm *dll, int *rc)
{
  dllhandle *handle;
  void (*fptr1)(void);
  *rc = 0;
   /* Load the DLL                                  */
  handle = dllload(dll->dll_loadmod_name);
  if (handle == NULL) {
     perror("A1CCDLGT failed on call to load DLL./n");
     *rc = 1;
     return;
  }

  /* Get the address of the function              */
  fptr1 = (void (*)(void))
          dllqueryfn(handle,dll->dll_func_name);
  if (fptr1 == NULL) {
     perror("A1CCDLGT failed on retrieving function./n");
     *rc = 2;
     return;
  }
  /* Return the function pointer                  */
  dll->fptr = fptr1;
  return;
}
```

C/C++ プログラムでの COBOL DLL の使用

DLL の COBOL サポートは、COBOL EXTERNAL データを除いて、z/OS C/C++ 製品の DLL サポートと相互協調で処理します。特に、COBOL アプリケーションは、C/C++ DLL からエクスポートされた関数を呼び出すことができ、C/C++ アプリケーションは、COBOL DLL からエクスポートされた COBOL プログラムを呼び出すことができます。

EXTERNAL 属性で宣言された COBOL データ項目は、DLL サポートとは無関係です。これらのデータ項目は、プログラムが DLL 内にあるかどうかに関係なく、それらを宣言する実行単位内の任意の COBOL プログラムから名前によってアクセスすることができます。

COBOL オプション DLL、RENT、および EXPORTALL は、C/C++ DLL、RENT、および EXPORTALL オプションとほとんど同じように機能します。 (DLL オプションは C にしか適用されません。) ただし、C/C++ コンパイラーは、デフォルトでは、DLL 対応コードを生成します。

C/C++ DLL 関数ポインターを COBOL に渡し、C/C++ 関数ポインターを 関数ポインター・データ項目として受け取り、それを COBOL 内で使用することができま

す。次の例は、サービスへの関数ポインターを戻す C 関数への COBOL 呼び出しと、そのサービスへの COBOL 呼び出しを示しています。

```
Identification Division.
Program-id. Demo.
Data Division.
Working-Storage section.
01  fp usage function-pointer.
Procedure Division.
    Call "c-function" returning fp.
    Call fp.
```

関連タスク

538 ページの『DLL を作成するためのプログラムのコンパイル』
539 ページの『DLL のリンク』

関連参照

359 ページの『DLL』
362 ページの『EXPORTALL』
383 ページの『RENT』
EXTERNAL 節 (*Enterprise COBOL 言語解説書*)

OO COBOL アプリケーションでの DLL の使用

DLL、THREAD、RENT、および DBCS コンパイラー・オプションを使用して、各 COBOL クラス定義をコンパイルしてから、RENT バインダー・オプションを使用して、それを別個の DLL モジュールにリンク・エディットしなければなりません。

関連タスク 329 ページの『第 16 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行』 538 ページの『DLL を作成するためのプログラムのコンパイル』 539 ページの『DLL のリンク』

関連参照

359 ページの『DLL』
396 ページの『THREAD』
383 ページの『RENT』
357 ページの『DBCS』

第 27 章 マルチスレッド化のための COBOL プログラムの準備

バッチ、TSO、IMS、または UNIX で、プロセス内の複数のスレッドで COBOL プログラムを実行することができます。

マルチスレッド化を実行するための明示的な COBOL 言語は存在しません。したがって、THREAD コンパイラー・オプションでコンパイルします。

COBOL はプログラム・スレッドの管理を直接サポートしません。ただし、マルチスレッド化アプリケーション・サーバー、C/C++ ドライバー・プログラムを使用してスレッドを作成するアプリケーション、Java と相互協調処理して Java スレッドを使用するプログラム、および PL/I タスキングを使用するアプリケーションにおいて、THREAD コンパイラー・オプションでコンパイルする COBOL プログラムを実行することができます。すなわち、他のプログラムは COBOL プログラムを呼び出して、COBOL プログラムをプロセス内の複数のスレッドで、またはスレッド内の複数のプログラム起動インスタンスとして、実行させることができます。スレッド化されたアプリケーションは、単一の 言語環境プログラム・エンクレーブ内で実行される必要があります。

LOCAL-STORAGE または WORKING-STORAGE の選択: マルチスレッド化プログラムは再帰的プログラムとしてコーディングしなければならないので、データの永続性は、再帰的プログラムの永続性となります。

- LOCAL-STORAGE SECTION 内のデータ項目は、プログラム起動のインスタンスごとに自動的に割り振られます。あるプログラムが複数のスレッドにおいて同時に実行されると、起動ごとに個別の LOCAL-STORAGE データのコピーを使用します。
- WORKING-STORAGE SECTION 内のデータ項目は、プログラムごとに一度割り振られるため、最後に使われた状態がプログラムのすべての起動において使用可能です。

個々のプログラム起動インスタンスに分離するデータの場合、そのデータを LOCAL-STORAGE SECTION で定義します。一般的に、この選択はスレッド化プログラム内の作業データに適しています。データを WORKING-STORAGE に宣言し、プログラムがそのデータの内容を変更する場合には、以下のアクションのどちらかを実行する必要があります。

- WORKING-STORAGE 内のデータに複数のスレッドから同時にアクセスしないようにアプリケーションを構成します。
- データに別々のスレッドから同時にアクセスする場合には、適切な直列化コードを書き込みます。

関連概念
550 ページの『マルチスレッド化』

関連タスク
551 ページの『マルチスレッド化サポートのための THREAD の選択』
552 ページの『マルチスレッド化されたプログラムへの制御権移動』
552 ページの『マルチスレッド化されたプログラムの終了』

553 ページの『マルチスレッド化によるファイルの処理』
556 ページの『マルチスレッド化による COBOL 制限の処理』

関連参照
396 ページの『THREAD』
PROGRAM-ID 段落 (*Enterprise COBOL 言語解説書*)

マルチスレッド化

マルチスレッド化の COBOL サポートを使用するには、プロセス、スレッド、実行単位、プログラム起動インスタンスの相互関係を理解する必要があります。

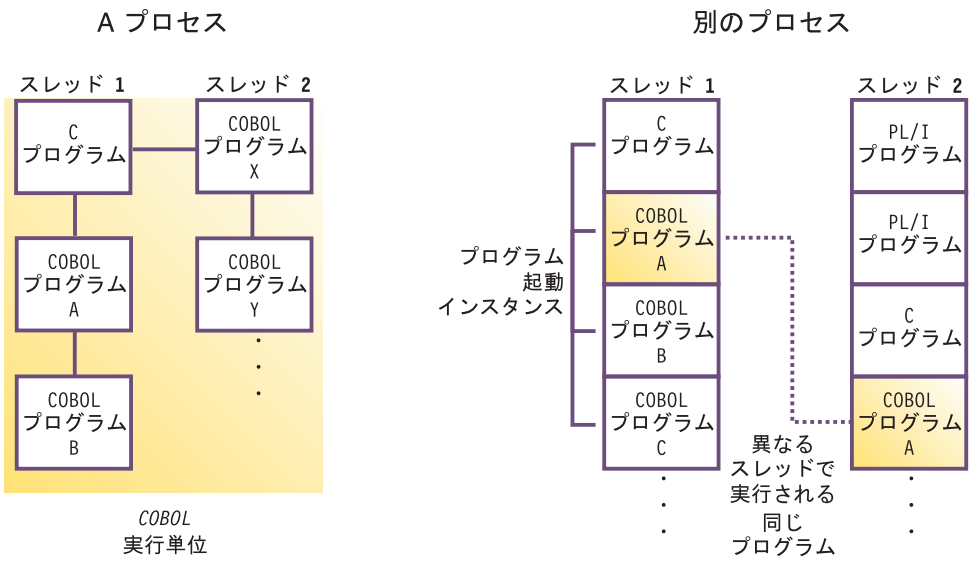
オペレーティング・システムおよびマルチスレッド化アプリケーションは、*プロセス* 内の実行フローを処理することができます。実行フローとは、プログラムのすべて、または一部が実行時に発生する一連のイベントです。 1 つのプロセス内で実行されるプログラムはリソースを共用することができます。プロセスは操作することができます。例えば、システムがプロセスの実行に使用する時間において、プロセスに高い、または低い優先順位を持たせることができます。

プロセス内で、アプリケーションは 1 つ以上の*スレッド* を開始することができます。スレッドはそれぞれ、そのスレッドを制御するコンピューター命令のストリームです。マルチスレッド化プロセスは、1 つの命令ストリーム (スレッド) で始まり、タスクを実行する他の命令ストリームを後で作成することができます。これらの複数のスレッドは並行して実行することができます。スレッド内では、実行プログラムの間で制御権が移動します。

マルチスレッド化環境で、COBOL *実行単位* は、実行中のアクティブな COBOL プログラムが含まれているスレッドを含むプロセスの部分です。COBOL 実行単位は、スレッドのいずれかの実行スタックでアクティブな COBOL プログラムがなくなるまで継続します。例えば、呼び出される COBOL プログラムには GOBACK ステートメントが含まれていて、C プログラムに制御を戻します。実行単位内では、COBOL プログラムは非 COBOL プログラムを呼び出すことができ、逆に、非 COBOL プログラムは COBOL プログラムを呼び出すことができます。

スレッド内では、別々の COBOL および 非 COBOL プログラムの間で制御が移動します。例えば、COBOL プログラムは別の COBOL プログラムまたは C プログラムを呼び出すことができます。別々に呼び出されたプログラムはそれぞれ、*プログラム起動インスタンス* です。特定プログラムのプログラム起動インスタンスは、指定したプロセス内で複数のスレッドに存在することができます。

次の図に、プロセス、スレッド、実行単位、およびプログラム起動インスタンスの間の関係を示します。



関連概念
言語環境プログラム・プログラミング・ガイド (プログラム管理モデル、基礎知識: スレッド)

関連タスク
『マルチスレッド化サポートのための THREAD の選択』
552 ページの『マルチスレッド化されたプログラムへの制御権移動』
552 ページの『マルチスレッド化されたプログラムの終了』
553 ページの『マルチスレッド化によるファイルの処理』
556 ページの『マルチスレッド化による COBOL 制限の処理』

関連参照
396 ページの『THREAD』

マルチスレッド化サポートのための THREAD の選択

マルチスレッド化をサポートするために、THREAD コンパイラー・オプションを使用します。プログラムが 1 つのアプリケーションによって単一プロセスの複数のスレッドで呼び出される場合には、THREAD を使用します。ただし、THREAD は、シリアライゼーション・ロジックが自動的に生成されるため、パフォーマンスに悪影響を与える可能性があります。

COBOL プログラムを複数のスレッドで実行するには、THREAD コンパイラー・オプションを使用して、実行単位内の COBOL プログラムのすべてをコンパイルしなければなりません。また、RENT 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、それらをバインダーまたはリンケージ・エディターの RENT オプションとリンクする必要があります。

オブジェクト指向 (OO) のクライアントとクラスをコンパイルする場合は、 THREAD オプションを使用します。

言語制限: THREAD オプションを使用する場合は、ある特定の言語エレメントを使用することができません。詳細は、下記の関連参照を参照してください。

再帰: THREAD 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する前に、PROGRAM-ID 段落で RECURSIVE 句を指定する必要があります。指定しなければ、エラーが発生します。

関連タスク 20 ページの『再帰的またはマルチスレッド化されたプログラムでのデータの共用』 329 ページの『UNIX のもとでのオブジェクト指向アプリケーションのコンパイル』

関連参照
396 ページの『THREAD』

マルチスレッド化されたプログラムへの制御権移動

マルチスレッド化環境用に COBOL プログラムを書くときは、適切なプログラム・リンケージ・ステートメントを選択します。

単一スレッド環境の場合のように、実行単位内で最初に呼び出されるとき、および呼び出されるプログラムに対する CANCEL 後に最初に呼び出されるときは、呼び出されるプログラムは初期状態にあります。CANCEL ステートメントで名前を付けるプログラムがどのスレッドにおいてもアクティブでないことを確認してください。アクティブ・プログラムを取り消そうとした場合には、重大度 3 の言語環境プログラム条件が発生します。

スレッド化アプリケーションに事前初期設定が必要である場合には、言語環境プログラム・サービス (CEEPIPI インターフェース) を使用します。事前初期設定のための COBOL 固有のインターフェース (ランタイム・オプション RTEREUS と、関数 IGZERRE および ILBOSTP0) を使用して、THREAD オプションでコンパイルしたプログラムから再使用可能環境を設定することはできません。

関連概念
言語環境プログラム・プログラミング・ガイド (言語環境プログラム終了: エンクレーブ終了)

関連タスク
『マルチスレッド化されたプログラムの終了』
500 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』

マルチスレッド化されたプログラムの終了

GOBACK、EXIT PROGRAM、または STOP RUN を使用して、マルチスレッド・プログラムを終了させることができます。

プログラムの呼び出し側に戻るには、GOBACK を使用します。あるスレッド内の最初のプログラムから GOBACK を使用するとき、そのスレッドは終了します。そのスレッドがあるエンクレーブ内の初期スレッドである場合には、そのエンクレーブは終了します。

メインプログラムからの場合を除き、GOBACK と同様に、EXIT PROGRAM を使用します。メインプログラムでは EXIT PROGRAM は無効です。

言語環境プログラムのエンクレーブ全体を終了し、メインプログラム (オペレーティング・システムであると考えられます) の呼び出し側に制御を戻すには、STOP RUN を使用します。エンクレーブ内で実行中のすべてのスレッドが終了します。

関連概念
言語環境プログラム・プログラミング・ガイド (言語環境プログラム終了: エンクレーブ終了)

関連タスク
500 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』

マルチスレッド化によるファイルの処理

スレッド化アプリケーションでは、QSAM、VSAM、および行順次ファイルでの入出力のために COBOL ステートメントをコーディングすることができます。

各ファイル定義 (FD) には暗黙のシリアライゼーション・ロックがあります。このロックは、以下のステートメントの実行に関連付けられた入力または出力操作の間に、自動シリアライゼーション・ロジックとともに使用されます。

- OPEN
- CLOSE
- READ
- WRITE
- REWRITE
- START
- DELETE

また、自動シリアライゼーションは、以下のステートメントに関連付けられた暗黙の MOVE の場合にも発生します。

```
WRITE record-name FROM identifier
READ file-name INTO identifier
```

自動シリアライゼーションは、以下の条件付き句内で指定されるステートメントには適用されません。

- AT END
- NOT AT END
- INVALID KEY
- NOT INVALID KEY
- AT END-OF-PAGE
- NOT AT END-OF-PAGE

関連概念
554 ページの『ファイル定義 (FD) ストレージ』

関連タスク
183 ページの『QSAM ファイルのクローズ』
217 ページの『VSAM ファイルのクローズ』

268 ページの『ERROR 宣言のコーディング』
『マルチスレッド化によるファイル・アクセスのシリアライズ』

ファイル定義 (FD) ストレージ

すべてのプログラム起動の際に、ファイル定義に関連付けられたストレージ (FD レコードや SAME RECORD AREA 節に関連付けられたレコード域) は割り振られ、最後に使われた状態で使用可能です。

実行のすべてのスレッドはこのストレージを共用します。OPEN、CLOSE、READ、WRITE、REWRITE、START、および DELETE ステートメントの実行中は、このストレージの自動シリアライゼーションに依存することができますが、これらのステートメントの使用の間は依存することはできません。

関連タスク
『マルチスレッド化によるファイル・アクセスのシリアライズ』

マルチスレッド化によるファイル・アクセスのシリアライズ

自動シリアライゼーションの利点を最大に活用し、固有のシリアライゼーション・ロジックの明示的書き込みを避けるには、スレッド化プログラム内のファイルにアクセスするときに、推奨されるいずれかのファイル編成および使用パターンを使用します。

以下のいずれかのファイル編成を使用する。

- 順次編成
- 行順次編成
- 順次アクセスによる相対編成
- 順次アクセスによる索引編成

入力に次のパターンを使用します。

```
  OPEN INPUT fn
  . . .
  READ fn INTO local-storage-item
  . . .
* Process the record from the local-storage item
  . . .
  CLOSE fn
```

出力に次のパターンを使用します。

```
  OPEN OUTPUT fn
  . . .
* Construct output record in local-storage item
  . . .
  WRITE rec FROM local-storage-item
  . . .
  CLOSE fn
```

他の使用パターンの場合には、以下のアクションのいずれかを実行します。

- アプリケーション・ロジックの安全を検証します。プログラムの 2 つのインスタンスが別々のスレッドで決して同時にアクティブにならないことを確認します。

- POSIX サービスに対する呼び出しを使用して、明示的シリアライゼーション・ロジックをコーディングします。

複数のスレッドからファイルにアクセスする際にシリアライゼーション問題の発生を避けるには、LOCAL-STORAGE SECTION で、ファイルに関連付けられたデータ項目 (ファイル状況データ項目やキー引数) を定義します。

『例: マルチスレッド化によるファイル入出力の使用パターン』

関連タスク
493 ページの『UNIX/POSIX API の呼び出し』

例: マルチスレッド化によるファイル入出力の使用パターン

以下の例では、マルチスレッド化アプリケーションにおいてファイル入出力の推奨使用パターンから逸脱した際の明示的シリアライゼーション・ロジックの必要性について説明します。また、これらの例では、シリアライゼーションを適切に処理できない結果として発生する予期しない動作についても説明します。

それぞれの例において、サンプル操作を含むプログラムの 2 つのインスタンスが 1 つの実行単位内で、2 つの異なるスレッド上で実行されています。

```
READ F1
. . .
REWRITE R1
```

上記の例で、2 番目のスレッドは、最初のスレッド上で READ ステートメントが実行された後に (ただし、最初のスレッド上で REWRITE ステートメントが実行される前に) READ ステートメントを実行すると考えられます。REWRITE ステートメントは、意図したレコードの更新は行わない可能性があります。目的の結果が確実に出るようにするには、明示的シリアライゼーション・ロジックを書き込みます。

```
 READ F1
 . . .
* Process the data in the FD record description entry for F1
 . . .
```

上記の例で、2 番目のスレッドは、最初のスレッドが FD レコード記述項目内のレコードをまだ処理中に READ ステートメントを実行すると考えられます。 2 番目の READ ステートメントは、最初のスレッドが処理中であるレコードをオーバーレイする可能性があります。この問題を回避するには、次の推奨技法を使用してください。

```
READ F1 INTO LOCAL-STORAGE-item
```

その他のケース: 後に READ NEXT が続く START や、後に DELETE が続く READ など、一連の関連した入出力操作を必要とするその他の使用パターンについても同様に考慮してください。適切な手順を実行して、ファイル入出力の正しい処理を確実に行います。

マルチスレッド化による COBOL 制限の処理

一部の COBOL アプリケーションは、サブシステムまたは他のアプリケーションに依存します。マルチスレッド化環境において、これらの依存性などに起因して、COBOL プログラムに対するいくつかの制限が発生します。

一般的に、実行単位内のアプリケーションに可視であるリソースへのアクセスを同期化する必要があります。この要件の例外には、DISPLAY および ACCEPT があります。これらは複数のスレッドから使用でき、また推奨使用パターンを持つサポート対象の COBOL ファイル入出力ステートメントからも使用できます。これらに対する同期化は、すべてランタイム環境によって提供されます。

CICS: マルチスレッド化アプリケーションは CICS 環境において実行することはできません。CICS 環境では、THREAD オプションを使用してコンパイルし、複数のスレッドまたは PL/I タスクを持たないアプリケーションの一部である COBOL プログラムを実行できます。

再帰的: マルチスレッド化アプリケーション内のプログラムは再帰的プログラムとしてコーディングする必要があります。したがって、ネストされたプログラムをコーディングしないなど、再帰的プログラムに適用される制限やプログラミング上の考慮事項を忠実に守る必要があります。

再入可能性: マルチスレッド化プログラムは、RENT コンパイラー・オプションを使用してコンパイルし、それらをバインダーまたはリンケージ・エディターの RENT オプションとリンクしなければなりません。

POSIX および PL/I: マルチスレッド化アプリケーションで POSIX スレッドを使用する場合には、言語環境プログラム・ランタイム・オプション POSIX(ON) を指定しなければなりません。アプリケーションが PL/I タスキングを使用する場合には、POSIX(OFF) を指定しなければなりません。POSIX スレッドと PL/I タスクを同一のアプリケーションに混在させることはできません。

PL/I タスク: 複数の PL/I タスクを含むアプリケーションに COBOL プログラムを組み込む場合の指針を以下に示します。

- 複数の PL/I タスクで実行する COBOL プログラムはすべて、THREAD オプションを指定してコンパイルします。NOTHREAD オプションを指定してコンパイルされた COBOL プログラムが 1 つでもあると、どの COBOL プログラムも単一の PL/I タスクでしか実行できなくなります。
- THREAD オプションを指定してコンパイルされた COBOL プログラムは、1 つ以上の PL/I タスクから呼び出すことができます。しかし、PL/I プログラムからの COBOL プログラムの呼び出しでは、TASK オプションや EVENT オプションを指定することができません。PL/I タスク呼び出しでは、最初に PL/I のプログラムまたは関数を呼び出し、そこから COBOL プログラムを呼び出す必要があります。このような間接呼び出しが必要になるのは、TASK オプションまたは EVENT オプションを含む PL/I の CALL ステートメントのターゲットとして COBOL プログラムを指定することができないからです。
- COBOL プログラムから STOP RUN ステートメントを発行したり、PL/I プログラムから STOP ステートメントを発行すると、言語環境プログラムのエンクレーブ全体 (すべてのタスク実行を含む) が終了してしまいます。

- PL/I タスクを含む実行単位では、明示的な POSIX スレッド化 (pthread_create() の呼び出し) をコーディングしないでください。

C および言語環境プログラム対応アセンブラー: マルチスレッド COBOL プログラムは、C プログラムおよび言語環境プログラム対応アセンブラー・プログラムがマルチスレッド実行用に適切にコーディングされている場合、同じ実行単位内でそれらのプログラムと結合することができます。

AMODE: マルチスレッド化アプリケーションを AMODE 31 で実行しなければなりません。AMODE 24 を複数のスレッドまたは PL/I タスクを持たないアプリケーションの一部として、THREAD オプションでコンパイルした COBOL プログラムを実行することができます。

非同期シグナル: スレッド化アプリケーションにおいて、COBOL プログラムは、非同期シグナルまたは割り込みによって割り込まれる場合があります。プログラムにそのような割り込みを容認できないロジックが含まれている場合には、そのロジックが継続する間はそうした割り込みを使用不可にする必要があります。C/C++ 関数を呼び出して、シグナル・マスクを適切に設定します。

古い COBOL プログラム: マルチスレッド化アプリケーションの複数のスレッド上で COBOL プログラムを実行するには、それらを Enterprise COBOL でコンパイルし、THREAD オプションを使用する必要があります。古いコンパイラーでコンパイルしたプログラムを実行する場合には、以下の制限を守る必要があります

- OS/VS COBOL プログラムを初期スレッド (IPT) 上にのみ含むアプリケーションを実行すること。
- 他の古いコンパイラーによって、1 つのスレッド上のみでコンパイルしたプログラムを含むアプリケーションを実行すること (ただし、スレッドは初期スレッド以外のスレッドでも構いません)。

IGZBRDGE、IGZETUN、および IGZEOPT: 静的呼び出しを動的呼び出しに変換するためのマクロである IGZBRDGE を、THREAD オプションでコンパイルしたプログラムと併用してはなりません。このマクロはサポートされません。 THREAD オプションでメインプログラムをコンパイルした、アプリケーションに対して、モジュール IGZETUN (ストレージ・チューニング用) または IGZEOPT (ランタイム・オプション用) を使用してはなりません。これらの CSECT は無視されます。

UPSI スレッド: アプリケーション内のすべてのプログラムおよびすべてのスレッドは UPSI スイッチの単一コピーを共用します。スレッド化アプリケーションでスイッチを変更する場合には、適切なシリアライゼーション・ロジックをコーディングする必要があります。

関連タスク

515 ページの『再帰呼び出しの実行』
554 ページの『マルチスレッド化によるファイル・アクセスのシリアライズ』

XL C/C++ プログラミング・ガイド (z/OS UNIX システム・サービス・アプリケーションでのスレッドの使用)

Language Environment Writing ILC Applications

第 5 部 XML と COBOL の連携

第 28 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、高速 XML パーサーとの間の COBOL 言語インターフェースです。次のように XMLPARSE コンパイラー・オプションを使用して、アプリケーションに適切なパーサーを選択します。

- XMLPARSE(XMLSS) は、z/OS XML System Services パーサーを選択します。このオプションによって、名前空間の処理やテキスト・フラグメントの国別文字表現 (Unicode UTF-16) への変換などの拡張機能が提供されます。
- XMLPARSE(COMPAT) は、COBOL ライブラリーに組み込まれた XML パーサーを選択します。このオプションによって、Enterprise COBOL バージョン 3 での XML 構文解析との互換性が得られます。

XML 入力を処理するには、XML パーサーとの間で制御を受け渡しする必要があります。このような制御の受け渡しを開始するには、 XML PARSE ステートメントを使用します。このステートメントでは、XML パーサーから制御を受け取り、パーサー・イベントを処理する処理プロシージャーを指定します。

処理プロシージャーで特殊レジスターを使用して、パーサーと情報を交換します。

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、文書および処理プロシージャーを識別します。
- XML PARSE ステートメントの ENCODING 句は、XML 文書のエンコードを指定します。
- 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。
- 以下の特殊レジスターは、情報の受け渡しを行います。
  - 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。
  - XML-NAMESPACE または XML-NNAMESPACE は、NAMESPACE-DECLARATION XML イベント、または名前空間のエレメント名または属性名の名前空間 ID を受け取ります。
  - XML-NAMESPACE-PREFIX または XML-NNAMESPACE-PREFIX は、NAMESPACE-DECLARATION XML イベント、または接頭部のエレメント名または属性名の名前空間接頭部を受け取ります。

XML 名前空間特殊レジスターは、処理プロシージャー外では未定義です。

XML-PARSE ステートメントの ENCODING 句および RETURNING NATIONAL 句は、XMLPARSE(XMLSS) コンパイラー・オプションが有効である場合にのみ使用できます。

リンク・エディットの考慮事項: XML PARSE ステートメントを含んだ COBOL プログラムは、AMODE 31 を使用してリンク・エディットする必要があります。

関連概念
『COBOL での XML パーサー』

関連タスク
563 ページの『XML 文書へのアクセス』
564 ページの『XML 文書の構文解析』
582 ページの『XML 文書のエンコード方式についての理解』
588 ページの『XML PARSE の例外処理』
592 ページの『XML 構文解析の終了』

関連参照
777 ページの『付録 D. XML 参照資料』
401 ページの『XMLPARSE』
Extensible Markup Language (XML)

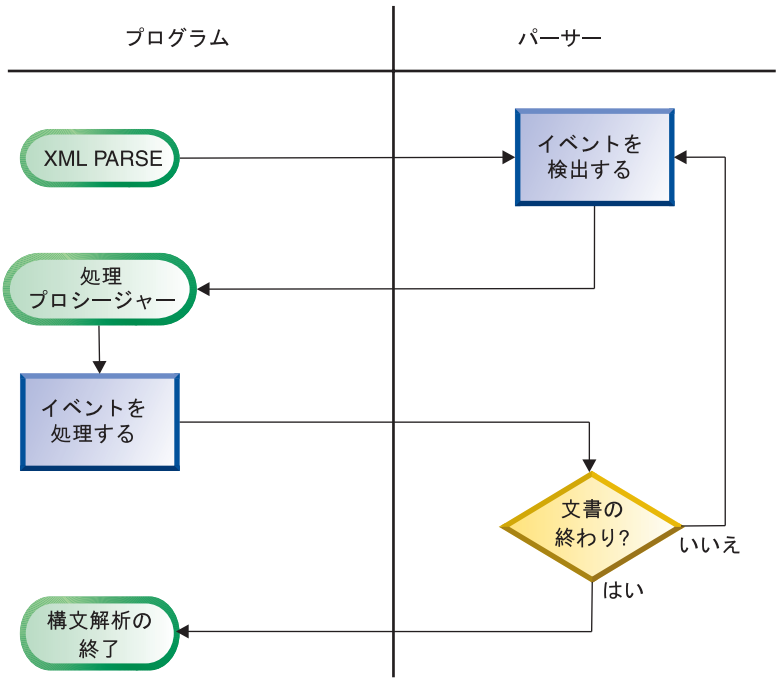
COBOL での XML パーサー

Enterprise COBOL 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 パーサーが文書内の (XML イベントに関連付けられた) 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。それぞれの XML イベントを処理するために独自のプロシージャーをコーディングします。この操作の間中、制御がパーサーとプロシージャーの間を行き来します。

パーサーとの受け渡しを開始するには、XML PARSE ステートメントを使用します。このステートメントに処理プロシージャーを指定します。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、文書内で検出した 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制御をパーサーに返します。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。 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。



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 パーサーが XML 文書を構文解析する際は、XML 文書のさまざまな側面が整形式になってい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。

関連タスク
『XML 文書へのアクセス』
564 ページの『XML 文書の構文解析』
566 ページの『XML を処理するためのプロシージャーの作成』
582 ページの『XML 文書のエンコード方式についての理解』
588 ページの『XML PARSE の例外処理』
592 ページの『XML 構文解析の終了』

関連参照
XML 仕様

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。文書を獲得する一般的方法は、WebSphere MQ メッセージ、CICS 一時キューまたは通信域、あるいは IMS メッセージ処理キューから検索するという方法です。

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。

XMLPARSE(XMLSS) オプションが有効である場合には、テキストを 1 レコード (またはセグメント) ずつパーサーに渡すことによって、 XML 文書をファイルから構文解析することができます。この機能は、非常に大きな XML 文書やデータ・セット内にある XML 文書を構文解析する場合に便利です。

関連タスク
454 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』
481 ページの『第 22 章 COBOL プログラムの開発 (IMS の場合)』

関連参照
572 ページの『XML 文書を 1 セグメントずつ構文解析』
401 ページの『XMLPARSE』

XML 文書の構文解析

XML 文書を構文解析するには、XML PARSE ステートメントを使用して、構文解析する XML 文書、構文解析時に発生する XML イベントの処理用プロシージャーを指定します。また、次のコードの断片で示しているように、ON EXCEPTION 句をコーディングすることによって、構文解析の終了後にアクションを行うようにすることもできます。

```
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
```

XML PARSE ステートメントで、まず XML 文書文字ストリームを含む*構文データ項目* (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、 国別カテゴリーの基本データ項目または国別グループ項目としてデータ項目を定義します。それ以外の場合には、英数字グループ項目または基本英数字データ項目として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は、XML 文書は、Unicode UTF-16BE、CCSID 1200 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 文書のコード化文字セットに関する、下記の関連参照にリストされている、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。コード・ページの詳細については、XML 文書のエンコードの理解に関する下記の関連参照を参照してください。

次に、文書から検出した XML イベントを処理するプロシージャーの名前を指定します (上の例では XMLEVENT-HANDLER)。

XMLPARSE(XMLSS) コンパイラー・オプションが有効である場合には、 XMLPARSE ステートメントの ENCODING 句を使用して、文書の CCSID を指定することができます。また、RETURNING NATIONAL 句を使用して、パーサーに UTF-8 または 1 バイト文字を国別文字に自動変換させて、処理プロシージャーに返すこともできます。

さらに、構文解析の終了時に制御を受け取る以下の句のいずれかまたは両方を指定できます。

- 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
- XMLPARSE(XMLSS) が有効である場合には、パーサーは文書内のエラーを検出して、(例外の種類に関係なく) EXCEPTION イベントをシグナル通知します。
- XMLPARSE(XMLSS) が有効である場合には、パーサーは END-OF-INPUT イベントをシグナル通知します。処理プロシージャーは、まだゼロに設定された特殊レジスター XML-CODE をパーサーに返し、これ以上 XML データがパーサーに渡されないことを示します。
- XMLPARSE(COMPAT) が有効である場合には、パーサーはエンコード競合 EXCEPTION イベントをシグナル通知します。処理プロシージャーは、パーサーに戻るまで、特殊レジスター XML-CODE をゼロまたは正しい CCSID に初期化しません。
- XMLPARSE(COMPAT) が有効である場合には、パーサーは文書内のエラーを検出し、EXCEPTION イベント (エンコード競合以外) をシグナル通知します。処理プロシージャーは、パーサーに戻るまで、特殊レジスター XML-CODE をゼロに初期化しません。
- パーサーに戻る前に、XML-CODE 特殊レジスターを -1 に設定して、構文解析プロセスを故意に終了した場合。

特殊レジスター: XML-EVENT 特殊レジスターを使用して、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。パーサーは、特殊レジスター XML-TEXT または XML-NTEXT に入っているイベントの内容を渡します。また、XMLPARSE(XMLSS) オプションが有効である場合には、パーサーは特殊レジスター XML-NAMESPACE または XML-NNAMESPACE を XML イベントに関連した名前空間 ID に設定します (存在する場合)。パーサーは XML-NAMESPACE-PREFIX または XML-NNAMESPACE-PREFIX 特殊レジスターを関連接頭部に設定します。

関連概念

568 ページの『XML-CODE』

関連タスク
582 ページの『XML 文書のエンコード方式についての理解』
『XML を処理するためのプロシージャーの作成』

関連参照
584 ページの『XML 文書のコード化文字セット』
568 ページの『XML-EVENT』
401 ページの『XMLPARSE』
XML PARSE ステートメント (*Enterprise COBOL 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、次の表に示す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。

これらの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。

表 68. **XML パーサーが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]	可変長基本カテゴリー英数字項目。サイズ制限 134,180,862 バイト。	XML PARSE ID[3] 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。サイズ制限 67,090,431 国別文字 (134,180,862 バイト)	XML PARSE ID[3] 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)
XML-NAMESPACE[1, 4]	可変長基本カテゴリー英数字項目。サイズ制限 32,768 バイト。	XML 文書が英数字データ項目である場合に、NAMESPACE-DECLARATION XML イベントまたは名前空間に存在するエレメントまたは属性名の名前空間 ID。[3]
XML-NNAMESPACE[1, 4]	可変長基本カテゴリー国別項目。サイズ制限 16,384 国別文字 (32,768 バイト)	XML 文書が国別データ項目である場合、または RETURNING NATIONAL 句が XML PARSE ステートメントで指定されている場合の、NAMESPACE-DECLARATION XML イベントまたは名前空間に存在するエレメントまたは属性名の名前空間 ID。
XML-NAMESPACE-PREFIX[1, 4]	可変長基本カテゴリー国別項目。サイズ制限 4,096 バイト	XML 文書が英数字データ項目である場合の、NAMESPACE-DECLARATION XML イベント、またはデフォルトではない名前空間に存在するエレメントまたは属性名の接頭部 (存在する場合)。[3]

表 68. **XML パーサーが使用する特殊レジスター** *(続き)*

特殊レジスター	暗黙的な定義および使用法	内容
XML-NNAMESPACE-PREFIX[1, 4]	可変長基本カテゴリー国別項目。サイズ制限 2,048 国別文字 (4,096 バイト)	XML 文書が国別データ項目である場合、または RETURNING NATIONAL 句が XML PARSE ステートメントに指定されている場合の、NAMESPACE-DECLARATION XML イベント、またはデフォルトではない名前空間に存在するエレメントまたは属性名の接頭部 (存在する場合)。

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で XML GENERATE ステートメントをコーディングする場合、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. 英数字データ項目で、RETURNING NATIONAL 句を XML PARSE ステートメントに指定した場合、テキストは対応する国別特殊レジスターで返されます。 RETURNING NATIONAL 句は、XMLPARSE(XMLSS) オプションが有効である場合にのみ指定できます。
4. パーサーは、XMLPARSE(XMLSS) オプションが有効である場合に、名前空間特殊レジスターを設定します。

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。ただし、INVOKE または CALL ステートメントを使用して、処理プロシージャーからメソッドまたは最外部のプログラムに制御が渡る場合、ターゲットとなるメソッドまたは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 INVOKE または CALL ステートメントで渡されたメソッドまたは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。

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。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。

575 ページの『例: XML の処理用プログラム』

関連概念

568 ページの『XML-CODE』
569 ページの『XML-TEXT および XML-NTEXT』
570 ページの『XML-NAMESPACE および XML-NNAMESPACE』
570 ページの『XML-NAMESPACE-PREFIX および XML-NNAMESPACE-PREFIX』

関連タスク

571 ページの『XML テキストの COBOL データ項目への変換』
147 ページの『国別 (Unicode) 表現との間の変換』

関連参照

568 ページの『XML-EVENT』
777 ページの『継続を許可する XML PARSE 例外』
782 ページの『継続を許可しない XML PARSE 例外』

401 ページの『XMLPARSE』
XML PARSE ステートメント (*Enterprise COBOL 言語解説書*)

XML-EVENT

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ます。パーサーはその 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、パーサーはイベントに関する追加情報が入った他の特殊レジスターを渡します。ほとんどの場合、パーサーは XML-TEXT または XML-NTEXT 特殊レジスターを、イベントを引き起こした XML フラグメントに設定します。

XMLPARSE(COMPAT) オプションが有効であれば、XML 文書が国別データ項目である場合、またはパーサーが Unicode 文字参照を検出した場合には、パーサーは XML-NTEXT を設定します。それ以外の場合には、パーサーは XML-TEXT を設定します。

XMLPARSE(XMLSS) オプションが有効であれば、RETURNING NATIONAL 句が XML PARSE ステートメントに指定されている場合、または XML 文書が国別データ項目である場合には、パーサーは XML-NTEXTを設定します。それ以外の場合には、パーサーは XML-TEXT を設定します。

XMLPARSE(XMLSS) コンパイラー・オプションが有効である場合には、パーサーは NAMESPACE-DECLARATION イベントで、名前空間に存在する名前を検出した場合に、名前空間特殊レジスターを設定します。

すべての XML イベントの詳細説明については、下記の関連参照の XML-EVENT を参照してください。

エンコード競合などの場合には、パーサーは、XML-CODE 特殊レジスターでイベントに関する情報を提供します。

574 ページの『例: 単純な文書の構文解析』

関連タスク
564 ページの『XML 文書の構文解析』

関連参照 401 ページの『XMLPARSE』 (コンパイラー・オプション)
XML-EVENT (*Enterprise COBOL 言語解説書*)

XML-CODE

パーサーから XML PARSE ステートメントに制御が戻されるとき、特殊レジスター XML-CODE には、パーサー (または XML-CODE を -1 に設定した場合には処理プロシージャー) によって設定された最新の値が入っています。

EXCEPTION イベント以外のすべてのイベントにおいて、XML-CODE の値はゼロです。処理プロシージャーが、 EXCEPTION 以外のイベントで制御をパーサーに返す前に XML-CODE を -1 に設定した場合、処理はユーザーが開始した COBOL 例外条件で停止します。

EXCEPTION イベントの場合、特殊レジスター XML-CODE は例外コードに設定されます。

XMLPARSE(COMPAT) が有効である場合のエンコード競合例外では、処理プロシージャーは、パーサーに返す前に XML-CODE を有効な値に初期化することがあります。XML-CODE を他のゼロ以外の値に初期化した場合、または他の例外のために初期化した場合には、パーサーは XML-CODE を元の例外コードに設定します。

関連タスク
566 ページの『XML を処理するためのプロシージャーの作成』
588 ページの『XML PARSE の例外処理』

関連参照
777 ページの『付録 D. XML 参照資料』
XML-CODE (*Enterprise COBOL 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。次の場合にパーサーは XML-NTEXT を設定します。

- XML 文書が国別データ項目である場合
- XMLPARSE(XMLSS) オプションが有効で、RETURNING NATIONAL 句が XML PARSE ステートメントで指定されている場合
- XMLPARSE(COMPAT) オプションが有効で、 ATTRIBUTE-NATIONAL-CHARACTER または CONTENT-NATIONAL-CHARACTER イベントが発生した場合

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*国別文字*の数を決定するには、LENGTH 組み込み関数 (例: 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。国別文字数は、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 (両方ともバイト数を返します)。

関連概念
568 ページの『XML-CODE』
568 ページの『XML-EVENT』

関連タスク
566 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 401 ページの『XMLPARSE』

XML-NAMESPACE および XML-NNAMESPACE

XMLPARSE(XMLSS) オプションが有効である場合には、XML パーサーは XML-NAMESPACE 特殊レジスターまたは XML-NNAMESPACE 特殊レジスターを次のものの名前空間 ID に設定します。

- NAMESPACE-DECLARATION XML イベント
- 名前空間内のエレメント名または属性名

パーサーは、XML 文書が国別データ項目である場合、または RETURNING NATIONAL 句が XML PARSE ステートメントに指定されている場合には、 XML-NNAMESPACE を設定します。それ以外の場合には、パーサーは、XML-NAMESPACE を設定します。

特殊レジスター XML-NAMESPACE と XML-NNAMESPACE は、相互に排他的です。パーサーが XML-NAMESPACE を設定した場合、XML-NNAMESPACE は空で長さゼロになります。パーサーが XML-NNAMESPACE を設定した場合、XML-NAMESPACE は空で長さゼロになります。

XML-NNAMESPACE 内の*国別文字*の数を決定するには、LENGTH 組み込み関数 (例: LENGTH(XML-NNAMESPACE)) を使用します。 XML-NNAMESPACE 内のバイト数を決定するには、特殊レジスター LENGTH OF XML-NNAMESPACE を使用します。国別文字数は、バイト数とは異なります。

XML-NAMESPACE 内のバイト数を決定するには、特殊レジスター LENGTH OF XML-NAMESPACE または LENGTH 組み込み関数を使用します (両方ともバイト数を返します)。

関連概念
568 ページの『XML-CODE』
『XML-NAMESPACE-PREFIX および XML-NNAMESPACE-PREFIX』
569 ページの『XML-TEXT および XML-NTEXT』

関連タスク
566 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
401 ページの『XMLPARSE』
XML-EVENT (*Enterprise COBOL 言語解説書*)

XML-NAMESPACE-PREFIX および XML-NNAMESPACE-PREFIX

XMLPARSE(XMLSS) オプションが有効である場合には、XML パーサーは、次の場合に XML-NAMESPACE-PREFIX 特殊レジスターまたは XML-NNAMESPACE-PREFIX 特殊レジスターを設定します。

- 名前空間接頭部も定義する NAMESPACE-DECLARATION XML イベントの場合
- 名前空間のエレメント名または属性名が接頭部を持つ場合

パーサーは、XML 文書が国別データ項目である場合、または RETURNING NATIONAL 句が XML PARSE ステートメントに指定されている場合には、 XML-NNAMESPACE-PREFIX を設定します。それ以外の場合には、パーサーは、XML-NAMESPACE-PREFIX を設定します。

特殊レジスター XML-NAMESPACE-PREFIX と XML-NNAMESPACE-PREFIX は、相互に排他的です。パーサーが XML-NAMESPACE-PREFIX を設定した場合、XML-NNAMESPACE-PREFIX は空で長さゼロになります。パーサーが XML-NNAMESPACE-PREFIX を設定した場合、XML-NAMESPACE-PREFIX は空で長さゼロになります。

XML-NNAMESPACE-PREFIX 内の*国別文字*の数を決定するには、LENGTH 組み込み関数 (例: LENGTH(XML-NNAMESPACE-PREFIX)) を使用します。 XML-NNAMESPACE-PREFIX 内のバイト数を決定するには、特殊レジスター LENGTH OF XML-NNAMESPACE-PREFIX を使用します。国別文字数は、バイト数とは異なります。

XML-NAMESPACE-PREFIX 内のバイト数を決定するには、特殊レジスター LENGTH OF XML-NAMESPACE-PREFIX または LENGTH 組み込み関数を使用します (両方ともバイト数を返します)。

関連概念
568 ページの『XML-CODE』
570 ページの『XML-NAMESPACE および XML-NNAMESPACE』
569 ページの『XML-TEXT および XML-NTEXT』

関連タスク
566 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
401 ページの『XMLPARSE』
XML-EVENT (*Enterprise COBOL 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。右端に配置する場合は、COBOL 項目の宣言に 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 では同じものを意味しますが、COBOL 送信フィールドとしてはまったく別の宣言となる必要があります。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関数を使用すると、高度な柔軟性が簡単に実現できます。
  - 組み込み関数 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。
  - 組み込み関数 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
139 ページの『COBOL での国別データ (Unicode) の使用』
566 ページの『XML を処理するためのプロシージャーの作成』

XML 文書を 1 セグメントずつ構文解析

パーサーに XML テキストを 1 セグメントずつ渡すことによって、XML 文書を構文解析することができます。この手法を適用するのは主に次の 2 とおりの場合です。

- 非常に大きな文書を処理する場合
- データ・セット内にある XML 文書を 1 レコードずつ処理する場合

この機能を使用するには、XMLPARSE(XMLSS)コンパイラー・オプションを有効にしてプログラムをコンパイルする必要があります。

1 セグメントずつ XML 文書を構文解析するには、*構文解析データ項目*を XML 文書の最初のセグメントに初期化してから、XML PARSE ステートメントを実行します。パーサーは通常の場合と同様に、XML テキストを処理してから、XML イベントを処理プロシージャーに返します。テキスト・セグメントの最後で、パーサーは、XML-CODE をゼロに設定して、END-OF-INPUT XML イベントをシグナル通知します。処理対象の文書のセグメントがまだ存在する場合には、処理プロシージャーで XML データの次のセグメントを構文解析データ項目に移動し、XML-CODE を 1 に設定してから、パーサーに戻ります。 XML セグメントの最後をパーサーにシグナル通知するには、XML-CODE をゼロに設定したままにしてパーサーに戻ります。

構文解析データ項目の長さはセグメントごとに評価され、これによってセグメント長が決定されます。

推奨: XML 文書セグメントが可変長である場合、構文解析データ項目には可変長項目を指定します。たとえば、可変長の XML セグメントの場合、構文解析データ項目には次のようなものが考えられます。

- OCCURS DEPENDING ON 節を含んだ、可変長グループ項目
- 参照変更された項目
- FD レコード (ここで、FD は、RECORD IS VARYING DEPENDING ON 節を指定し、依存データ項目が FD レコードの参照修飾子または ODO オブジェクトで長さとして使用されます)

XML 文書をパーサーに複数のセグメントで送信した場合、文書コンテンツは、単一イベントで 1 つの大きなフラグメントとしてではなく、複数のイベントで複数のフラグメントとして、処理プロシージャーに返されることがあります。

たとえば、コンテンツ文字のストリングの真ん中を分割点として 2 つのセグメントに文書を分割した場合、パーサーは 2 つの別個の CONTENT-CHARACTERS イベントでコンテンツを返します。処理プロシージャーは、必要に応じてアプリケーションでコンテンツのストリングを再組み立てする必要があります。開始エレメント・タグ、属性名、名前空間宣言、および終了エレメント・タグは、文書の 2 セグメントに分割された場合でも常に、単一イベントで処理プロシージャーに送信されます。

セグメントの分割がマルチバイト文字のバイトを横断して行われた場合、パーサーは、単一イベントで送信するために、分割を検出して文字を再組み立てします。

QSAM または VSAM ファイルに保管された XML 文書は、次のように処理することができます。

1. ファイルを開いて、XML 文書の最初のレコードを読み取る。
2. FD レコードを *identifier-1* として、XML PARSE ステートメントを実行する。
3. END-OF-INPUT イベントを処理するための処理プロシージャー・ロジックで、XML 文書の次のレコードを読み取って、*identifier-1* に格納する。ファイルの終わり (ファイル状況コード 10) ではない場合、XML-CODE を 1 に設定して、パーサーに戻る。ファイルの終わりである場合、XML-CODE をゼロに設定したままでパーサーに戻る。
4. END-OF-DOCUMENT イベントの処理プロシージャー・ロジックで、ファイルを閉じる。

使用上の注意: XML 文書のルート・エレメントの後には各種情報 (任意の順序の 0 個以上のコメントまたは処理命令) が続く可能性があります。ただし、1 セグメントずつ文書を構文解析した場合、パーサーは、セグメントの最後の項目が不完全である場合にのみ、ルート・エレメントの終了タグの処理後に、END-OF-INPUT XML イベントをシグナル通知します。セグメントが完全な XML 項目 (ルート・エレメント終了タグ、またはその後ろに完全なコメントまたは処理命令がある場合など) で終了している場合、項目自体のイベントの後に発生する、次の XML イベントは、END-OF-DOCUMENT XML イベントになります。

推奨: ルート・エレメントの最後に続けて XML データのセグメントを提供するには、各セグメントの最後に、少なくとも最初がスペースではない文字の XML 項目を組み込んでください。パーサーが処理する最後のセグメントにのみ、完全な項目を組み込んでください。たとえば、次の例 (各行が XML 文書の 1 セグメント) では、テキスト This comment ends this segment が入っているセグメントが構文解析される最後のセグメントです。

```
<Tagline>
COBOL is the language of the future!
</Tagline> <
!--First comment--
> <?pi data?> <!-
-This comment ends this segment-->
<!-- This segment is not included in the parse-->
```

580 ページの『例: XML 文書を 1 セグメントずつ構文解析』

関連参照
401 ページの『XMLPARSE』 (コンパイラー・オプション)
568 ページの『XML-EVENT』
XML-EVENT (*Enterprise COBOL 言語解説書*)

XML PARSE の例

次の例では、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。

以下の例を使用して、XML PARSE の基本的な使用方法、および次のような XMLPARSE(XMLSS) の特殊な使用方法を理解してください。

- 名前空間を含んだ文書の構文解析
- 文書を 1 セグメントずつ構文解析

『例: 単純な文書の構文解析』
575 ページの『例: XML の処理用プログラム』
578 ページの『例: 名前空間を使用する XML 文書の構文解析』
580 ページの『例: XML 文書を 1 セグメントずつ構文解析』

例: 単純な文書の構文解析

この例では、基本的な XML 文書の構文解析におけるイベントのフローおよび特殊レジスター XML-TEXT の関連コンテンツを説明します。

COBOL プログラムには次のようなデータ項目 Doc の基本的な 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。結果を下表に示します。

表 69. **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	タイプ
ATTRIBUTE-CHARACTERS	short
CONTENT-CHARACTERS	Hello, World!
END-OF-ELEMENT	msg

表 69. **XML イベントおよび特殊レジスター** *(続き)*

XML-EVENT	XML-TEXT
END-OF-DOCUMENT	

関連概念
569 ページの『XML-TEXT および XML-NTEXT』

関連参照 568 ページの『XML-EVENT』
401 ページの『XMLPARSE』 (コンパイラー・オプション)
XML PARSE (*Enterprise COBOL 言語解説書*)

例: XML の処理用プログラム

以下の例では、XML PARSE ステートメントおよび処理プロシージャーの使用方法を示します。

この XML 文書は、構文解析のフローを確認する目的のソースです。
XMLPARSE(XMLSS) コンパイラー・オプションおよび XMLPARSE(COMPAT) コンパイラー・オプションの両方のプログラムの出力を以下に示します。 XML 文書とプログラムの出力結果を比較して、パーサーと処理プロシージャーとの間の相互作用を確認し、イベントと文書フラグメントを突き合わせてください。

```
 cbl codepage(1047)
 Identification division.
   Program-id. XMLSAMPL.

 Data division.
  Working-storage section.
*****************************************************************
* XML document, encoded as initial values of data items.        *
*****************************************************************
   1 xml-document.
    2 pic x(39) value '<?xml version="1.0" encoding="IBM-1047"'.
    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(35) value '  <bread type="baker&apos;s best"/>'.
    2 pic x(41) value '  <?spread please use real mayonnaise  ?>'.
    2 pic x(31) value '  <meat>Ham &amp; turkey</meat>'.
    2 pic x(40) value '  <filling>Cheese, lettuce, tomato, etc.'.
    2 pic x(10) value '</filling>'.
    2 pic x(35) value '  <![CDATA[We should add a <relish>'.
    2 pic x(22) value ' element in future!]]>'.
    2 pic x(31) value '  <listprice>$4.99 </listprice>'.
    2 pic x(27) value '  <discount>0.10</discount>'.
    2 pic x(11) value '</sandwich>'.
   1 xml-document-length computational pic 999.

*****************************************************************
* Sample data definitions for processing numeric XML content.   *
*****************************************************************
   1 current-element pic x(30).
   1 xfr-ed pic x(9) justified.
   1 xfr-ed-1 redefines xfr-ed pic 999999.99.
   1 list-price computational pic 9v99 value 0.
   1 discount computational pic 9v99 value 0.
   1 display-price pic $$9.99.

 Procedure division.
  Mainline section.
```

```
     XML parse xml-document processing procedure xml-handler
       On exception
         Display 'XML document error ' XML-Code
       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 '}'
         Move XML-Text to current-element
       When 'CONTENT-CHARACTERS'
         Display 'Content characters: {' XML-Text '}'
* ==> Transform XML content to operational COBOL data item...
         evaluate current-element
           When 'listprice'
* ==> Using function NUMVAL-C...
             Compute list-price = function numval-c(XML-Text)
           When 'discount'
* ==> Using de-editing of a numeric edited item...
             Move XML-Text to xfr-ed
             Move xfr-ed-1 to discount
         End-evaluate
       When 'END-OF-ELEMENT'
         Display 'End element tag: {' XML-Text '}'
         Move spaces to current-element
       When 'START-OF-DOCUMENT'
         Display 'Start of document'
       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: {' XML-Text '}'
       When 'END-OF-CDATA-SECTION'
         Display 'End of CData: {' XML-Text '}'
       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 xml-document-length = function length (XML-Text)
        Display 'Exception ' XML-Code ' at offset '
            xml-document-length '.'
      When other
        Display 'Unexpected XML event: ' XML-Event '.'
    End-evaluate
    .
End program XMLSAMPL.
```

XMLPARSE(XMLSS) を使用した構文解析の出力例:

以下の出力結果では、構文解析の各イベントが、どの文書フラグメントから発生しているかを確認することができます。

```
Start of document
Version: {1.0}
Encoding: {IBM-1047}
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'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 & 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: {}
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!
```

XMLPARSE(COMPAT) を使用した構文解析の出力例:

以下の出力結果では、構文解析の各イベントが、どの文書フラグメントから発生しているかを確認することができます。

```
Start of document
Version: {1.0}
Encoding: {IBM-1047}
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!
```

例: 名前空間を使用する XML 文書の構文解析

この例では、XMPARSE(XMLSS) オプションが有効である場合に使用できる XML 構文解析の機能を説明します。

以下の例では、名前空間 ID および名前空間接頭部を使用して、エレメント名および属性名を修飾しています。この修飾によって、同じ名前を複数のコンテキストで使用することが可能になります。author の title (作者の称号: Mr) および book の title (本のタイトル: *Writing COBOL for Fun and Profit*) として title を使用していることに注目してください。

表 70 は、関連 XML 特殊レジスターのコンテンツとともに、処理プロシージャーがパーサーから受け取る一連のイベントのリストです。

サンプル XML 文書には、複数の名前空間宣言 (デフォルト名前空間、および接頭部 (bk、pi、およびisbn) がある 3 つの名前空間 ID) が含まれています。エレメント ″comment″ にはデフォルト名前空間が空ストリングに設定されていること (xmlns='') に注目してください。これによってデフォルト名前空間が*非宣言化*され、その結果、デフォルト名前空間が存在しないようになります。

サンプル XML 文書

```
<section
 xmlns="http://www.ibm.com/events"
 xmlns:bk="urn:loc.gov:books"
 xmlns:pi="urn:personalInformation"
 xmlns:isbn='urn:ISBN:0-395-36341-6'>
  <title>Book-Signing Event</title>
  <signing>
    <bk:author pi:title="Mr" pi:name="Tom Ross"/>
    <book bk:title="Writing COBOL for Fun and Profit" isbn:number="0426070806"/>
    <comment xmlns=''>What a great issue!</comment>
  </signing>
  </section>
```

XML PARSE の結果

表 70. **XML イベントおよび特殊レジスター**

XML-EVENT	XML-TEXT	XML-NAMESPACE-PREFIX	XML-NAMESPACE
START-OF-DOCUMENT			
START-OF-ELEMENT	セクション		http://www.ibm.com/events
NAMESPACE-DECLARATION			http://www.ibm.com/events
NAMESPACE-DECLARATION		bk	urn:loc.gov:books
NAMESPACE-DECLARATION		pi	urn:personalInformation
NAMESPACE-DECLARATION		isbn	urn:ISBN:0-395-36341-6
START-OF-ELEMENT	title		http://www.ibm.com/events
CONTENT-CHARACTERS	Book-Signing Event		
END-OF-ELEMENT	title		http://www.ibm.com/events
START-OF-ELEMENT	signing		http://www.ibm.com/events
START-OF-ELEMENT	author	bk	urn:loc.gov:books
ATTRIBUTE-NAME	title	pi	urn:personalInformation
ATTRIBUTE-CHARACTERS	Mr		
ATTRIBUTE-NAME	name	pi	urn:personalInformation
ATTRIBUTE-CHARACTERS	Tom Ross		
END-OF-ELEMENT	author	bk	urn:loc.gov:books
START-OF-ELEMENT	book		http://www.ibm.com/events
ATTRIBUTE-NAME	title	bk	urn:loc.gov:books
ATTRIBUTE-CHARACTERS	Writing COBOL for Fun and Profit		
ATTRIBUTE-NAME	number	isbn	urn:ISBN:0-395-36341-6

表 70. **XML イベントおよび特殊レジスター** *(続き)*

XML-EVENT	XML-TEXT	XML-NAMESPACE-PREFIX	XML-NAMESPACE
ATTRIBUTE-CHARACTERS	0426070806		
END-OF-ELEMENT	book		http://www.ibm.com/events
START-OF-ELEMENT	comment		
NAMESPACE-DECLARATION			
CONTENT-CHARACTERS	What a great issue!		
END-OF-ELEMENT	comment		
END-OF-ELEMENT	signing		http://www.ibm.com/events
END-OF-ELEMENT	セクション		http://www.ibm.com/events
END-OF-DOCUMENT			

関連概念
569 ページの『XML-TEXT および XML-NTEXT』
570 ページの『XML-NAMESPACE および XML-NNAMESPACE』
570 ページの『XML-NAMESPACE-PREFIX および XML-NNAMESPACE-PREFIX』

関連参照
568 ページの『XML-EVENT』
401 ページの『XMLPARSE』 (コンパイラー・オプション)
XML-EVENT (*Enterprise COBOL 言語解説書*)

例: XML 文書を 1 セグメントずつ構文解析

下のサンプル・プログラムでは、XML 文書の 1 セグメントずつの構文解析を示します。例では、ファイルの XML コンテンツ、XML テキストを読み取りパーサーに送信するプログラム、および入力レコードを構文解析した結果の一連のイベントが示されています。

この機能を使用するには、XMLPARSE(XMLSS)コンパイラー・オプションを有効にしてプログラムをコンパイルする必要があります。

サンプル・プログラムでは、XML 文書のレコード (セグメント) をファイル (INFILE) から読み取り、そのレコードをパーサーに XML PARSE ステートメントを使用して渡しています。パーサーは、XML イベントごとに XML を処理して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がイベントを処理し、パーサーに返します。

セグメントの最後で、パーサーは XML-EVENT を END-OF-INPUT に設定し、XML-CODE をゼロに設定し、制御を処理プロシージャーに渡します。処理プロシージャーは次の XML レコード読み取って構文解析データ項目に入れ、XML-CODE を 1 に設定してパーサーに返します。

構文解析結果の表示では、処理プロシージャーは入力の各レコードを表示し、その後に一連の XML イベントおよび XML-TEXT 内の関連テキスト・フラグメントを表示します。 XML-TEXT のコンテンツは中括弧 ({}) で表示されます。空の中括弧は、XML-TEXT が空であることを示します。 (イベント番号 08 の余分なゼロ長

CONTENT-CHARACTERS XML イベントに注意してください。 XML テキストを細切れに入力するとこのような例外が発生することがよくあります。)

このように、処理プロシージャーとパーサーは、READ ステートメントがファイル終了状況コードを返すまで、交互に処理を行います。処理プロシージャーは、セグメント処理の最後を示す、まだゼロに設定された XML-CODE をパーサーに返します。

INFILE:

```
<?xml version='1.0'?>
<Tagline>
COBOL is the language of the future!
</Tagline>
```

プログラム:

```
  Identification division.
  Program-id. PARSESEG.
  Environment division.
  Input-output section.
  File-control.
      Select Input-XML
       Assign to infile
       File status is Input-XML-status.
  Data division.
  File section.
  FD Input-XML
      Record is varying from 1 to 255 depending on Rec-length
      Recording mode V.
  1 fdrec.
    2 pic X occurs 1 to 255 depending on Rec-length .
  Working-storage section.
  1 Event-number comp pic 99.
  1 Rec-length comp-5 pic 9(4).
  1 Input-XML-status pic 99.
  Procedure division.
      Open input Input-XML
      If Input-XML-status not = 0
        Display 'Open failed, file status: '  Input-XML-status
        Goback
      End-if
      Read Input-XML
      If Input-XML-status not = 0
        Display 'Read failed, file status: '  Input-XML-status
        Goback
      End-if
      Move 0 to Event-number
      Display 'Starting with: ' fdrec
      Display 'Event number and name    Content of XML-text'
      XML parse fdrec processing procedure Handle-parse-events
      Close Input-XML
      Goback
      .
  Handle-parse-events.
      Add 1 to Event-number
      Display '  ' Event-number ': ' XML-event '{' XML-text '}'
      Evaluate XML-event
        When 'END-OF-INPUT'
          Read Input-XML
          Evaluate Input-XML-status
            When 0
              Move 1 to XML-code
              Display 'Continuing with: ' fdrec
            When 10
              Display 'At EOF; no more input.'
```

```
                When other
                  Display 'Read failed, file status:' Input-XML-status
                  Goback
              End-evaluate
            When other
              Continue
          End-evaluate
               .
  End program PARSESEG.
```

結果:

```
Starting with:   <?xml version='1.0'?>
Event number and name      Content of {XML-TEXT}
  01: START-OF-DOCUMENT      {}
  02: VERSION-INFORMATION    {1.0}
  03: END-OF-INPUT           {}
Continuing with:    <Tagline>
  04: START-OF-ELEMENT       {Tagline}
  05: END-OF-INPUT           {}
Continuing with:    COBOL is the language of the future!
  06: CONTENT-CHARACTERS     {COBOL is the language of the future!}
  07: END-OF-INPUT           {}
Continuing with:    </Tagline>
  08: CONTENT-CHARACTERS     {}
  09: END-OF-ELEMENT         {Tagline}
  10: END-OF-DOCUMENT        {}
```

XML 文書のエンコード方式についての理解

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を使用してエンコードされている必要があります。特定の構文解析でサポートされているエンコードは、次の点に依存します。

- XML 文書が入っているデータ項目のカテゴリー
- XMLPARSE コンパイラー・オプションの設定
- XML PARSE ステートメントに指定されたオプションの句。

国別データ項目に入っている XML 文書の場合、サポートされているコード・ページは、Unicode UTF-16BE (ビッグ・エンディアン)、CCSID 1200 です。

英数字データ項目に入っている XML 文書の場合、XMLPARSE(XMLSS) コンパイラー・オプションが有効であるときにサポートされているコード・ページは次のとおりです。

- RETURNING NATIONAL 句が XML PARSE で指定されている場合: Unicode UTF-8、または z/OS Unicode Services で Unicode UTF-16 への変換がサポートされているすべての EBCDIC または ASCII コード・ページ。
- RETURNING NATIONAL 句が指定されていない場合: Unicode UTF-8、または XML 文書のコード化文字セットに関する関連参照でリストされている、すべての 1 バイト EBCDIC コード・ページ。

英数字データ項目に入っている XML 文書の場合、XMLPARSE(COMPAT) コンパイラー・オプションが有効であるときにサポートされているコード・ページは、XML 文書のコード化文字セットに関する関連参照で指定されています。

入力 XML 文書のエンコードの判別

パーサーは、XML 文書を正しく処理するために、そのエンコードを認識している必要があります。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。

次に示すとおり、XML 文書のエンコードの判別では、複数のエンコードに関する情報源を使用します。

- XMLPARSE(XMLSS) オプションが有効である場合:
  - XML 文書が入っているデータ項目のデータ・タイプ
  - XML PARSE ステートメントのオプションの ENCODING 句
  - CODEPAGE コンパイラー・オプションで指定された CCSID
- XMLPARSE(COMPAT) オプションが有効である場合:
  - XML 文書が入っているデータ項目のデータ・タイプ
  - XML 文書内で指定されているエンコード宣言
  - CODEPAGE コンパイラー・オプションで指定された CCSID
  - 文書の最初の数バイトを検査することによって判別した、XML 文書の実際のエンコード

XMLPARSE(XMLSS) オプションが有効である場合:

- XML 文書内で指定されているエンコード宣言は無視されます。
- 国別データ項目に入っている XML 文書の場合、XML PARSE ステートメントの ENCODING 句は、省略するか CCSID 1200 を指定する必要があります。 CODEPAGE コンパイラー・オプションで指定された CCSID は無視されます。実際の文書エンコードが Unicode UTF-16BE ではない場合、パーサーは、XML 例外イベントをシグナル通知します。
- 英数字データ項目に入っている XML 文書の場合、XML PARSE ステートメントの ENCODING 句で指定された CCSID は、CODEPAGE コンパイラー・オプションをオーバーライドします。
- XML PARSE ステートメントに ENCODING 句が含まれている場合、指定された CCSID は、 CODEPAGE コンパイラー・オプションで指定された CCSID をオーバーライドします。実際の文書エンコードが指定された CCSID と一致しない場合、パーサーは構文解析の開始時に XML 例外イベントを提起します。

関連タスク

586 ページの『コード・ページの指定』
585 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

401 ページの『XMLPARSE』
584 ページの『XML 文書のコード化文字セット』

XML 文書のコード化文字セット

XML 文書は、下記のサポートされたコード・ページのいずれかでエンコードする必要があります。

国別データ項目で生成または構文解析される XML 文書は、Unicode UTF-16、CCSID 1200 でエンコードする必要があります。

XML GENERATE ステートメントの場合、英数字データ項目で生成される文書は、Unicode UTF-8、CCSID 1208、または下表で示された 1 バイト EBCDIC 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。表で示された CCSID は、XML GENERATE ステートメントの ENCODING 句でコーディングできます。

XML PARSE ステートメントの場合、英数字データ項目の文書は次のコード・ページでエンコードする必要があります。

- XMLPARSE(XMLSS) の場合:
  - RETURNING NATIONAL 句が XML PARSE ステートメントで指定されている場合、z/OS Unicode サービスで Unicode UTF-16 への変換がサポートされている任意の EBCDIC または ASCII コード・ページ
  - RETURNING NATIONAL 句が XML PARSE ステートメントで指定されていない場合、 Unicode UTF-8、CCSID 1208、または下表で示されているいずれかの 1 バイト EBCDIC コード・ページ
- XMLPARSE(COMPAT) の場合: 下表で示されているいずれかの 1 バイト EBCDIC コード・ページ

XML PARSE ステートメントの ENCODING 句には、上述の XML PARSE の場合のように、サポートされた任意の CCSID をコーディングすることができます。

XML GENERATE および XML PARSE で使用する場合には、CODEPAGE コンパイラー・オプションには、この表で示された任意の CCSID (1208 を除く) をコーディングすることができます。

表 71. **XML 文書のコード化文字セット**

CCSID	説明
1208	Unicode UTF-8[1]
1047	Latin 1 / オープン・システム
1140, 37	USA、カナダ、. . . ユーロ国別拡張コード・ページ (ECECP)、国別拡張コード・ページ (CECP)
1141, 273	オーストリア、ドイツ ECECP、CECP
1142, 277	デンマーク、ノルウェー ECECP、CECP
1143, 278	フィンランド、スウェーデン ECECP、CECP
1144, 280	イタリア ECECP、CECP
1145, 284	スペイン、ラテンアメリカ (スペイン語圏) ECECP、CECP
1146, 285	英国 ECECP、CECP
1147, 297	フランス ECECP、CECP
1148, 500	国際 ECECP、CECP
1149, 871	アイスランド ECECP、CECP

表 71. **XML 文書のコード化文字セット** *(続き)*

CCSID	説明
1. Unicode UTF-8 (CCSID 1208) は、XMLPARSE(XMLSS) コンパイラー・オプションが有効である場合に、XML PARSE ステートメントでサポートされます。	

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 (Unicode 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。

関連タスク
147 ページの『国別 (Unicode) 表現との間の変換』
586 ページの『コード・ページの指定』

関連参照
350 ページの『CODEPAGE』

UTF-8 でエンコードされた XML 文書の構文解析

XMLPARSE(XMLSS) コンパイラー・オプションが有効である場合には、 UTF-8 でエンコードされた XML 文書を、一部の追加要件が適用される点を除いて、他の XML 文書を構文解析するのと同じように構文解析することができます。

UTF-8 でエンコードされた XML 文書を構文解析するには、次のコードの断片で示すように、XML PARSE ステートメントの ENCODING 句に CCSID 1208 を指定する必要があります。

```
XML PARSE xml-document
    WITH ENCODING 1208
    PROCESSING PROCEDURE xml-event-handler
    . . .
END-XML
```

WORKING-STORAGE または LOCAL-STORAGE に、英数字データ項目または英数字グループ項目として、xml-document を定義します。

デフォルトでは、パーサーは、英数字 XML 特殊レジスター XML-TEXT、XML-NAMESPACE、および XML-NAMESPACE-PREFIX で XML 文書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。

UTF-8 文書フラグメントの処理は、XML PARSE ステートメントで RETURNING NATIONAL 句を指定することによって信頼性を高めることができます。 RETURNING NATIONAL 句を使用すると、XML 文書フラグメントは、効率的に UTF-16 エンコードに変換され、国別特殊レジスター XML-NTEXT、XML-NNAMESPACE、および

XMLNNAMESPACE-PREFIX でアプリケーションに返されます。その後、国別データ項目の XML テキスト・フラグメントを効率的に処理することができます。 (国別データ項目の UTF-16 エンコードによって、COBOL での Unicode 処理が非常に簡素化します。)

次のコードの断片では、UTF-8 XML 文書の構文解析における ENCODING 句と RETURNING NATIONAL 句、両方の使用方法を説明します。

```
XML PARSE xml-document
    WITH ENCODING 1208  RETURNING NATIONAL
    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
```

関連参照 401 ページの『XMLPARSE』
569 ページの『XML-TEXT および XML-NTEXT』
570 ページの『XML-NAMESPACE および XML-NNAMESPACE』
570 ページの『XML-NAMESPACE-PREFIX および XML-NNAMESPACE-PREFIX』
XML PARSE ステートメント (*Enterprise COBOL 言語解説書*)

XML マークアップ内のコード・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 コード・ページ CCSID の特殊文字とそれぞれの 16 進値を示します。

表 72. **コード・ページ 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'

コード・ページの指定

英数字データ項目の XML 文書を構文解析するためのコード・ページを指定する方法として推奨されるのは、文書からエンコード宣言を省略し、次のエンコード指定に依存する方法です。

- XMLPARSE(XMLSS) オプションが有効である場合: XML PARSE ステートメントの ENCODING 句、または CODEPAGE コンパイラー・オプション
- XMLPARSE(COMPAT) オプションが有効である場合: CODEPAGE コンパイラー・オプション

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 文書のやりとりをする際に、伝送プロセスでやむなく発生する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なくなります。

サポートされているコード・ページの詳細については、XML 文書のエンコードと XML 文書のコード化文字セットの理解に関する関連参照を参照してください。

XMLPARSE(COMPAT) の場合:

推奨されてはいませんが、XML 宣言には、XML 文書に対するエンコード情報を指定することができます。大部分の XML 文書は XML 宣言で開始されます。 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。

エンコード宣言を含む XML 宣言の例を以下に示します。

```
<?xml version="1.0" encoding="ibm-1140" ?>
```

エンコード宣言は、以下のいずれかの方法で指定することができます。

- 先行ゼロ付きまたは先行ゼロなしの CCSID 番号に、オプションで接頭部として次のいずれかのストリング (大文字および小文字の混合は自由) を指定します。
  - IBM-
  - IBM_
  - CCSID-
  - CCSID_
- 下表に示されている別名のいずれかを使用します。別名は、大文字と小文字を混合させてコーディングできます。

表 73. **XML エンコード宣言の別名**

コード・ページ	サポートされる別名
037	EBCDIC-CP-US、EBCDIC-CP-CA、EBCDIC-CP-WT、EBCDIC-CP-NL
500	EBCDIC-CP-BE、EBCDIC-CP-CH
1200	UTF-16
1208	UTF-8

関連タスク

582 ページの『XML 文書のエンコード方式についての理解』
585 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

584 ページの『XML 文書のコード化文字セット』

XML PARSE の例外処理

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ます。特定の例外コードおよびそれに続いて実行できるアクションは、XMLPARSE コンパイラー・オプションの設定によって異なります。

XMLPARSE(XMLSS) の場合

パーサーが XML-CODE 特殊レジスターで返す例外コードは、z/OS XML System Services パーサーが生成する戻りコードと理由コードで構成されます。戻りコードと理由コードはそれぞれハーフワード・バイナリー値です。 XML-CODE の値はこれらの 2 つの値を連結したものです。たとえば、次の XML 文書は、エレメント終了タグ mmsg がエレメント開始タグ msg と一致していないため、正しい構成ではありません。

```
<msg>Hello</mmsg>
```

この文書の z/OS XML System Services パーサーからの戻りコードは、12 (16 進数の 000C) です。理由コードは、16 進数の 3035 (XRSN_ENDTAG_NAME_MISMATCH) です。この 2 つの値を連結した 16 進数の 000C3035 が、特殊レジスター XML-CODE で、処理プロシージャーに返されます。

パーサーの戻りコードおよび理由コードは、「*z/OS XML System Services Users Guide and Reference*」の付録で 16 進値で文書化されています。

処理プロシージャーでは、例外イベントを処理できず、構文解析を再開できません。処理プロシージャーが EXCEPTION イベントからパーサーに戻るとき、パーサーはイベントをそれ以上シグナル通知しません。パーサーは、XML PARSE ステートメントの ON EXCEPTION 句で指定したステートメントに制御を渡します。 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。XML-CODE には、パーサーが設定した元の例外コードが含まれます。

XMLPARSE(COMPAT) の場合

XML パーサーが XML-CODE に入れて渡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イベントを処理するには、次の手順に従います。

1. XML-CODE の内容を検査します。
2. 必要に応じて例外を処理します。
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 (エンコードの矛盾が検出されたことを示します) のみです。

- 50-99
- 100,001 から 165,535

XML-CODE に渡される例外コードが 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。)

この範囲の例外発生後の構文解析の終了時に、ON EXCEPTION 句に指定されたステートメントがあればこれに制御が渡され、そうでなければ、XML PARSE ステートメントの終わりに制御が渡されます。特殊レジスター XML-CODE には、パーサーが設定した最新の例外のコードが格納されます。

この他の例外では、パーサーは追加のイベントをシグナル通知せず、ON EXCEPTION 句に指定されているステートメントに制御を渡します。この場合、パーサーに制御を返す前に処理プロシージャーで XML-CODE を初期化したとしても、XML-CODE には元の例外番号が入ります。

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権を移動します。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 (構文解析の正常終了)。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渡されます。特殊レジスター XML-CODE はゼロに設定されます。

関連概念
『XML パーサーによるエラーの処理方法』
568 ページの『XML-CODE』

関連タスク
566 ページの『XML を処理するためのプロシージャーの作成』
582 ページの『XML 文書のエンコード方式についての理解』
591 ページの『コード・ページの矛盾の処理』

関連参照
777 ページの『継続を許可する XML PARSE 例外』
782 ページの『継続を許可しない XML PARSE 例外』
401 ページの『XMLPARSE』

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提供します。

- 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。

XMLPARSE(XMLSS) の場合、例外コードは下記の関連参照の z/OS XML System Services で記述されています。 XMLPARSE(COMPAT) の場合、例外コードは下記の関連参照の XML PARSE 例外で記述されています。

- XMLPARSE(COMPAT) が有効である場合には、 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。
- XMLPARSE(XMLSS) が有効である場合には、XML-TEXT または XML-NTEXT には、エラーまたは異常の検出ポイントまでの文書テキストが含まれています。 XML 文書を 1 セグメントずつ処理した場合には、該当する特殊レジスターには現在のセグメントのみが含まれます。

すべての他の XML 特殊レジスターは空で、長さゼロです。

XMLPARSE(XMLSS) の場合

XMLPARSE(XMLSS) が有効である場合には、パーサーに戻る前に XML-CODE をゼロに設定しても、例外発生後に構文解析を続行することはできません。処理プロシージャーからパーサーに戻ると、パーサーは、XML PARSE ステートメントの ON EXCEPTION 句が指定されている場合はその句に制御を渡します。指定されていない場合には、パーサーは XML PARSE ステートメントの最後に制御を渡します。 XML-CODE には、パーサーが設定した元の例外コードが含まれます。

XMLPARSE(COMPAT) の場合

XMLPARSE(COMPAT) が有効である場合、例外コードの数字が次の範囲のいずれかである場合には、処理プロシージャーで例外を処理して、構文解析を続行することができます。

- 1-99
- 100,001 から 165,535

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。

1 から 49 の範囲の例外は XML 仕様に基づく致命的エラーです。したがって、ユーザ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ることはできません。ただし、パーサーでは、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が検出されるまで、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
568 ページの『XML-CODE』

関連タスク
582 ページの『XML 文書のエンコード方式についての理解』
588 ページの『XML PARSE の例外処理』
591 ページの『コード・ページの矛盾の処理』
592 ページの『XML 構文解析の終了』

関連参照
777 ページの『継続を許可する XML PARSE 例外』
782 ページの『継続を許可しない XML PARSE 例外』
401 ページの『XMLPARSE』

コード・ページの矛盾の処理

エンコード競合例外を処理する方法は、XMLPARSE コンパイラー・オプションの設定に依存します。

XMLPARSE(XMLSS) の場合

XMLPARSE(XMLSS) コンパイラー・オプションが有効である場合、パーサーはエンコード競合またはその他のタイプの例外のために処理を続行しません。 XML-CODE の値に加えた変更はすべて無視されます。パーサーが XML PARSE ステートメントに戻る際の XML-CODE の値は、元の例外コードです。

ヒント: エンコード競合例外が発生した場合、z/OS XML System Services 理由コードは通常 XRSN_PARM_ENCODING_SPEC_INVALID です。ただし、競合が構文解析の開始後に検出された場合、異なる理由コードを取得する場合もあります (正しくない文字を示す理由コードなど)。

XMLPARSE(COMPAT) の場合

XMLPARSE(COMPAT) コンパイラー・オプションが有効である場合、処理プロシージャーは、文書エンコード競合の例外を処理できる可能性があります。 文書項目が英数字であり、XML-CODE の例外コードが 100,001 から 165,535 で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 を減算した値を文書のコード・ページに指定して、構文解析を行うこともできます。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、パーサーは外部コード・ページ (CODEPAGE コンパイラー・オプションの値) を使用します。
- 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 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
568 ページの『XML-CODE』
589 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
582 ページの『XML 文書のエンコード方式についての理解』
588 ページの『XML PARSE の例外処理』

関連参照
777 ページの『継続を許可する XML PARSE 例外』
782 ページの『継続を許可しない XML PARSE 例外』
401 ページの『XMLPARSE』

XML 構文解析の終了

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 文書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、文書を十分に確認済みである場合、あるいは文書に何らかの不規則性があるためにそれ以上処理を続けても意味がないことがわかった場合に使用します。

この場合、パーサーはそれ以上イベント (XML 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 が -1 であるかどうかを検査できます。これは、ユーザーが意図的に構文解析を終了したことを示します。 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、XMLPARSE(COMPAT) オプションが有効である場合には、 XML-CODE を変更せずに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。

XMLPARSE(XMLSS) オプションが有効である場合には、例外イベント発生後に構文解析は常に終了します。

関連概念
568 ページの『XML-CODE』
589 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
588 ページの『XML PARSE の例外処理』

関連参照
XML-CODE (*Enterprise COBOL 言語解説書*)

第 29 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 文書をエンコードするコード・ページ
- 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)
- 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、メッセージとして WebSphere MQ へ渡したり、後で変換するために CICS 通信域へ伝送したりできます。

リンク・エディットの考慮事項: XML GENERATE ステートメントを含んだ COBOL プログラムは、AMODE 31 を使用してリンク・エディットする必要があります。

関連タスク
『XML 出力の生成』
597 ページの『生成される XML 出力のエンコードの制御』
598 ページの『XML 出力生成時のエラーの処理』
604 ページの『XML 出力の拡張』

関連参照
Extensible Markup Language (XML)

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) として宣言できます。

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 文書内に含まれる各基本データ項目が (このようなデータ項目が FILLER 以外の名前を持ち、データ記述項目に OCCURS 節がない場合)、子エレメントとしてではなく、 すぐ上位のデータ項目に対応する XML エレメントの属性として表されます。

たとえば、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、ENCODING 句をコーディングして、生成される XML 文書の CCSID を指定することもできます。 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および CODEPAGE コンパイラー・オプション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) またはプログラムで有効になっている CODEPAGE コンパイラー・オプション (ENCODING 句が指定されていない場合) から派生します。

たとえば、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 CCSID 1208 (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 1208
        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・エレメントで宣言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。

たとえば、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 (:) のない、有効な XML 名である必要があります。詳細については、名前空間に関する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。

たとえば、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>
```

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。

599 ページの『例: XML の生成』

関連概念
Uniform Resource Identifier (URI): Generic Syntax

関連タスク
『生成される XML 出力のエンコードの制御』
598 ページの『XML 出力生成時のエラーの処理』
151 ページの『UTF-8 データの処理』

関連参照
データのクラスおよびカテゴリー (*Enterprise COBOL 言語解説書*)
XML GENERATE ステートメント (*Enterprise COBOL 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 句を使用してコード・ページ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* 句を指定して出力文書のコード化文字セット ID (CCSID) を指定した場合、 *codepage* には、XML 文書のコード化文字セットに関する下記の関連参照で示されている COBOL XML 処理でサポートされているコード・ページのいずれかを指定する符号なし整数データ項目または符号なし整数リテラルを指定する必要があります。

- 生成された XML を受け取るデータ項目のカテゴリーが国別である場合、WITH ENCODING 句には 1200 (Unicode UTF-16 の CCSID)を指定する必要があります。

- 受け取る ID のカテゴリーが英数字の場合、WITH ENCODING 句には CCSID 1208 またはサポートされている EBCDIC コード・ページの CCSID を指定する必要があります。

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 74. **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
英数字	ソースのコンパイル時に CODEPAGE コンパイラー・オプションで指定された有効なコード・ページ
国別	UTF-16 ビッグ・エンディアン (UTF-16BE、CCSID 1200)

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。

関連参照 350 ページの『CODEPAGE』
584 ページの『XML 文書のコード化文字セット』
XML GENERATE の操作 (*Enterprise COBOL 言語解説書*)

XML 出力生成時のエラーの処理

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
118 ページの『データ項目のサブストリングの参照』

関連参照
785 ページの『XML GENERATE 例外』

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
|                 xml-declaration 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

```
Process xmlparse(xmlss), codepage(37)
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
|                                If xml-namespace-prefix = space
|                                  String '<' xml-text delimited by size
|                                      into buffer with pointer pose
|                                Else
|                                  String '<' xml-namespace-prefix ':' xml-text
|                                      delimited by size into buffer with pointer pose
|                                End-if
|                                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
|                                If xml-namespace-prefix = space
|                                  String xml-text '="' delimited by size into buffer
|                                      with pointer posd
|                                Else
|                                  String xml-namespace-prefix ':' xml-text '="'
|                                      delimited by size into buffer with pointer posd
|                                End-if
|                                Set attribute to true
|                              When 'NAMESPACE-DECLARATION'
|                                If element
|                                  String ' ' delimited by size into buffer
|                                      with pointer posd
|                                Else
|                                  String '" ' delimited by size into buffer
|                                      with pointer posd
|                                End-if
|                                If xml-namespace-prefix = space
|                                  String 'xmlns="' xml-namespace delimited by size
|                                      into buffer with pointer posd
|                                Else
|                                  String 'xmlns:' xml-namespace-prefix '="' xml-namespace
|                                      delimited by size into buffer with pointer posd
|                                End-if
|                                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
                                  If xml-namespace-prefix = space
                                    String '</' xml-text '>' delimited by size
                                        into buffer with pointer posd
                                  Else
                                    String '</' xml-namespace-prefix ':' xml-text '>'
                                        delimited by size into buffer with pointer posd
                                  End-if
                              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="IBM-037"?>
<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
561 ページの『第 28 章 XML 入力の処理』

関連参照 401 ページの『XMLPARSE』
XML GENERATE の操作 (*Enterprise COBOL 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。

以下に例を示します。

- 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。
- データ定義が必要なデータ型ではありません。再定義 (XML GENERATE ステートメントでは無視されます) のみが適切なフォーマットになっていると思われます。
- 無関係な従属グループ内で必要データ項目があまりに深くネストされています。XML 出力は、階層ではなく、デフォルトで行われるように「平ら」にする必要があります。
- 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。
- グループ項目は必要情報を含んでいますが、順序が正しくありません。

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。

幾つかの利点のある代替方法として、元のグループ・データ項目の再定義を準備し、その再定義から XML 出力を生成するという方法があります。このためには、元のデータ記述セットを出発点にして、以下の変更を行ってください。

- 基本データ項目の名前を FILLER に変更するか、またはそれらの名前を削除することにより、生成された XML から基本データ項目を除外します。
- 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。
- 不要な中間グループ項目を除去して、階層を平らにします。
- 種々のデータ型を指定して、必要なトリミング動作が行われるようにします。
- 一連の XML GENERATE ステートメントを使用して、異なった出力順序を選択します。

上記の変更を最も安全に行うには、1 つ以上の REPLACE コンパイラー指示ステートメントを伴う元の宣言の別のコピーを使用します。 以下に示す例は、その方法を示しています。

『例: XML 出力の拡張』

XML 文書を生成する際に、エレメントまたは属性名および値の中にハイフンを含んでいるものがあることもあります。エレメントまたは属性値に含まれるハイフンはそのままにして、エレメントまたは属性名のハイフンを下線に変換することがあります。以下に示す例は、その方法を示しています。

608 ページの『例: エレメントまたは属性名のハイフンを下線に変換する』

関連参照
XML GENERATE の操作 (*Enterprise COBOL 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、訂正可能な幾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 X(02).
            25  CDR-LIFE-BP-DASH   PIC X(02).
            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(02).
            25  CDR-LIFE-QTY-SPACE1  PIC X(02).
            25  FILLER               PIC X(02).
        20  CDR-LIFE-BASE-QTY-ED    REDEFINES
                CDR-LIFE-BASE-QTY PIC ZZ,ZZZ,ZZZ.ZZZ.
        20  CDR-LIFE-BASE-VALUE   PIC X(15).
        20  CDR-LIFE-BASE-VALUE-ED  REDEFINES
                CDR-LIFE-BASE-VALUE
                                  PIC $(4),$$$,$$9.99.
    15  CDR-LIFE-BASE-TOT-VALUE-LINE.
        20  CDR-LIFE-BASE-TOT-VALUE   PIC X(15).
```

このデータ構造に幾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 (599 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>$2</CDR-LIFE-BP-SPACE>
      <CDR-LIFE-BP-DASH>3.</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>       1</CDR-LIFE-QTY-SPACE>
      <CDR-LIFE-QTY-DASH>23</CDR-LIFE-QTY-DASH>
      <CDR-LIFE-QTY-SPACE1>.0</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>$3</CDR-LIFE-BP-SPACE>
      <CDR-LIFE-BP-DASH>4.</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>       2</CDR-LIFE-QTY-SPACE>
      <CDR-LIFE-QTY-DASH>34</CDR-LIFE-QTY-DASH>
      <CDR-LIFE-QTY-SPACE1>.0</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-LINE>
    <CDR-LIFE-BASE-TOT-VALUE>Very high!</CDR-LIFE-BASE-TOT-VALUE>
  </CDR-LIFE-BASE-TOT-VALUE-LINE>
</CDR-LIFE-BASE-VALUES-BOX>
```

生成されたこの XML には幾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。
- 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、ストレージを次のように再定義することにより修正できます。

```
1 BaseValues redefines CDR-LIFE-BASE-VALUES-BOX.
 2 BaseValueDate pic x(8).
 2 BaseValueLine occurs 2 times.
  3 Description pic x(15).
  3 pic x(9).
  3 BaseDate pic x(8).
  3 BasePrice pic x(6) justified.
  3 BaseQuantity pic x(14) justified.
  3 BaseValue pic x(15) justified.
 2 TotalValue pic x(15).
```

上記のデータ値の定義のセットから XML を生成してフォーマット設定した結果は、一層便利なものになっています。

```
<BaseValues>
  <BaseValueDate>01/02/03</BaseValueDate>
  <BaseValueLine>
    <Description>First</Description>
    <BaseDate>01/01/01</BaseDate>
    <BasePrice>$23.00</BasePrice>
    <BaseQuantity>123.000</BaseQuantity>
    <BaseValue>$765.00</BaseValue>
  </BaseValueLine>
  <BaseValueLine>
    <Description>Second</Description>
    <BaseDate>02/02/02</BaseDate>
    <BasePrice>$34.00</BasePrice>
    <BaseQuantity>234.000</BaseQuantity>
    <BaseValue>$654.00</BaseValue>
  </BaseValueLine>
  <TotalValue>Very high!</TotalValue>
</BaseValues>
```

上に示すように、元のデータ定義を直接再定義できます。通常、元の定義を使用するほうがより安全ですが、コンパイラーのテキスト操作機能を使用して、元の定義を適宜に変更することができます。その例を、以下の REPLACE コンパイラー指示ステートメントで示します。この REPLACE ステートメントは複雑に見えますが、元のデータ定義が変更されると、自己保持するという利点があります。

```
replace ==CDR-LIFE-BASE-VALUES-BOX== by
            ==BaseValues redefines CDR-LIFE-BASE-VALUES-BOX==
        ==CDR-LIFE-BASE-VAL-DATE== by ==BaseValueDate==
        ==CDR-LIFE-BASE-VALUE-LINE== by ==BaseValueLine==
        ==20  CDR-LIFE-BASE-DESC.== by ====
        ==CDR-LIFE-BASE-DESC1== by ==Description==
        ==CDR-LIFE-BASE-LIT== by ====
        ==CDR-LIFE-BASE-DTE== by ==BaseDate==
        ==20  CDR-LIFE-BASE-PRICE.== by ====
        ==25  CDR-LIFE-BP-SPACE PIC X(02).== by ====
        ==25  CDR-LIFE-BP-DASH PIC X(02).== by ====
        ==25  CDR-LIFE-BP-SPACE1 PIC X(02).== by ====
        ==CDR-LIFE-BASE-PRICE-ED== by ==BasePrice==
        ==REDEFINES CDR-LIFE-BASE-PRICE PIC $$$.$$.== by
            ==pic x(6) justified.==
        ==20  CDR-LIFE-BASE-QTY.
            25  CDR-LIFE-QTY-SPACE PIC X(08).
            25  CDR-LIFE-QTY-DASH PIC X(02).
            25  CDR-LIFE-QTY-SPACE1 PIC X(02).
            25  FILLER PIC X(02).== by ====
        ==CDR-LIFE-BASE-QTY-ED== by ==BaseQuantity==
        ==REDEFINES CDR-LIFE-BASE-QTY PIC ZZ,ZZZ,ZZZ.ZZZ.== by
            ==pic x(14) justified.==
        ==CDR-LIFE-BASE-VALUE-ED== by ==BaseValue==
        ==20  CDR-LIFE-BASE-VALUE PIC X(15).== by ====
        ==REDEFINES CDR-LIFE-BASE-VALUE PIC $(4),$$$,$$9.99.==
            by ==pic x(15) justified.==
        ==CDR-LIFE-BASE-TOT-VALUE-LINE. 20== by ====
        ==CDR-LIFE-BASE-TOT-VALUE== by ==TotalValue==.
```

元の定義セットの 2 番目のインスタンスが後に続いているこの REPLACE ステートメントの結果は、上に示されているグループ項目 BaseValues の推奨再定義と似ています。この REPLACE ステートメントは、不要な定義を除去し、保持すべき定義の変更を行うさまざまな手法の例を示しています。それぞれの状況に適したいずれかの手法をご利用ください。

関連参照
XML GENERATE の操作 (*Enterprise COBOL 言語解説書*)
REPLACE ステートメント (*Enterprise COBOL 言語解説書*)

例: エレメントまたは属性名のハイフンを下線に変換する

ハイフンを含んでいるデータ名を持つ項目のあるデータ構造から XML 文書を生成すると、生成された XML にはハイフンを含んでいるエレメントまたは属性名が含まれることになります。この例は、エレメントまたは属性値内にあるハイフンを変更せずに、そのようなハイフンを下線に変換する方法を示しています。

```
1 Customer-Record.
 2 Customer-Number  pic 9(9).
 2 First-Name       pic x(10).
 2 Last-Name        pic x(20).
```

上記のデータ構造に幾つかのサンプル値を取り込み、XML をそれから生成し、その後プログラム Pretty (599 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。

```
<Customer-Record>
  <Customer-Number>12345</Customer-Number>
  <First-Name>John</First-Name>
  <Last-Name>Smith-Jones</Last-Name>
</Customer-Record>
```

エレメント名はハイフンを含み、エレメント Last-Name の内容もハイフンを含んでいます。

この XML 文書がデータ項目 xmldoc の内容であり、charcnt が XML 文書の長さに設定されていると想定した場合、以下のコードを使用することにより、エレメント名の中のすべてのハイフンを下線に変更し、エレメント値は未変更のままにすることができます。

```
1 xmldoc              pic x(16384).
1 charcnt     comp-5  pic 9(5).
1 pos         comp-5  pic 9(5).
1 tagstate    comp-5  pic 9  value zero.
1 quotestate  comp-5  pic 9  value zero.

. . .
dash-to-underscore.
  perform varying pos from 1 by 1
          until pos > charcnt
    if xmldoc(pos:1) = '<'
      move 1 to tagstate
    end-if
    if tagstate = 1
      if xmldoc(pos:1) = '"'
        if quotestate = 0
          move 1 to quotestate
        else
          move 0 to quotestate
        end-if
      end-if
    end-if
    if tagstate = 1 and quotestate = 0 and xmldoc(pos:1) = '-'
      move '_' to xmldoc(pos:1)
    else
      if xmldoc(pos:1) = '>'
```

```
        move 0 to tagstate
      end-if
    end-if
  end-perform.
```

以下に示すように、データ項目 xmldoc 内の改訂された XML 文書では、エレメント名内のハイフンは下線になっていますが、エレメント値内のハイフンはそのままです。

```
<Customer_Record>
  <Customer_Number>12345</Customer_Number>
  <First_Name>John</First_Name>
  <Last_Name>Smith-Jones</Last_Name>
</Customer_Record>
```

第 6 部 オブジェクト指向プログラムの開発

第 30 章 オブジェクト指向プログラムの作成

オブジェクト指向 (OO) プログラムを書く際には、必要とするクラス、およびクラスが作業を行うのに必要なメソッドとデータを決定する必要があります。

OO プログラムは、オブジェクト (状態と動作をカプセル化するエンティティー) ならびにオブジェクトのクラス、メソッド、およびデータに基づいています。クラスは、オブジェクトの状態および機能を定義するテンプレートです。通常、プログラムは、あるクラスの複数のオブジェクト・インスタンス (または単にインスタンス)、つまりそのクラスのメンバーである複数のオブジェクトを作成し、それを扱う仕事をします。それぞれのインスタンスの状態はインスタンス・データと呼ばれるデータに保管されます。各インスタンスの機能は、インスタンス・メソッドと呼ばれます。クラスでは、そのクラスのすべてのインスタンスが共用するデータ (ファクトリーまたは*静的*データと呼ばれる)、およびいずれのオブジェクト・インスタンスとも無関係にサポートされるメソッド (ファクトリーまたは*静的*メソッドと呼ばれる) を定義できます。

Enterprise COBOL を使用して、以下のことを行うことができます。

- メソッドとデータを COBOL でインプリメントした状態で、クラスを定義する。
- Java および COBOL クラスのインスタンスを作成する。
- Java および COBOL オブジェクトにメソッドを呼び出す。
- Java クラスまたはほかの COBOL クラスから継承するクラスを書き込む。
- 多重定義メソッドを定義して呼び出す。

Enterprise COBOL プログラムで、Java Native Interface (JNI) が提供するサービスを呼び出して、COBOL 言語で直接使用可能な基本 OO 機能に加えて、Java 指向機能を取得できます。

Enterprise COBOL クラスでは、CALL ステートメントをコーディングして、プロシージャー型 COBOL プログラムとインターフェースを取ることができます。したがって、COBOL クラス定義構文は、プロシージャー型 COBOL ロジックのラッパー・クラスを書き込むために特に役立ち、Java から既存の COBOL コードにアクセスすることを可能にします。

Java コードは、COBOL クラスのインスタンスを作成したり、これらのクラスのメソッドを呼び出したり、COBOL クラスを拡張したりすることができます。

オブジェクト指向 COBOL プログラムや Java プログラムの開発や実行は、z/OS UNIX 環境で行うことが推奨されます。

制約事項:

- COBOL クラス定義およびメソッドは、EXEC SQL ステートメントを含むことができず、SQL コンパイラー・オプションを使用してコンパイルすることもできません。

- COBOL クラス定義およびメソッドは、EXEC CICS ステートメントを含むことができず、CICS 環境で実行することができません。CICS コンパイラー・オプションを使用してコンパイルすることができません。

『例: 口座』

関連タスク 617 ページの『クラスの定義』 622 ページの『クラス・インスタンス・メソッドの定義』 631 ページの『クライアントの定義』 644 ページの『サブクラスの定義』 648 ページの『ファクトリー・セクションの定義』 329 ページの『第 16 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行』 *Enterprise COBOL コンパイラーおよびランタイム 移行ガイド* (IBM COBOL ソース・プログラムのアップグレード)

関連参照
The Java Language Specification

例: 口座

銀行にカスタマーが口座を開設し、その口座で預金や引き出しを行うときの例を見てみましょう。口座は、Account という名前の汎用クラスで表します。多数のカスタマーが存在します。したがって、Account クラスの複数インスタンスが同時に存在すると考えることができます。

必要とするクラスを判別したら、次のステップでは、それらのクラスがそれぞれの作業を実行するために必要なメソッドを判別します。 Account クラスは以下のサービスを提供する必要があります。

- 口座を開設する。
- 現在の収支を取る。
- 口座に預金する。
- 口座から預金を引き出す。
- 口座状況を報告する。

Account クラスの以下のメソッドは、上記の要件を満たします。

init 口座を開設し、それに口座番号を割り当てます。

getBalance
口座の現在の収支を戻します。

credit 指定の金額を口座に預金します。

debit 指定の金額を口座から引き出します。

print 口座番号と勘定残高を表示します。

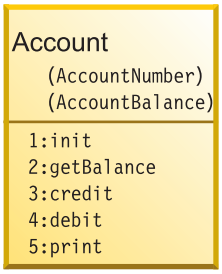
Account クラスとそのメソッドを設計すると、クラスはいくつかのインスタンス・データを保持する必要があることがわかります。一般に、Account オブジェクトには次のようなインスタンス・データが必要です。

- 口座番号
- 勘定残高

- カスタマー情報: 名前、住所、自宅の電話番号、勤務先電話番号、社会保障番号など

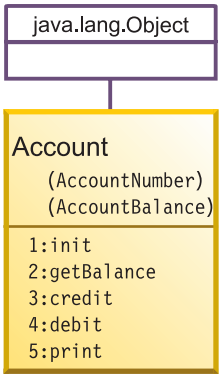
ただし、上記の例を単純化するため、口座番号と勘定残高は、Account クラスが必要とする唯一のインスタンス・データであると想定します。

クラスやメソッドを設計するとき、ダイアグラムは役に立ちます。次のダイアグラムで、Account クラスの設計における最初の試みを示します。



ダイアグラムにおいて、括弧内のワードはインスタンス・データの名前です。番号とコロンに続くワードは、インスタンス・メソッドの名前です。

以下の構造は、クラスの相互関係を示しており、*継承の階層* と呼ばれています。Account クラスはクラス java.lang.Object から直接継承します。



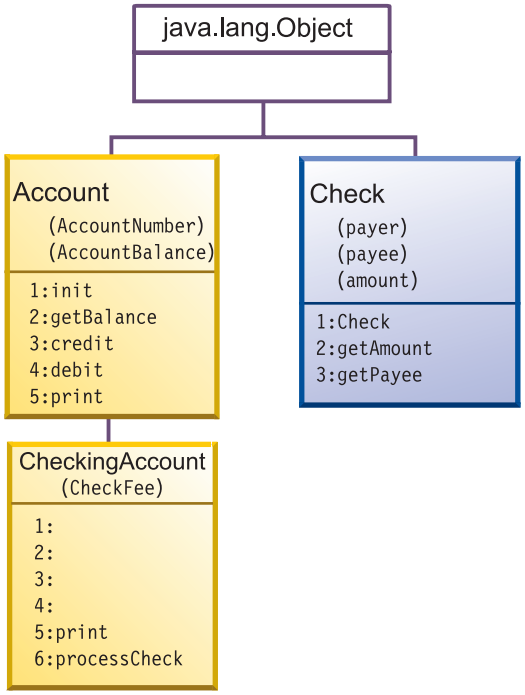
サブクラス

上記の口座の例において、Account は汎用クラスです。ただし、銀行は、当座預金、普通預金、住宅ローンなど、多くの種類の口座を用意することができます。これらの口座のすべてには、口座の一般的特性がある一方で、すべての種類の口座に必ずしも共有されない追加特性が含まれている場合があります。

例えば、CheckingAccount クラスには、すべての口座が持つ口座番号や勘定残高に加えて、口座に書き込まれる各当座に適用される当座手数料がある場合があります。また、CheckingAccount クラスには、当座を処理する (すなわち、金額を読み取る、支払人の借方に記入する、受取人の貸方に記入するなど) メソッドも必要です。したがって、CheckingAccount を Account のサブクラスとして定義し、そのサブクラスが必要とする追加のインスタンス・データやインスタンス・メソッドをそのサブクラスで定義することは意味があります。

CheckingAccount クラスを設計すると、当座をモデル化するクラスの必要性があることに気が付きます。クラス Check のインスタンスは、少なくとも、支払人、受取人、および当座の金額のインスタンス・データを必要とします。

実際のオブジェクト指向のアカウント・システムでは多くの追加クラス (ならびにデータベースおよびトランザクション処理ロジック) を設計する必要があるでしょうが、ここでは例を単純化するために省略されています。継承更新ダイアグラムを以下に示します。



番号とコロンの後にメソッド名が記されていないものは、その番号のメソッドがスーパークラスから継承されていることを示します。

多重継承: OO COBOL アプリケーションでは*多重継承*を使用できません。定義するすべてのクラスは、厳密に 1 つの親を持つ必要があります。java.lang.Object は、すべての継承の階層のルートになければなりません。したがって、OO COBOL アプリケーション内で定義されるオブジェクト指向システムのクラス構造はツリー状です。

629 ページの『例: メソッドの定義』

関連タスク

617 ページの『クラスの定義』
622 ページの『クラス・インスタンス・メソッドの定義』
644 ページの『サブクラスの定義』

クラスの定義

COBOL クラス定義は、IDENTIFICATION DIVISION および ENVIRONMENT DIVISION、その後にオプションのファクトリー定義とオプションのオブジェクト定義、さらにその後に END CLASS マーカーが続く構成となっています。

表 75. **クラス定義の構成**

セクション	目的	構文
IDENTIFICATION DIVISION (必須)	クラスの名前。継承情報を提供する。	618 ページの『クラス定義用の CLASS-ID 段落』 (必須) AUTHOR 段落 (オプション) INSTALLATION 段落 (オプション) DATE-WRITTEN 段落 (オプション) DATE-COMPILED 段落 (オプション)
ENVIRONMENT DIVISION (必須)	コンピューター環境を記述する。クラス定義内で使用されるクラス名を、コンパイル単位の外側で判明している、対応する外部クラス名に関連付ける。	CONFIGURATION SECTION (必須) 619 ページの『クラス定義用の REPOSITORY 段落』 (必須) SOURCE-COMPUTER 段落 (オプション) OBJECT-COMPUTER 段落 (オプション) SPECIAL-NAMES 段落 (オプション)

ファクトリー定義 (オプション)	クラスのすべてのインスタンスが共有するデータとオブジェクト・インスタンスとは別々にサポートされるメソッドを定義する。	`IDENTIFICATION DIVISION.`
		`FACTORY.`
		`  DATA DIVISION.`
		`  WORKING-STORAGE SECTION.`
		`*   (Factory data here)`
		`  PROCEDURE DIVISION.`
		`*   (Factory methods here)`
		`END FACTORY.`

オブジェクト定義 (オプション)	インスタンス・データとインスタンス・メソッドを定義する。	`IDENTIFICATION DIVISION.`
		`OBJECT.`
		`  DATA DIVISION.`
		`  WORKING-STORAGE SECTION.`
		`*   (Instance data here)`
		`  PROCEDURE DIVISION.`
		`*   (Instance methods here)`
		`END OBJECT.`

SOURCE-COMPUTER、OBJECT-COMPUTER、または SPECIAL-NAMES 段落をクラス CONFIGURATION SECTION に指定すると、それらの段落は、そのクラスが導入するすべてのメソッドを含む、クラス定義全体に適用されます。

クラス CONFIGURATION SECTION は、プログラム CONFIGURATION SECTION と同じ記入項目で構成されています。ただし、クラス CONFIGURATION SECTION には INPUT-OUTPUT SECTION を含めることはできません。INPUT-OUTPUT SECTION の定義は、クラス・レベルでその定義を行うのではなく、それを必要とする個々のメソッドにおいてのみ行います。

上記で説明したように、インスタンス・データとメソッドの定義は、そのクラス定義の OBJECT 段落内で、それぞれ、DATA DIVISION および PROCEDURE DIVISION において、行います。個別のオブジェクト・インスタンスにではなく、クラス自体に関連付けるデータとメソッドを必要とするクラスに、クラス定義の FACTORY 段落内で、別々の DATA DIVISION および PROCEDURE DIVISION を定義します。

各 COBOL クラス定義は、別々のソース・ファイルになければなりません。

621 ページの『例: クラスの定義』

関連タスク 620 ページの『クラス・インスタンス・データ定義用の WORKING-STORAGE SECTION』 622 ページの『クラス・インスタンス・メソッドの定義』 644 ページの『サブクラスの定義』 648 ページの『ファクトリー・セクションの定義』 7 ページの『コンピューター環境の記述』 329 ページの『第 16 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行』

関連参照
COBOL クラス定義構成 (*Enterprise COBOL 言語解説書*)

クラス定義用の CLASS-ID 段落

クラスを指定し、それに継承情報を提供するには、IDENTIFICATION DIVISION の CLASS-ID 段落を使用してください。

```
   Identification Division.
必要
 Class-id. Account inherits Base.
必要
```

以下のクラスを識別するには、CLASS-ID 段落を使用してください。

- 定義中のクラス (上記の例の Account)。
- 定義中のクラスがその特性を継承する元の即時スーパークラス。スーパークラスは、Java または COBOL でインプリメントされます。

  上記の例において、inherits Base では、Account クラスは、クラス定義内で Base として認識されているクラスからメソッドとデータを継承することを示します。オブジェクト指向 COBOL プログラムにおいて、名前 Base は java.lang.Object を参照するために使用することをお勧めします。

クラス名では 1 バイト文字を使用する必要があり、クラス名は COBOL ユーザー定義語の通常の形成規則に準拠している必要があります。

ENVIRONMENT DIVISION の CONFIGURATION SECTION で REPOSITORY 段落を使用して、スーパークラス名 (例の Base) を外部に判明しているスーパークラス名 (Base の場合の java.lang.Object) に関連付けます。また、オプションとして、定義中のクラスの名前 (例の Account) を REPOSITORY 段落に指定し、その対応する外部クラス名にそれを関連付けることもできます。

すべてのクラスを java.lang.Object クラスから直接的または間接的に引き出さなければなりません。

関連タスク
619 ページの『クラス定義用の REPOSITORY 段落』

関連参照
CLASS-ID 段落 (*Enterprise COBOL 言語解説書*)
ユーザー定義語 (*Enterprise COBOL 言語解説書*)

クラス定義用の REPOSITORY 段落

指定された語をクラス定義内で使用するときにその語がクラス名であることをコンパイラーに宣言する場合、さらに必要に応じてクラス名を対応する外部クラス名 (コンパイル単位の外側で認識されているクラス名) に関係付ける場合に、REPOSITORY 段落を使用してください。

外部クラス名は大/小文字が区別されます。したがって、Java 形成規則に準拠しなければなりません。例えば、Account クラス定義において、以下のようにコーディングします。

```
Environment Division. 必要
Configuration Section. 必要
Repository. 必要
    Class Base is "java.lang.Object"  Required
    Class Account is "Account".       Optional
```

REPOSITORY 段落記入項目では、クラス定義内で Base および Account として参照されるクラスの外部クラス名は、それぞれ、java.lang.Object および Account であることが示されます。

REPOSITORY 段落では、クラス定義において明示的に参照するそれぞれのクラス名ごとに記入項目をコーディングする必要があります。以下に例を示します。

- ベース
- 定義中のクラスが継承する元のスーパークラス
- クラス定義内のメソッドで参照するクラス

REPOSITORY 段落記入項目において、名前に非 COBOL 文字が含まれている場合には、外部クラス名を指定しなければなりません。Java パッケージ の一部である参照クラスごとに外部クラス名も指定しなければなりません。そのようなクラスごとに、外部クラス名をパッケージの完全修飾名として指定し、後にピリオド (.) が付き、続いてその後に Java クラスの単純名が付きます。例えば、Object クラスは java.lang パッケージの一部です。したがって、上記に示したように、その外部名を java.lang.Object として指定します。

REPOSITORY 段落で指定する外部クラス名は、完全修飾 Java クラス名の形成規則に準拠した英数字リテラルでなければなりません。

REPOSITORY 段落記入項目に外部クラス名を組み込まない場合、外部クラス名は以下の方法でクラス名から作成されます。

- クラス名は大文字に変換されます。
- 各ハイフンはゼロに変更されます。
- 数字の場合、最初の文字は以下のように変更されます。
  - 1 から 9 は A から I に変更されます。
  - 0 は J に変換されます。

外部名は英大/小文字混合で名前付けされます。したがって、上記の例で、クラス Account は外部的には Account (英大/小文字混合) として認識されます。

オプションとして、定義中のクラス (上記の例の Account) の記入項目を REPOSITORY 段落に組み込むことができます。外部クラス名が非 COBOL 文字を含

む場合は定義中のクラスの記入項目を組み込む必要があり、クラスが Java パッケージの一部となる場合には、完全パッケージ修飾クラス名を指定する必要があります。

『例: 外部クラス名および Java パッケージ』

関連タスク
669 ページの『Java 用の配列およびストリングの宣言』

関連参照
REPOSITORY 段落 (*Enterprise COBOL 言語解説書*)
The Java Language Specification (Identifiers)
The Java Language Specification (Packages)

例: 外部クラス名および Java パッケージ

次の例では、REPOSITORY 段落内の記入項目から外部クラス名を決定する方法を説明します。

```
Environment division.
Configuration section.
Repository.
    Class Employee is "com.acme.Employee"
    Class JavaException is "java.lang.Exception"
    Class Orders.
```

次の表には、ローカル・クラス名 (クラス定義内で使用されるクラス名)、そのクラスを含む Java パッケージ、および関連付けられた外部クラス名が記述されています。

ローカル・クラス名	Java パッケージ	外部クラス名
Employee	com.acme	com.acme.Employee
JavaException	java.lang	java.lang.Exception
Orders	(名前付けなし)	ORDERS

外部クラス名 (REPOSITORY 段落記入項目内のクラス名の後の名前とオプションの IS) は、パッケージ (ある場合) の完全修飾名から構成され、後にピリオドが付き、続いてその後にクラスの単純名が付きます。

関連タスク
619 ページの『クラス定義用の REPOSITORY 段落』

関連参照
REPOSITORY 段落 (*Enterprise COBOL 言語解説書*)

クラス・インスタンス・データ定義用の WORKING-STORAGE SECTION

COBOL クラスが必要とする インスタンス・データ、つまりクラスの各インスタンスに割り当てられるデータを記述するには、OBJECT 段落の DATA DIVISION にある WORKING-STORAGE SECTION を使用します。

IDENTIFICATION DIVISION 宣言の直前に入れる必要がある OBJECT キーワードは、クラスのインスタンス・データおよびインスタンス・メソッドの定義の開始を示します。例えば、Account クラスのインスタンス・データの定義は、以下のようになります。

```
Identification division.
Object.
  Data division.
  Working-storage section.
  01 AccountNumber  pic 9(6).
  01 AccountBalance pic S9(9) value zero.
  . . .
End Object.
```

インスタンス・データは、オブジェクト・インスタンスが作成されるときに割り振られ、Java ランタイムによるインスタンスのガーベッジ・コレクションが行われるまで存在します。

上記に示すように、単純インスタンス・データの初期化は、VALUE 節を使用して行うことができます。より複雑なインスタンス・データの初期化は、カスタマイズしたメソッドをコーディングし、クラスのインスタンスを作成して初期化して行うことができます。

COBOL インスタンス・データは、Java private 非静的メンバー・データと同等です。他のクラスまたはサブクラス (同じクラス内のファクトリー・メソッドがあればそのメソッドも) は、COBOL インスタンス・データを直接参照することはできません。インスタンス・データは、OBJECT 段落で定義するすべてのインスタンス・メソッドにグローバルです。OBJECT 段落の外側からインスタンス・データにアクセス可能にしたい場合には、アクセスを可能にするために属性 (get または set) インスタンス・メソッドを定義します。

インスタンス・データ宣言のための WORKING-STORAGE SECTION の構文は、プログラムにおける場合と一般的に同じですが、次のような例外があります。

- EXTERNAL 属性を使用することはできません。
- GLOBAL 属性を使用できますが、効力はありません。

関連タスク

641 ページの『クラスのインスタンスの作成および初期化』
643 ページの『クラスのインスタンスの解放』
650 ページの『ファクトリー・メソッドの定義』
628 ページの『属性 (get および set) メソッドのコーディング』

例: クラスの定義

次の例では、Account クラスの定義での最初の試みを示します。ただし、メソッド定義は除きます。

```
cbl dll,thread,pgmname(longmixed)
Identification Division.
Class-id. Account inherits Base.
Environment Division.
Configuration section.
Repository.
    Class Base    is "java.lang.Object"
    Class Account is "Account".
```

```
*
 Identification division.
 Object.
  Data division.
  Working-storage section.
  01 AccountNumber  pic 9(6).
  01 AccountBalance pic S9(9) value zero.
*
  Procedure Division.
*
*   (Instance method definitions here)
*
 End Object.
*
 End class Account.
```

関連タスク 329 ページの『第 16 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行』 631 ページの『クライアントの定義』

クラス・インスタンス・メソッドの定義

クラス定義の OBJECT 段落の PROCEDURE DIVISION に COBOL インスタンス・メソッドを定義します。インスタンス・メソッドは、クラスのそれぞれのオブジェクト・インスタンスごとにサポートされる操作を定義します。

COBOL インスタンス・メソッド定義は、4 つの部 (COBOL プログラムに類似) とその後の END METHOD マーカーから構成されます。

表 76. **インスタンス・メソッド定義の構成**

除算	目的	構文
IDENTIFICATION (必須)	メソッドの名前を指定する。	623 ページの『クラス・インスタンス・メソッド定義用の METHOD-ID 段落』 (必須) AUTHOR 段落 (オプション) INSTALLATION 段落 (オプション) DATE-WRITTEN 段落 (オプション) DATE-COMPILED 段落 (オプション)
ENVIRONMENT (オプション)	メソッド内で使用されるファイル名を、オペレーティング・システムに認識された、対応するファイル名に関連付ける。	623 ページの『クラス・インスタンス・メソッド定義用の INPUT-OUTPUT SECTION』 (オプション)
DATA (オプション)	外部ファイルを定義する。データのコピーを割り振る。	624 ページの『クラス・インスタンス・メソッド定義用の DATA DIVISION』 (オプション)
PROCEDURE (オプション)	メソッドによって提供されるサービスを完了するために実行可能ステートメントをコーディングする。	625 ページの『クラス・インスタンス・メソッド定義用の PROCEDURE DIVISION』 (オプション)

定義: メソッドのシグニチャーは、メソッドの名前およびその仮パラメーターの数と型から構成されています。 (COBOL メソッドの仮パラメーターの定義は、そのメソッドの PROCEDURE DIVISION ヘッダーの USING 句で行います。)

クラス定義内では、各メソッド名を固有にする必要はありませんが、各メソッドに固有のシグニチャーを与える必要があります。(メソッドに同じ名前を与えても、別々のシグニチャーを与えると、メソッドを多重定義 することになります。)

COBOL インスタンス・メソッドは Java public 非静的メソッドと同じです。

629 ページの『例: メソッドの定義』

関連タスク 625 ページの『クラス・インスタンス・メソッド定義用の PROCEDURE DIVISION』 627 ページの『インスタンス・メソッドの多重定義』 626 ページの『インスタンス・メソッドのオーバーライド』 636 ページの『メソッドの呼び出し (INVOKE)』 647 ページの『サブクラス・インスタンス・メソッドの定義』 650 ページの『ファクトリー・メソッドの定義』

クラス・インスタンス・メソッド定義用の METHOD-ID 段落

インスタンス・メソッドを指定するには、METHOD-ID 段落を使用してください。IDENTIFICATION DIVISION 宣言とともに、METHOD-ID 段落の直前に置いて、メソッド定義の開始を表します。

例えば、Account クラス内の credit メソッドの定義は、以下のように開始します。

```
   Identification Division.
Method-id. "credit".
```

メソッド名を英数字または各国語リテラルとしてコーディングします。メソッド名は、大/小文字を区別して処理されるため、Java メソッド名の形成規則に準拠する必要があります。

他の Java または COBOL メソッドもしくはプログラム (すなわち、クライアント) は、メソッド名を使用してメソッドを呼び出します。

関連タスク

636 ページの『メソッドの呼び出し (INVOKE)』
139 ページの『COBOL での国別データ (Unicode) の使用』

関連参照

The Java Language Specification (Meaning of method names)
The Java Language Specification (Identifiers)
METHOD-ID 段落 (*Enterprise COBOL 言語解説書*)

クラス・インスタンス・メソッド定義用の INPUT-OUTPUT SECTION

インスタンス・メソッドの ENVIRONMENT DIVISION は、1 つのセクション INPUT-OUTPUT SECTION のみを持つことができます。このセクションは、メソッド定義で使用したファイル名をオペレーティング・システムに認識された、対応するファイル名に関連付けます。

例えば、情報をファイルから読み取ったメソッドを Account クラスが定義した場合には、その Account クラスは、以下のようにコーディングされる INPUT-OUTPUT SECTION を持つと考えられます。

```
Environment Division.
Input-Output Section.
File-Control.
    Select account-file Assign AcctFile.
```

メソッドの INPUT-OUTPUT SECTION の構文は、プログラムの INPUT-OUTPUT SECTION の構文と同じです。

関連タスク
7 ページの『コンピューター環境の記述』

関連参照
INPUT-OUTPUT SECTION (*Enterprise COBOL 言語解説書*)

クラス・インスタンス・メソッド定義用の DATA DIVISION

インスタンス・メソッドの DATA DIVISION は、4 つのセクション (FILE SECTION、LOCAL-STORAGE SECTION、 WORKING-STORAGE SECTION、および LINKAGE SECTION) の任意のもので構成されます。

FILE SECTION
メソッド FILE SECTION では EXTERNAL ファイルしか定義できないことを除けば、プログラム FILE SECTION と同じです。

LOCAL-STORAGE SECTION
メソッドの呼び出しごとに、LOCAL-STORAGE データの別個のコピーを割り振り、そのメソッドからの戻り時に解放します。メソッド LOCAL-STORAGE SECTION は、プログラム LOCAL-STORAGE SECTION に類似しています。

データ項目上の VALUE 節を指定すると、メソッドの呼び出しのたびに、項目はその値に初期化されます。

WORKING-STORAGE SECTION
WORKING-STORAGE データの単一コピーが割り振られます。データは、実行単位が終了するまで、最後に使われた状態で持続します。メソッドの呼び出しのたびに、呼び出しているオブジェクトまたはスレッドに関係なく、データの同じ単一コピーを使用します。メソッド WORKING-STORAGE SECTION は、プログラム WORKING-STORAGE SECTION に類似しています。

データ項目上の VALUE 節を指定すると、メソッドの最初の呼び出しのときに、項目はその値に初期化されます。データ項目に対する EXTERNAL 節を指定することができます。

LINKAGE SECTION
プログラム LINKAGE SECTION と同じです。

インスタンス・メソッドの DATA DIVISION および OBJECT 段落の DATA DIVISION の両方において、データ項目を同じ名前で定義した場合、そのデータ名に対するメソッド内の参照は、そのメソッド・データ項目だけを参照します。メソッド DATA DIVISION が優先します。

関連タスク
14 ページの『データの記述』
532 ページの『EXTERNAL 節によるデータの共用』

関連参照
DATA DIVISION 概要 (*Enterprise COBOL 言語解説書*)

クラス・インスタンス・メソッド定義用の PROCEDURE DIVISION

インスタンス・メソッドが提供するサービスをインプリメントするための実行可能ステートメントを、インスタンス・メソッドの PROCEDURE DIVISION にコーディングしてください。

プログラムの PROCEDURE DIVISION でコーディングできるメソッドの PROCEDURE DIVISION において、ほとんどの COBOL ステートメントをコーディングすることができます。ただし、メソッドで以下のステートメントをコーディングすることはできません。

- ENTRY
- EXIT PROGRAM
- 標準 COBOL 85 の以下の古くなったエレメント:
  - ALTER
  - プロシージャー名が指定されていない GOTO
  - SEGMENT-LIMIT
  - USE FOR DEBUGGING

さらに、すべての COBOL クラス定義を THREAD コンパイラー・オプションを使用してコンパイルする必要があるため、SORT または MERGE ステートメントは COBOL メソッドで使用できません。

インスタンス・メソッドで EXIT METHOD または GOBACK ステートメントをコーディングして、呼び出し側のクライアントに制御を戻すことができます。両方のステートメントに同じ効果があります。メソッドが呼び出されるときに RETURNING 句が指定されていると、EXIT METHOD または GOBACK ステートメントは、呼び出し側のクライアントにデータの値を戻します。

各メソッドの PROCEDURE DIVISION では、暗黙の EXIT METHOD が最後のステートメントとして生成されます。

メソッドで STOP RUN を指定することができます。この指定を行うと、実行単位内で実行しているすべてのスレッドを含め、実行単位全体が終了します。

メソッド定義の終了は、END METHOD マーカーを使用して行う必要があります。例えば、次のステートメントは credit メソッドの終わりを示します。

```
End method "credit".
```

渡された引数の取得のための USING 句: メソッドの PROCEDURE DIVISION ヘッダーの USING 句に、メソッドの仮パラメーター (ある場合) を指定してください。引

数が BY VALUE で渡される指定をしなければなりません。各パラメーターは、メソッドの LINKAGE SECTION で、レベル 01 またはレベル 77 項目として定義します。各パラメーターのデータ型は、Java と相互運用可能な型のいずれかでなければなりません。

値を返すための RETURNING 句: メソッドの PROCEDURE DIVISION ヘッダーの RETURNING 句に、メソッドの結果として戻されるデータ項目 (ある場合) を指定してください。データ項目は、メソッドの LINKAGE SECTION で、レベル 01 または レベル 77 項目として定義します。戻り値のデータ型は、Java と相互運用可能な型のいずれかでなければなりません。

関連タスク 669 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』 『インスタンス・メソッドのオーバーライド』 627 ページの『インスタンス・メソッドの多重定義』 635 ページの『オブジェクト参照の比較および設定』 636 ページの『メソッドの呼び出し (INVOKE)』 329 ページの『第 16 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行』

関連参照
396 ページの『THREAD』
手続き部ヘッダー (*Enterprise COBOL 言語解説書*)

インスタンス・メソッドのオーバーライド

サブクラスで定義されたインスタンス・メソッドは、そのサブクラスで利用できるのであれば (これら 2 つのメソッドが同じシグニチャーを持っている場合)、継承されたインスタンス・メソッドをオーバーライドすると言います。

スーパークラス・インスタンス・メソッド m1 を COBOL サブクラスでオーバーライドするには、スーパークラス・メソッドと名前が同じで、その PROCEDURE DIVISION USING 句 (ある場合) の仮パラメーターの数およびタイプがスーパークラス・メソッドと同じであるサブクラスでインスタンス・メソッド m1 を定義します。(スーパークラス・メソッドが Java でインプリメントされる場合には、対応する Java パラメーターのデータ型と相互運用可能な仮パラメーターをコーディングする必要があります。) クライアントがサブクラスのインスタンスで m1 を呼び出すとき、スーパークラス・メソッドではなく、サブクラス・メソッドが呼び出されます。

例えば、Account クラスは、その LINKAGE SECTION および PROCEDURE DIVISION ヘッダーが以下のような、メソッド debit を定義します。

```
Linkage section.
01 inDebit    pic S9(9) binary.
Procedure Division using by value inDebit.
```

CheckingAccount サブクラスを定義し、Account スーパークラスで定義された debit メソッドをオーバーライドする debit メソッドをそれに持たす場合には、pic S9(9) binary として指定された入力パラメーターを必ず 1 つ持つサブクラス・メソッドを定義します。クライアントが、CheckingAccount インスタンスへのオブジェクト参照を使用して、debit を呼び出すと、CheckingAccount debit メソッド (Account スーパークラス内の debit メソッドではなく) が呼び出されます。

メソッド戻り値の有無および PROCEDURE DIVISION RETURNING 句 (ある場合) で使われる戻り値のデータ型は、サブクラス・インスタンス・メソッドとオーバーライドしたスーパークラス・インスタンス・メソッドにおいて同一でなければなりません。

インスタンス・メソッドは、COBOL スーパークラスのファクトリー・メソッドをオーバーライドしてはならないし、Java スーパークラスの静的メソッドをオーバーライドすることもできません。

629 ページの『例: メソッドの定義』

関連タスク 625 ページの『クラス・インスタンス・メソッド定義用の PROCEDURE DIVISION』 669 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』 636 ページの『メソッドの呼び出し (INVOKE)』 640 ページの『オーバーライドされたスーパークラス・メソッドの呼び出し』 644 ページの『サブクラスの定義』 651 ページの『ファクトリー・メソッドまたは静的メソッドの隠蔽』

関連参照
The Java Language Specification (Inheritance, overriding, and hiding)

インスタンス・メソッドの多重定義

クラスでサポートされる 2 つのメソッド (クラスで定義されているか、スーパークラスから継承されたかにかかわらず) は、それらが同じ名前を持っており、シグニチャーが異なる場合には、*多重定義されている*と言います。

例えば、別々の一組のパラメーターを使用してデータを初期化するために、クライアントが別々の版のメソッドを呼び出せるようにするときは、メソッドを多重定義します。

メソッドを多重定義するには、その PROCEDURE DIVISION USING 句 (ある場合) の仮パラメーターの数や型が、同じクラスでサポートされる同一の名前のメソッドと異なるメソッドを定義します。例えば、Account クラスは、必ず 1 つの仮パラメーターを持つインスタンス・メソッド init を定義します。init メソッドの LINKAGE SECTION および PROCEDURE DIVISION ヘッダーは、以下のようになります。

```
Linkage section.
01 inAccountNumber pic S9(9) binary.
Procedure Division using by value inAccountNumber.
```

クライアントはこのメソッドを呼び出し、inAccountNumber のデータ型と一致する引数を必ず 1 つ渡して、指定の口座番号 (およびデフォルトの勘定残高ゼロ) で Account インスタンスを初期化します。

ただし、Account クラスでは、例えば、開始の勘定残高も指定できる別の仮パラメーターを持つ、2 番目のインスタンス・メソッド init を定義することができます。この init メソッドの LINKAGE SECTION および PROCEDURE DIVISION ヘッダーは、以下のようになります。

```
Linkage section.
01 inAccountNumber pic S9(9) binary.
01 inBalance       pic S9(9) binary.
Procedure Division using by value inAccountNumber
                                  inBalance.
```

クライアントは、目的のメソッドのシグニチャーと一致する引数を渡して、いずれかの init メソッドを呼び出すことができます。

メソッド戻り値の有無は、多重定義したメソッドと共通している必要はありません。また、PROCEDURE DIVISION RETURNING 句 (ある場合) で指定する戻り値のデータ型は、多重定義したメソッドと同一である必要はありません。

ファクトリー・メソッドの多重定義は、インスタンス・メソッドを多重定義する場合とまったく同じ方法で行うことができます。

多重定義メソッドの定義および多重定義メソッドの呼び出しの解決に関する規則は、対応する Java 規則に基づきます。

関連タスク
636 ページの『メソッドの呼び出し (INVOKE)』
650 ページの『ファクトリー・メソッドの定義』

関連参照
The Java Language Specification (Overloading)

属性 (get および set) メソッドのコーディング

X のための accessor (get) メソッドおよび mutator (set) メソッドをコーディングして X を定義するクラスの外側から、インスタンス変数 X へのアクセスを提供することができます。

COBOL のインスタンス変数は、*private* です。インスタンス変数を定義するクラスは、インスタンス変数を完全にカプセル化します。したがって、直接アクセスできるのは、同じ OBJECT 段落で定義するインスタンス・メソッドだけです。通常、優れた設計のオブジェクト指向アプリケーションは、クラスの外側からインスタンス変数にアクセスする必要はありません。

public インスタンス変数の概念は、Java および他のオブジェクト指向言語で定義されており、クラス属性の概念は CORBA で定義されていますが、どちらの概念も COBOL には直接サポートされていません。(CORBA *属性* は、変数が読み取り専用でない場合に、変数の値にアクセスするための自動生成 get メソッドと変数の値を変更するための自動生成 set メソッドを持つ、インスタンス変数です。)

629 ページの『例: ゲット・メソッドのコーディング』

関連タスク 620 ページの『クラス・インスタンス・データ定義用の WORKING-STORAGE SECTION』 21 ページの『データの処理』

例: ゲット・メソッドのコーディング

次の例は、インスタンス変数 AccountBalance の値をクライアントに戻すインスタンス・メソッド getBalance の、Account クラスにおける定義を示しています。getBalance および AccountBalance は、Account クラス定義の OBJECT 段落で定義されます。

```
 Identification Division.
 Class-id. Account inherits Base.
*  (ENVIRONMENT DIVISION not shown)
*  (FACTORY paragraph not shown)
*
 Identification division.
 Object.
  Data division.
  Working-storage section.
  01 AccountBalance pic S9(9) value zero.
*   (Other instance data not shown)
*
  Procedure Division.
*
   Identification Division.
   Method-id. "getBalance".
   Data division.
   Linkage section.
   01 outBalance pic S9(9) binary.
*
   Procedure Division returning outBalance.
     Move AccountBalance to outBalance.
   End method "getBalance".
*
*  (Other instance methods not shown)
 End Object.
*
 End class Account.
```

例: メソッドの定義

次の例は、直前の例に、Account クラスのインスタンス・メソッド定義を追加し、Java Check クラスの定義を示します。

(直前の例とは、 621 ページの『例: クラスの定義』のことです。)

Account クラス

```
 cbl dll,thread,pgmname(longmixed)
 Identification Division.
 Class-id. Account inherits Base.
 Environment Division.
 Configuration section.
 Repository.
     Class Base    is "java.lang.Object"
     Class Account is "Account".
*
*  (FACTORY paragraph not shown)
*
 Identification division.
 Object.
  Data division.
  Working-storage section.
  01 AccountNumber  pic 9(6).
  01 AccountBalance pic S9(9) value zero.
*
```

```
  Procedure Division.
*
*    init method to initialize the account:
   Identification Division.
   Method-id. "init".
   Data division.
   Linkage section.
   01 inAccountNumber pic S9(9) binary.
   Procedure Division using by value inAccountNumber.
     Move inAccountNumber to AccountNumber.
   End method "init".
*
*    getBalance method to return the account balance:
   Identification Division.
   Method-id. "getBalance".
   Data division.
   Linkage section.
   01 outBalance pic S9(9) binary.
   Procedure Division returning outBalance.
     Move AccountBalance to outBalance.
   End method "getBalance".
*
*    credit method to deposit to the account:
   Identification Division.
   Method-id. "credit".
   Data division.
   Linkage section.
   01 inCredit   pic S9(9) binary.
   Procedure Division using by value inCredit.
     Add inCredit to AccountBalance.
   End method "credit".
*
*    debit method to withdraw from the account:
   Identification Division.
   Method-id. "debit".
   Data division.
   Linkage section.
   01 inDebit    pic S9(9) binary.
   Procedure Division using by value inDebit.
     Subtract inDebit from AccountBalance.
   End method "debit".
*
*    print method to display formatted account number and balance:
   Identification Division.
   Method-id. "print".
   Data division.
   Local-storage section.
   01 PrintableAccountNumber  pic ZZZZZZ999999.
   01 PrintableAccountBalance pic $$$$,$$$,$$9CR.
   Procedure Division.
     Move AccountNumber  to PrintableAccountNumber
     Move AccountBalance to PrintableAccountBalance
     Display " Account: " PrintableAccountNumber
     Display " Balance: " PrintableAccountBalance.
   End method "print".
*
 End Object.
*
 End class Account.
```

Check クラス

```
/**
 * A Java class for check information
 */
public class Check {
  private CheckingAccount payer;
```

```
  private Account         payee;
  private int             amount;

  public Check(CheckingAccount inPayer, Account inPayee, int inAmount) {
    payer=inPayer;
    payee=inPayee;
    amount=inAmount;
  }

  public int getAmount() {
    return amount;
  }

  public Account getPayee() {
    return payee;
  }
}
```

関連タスク 329 ページの『第 16 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行』

クライアントの定義

クラス内で 1 つ以上のメソッドからサービスを要求するプログラムまたはメソッドは、そのクラスの *クライアント* と呼ばれます。

COBOL クライアントまたは Java クライアントで、以下のことを行うことができます。

- Java および COBOL クラスのオブジェクト・インスタンスを作成する。
- Java および COBOL オブジェクトにインスタンス・メソッドを呼び出す。
- COBOL ファクトリー・メソッドおよび Java 静的メソッドを呼び出す。

COBOL クライアントでは、Java Native Interface (JNI) が提供するサービスを呼び出すこともできます。

COBOL クライアント・プログラムは、次のような通常の 4 つの部分で成り立っています。

表 77. **COBOL クライアントの構成**

除算	目的	構文
IDENTIFICATION (必須)	クライアントの名前。	通常のようにコーディング。ただし、クライアント・プログラムは以下のとおりにする必要があります。 • 再帰的 (PROGRAM-ID 段落内の RECURSIVE 宣言) • スレッド対応 (THREAD オプションでコンパイル、スレッド化アプリケーション用コーディングの指針に準拠)

表 77. **COBOL クライアントの構成** *(続き)*

除算	目的	構文
ENVIRONMENT (必須)	コンピューター環境を記述する。クライアント内で使用されるクラス名を、コンパイル単位の外側で判明している、対応する外部クラス名に関連付ける。	CONFIGURATION SECTION (必須) 『クライアント定義用の REPOSITORY 段落』 (必須)
DATA (オプション)	クライアントが必要とするデータを記述する。	633 ページの『クライアント定義用の DATA DIVISION』 (オプション)
PROCEDURE (オプション)	クラスのインスタンスの作成、オブジェクト参照データ項目の操作、メソッドの呼び出し。	INVOKE、IF、および SET ステートメントを使用してコーディング

THREAD コンパイラー・オプションを使用して、オブジェクト指向構文を含む、または Java と相互協調処理する、すべての COBOL プログラムをコンパイルする必要があるため、以下の言語エレメントは COBOL クライアントで使用できません。

- SORT または MERGE ステートメント
- ネストされたプログラム

THREAD コンパイラー・オプションを使用してコンパイルするプログラムは、再帰的である必要があります。各 OO COBOL クライアント・プログラムの PROGRAM-ID 段落で、RECURSIVE 節を指定しなければなりません。

643 ページの『例: クライアントの定義』

関連タスク 329 ページの『第 16 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行』 549 ページの『第 27 章 マルチスレッド化のための COBOL プログラムの準備』 663 ページの『第 31 章 Java メソッドとの通信』 669 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』 641 ページの『クラスのインスタンスの作成および初期化』 635 ページの『オブジェクト参照の比較および設定』 636 ページの『メソッドの呼び出し (INVOKE)』 652 ページの『ファクトリー・メソッドまたは静的メソッドの呼び出し』

関連参照
396 ページの『THREAD』

クライアント定義用の REPOSITORY 段落

指定された語を COBOL クライアント内で使用するときにその語がクラス名であることをコンパイラーに宣言する場合、さらに必要に応じてクラス名を対応する外部クラス名 (コンパイル単位の外側で認識されているクラス名) に関係付ける場合に、REPOSITORY 段落を使用してください。

外部クラス名は大/小文字が区別されます。したがって、Java 形成規則に準拠しなければなりません。例えば、Account および Check クラスを使用するクライアント・プログラムにおいて、以下のようにコーディングすることがあります。

```
Environment division. 必要
Configuration section. 必要
  Source-Computer.  IBM-390.
  Object-Computer.  IBM-390.
Repository. 必要
    Class Account is "Account"
    Class Check   is "Check".
```

REPOSITORY 段落記入項目では、クライアント内で Account および Check として参照されるクラスの外部クラス名は、それぞれ、Account および Check であることが示されます。

REPOSITORY 段落では、クライアントにおいて明示的に参照するそれぞれのクラス名ごとに記入項目をコーディングする必要があります。REPOSITORY 段落記入項目において、名前に非 COBOL 文字が含まれている場合には、外部クラス名を指定しなければなりません。

Java パッケージの一部である参照クラスごとに外部クラス名を指定しなければなりません。そのようなクラスごとに、外部クラス名をパッケージの完全修飾名として指定し、後にピリオド (.) が付き、続いてその後に Java クラスの単純名が付きます。

REPOSITORY 段落で指定する外部クラス名は、完全修飾 Java クラス名の形成規則に準拠した英数字リテラルでなければなりません。

REPOSITORY 段落記入項目に外部クラス名を組み込まない場合、外部クラス名の作成は、クラス定義で外部クラス名が REPOSITORY 段落記入項目に組み込まれていない場合と同じ方法で、クラス名から行われます。外部名は英大/小文字混合で名前付けされます。したがって、上記の例で、クラス Account およびクラス Check は、それぞれ、外部的には Account および Check (英大/小文字混合) として認識されます。

CONFIGURATION SECTION の SOURCE-COMPUTER、OBJECT-COMPUTER、および SPECIAL-NAMES 段落はオプションです。

関連タスク
619 ページの『クラス定義用の REPOSITORY 段落』

関連参照
REPOSITORY 段落 (*Enterprise COBOL 言語解説書*)
The Java Language Specification (Identifiers)
The Java Language Specification (Packages)

クライアント定義用の DATA DIVISION

クライアントが必要とするデータを記述するには、DATA DIVISION の任意のセクションを使用できます。

```
Data Division.
Local-storage section.
01  anAccount        usage object reference Account.
01  aCheckingAccount usage object reference CheckingAccount.
01  aCheck           usage object reference Check.
01  payee            usage object reference Account.
. . .
```

クライアントはクラスを参照するので、*オブジェクト参照* (つまり、クラスのインスタンスへの参照) と呼ばれる 1 つ以上の特別なデータ項目を必要とします。インスタンス・メソッドへのすべての要求は、メソッドがサポートされる (すなわち、継承によって定義されているか、または使用可能である) クラスのインスタンスへのオブジェクト参照が必要です。COBOL クラスのインスタンスを参照する場合と同じ構文を使用して、オブジェクト参照をコーディングして Java クラスのインスタンスを参照します。上記の例では、`usage object reference` という句は、オブジェクト参照データ項目を表します。

上記コードの 4 つのオブジェクト参照はすべて、クラス名が `OBJECT REFERENCE` 句の後に現れるので、*型式化*オブジェクト参照と呼ばれます。型式化オブジェクト参照の参照先は、`OBJECT REFERENCE` 句で名前付けしたクラスのインスタンス、またはそのサブクラスのいずれか 1 つに限られます。したがって、anAccount は、Account クラスのインスタンス、またはそのサブクラスのいずれかを参照できますが、ほかのクラスのインスタンスを参照することはできません。同様に、aCheck の参照先は、Check クラスのインスタンス、またはそのサブクラスのインスタンスに限られます。

別の型のオブジェクト参照 (上に示されていない) には、`OBJECT REFERENCE` 句の後にクラス名がありません。そのような参照を*汎用* オブジェクト参照と呼びます。すべてのクラスのインスタンスを参照できるという意味です。汎用オブジェクト参照は、非常に限られた環境 (`INVOKE` *`class-name`* `NEW . . .` ステートメントの `RETURNING` 句で使用するとき) でしか Java との相互運用が可能ではないので、汎用オブジェクト参照をコーディングするのは避けてください。

`OBJECT REFERENCE` 句で使用するクラス名は、`CONFIGURATION SECTION` の `REPOSITORY` 段落で定義しなければなりません。

関連タスク 『LOCAL-STORAGE または WORKING-STORAGE の選択』 669 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』 636 ページの『メソッドの呼び出し (INVOKE)』 632 ページの『クライアント定義用の REPOSITORY 段落』

関連参照
RETURNING 句 (*Enterprise COBOL 言語解説書*)

LOCAL-STORAGE または WORKING-STORAGE の選択

一般に、クライアント・プログラムが必要とする作業データを定義する場合に、`WORKING-STORAGE SECTION` を使用できます。しかし、プログラムがマルチスレッドで同時に実行できるような場合、代わりに `LOCAL-STORAGE SECTION` にデータを定義することができます。

各スレッドは、`LOCAL-STORAGE` データの別々のコピーへのアクセス権を持っていますが、`WORKING-STORAGE` データの単一のコピーへのアクセス権は共用します。`WORKING-STORAGE SECTION` でデータを定義する場合には、データへのアクセスを同期化するか、または、2 つのスレッドが同時にそのデータへアクセスしないようにする必要があります。

関連タスク
549 ページの『第 27 章 マルチスレッド化のための COBOL プログラムの準備』

オブジェクト参照の比較および設定

条件ステートメントまたは JNI サービス IsSameObject の呼び出しをコーディングすることによってオブジェクト参照を比較できます。また、SET ステートメントを使用して、オブジェクト参照を設定できます。

例えば、次の IF ステートメントのいずれかをコーディングして、オブジェクト参照 anAccount がオブジェクト・インスタンスをまったく参照していないことをチェックします。

```
If anAccount = Null . . .
If anAccount = Nulls . . .
```

IsSameObject の呼び出しをコーディングして、2 つのオブジェクト参照 (object1 と object2) が同じオブジェクト・インスタンスを指すかどうか、あるいはそれぞれがオブジェクト・インスタンスを指さないかどうかを検査することができます。引数および戻り値が Java と相互運用可能であることを確認し、呼び出し可能サービスへのアドレス可能度を確立するには、IsSameObject への呼び出しの前に、以下のデータ定義およびステートメントをコーディングします。

```
Local-storage Section.
. . .
01  is-same Pic X.
    88 is-same-false Value X'00'.
    88 is-same-true  Value X'01' Through X'FF'.
Linkage Section.
    Copy JNI.
Procedure Division.
    Set Address Of JNIEnv To JNIEnvPtr
    Set Address Of JNINativeInterface To JNIEnv
    Call IsSameObject Using By Value JNIEnvPtr object1 object2
                      Returning is-same
    If is-same-true . . .
```

メソッド内では、オブジェクト参照と SELF を比較する IsSameObject の呼び出しをコーディングすることにより、メソッドが呼び出されたオブジェクト・インスタンスをオブジェクト参照が指すかどうかを検査できます。

上記の代わりに、Java equals メソッド (java.lang.Object からの継承) を呼び出して、2 つのオブジェクト参照が同一のオブジェクト・インスタンスを参照するかを決定することができます。

SET ステートメントを使用することにより、オブジェクト参照がオブジェクト・インスタンスを指さないようにすることができます。以下に例を示します。

```
Set anAccount To Null.
```

また、SET ステートメントを使用して、1 つのオブジェクト参照が別のオブジェクト参照と同じインスタンスを参照するように設定することもできます。以下に例を示します。

```
Set anotherAccount To anAccount.
```

この SET ステートメントによって、anotherAccount は anAccount と同じオブジェクト・インスタンスを参照します。受け取り側 (anotherAccount) が汎用オブジェクト参照である場合、送り出し側 (anAccount) は、汎用オブジェクト参照か、または型式化オブジェクト参照のどちらかになります。受け取り側が型式化オブジェクト参照である場合は、送り出し側も、受け取り側と同じクラス、または、そのサブクラスのいずれか 1 つにバインドされた、型式化オブジェクト参照でなければなりません。

メソッド内では、オブジェクト参照を SELF に設定して、メソッドが呼び出されたオブジェクト・インスタンスをオブジェクト参照が参照するように設定することができます。以下に例を示します。

```
Set anAccount To Self.
```

関連タスク 669 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』 663 ページの『JNI サービスへのアクセス』

関連参照
The Java Native Interface (IsSameObject)

メソッドの呼び出し (INVOKE)

Java クライアントでは、COBOL でインプリメントされたクラスのオブジェクト・インスタンスを作成し、標準 Java 構文を使用して、それらのオブジェクトでメソッドを呼び出すことができます。 COBOL クライアントでは、INVOKE ステートメントをコーディングすることにより、Java または COBOL クラスで定義されたメソッドを呼び出すことができます。

```
Invoke Account "createAccount"
   using by value 123456
   returning anAccount
Invoke anAccount "credit" using by value 500.
```

上の最初の例の INVOKE ステートメントは、クラス名 Account を使用して、createAccount という名前のメソッドを呼び出します。このメソッドは、Account クラスで定義または継承されている必要があります。また、次のいずれかの型である必要があります。

- Java 静的メソッド
- COBOL ファクトリー・メソッド

using by value 123456 という句は、123456 が、メソッドの入力引数であり、値により受け渡されることを表します。入力引数 123456 と戻されるデータ項目 anAccount は、(多重定義されているかもしれない) createAccount メソッドの仮パラメーターおよび戻りの型の定義にそれぞれ準拠している必要があります。

2 番目の INVOKE ステートメントは、戻されるオブジェクト参照 anAccount を使用して、インスタンス・メソッド credit (Account クラスに定義されている) を呼び出します。入力引数 500 は、(おそらく多重定義された) credit メソッドの仮パラメーターの定義に準拠しなければなりません。

実行時におけるその値がターゲット・メソッドのシグニチャー内のメソッド名と一致するリテラルとして、または ID として呼び出すメソッドの名前をコーディング

します。メソッド名は、英数字または国別リテラルであるか、あるいはカテゴリー英字、英数字、または国別のデータ項目でなければならず、解釈されるときには大/小文字が区別されます。

INVOKE ステートメントを (上記の 2 番目の例のステートメントのように) オブジェクト参照を使用してコーディングする場合、そのステートメントは次の 2 つの形式のうちのいずれかで始まります。

```
Invoke objRef "literal-name"  . . .
Invoke objRef identifier-name . . .
```

メソッド名が ID である場合には、オブジェクト参照 (objRef) を、指定する型のない USAGE OBJECT REFERENCE として、すなわち、汎用オブジェクト参照として、定義しなければなりません。

呼び出されたメソッドが、オブジェクト参照の参照先のクラスでサポートされない場合、重大度 3 の言語環境プログラム条件が実行時に発生します。ただし、INVOKE ステートメントで ON EXCEPTION 句をコーディングした場合は別です。

オプションの範囲終了符号 END-INVOKE を INVOKE ステートメントで使用することができます。

INVOKE ステートメントは RETURN-CODE 特殊レジスターを設定しません。

関連タスク 『引数の引き渡し用の USING 句』 640 ページの『戻り値の取得用の RETURNING 句』 625 ページの『クラス・インスタンス・メソッド定義用の PROCEDURE DIVISION』 669 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』 640 ページの『オーバーライドされたスーパークラス・メソッドの呼び出し』 652 ページの『ファクトリー・メソッドまたは静的メソッドの呼び出し』

関連参照
INVOKE ステートメント (*Enterprise COBOL 言語解説書*)

引数の引き渡し用の USING 句

引数をメソッドに渡す場合、INVOKE ステートメントの USING 句に引数を指定してください。それぞれの引数のデータ型が、意図されたターゲット・メソッドの対応する仮パラメーターの型と一致するように、それぞれの引数のデータ型をコーディングしてください。

表 78. **COBOL クライアントでの引数の合致**

ターゲット・メソッドのプログラミング言語	引数はオブジェクト参照ですか	引数の DATA DIVISION 定義を次のようにコーディングします	制約事項
COBOL	いいえ	対応する仮パラメーターの定義と同じ	
Java	いいえ	対応する Java パラメーターと相互運用可能	

表 78. **COBOL クライアントでの引数の合致** *(続き)*

ターゲット・メソッドのプログラミング言語	引数はオブジェクト参照ですか	引数の DATA DIVISION 定義を次のようにコーディングします	制約事項
COBOL または Java	はい	ターゲット・メソッドの対応するパラメーターと同じクラスに型式化されるオブジェクト参照	COBOL クライアントでは (Java クライアントとは異なり)、引数のクラスを、対応するパラメーターのクラスのサブクラスにすることができません。

SET ステートメントまたは REDEFINES 節を使用して、オブジェクト参照引数を対応する仮パラメーターの型と一致させる方法については、以下に参照されている例を参照してください。

『例: COBOL クライアントからの規格合致オブジェクト参照の引数の引き渡し』

ターゲット・メソッドが多重定義されている場合、引数のデータ型は、同じ名前を持つメソッドの中から選択するために使用されます。

引数が BY VALUE で渡される指定をしなければなりません。言い換えれば、引数は、呼び出されるメソッドの対応する仮パラメーターが変更されても影響を受けません。

各引数のデータ型は、Java と相互運用可能な型のいずれかでなければなりません。

関連タスク 625 ページの『クラス・インスタンス・メソッド定義用の PROCEDURE DIVISION』 627 ページの『インスタンス・メソッドの多重定義』 669 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』 521 ページの『データの受け渡し』

関連参照
INVOKE ステートメント (*Enterprise COBOL 言語解説書*)
SET ステートメント (*Enterprise COBOL 言語解説書*)
REDEFINES 節 (*Enterprise COBOL 言語解説書*)

例: COBOL クライアントからの規格合致オブジェクト参照の引数の引き渡し

以下の例は、COBOL クライアントのオブジェクト参照引数を、呼び出されるメソッドに対応する仮パラメーターの予想クラスに合致させる方法を示しています。

クラス C は、1 つのパラメーター (クラス java.lang.Object のオブジェクトへの参照) を持つメソッド M を定義します。

```
. . .
Class-id. C inherits Base.
. . .
Repository.
    Class Base       is "java.lang.Object"
    Class JavaObject is "java.lang.Object".
Identification division.
```

```
 Factory.
 . . .
  Procedure Division.
   Identification Division.
   Method-id. "M".
   Data division.
   Linkage section.
   01 obj object reference JavaObject.
   Procedure Division using by value obj.
   . . .
```

メソッド M を呼び出すには、COBOL クライアントは、クラス java.lang.Object のオブジェクトへの参照である引数を渡す必要があります。以下のクライアントは、データ項目 aString を定義していますが、これを M に引数として渡すことができません。aString は、クラス java.lang.String のオブジェクトへの参照だからです。クライアントはまず SET ステートメントを使用して、aString をデータ項目 anObj (クラス java.lang.Object のオブジェクトへの参照) に割り当てます。 (java.lang.String は java.lang.Object のサブクラスなので、この SET ステートメントは正しいものです。) その後クライアントは anObj を引数として M に渡します。

```
 . . .
 Repository.
     Class jstring    is "java.lang.String"
     Class JavaObject is "java.lang.Object".
 Data division.
 Local-storage section.
 01  aString object reference jstring.
 01  anObj   object reference JavaObject.
*
 Procedure division.
     . . . (statements here assign a value to aString)
     Set anObj to aString
     Invoke C "M"
       using by value anObj
```

SET ステートメントを使用して、クラス java.lang.Object のオブジェクトへの参照として anObj を取得する代わりに、クライアントは、以下のように REDEFINES 節で aString および anObj を定義することができます。

```
 . . .
 01  aString object reference jstring.
 01  anObj   redefines aString object reference JavaObject.
```

クライアントが値をデータ項目 aString (つまり、クラス java.lang.String のオブジェクトへの有効な参照) を割り当てた後、anObj を引数として M に渡すことができます。REDEFINES 節を使用して引数を合致させる例については、以下に参照されている例を参照してください。

676 ページの『例: COBOL で書かれた J2EE クライアント』

関連タスク 669 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』 625 ページの『クラス・インスタンス・メソッド定義用の PROCEDURE DIVISION』

関連参照 INVOKE ステートメント (*Enterprise COBOL 言語解説書*) SET ステートメント (*Enterprise COBOL 言語解説書*) REDEFINES 節 (*Enterprise COBOL 言語解説書*)

戻り値の取得用の RETURNING 句

データ項目がメソッドの結果として戻される場合、その項目を、INVOKE ステートメントの RETURNING 句に指定してください。戻される項目は、クライアントの DATA DIVISION で定義します。

INVOKE ステートメントの RETURNING 句に指定する項目は、以下の表に示すように、ターゲット・メソッドが戻す型と合致している必要があります。

表 79. **COBOL クライアントでの戻されるデータ項目の合致**

ターゲット・メソッドのプログラミング言語	戻される項目はオブジェクト参照ですか	戻される項目の DATA DIVISION 定義を次のようにコーディングします
COBOL	いいえ	ターゲット・メソッドの RETURNING 項目の定義と同じ
Java	いいえ	戻された Java データ項目と相互運用可能
COBOL または Java	はい	ターゲット・メソッドが戻すオブジェクト参照と同じクラスに型式化されるオブジェクト参照

すべての場合において、戻り値のデータ型は、Java と相互運用可能になる型のうちのいずれかでなければなりません。

関連タスク 669 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』

関連参照
INVOKE ステートメント (*Enterprise COBOL 言語解説書*)

オーバーライドされたスーパークラス・メソッドの呼び出し

ときおりクラス内で、現行クラスで定義された同じシグニチャーを持つメソッドを呼び出す代わりに、オーバーライドされたスーパークラス・メソッドを呼び出さなければならないことがあります。

例えば、CheckingAccount クラスが、その即時スーパークラス Account に定義されている debit インスタンス・メソッドをオーバーライドすると想定します。次のステートメントをコーディングして、CheckingAccount クラスのメソッド内で Account の debit メソッドを呼び出すことができます。

```
Invoke Super "debit" Using By Value amount.
```

debit メソッドのシグニチャーに一致するように、amount を PIC S9(9) BINARY として定義します。

CheckingAccount クラスは、Account クラスに定義されている print メソッドをオーバーライドします。print メソッドには仮パラメーターがありません。したがって、CheckingAccount クラス内のメソッドは、次のステートメントでスーパークラス print メソッドを呼び出すことができます。

```
Invoke Super "print".
```

キーワード SUPER は、現行クラス内のメソッドではなく、スーパークラス・メソッドを呼び出すことを指示します。 (SUPER は、現在実行中のメソッドの起動に使用されているオブジェクトへの暗黙の参照です。)

614 ページの『例: 口座』

関連タスク
626 ページの『インスタンス・メソッドのオーバーライド』

関連参照
INVOKE ステートメント (*Enterprise COBOL 言語解説書*)

クラスのインスタンスの作成および初期化

Java または COBOL クラスで定義されたインスタンス・メソッドを使用するには、まずクラスのインスタンスを作成する必要があります。

クラス *class-name* の新しいインスタンスを生成するには、また、作成したオブジェクトへの参照 *object-reference* を取得するには、次の形式のステートメントをコーディングします (*object-reference* は、クライアントの DATA DIVISION で定義されます)。

```
INVOKE class-name NEW . . . RETURNING object-reference
```

メソッド内に INVOKE . . . NEW ステートメントをコーディングしており、戻されたオブジェクト参照の使用がメソッドの起動の期間に限定されていない場合は、JNI サービス NewGlobalRef を呼び出すことによって、戻されたオブジェクト参照をグローバル参照に変換しなければなりません。

```
Call NewGlobalRef using by value JNIEnvPtr object-reference
                  object-reference の戻り
```

NewGlobalRef を呼び出さない場合には、戻されたオブジェクト参照はあくまでもローカル参照にすぎないため、メソッドが戻った後で自動的に解放されます。

関連タスク
『Java クラスのインスタンス化』
642 ページの『COBOL クラスのインスタンス化』
663 ページの『JNI サービスへのアクセス』
666 ページの『ローカル参照とグローバル参照の管理』
633 ページの『クライアント定義用の DATA DIVISION』
636 ページの『メソッドの呼び出し (INVOKE)』
669 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』

関連参照
INVOKE ステートメント (*Enterprise COBOL 言語解説書*)

Java クラスのインスタンス化

Java クラスをインスタンス化するには、 INVOKE . . . NEW ステートメントの USING 句を RETURNING 句の直前にコーディングし、コンストラクターのシグニチャーと一致する引数の数および型を BY VALUE で渡すことにより、クラスがサポートするパラメーター化コンストラクターを呼び出します。

各引数のデータ型は、Java と相互運用可能な型のいずれかでなければなりません。デフォルトの (パラメーターなし) コンストラクターを呼び出すには、USING 句を省略します。

例えば、Check クラスのインスタンスを生成し、そのインスタンス・データを初期化し、生成した Check インスタンスへの参照 aCheck を取得するには、次のステートメントを COBOL クライアントにコーディングすることができます。

```
Invoke Check New
  using by value aCheckingAccount, payee, 125
  returning aCheck
```

関連タスク

636 ページの『メソッドの呼び出し (INVOKE)』
669 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』

関連参照

VALUE 節 (*Enterprise COBOL 言語解説書*)
INVOKE ステートメント (*Enterprise COBOL 言語解説書*)

COBOL クラスのインスタンス化

COBOL クラスをインスタンス化するには、型式化オブジェクト参照または汎用オブジェクト参照を、 INVOKE . . . NEW ステートメントの RETURNING 句に指定できます。ただし、USING 句をコーディングすることはできません。インスタンス・データは、クラス定義の VALUE 節に指定されたとおりに初期化されます。

したがって、INVOKE . . . NEW ステートメントは、単一のインスタンス・データのみを有する COBOL クラスのインスタンスを生成するときに役立ちます。例えば、次のステートメントは、Account クラスのインスタンスを作成し、Account クラス定義の OBJECT 段落の WORKING-STORAGE SECTION の VALUE 節に指定されたとおりに、インスタンス・データを初期化し、新しいインスタンスへの参照 outAccount を提供します。

```
Invoke Account New returning outAccount
```

VALUE 節だけを使用して初期化することができない COBOL クラス・データの初期化を可能にするには、COBOL クラスを設計する際に、FACTORY 段落にパラメーター化生成メソッドを定義し、OBJECT 段落にパラメーター化初期化メソッドを定義する必要があります。

1. パラメーター化ファクトリー生成メソッドで、以下の手順を実行します。
   a. INVOKE *class-name* NEW RETURNING *objectRef* をコーディングして、 *class-name* のインスタンスを作成し、VALUE 節を有するインスタンス・データ項目に初期値を与えます。
   b. パラメーター化した初期化メソッドをインスタンス (*objectRef*) 上で呼び出し、指定された引数 BY VALUE をファクトリー・メソッドに渡します。
2. 初期化メソッドで、ロジックをコーディングし、仮パラメーターを介して指定された値を使用して、インスタンス・データ初期化を完了します。

COBOL クラスのインスタンスを作成して適切に初期化するために、クライアントはパラメーター化ファクトリー・メソッドを呼び出し、BY VALUE で目的の引数を渡します。クライアントに戻されるオブジェクト参照は、ローカル参照です。メソッ

ド内にクライアント・コードがあり、戻されるオブジェクト参照を使用するのがそのメソッドの存続期間に限定されない場合、クライアント・コードは、JNI サービス NewGlobalRef を呼び出すことによって、戻されるオブジェクト参照をグローバル参照に変換しなければなりません。

653 ページの『例: ファクトリーの定義 (メソッドに関して)』

関連タスク
663 ページの『JNI サービスへのアクセス』
666 ページの『ローカル参照とグローバル参照の管理』
636 ページの『メソッドの呼び出し (INVOKE)』
648 ページの『ファクトリー・セクションの定義』

関連参照
VALUE 節 (*Enterprise COBOL 言語解説書*)
INVOKE ステートメント (*Enterprise COBOL 言語解説書*)

クラスのインスタンスの解放

任意のクラスの個々のオブジェクト・インスタンスを解放するために、アクションを実行する必要はありません。したがって、オブジェクト・インスタンスを解放するために有効な構文はありません。Java ランタイム・システムは自動的に *ガーベッジ・コレクション* を実行します。すなわち、使用されなくなったオブジェクトのメモリーを再利用します。

ただし、参照済みオブジェクトのガーベッジ・コレクションを許可するために、ネイティブ COBOL クライアント内のオブジェクトへのローカル参照またはグローバル参照を明示的に解放する必要がある場合があります。

関連タスク
666 ページの『ローカル参照とグローバル参照の管理』

例: クライアントの定義

次の例では、Account クラスの小さいクライアント・プログラムを示します。

プログラムは以下を実行します。

- ファクトリー・メソッド createAccount を呼び出して、デフォルト収支ゼロの Account インスタンスを作成します。
- インスタンス・メソッド `credit` を呼び出して、$500 をこの新規の口座に預金します。
- インスタンス・メソッド `print` を呼び出して、口座の状況を表示します。

(Account クラスは、629 ページの『例: メソッドの定義』に示されています。)

```
cbl dll,thread,pgmname(longmixed)
Identification division.
Program-id. "TestAccounts" recursive.
Environment division.
Configuration section.
Repository.
    Class Account is "Account".
```

```
 Data Division.
*  Working data is declared in LOCAL-STORAGE instead of
*  WORKING-STORAGE so that each thread has its own copy:
 Local-storage section.
 01  anAccount usage object reference Account.
*
 Procedure division.
 Test-Account-section.
     Display "Test Account class"
*  Create account 123456 with 0 balance:
     Invoke Account "createAccount"
       using by value 123456
       returning anAccount
*  Deposit 500 to the account:
     Invoke anAccount "credit" using by value 500
     Invoke anAccount "print"
     Display space
*
     Stop Run.
 End program "TestAccounts".
```

653 ページの『例: ファクトリーの定義 (メソッドに関して)』

関連タスク 650 ページの『ファクトリー・メソッドの定義』 652 ページの『ファクトリー・メソッドまたは静的メソッドの呼び出し』 329 ページの『第 16 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行』

サブクラスの定義

クラス (サブクラス、派生クラス、または子クラスと呼ばれる) を別のクラス (スーパークラス、基本クラス、または親クラスと呼ばれる) の特殊化クラスにすることができます。

サブクラスは、そのスーパークラスのメソッドおよびインスタンス・データを継承し、*is-a* 関係によって、そのスーパークラスに関連付けられています。例えば、サブクラス P がスーパークラス Q から継承し、サブクラス Q がスーパークラス S から継承した場合、P のインスタンスは Q のインスタンスであり、また (推移性によって) S のインスタンスでもあります。したがって、P のインスタンスは、Q と S のメソッドおよびデータを継承します。

サブクラスを使用することの利点:

- コードの再利用: 継承を通じて、サブクラスは、スーパークラスに既に存在するメソッドを再利用することができます。
- 特化: サブクラスでは、スーパークラスが処理しないケースを処理するために新規のメソッドを追加することができます。また、スーパークラスが必要としない新規のデータ項目を追加することもできます。
- アクションの変更: サブクラスは、スーパークラスにあるシグニチャーと同じシグニチャーのメソッドを定義して、スーパークラスから継承するメソッドをオーバーライドすることができます。メソッドをオーバーライドするときは、メソッドの実行内容について、いくつかの小さい変更を行うだけの場合、または全面的な変更を行う場合があります。

制約事項: COBOL プログラムでは*多重継承* を使用することはできません。定義する各 COBOL クラスには、Java または COBOL でインプリメントされた即時スー

パークラスは必ず 1 つだけでなければなりません。また、それぞれのクラスは、直接的または間接的に java.lang.Object から派生したものでなければなりません。継承のセマンティクスは、Java によって定義されます。

サブクラスの構造および構文は、クラス定義の構造および構文と同一です。サブクラス定義の OBJECT 段落内で、それぞれ、DATA DIVISION および PROCEDURE DIVISION に、インスタンス・データとメソッドを定義します。個別のオブジェクト・インスタンスにではなく、サブクラス自体に関連付けるデータとメソッドを必要とするサブクラスに、サブクラス定義の FACTORY 段落内で、別々の DATA DIVISION および PROCEDURE DIVISION を定義します。

COBOL インスタンス・データは private です。サブクラスが COBOL スーパークラスのインスタンス・データにアクセスすることができるのは、スーパークラスがそのアクセスを可能にするために属性 (get または set) インスタンス・メソッドを定義する場合に限られます。

614 ページの『例: 口座』
647 ページの『例: サブクラスの定義 (メソッドに関して)』

関連タスク

617 ページの『クラスの定義』
626 ページの『インスタンス・メソッドのオーバーライド』
628 ページの『属性 (get および set) メソッドのコーディング』
647 ページの『サブクラス・インスタンス・メソッドの定義』
648 ページの『ファクトリー・セクションの定義』

関連参照

The Java Language Specification (Inheritance, overriding, and hiding)
COBOL クラス定義構成 (*Enterprise COBOL 言語解説書*)

サブクラス定義用の CLASS-ID 段落

サブクラスを指定し、それが特性を継承する直接の Java または COBOL スーパークラスを示すには、CLASS-ID 段落を使用してください。

```
   Identification Division.
必要
 Class-id. CheckingAccount inherits Account.
必要
```

上記の例で、定義されるサブクラスは CheckingAccount です。CheckingAccount は、サブクラス定義において Account として認識されているクラスのすべてのメソッドを継承します。CheckingAccount メソッドが Account インスタンス・データにアクセスできるのは、Account クラスが、そのアクセスを可能にするために、属性 (get または set) メソッドを指定する場合に限られます。

ENVIRONMENT DIVISION の CONFIGURATION SECTION の REPOSITORY 段落に、即時スーパークラスの名前を指定しなければなりません。オプションとして、外部で認識されているクラスの名前にスーパークラス名を関連付けることができます。また、定義中のサブクラスの名前 (上記の例の CheckingAccount) を REPOSITORY 段落に指定し、その対応する外部クラス名にそれを関連付けることもできます。

関連タスク
618 ページの『クラス定義用の CLASS-ID 段落』
628 ページの『属性 (get および set) メソッドのコーディング』
『サブクラス定義用の REPOSITORY 段落』

サブクラス定義用の REPOSITORY 段落

指定された語をサブクラス定義内で使用するときにその語がクラス名であることをコンパイラーに宣言する場合、さらに必要に応じてクラス名を対応する外部クラス名 (コンパイル単位の外側で認識されているクラス名) に関係付ける場合に、REPOSITORY 段落を使用してください。

例えば、CheckingAccount サブクラス定義では、これらの REPOSITORY 段落記入項目は、サブクラス定義内で CheckingAccount、Check、および Account として参照されるクラスの外部クラス名が、それぞれ、CheckingAccount、 Check、および Account であることを示します。

```
Environment Division. 必要
Configuration Section. 必要
Repository. 必要
    Class CheckingAccount is "CheckingAccount" Optional
    Class Check           is "Check"           Required
    Class Account         is "Account". 必要
```

REPOSITORY 段落では、サブクラス定義において明示的に参照するそれぞれのクラス名ごとに記入項目をコーディングする必要があります。以下に例を示します。

- 定義中のサブクラスが継承する元のユーザー定義スーパークラス
- サブクラス定義内のメソッドで参照するクラス

サブクラス内の REPOSITORY 段落記入項目をコーディングする場合の規則は、クラス内の REPOSITORY 段落記入項目をコーディングする場合の規則と同一です。

関連タスク
619 ページの『クラス定義用の REPOSITORY 段落』

関連参照
REPOSITORY 段落 (*Enterprise COBOL 言語解説書*)

サブクラス・インスタンス・データ定義用の WORKING-STORAGE SECTION

スーパークラスに定義したインスタンスに加えてサブクラスが必要とするインスタンス・データを記述するには、サブクラス OBJECT 段落の DATA DIVISION にある WORKING-STORAGE SECTION を使用します。クラスにインスタンス・データを定義するときに使用する構文と同じ構文を使用します。

例えば、Account クラスの CheckingAccount サブクラスのインスタンス・データの定義は、以下のようになります。

```
Identification division.
Object.
 Data division.
```

```
 Working-storage section.
 01 CheckFee pic S9(9) value 1.
 . . .
End Object.
```

関連タスク 620 ページの『クラス・インスタンス・データ定義用の WORKING-STORAGE SECTION』

サブクラス・インスタンス・メソッドの定義

サブクラスは、そのスーパークラスのメソッドを継承します。サブクラス定義において、継承したメソッドと同じシグニチャーのインスタンス・メソッドを定義して、サブクラスが継承するインスタンス・メソッドをオーバーライドすることができます。また、サブクラスが必要とする新しいメソッドを定義することもできます。

サブクラス・インスタンス・メソッドの構造と構文は、クラス・インスタンス・メソッドの構造と構文と同一です。サブクラス定義の OBJECT 段落の PROCEDURE DIVISION にサブクラス・インスタンス・メソッドを定義します。

『例: サブクラスの定義 (メソッドに関して)』

関連タスク
622 ページの『クラス・インスタンス・メソッドの定義』
626 ページの『インスタンス・メソッドのオーバーライド』
627 ページの『インスタンス・メソッドの多重定義』

例: サブクラスの定義 (メソッドに関して)

次の例は、Account クラスの CheckingAccount サブクラスのインスタンス・メソッド定義を示しています。

processCheck メソッドは、Check クラスの Java インスタンス・メソッド getAmount および getPayee を呼び出して、チェック・データを取得します。Account クラスから継承した credit および debit インスタンス・メソッドを呼び出して、当座の受取人を貸し方に記入し、支払人を借方に記入します。

print メソッドは、Account クラスに定義されている print インスタンス・メソッドをオーバーライドします。オーバーライドした print メソッドを呼び出して、口座状況を表示し、また当座手数料も表示します。CheckFee は、サブクラスに定義するインスタンス・データ項目です。

(Account クラスは、 629 ページの『例: メソッドの定義』に示されています。)

CheckingAccount クラス (Account のサブクラス)

```
cbl dll,thread,pgmname(longmixed)
Identification Division.
Class-id. CheckingAccount inherits Account.
Environment Division.
Configuration section.
Repository.
    Class CheckingAccount is "CheckingAccount"
    Class Check           is "Check"
```

```
     Class Account         is "Account".
*
*  (FACTORY paragraph not shown)
*
 Identification division.
 Object.
  Data division.
  Working-storage section.
  01 CheckFee pic S9(9) value 1.
  Procedure Division.
*
*    processCheck method to get the check amount and payee,
*    add the check fee, and invoke inherited methods debit
*    to debit the payer and credit to credit the payee:
   Identification Division.
   Method-id. "processCheck".
   Data division.
   Local-storage section.
   01 amount pic S9(9) binary.
   01 payee usage object reference Account.
   Linkage section.
   01 aCheck usage object reference Check.
*
   Procedure Division using by value aCheck.
     Invoke aCheck "getAmount" returning amount
     Invoke aCheck "getPayee" returning payee
     Invoke payee  "credit" using by value amount
     Add checkFee to amount
     Invoke self   "debit"  using by value amount.
   End method "processCheck".
*
*    print method override to display account status:
   Identification Division.
   Method-id. "print".
   Data division.
   Local-storage section.
   01 printableFee pic $$,$$$,$$9.
   Procedure Division.
     Invoke super "print"
     Move CheckFee to printableFee
     Display " Check fee: " printableFee.
   End method "print".
*
 End Object.
*
 End class CheckingAccount.
```

関連タスク 329 ページの『第 16 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行』 636 ページの『メソッドの呼び出し (INVOKE)』 626 ページの『インスタンス・メソッドのオーバーライド』 640 ページの『オーバーライドされたスーパークラス・メソッドの呼び出し』

ファクトリー・セクションの定義

個別のオブジェクト・インスタンスではなく、クラス自身に関連付けるデータおよびメソッドを定義するためには、クラス定義の FACTORY 段落を使用します。

COBOL ファクトリー・データ は、Java private 静的データと同じです。データの単一コピーは、そのクラス用にインスタンス生成され、クラスのすべてのオブジェクト・インスタンスに共用されます。クラスのすべてのインスタンスからデータを

収集するときは、ごく一般的にファクトリー・データを使用します。例えば、ファクトリー・データ項目を定義して、作成するクラスのインスタンス数の現在高を集計できます。

COBOL ファクトリー・メソッド は Java public 静的メソッドと同じです。これらのメソッドは、どのオブジェクト・インスタンスとも無関係に、クラスによってサポートされます。 VALUE 節を使用しただけではインスタンス・データを初期化できないときは、ごく一般的に、ファクトリー・メソッドを使用して、オブジェクトの生成をカスタマイズします。

対照的に、クラスのそれぞれのオブジェクト・インスタンスごとに作成されるデータを定義したり、クラスのそれぞれのオブジェクト・インスタンスごとにサポートされるメソッドを定義したりする場合には、クラス定義の OBJECT 段落を使用します。

ファクトリー定義は、以下の 3 つの部から構成され、その後に END FACTORY ステートメントが続きます。

表 80. **ファクトリー定義の構成**

除算	目的	構文
IDENTIFICATION (必須)	ファクトリー定義の開始を示す。	IDENTIFICATION DIVISION. FACTORY.
DATA (オプション)	このクラス用に一度割り振られたデータを記述する (クラスのそれぞれのインスタンスごとに割り振られたデータとは正反対)。	『ファクトリー・データ定義用の WORKING-STORAGE SECTION』 (オプション)
PROCEDURE (オプション)	ファクトリー・メソッドを定義する。	ファクトリー・メソッドの定義: 650 ページの『ファクトリー・メソッドの定義』

653 ページの『例: ファクトリーの定義 (メソッドに関して)』

関連タスク
617 ページの『クラスの定義』
642 ページの『COBOL クラスのインスタンス化』
658 ページの『プロシージャー指向 COBOL プログラムのラッピング』
659 ページの『OO アプリケーションの構造化』

ファクトリー・データ定義用の WORKING-STORAGE SECTION

COBOL クラスが必要とする ファクトリー・データ、つまりクラスのすべてのオブジェクト・インスタンスに共有される、静的に割り当てられたデータを記述するには、FACTORY 段落の DATA DIVISION にある WORKING-STORAGE SECTION を使用します。

IDENTIFICATION DIVISION 宣言の直前に入れる必要がある FACTORY キーワードは、クラスのファクトリー・データおよびファクトリー・メソッドの定義の開始を示します。例えば、Account クラスのファクトリー・データの定義は、以下のようになります。

```
Identification division.
Factory.
 Data division.
 Working-storage section.
 01 NumberOfAccounts pic 9(6) value zero.
 . . .
End Factory.
```

上記に示すように、単純ファクトリー・データの初期化は、VALUE 節を使用して行うことができます。

COBOL ファクトリー・データ は、Java private 静的データと同じです。他のクラスまたはサブクラス (必要に応じて、同じクラス内のインスタンス・メソッドも) は、COBOL ファクトリー・データを直接参照することはできません。ファクトリー・データは、FACTORY 段落で定義するすべてのファクトリー・メソッドにグローバルです。FACTORY 段落の外側からファクトリー・データにアクセス可能にする場合には、アクセスを可能にするためにファクトリー属性 (get または set) メソッドを定義します。

関連タスク
628 ページの『属性 (get および set) メソッドのコーディング』
642 ページの『COBOL クラスのインスタンス化』

ファクトリー・メソッドの定義

クラス定義の FACTORY 段落の PROCEDURE DIVISION に COBOL ファクトリー・メソッド を定義します。ファクトリー・メソッドは、クラスのどのオブジェクト・インスタンスとは無関係に、クラスによってサポートされる操作を定義します。COBOL ファクトリー・メソッドは Java public 静的メソッドと同じです。

一般的には、そのインスタンスが複雑な初期化を必要とするクラスについて、すなわち、VALUE 節だけの使用では割り当てることができない値に対して、ファクトリー・メソッドを定義します。ファクトリー・メソッド内でインスタンス・メソッドを呼び出して、インスタンス・データを初期化することができます。ファクトリー・メソッドは、インスタンス・データに直接アクセスすることはできません。

ファクトリー属性 (get および set) メソッドをコーディングして、FACTORY 段落の外側からファクトリー・データにアクセス可能にすることができます。例えば、同じクラスのインスタンス・メソッドから、またはクライアント・プログラムからファクトリー・データにアクセス可能にすることができます。例えば、Account クラスはファクトリー・メソッド getNumberOfAccounts を定義して、口座数の現在の集計を戻すことができます。

ファクトリー・メソッドを使用して、Java プログラムからアクセス可能になるようにプロシージャー指向の COBOL プログラムをラップすることもできます。main という名前のファクトリー・メソッドをコーディングすることで、java コマンドを使用してオブジェクト指向アプリケーションを実行したり、Java の標準的な方法に従ってアプリケーションを構成したりすることができます。詳細については、関連タスクを参照してください。

ファクトリー・メソッドの定義では、インスタンス・メソッドを定義するときに用いる構文と同じ構文を使用します。COBOL ファクトリー・メソッド定義は、4 つの部 (COBOL プログラムに類似) とその後の END METHOD マーカーから構成されます。

表 81. **ファクトリー・メソッド定義の構成**

除算	目的	構文
IDENTIFICATION (必須)	クラス・インスタンス・メソッドの場合と同じ	クラス・インスタンス・メソッドの場合と同じ (必須)
ENVIRONMENT (オプション)	クラス・インスタンス・メソッドの場合と同じ	クラス・インスタンス・メソッドの場合と同じ
DATA (オプション)	クラス・インスタンス・メソッドの場合と同じ	クラス・インスタンス・メソッドの場合と同じ
PROCEDURE (オプション)	クラス・インスタンス・メソッドの場合と同じ	クラス・インスタンス・メソッドの場合と同じ

クラス定義内では、各ファクトリー・メソッド名を固有にする必要はありませんが、各ファクトリー・メソッドに固有のシグニチャーを与える必要があります。ファクトリー・メソッドの多重定義は、インスタンス・メソッドを多重定義する場合とまったく同じ方法で行うことができます。例えば、CheckingAccount サブクラスは、2 つの版のファクトリー・メソッド createCheckingAccount、すなわち、口座を初期化してデフォルトの収支ゼロを設定する版と、開始残高を渡せるようにする版を提供します。クライアントは、意図したメソッドのシグニチャーと一致する引数を渡して、createCheckingAccount メソッドのいずれかを呼び出すことができます。

ファクトリー・メソッドの DATA DIVISION および FACTORY 段落の DATA DIVISION の両方において、データ項目を同じ名前で定義した場合、そのデータ名に対するメソッド内の参照は、そのメソッド・データ項目だけを参照します。メソッド DATA DIVISION が優先します。

653 ページの『例: ファクトリーの定義 (メソッドに関して)』

関連タスク

659 ページの『OO アプリケーションの構造化』
658 ページの『プロシージャー指向 COBOL プログラムのラッピング』
642 ページの『COBOL クラスのインスタンス化』
622 ページの『クラス・インスタンス・メソッドの定義』
628 ページの『属性 (get および set) メソッドのコーディング』
627 ページの『インスタンス・メソッドの多重定義』
『ファクトリー・メソッドまたは静的メソッドの隠蔽』
652 ページの『ファクトリー・メソッドまたは静的メソッドの呼び出し』
482 ページの『IMS のもとでのオブジェクト指向 COBOL と Java の使用』

ファクトリー・メソッドまたは静的メソッドの隠蔽

サブクラスで定義されたファクトリー・メソッドは、そのサブクラスで利用できるのであれば、継承された COBOL または Java メソッドを (これら 2 つのメソッドが同じシグニチャーを持っている場合) *隠蔽*すると言います。

スーパークラス・ファクトリー・メソッド f1 を COBOL サブクラスで隠すには、スーパークラス・メソッドと名前が同じで、その PROCEDURE DIVISION USING 句 (ある場合) の仮パラメーターの数およびタイプがスーパークラス・メソッドと同じであるサブクラスでファクトリー・メソッド f1 を定義します。(スーパークラス・メソッドが Java でインプリメントされる場合には、対応する Java パラメーターのデータ型と相互運用可能な仮パラメーターをコーディングする必要があります。) クライアントがサブクラス名を使用して f1 を呼び出すとき、スーパークラス・メソッドではなく、サブクラス・メソッドが呼び出されます。

メソッド戻り値の有無および PROCEDURE DIVISION RETURNING 句 (ある場合) で使われる戻り値のデータ型は、サブクラス・ファクトリー・メソッドと隠されたスーパークラス・メソッドにおいて同一でなければなりません。

ファクトリー・メソッドは、Java または COBOL スーパークラスに、インスタンス・メソッドを隠してはいけません。

653 ページの『例: ファクトリーの定義 (メソッドに関して)』

関連タスク 669 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』 626 ページの『インスタンス・メソッドのオーバーライド』 636 ページの『メソッドの呼び出し (INVOKE)』

関連参照
The Java Language Specification (Inheritance, overriding, and hiding)
手続き部ヘッダー (*Enterprise COBOL 言語解説書*)

ファクトリー・メソッドまたは静的メソッドの呼び出し

COBOL ファクトリー・メソッドまたは Java 静的メソッドを COBOL メソッドまたはクライアント・プログラムで呼び出すには、クラス名を INVOKE ステートメントの第 1 オペランドとしてコーディングしてください。

例えば、クライアント・プログラムは次のステートメントをコーディングして、createCheckingAccount という名前の多重定義 CheckingAccount ファクトリー・メソッドの 1 つを呼び出して、口座番号 777777 および開始残高 $300 の当座預金を作成することができます。

```
Invoke CheckingAccount "createCheckingAccount"
  using by value 777777 300
  returning aCheckingAccount
```

ファクトリー・メソッドを定義する同じクラス内からファクトリー・メソッドを呼び出す場合にも、クラス名を INVOKE ステートメントの第 1 オペランドとして使用します。

実行時におけるその値がメソッド名であるリテラルとして、または ID として呼び出すメソッドの名前をコーディングします。メソッド名は、英数字または国別リテラルであるか、あるいはカテゴリー英字、英数字、または国別のデータ項目でなければならず、解釈されるときには大/小文字が区別されます。

呼び出されたメソッドが、INVOKE ステートメントで指定されたクラスでサポートされない場合、重大度 3 の言語環境プログラム条件が実行時に発生します。ただし、INVOKE ステートメントで ON EXCEPTION 句をコーディングした場合は別です。

USING 句で COBOL ファクトリー・メソッドまたは Java 静的メソッドに引数を渡すときの適合要件と、RETURNING 句で戻り値を受けるときの適合要件は、インスタンス・メソッドを呼び出す場合と同じです。

『例: ファクトリーの定義 (メソッドに関して)』

関連タスク 636 ページの『メソッドの呼び出し (INVOKE)』 139 ページの『COBOL での国別データ (Unicode) の使用』 669 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』

関連参照
INVOKE ステートメント (*Enterprise COBOL 言語解説書*)

例: ファクトリーの定義 (メソッドに関して)

次の例では、以前の例が、ファクトリー・データおよびメソッドの定義を示すように更新します。

以下の更新が示されます。

- Account クラスは、ファクトリー・データとパラメーター化ファクトリー・メソッド createAccount を追加します。こうすることで、渡される口座番号を使用して Account インスタンスの作成が可能になります。
- CheckingAccount サブクラスは、ファクトリー・データおよび多重定義のパラメーター化ファクトリー・メソッド createCheckingAccount を追加します。createCheckingAccount の 1 つのインプリメンテーションでデフォルトの収支ゼロの口座を初期化し、もう 1 つのインプリメンテーションで開始残高を渡せるようにします。クライアントは、目的のメソッドのシグニチャーと一致する引数を渡して、メソッドを呼び出すこともできます。
- TestAccounts クライアントは、Account および CheckingAccount クラスのファクトリー・メソッドによって提供されるサービスを呼び出して、Java Check クラスのインスタンスを生成します。
- TestAccounts クライアント・プログラムからの出力を表示します。

(以前の例とは、629 ページの『例: メソッドの定義』、 643 ページの『例: クライアントの定義』、および 647 ページの『例: サブクラスの定義 (メソッドに関して)』のことです。)

また、上記の例の完全なソース・コードが HFS の cobol/demo/oosample サブディレクトリーに入っています。一般的には、このソースへの絶対パスは、/usr/lpp/cobol/demo/oosample です。そこで MAKE ファイルを作成して、コードのコンパイルとリンクを行うことができます。

Account クラス

```
cbl dll,thread,pgmname(longmixed),lib
Identification Division.
Class-id. Account inherits Base.
```

```
 Environment Division.
 Configuration section.
 Repository.
     Class Base    is "java.lang.Object"
     Class Account is "Account".
*
 Identification division.
 Factory.
  Data division.
  Working-storage section.
  01 NumberOfAccounts pic 9(6) value zero.
*
  Procedure Division.
*
*    createAccount method to create a new Account
*    instance, then invoke the OBJECT paragraph's init
*    method on the instance to initialize its instance data:
   Identification Division.
   Method-id. "createAccount".
   Data division.
   Linkage section.
   01 inAccountNumber  pic S9(6) binary.
   01 outAccount object reference Account.
*      Facilitate access to JNI services:
     Copy JNI.
   Procedure Division using by value inAccountNumber
       returning outAccount.
*      Establish addressability to JNI environment structure:
     Set address of JNIEnv to JNIEnvPtr
     Set address of JNINativeInterface to JNIEnv
     Invoke Account New returning outAccount
     Invoke outAccount "init" using by value inAccountNumber
     Add 1 to NumberOfAccounts.
   End method "createAccount".
*
 End Factory.
*
 Identification division.
 Object.
  Data division.
  Working-storage section.
  01 AccountNumber  pic 9(6).
  01 AccountBalance pic S9(9) value zero.
*
  Procedure Division.
*
*    init method to initialize the account:
   Identification Division.
   Method-id. "init".
   Data division.
   Linkage section.
   01 inAccountNumber pic S9(9) binary.
   Procedure Division using by value inAccountNumber.
     Move inAccountNumber to AccountNumber.
   End method "init".
*
*    getBalance method to return the account balance:
   Identification Division.
   Method-id. "getBalance".
   Data division.
   Linkage section.
   01 outBalance pic S9(9) binary.
   Procedure Division returning outBalance.
     Move AccountBalance to outBalance.
   End method "getBalance".
*
*    credit method to deposit to the account:
```

```
   Identification Division.
   Method-id. "credit".
   Data division.
   Linkage section.
   01 inCredit   pic S9(9) binary.
   Procedure Division using by value inCredit.
     Add inCredit to AccountBalance.
   End method "credit".
*
*    debit method to withdraw from the account:
   Identification Division.
   Method-id. "debit".
   Data division.
   Linkage section.
   01 inDebit    pic S9(9) binary.
   Procedure Division using by value inDebit.
     Subtract inDebit from AccountBalance.
   End method "debit".
*
*    print method to display formatted account number and balance:
   Identification Division.
   Method-id. "print".
   Data division.
   Local-storage section.
   01 PrintableAccountNumber  pic ZZZZZZ999999.
   01 PrintableAccountBalance pic $$$$,$$$,$$9CR.
   Procedure Division.
     Move AccountNumber  to PrintableAccountNumber
     Move AccountBalance to PrintableAccountBalance
     Display " Account: " PrintableAccountNumber
     Display " Balance: " PrintableAccountBalance.
   End method "print".
*
 End Object.
*
 End class Account.
```

CheckingAccount クラス (Account のサブクラス)

```
 cbl dll,thread,pgmname(longmixed),lib
 Identification Division.
 Class-id. CheckingAccount inherits Account.
 Environment Division.
 Configuration section.
 Repository.
     Class CheckingAccount is "CheckingAccount"
     Class Check           is "Check"
     Class Account         is "Account".
*
 Identification division.
 Factory.
  Data division.
  Working-storage section.
  01 NumberOfCheckingAccounts pic 9(6) value zero.
*
  Procedure Division.
*
*    createCheckingAccount overloaded method to create a new
*    CheckingAccount instance with a default balance, invoke
*    inherited instance method init to initialize the account
*    number, and increment factory data tally of checking accounts:
   Identification Division.
   Method-id. "createCheckingAccount".
   Data division.
   Linkage section.
   01 inAccountNumber  pic S9(6) binary.
   01 outCheckingAccount object reference CheckingAccount.
```

```
*      Facilitate access to JNI services:
     Copy JNI.
   Procedure Division using by value inAccountNumber
       returning outCheckingAccount.
*      Establish addressability to JNI environment structure:
     Set address of JNIEnv to JNIEnvPtr
     Set address of JNINativeInterface to JNIEnv
     Invoke CheckingAccount New returning outCheckingAccount
     Invoke outCheckingAccount "init"
       using by value inAccountNumber
     Add 1 to NumberOfCheckingAccounts.
   End method "createCheckingAccount".
*
*    createCheckingAccount overloaded method to create a new
*    CheckingAccount instance, invoke inherited instance methods
*    init to initialize the account number and credit to set the
*    balance, and increment factory data tally of checking accounts:
   Identification Division.
   Method-id. "createCheckingAccount".
   Data division.
   Linkage section.
   01 inAccountNumber  pic S9(6) binary.
   01 inInitialBalance pic S9(9) binary.
   01 outCheckingAccount object reference CheckingAccount.
     Copy JNI.
   Procedure Division using by value inAccountNumber
                                     inInitialBalance
       returning outCheckingAccount.
     Set address of JNIEnv to JNIEnvPtr
     Set address of JNINativeInterface to JNIEnv
     Invoke CheckingAccount New returning outCheckingAccount
     Invoke outCheckingAccount "init"
       using by value inAccountNumber
     Invoke outCheckingAccount "credit"
       using by value inInitialBalance
     Add 1 to NumberOfCheckingAccounts.
   End method "createCheckingAccount".
*
 End Factory.
*
 Identification division.
 Object.
  Data division.
  Working-storage section.
  01 CheckFee pic S9(9) value 1.
  Procedure Division.
*
*    processCheck method to get the check amount and payee,
*    add the check fee, and invoke inherited methods debit
*    to debit the payer and credit to credit the payee:
   Identification Division.
   Method-id. "processCheck".
   Data division.
   Local-storage section.
   01 amount pic S9(9) binary.
   01 payee usage object reference Account.
   Linkage section.
   01 aCheck usage object reference Check.
   Procedure Division using by value aCheck.
     Invoke aCheck "getAmount" returning amount
     Invoke aCheck "getPayee" returning payee
     Invoke payee  "credit" using by value amount
     Add checkFee to amount
     Invoke self   "debit"  using by value amount.
   End method "processCheck".
*
*    print method override to display account status:
```

```
   Identification Division.
   Method-id. "print".
   Data division.
   Local-storage section.
   01 printableFee pic $$,$$$,$$9.
   Procedure Division.
     Invoke super "print"
     Move CheckFee to printableFee
     Display " Check fee: " printableFee.
   End method "print".
*
 End Object.
*
 End class CheckingAccount.
```

Check クラス

```
/**
 * A Java class for check information
 */
public class Check {
  private CheckingAccount payer;
  private Account         payee;
  private int             amount;

  public Check(CheckingAccount inPayer, Account inPayee, int inAmount) {
    payer=inPayer;
    payee=inPayee;
    amount=inAmount;
  }

  public int getAmount() {
    return amount;
  }

  public Account getPayee() {
    return payee;
  }
}
```

TestAccounts クライアント・プログラム

```
 cbl dll,thread,pgmname(longmixed)
 Identification division.
 Program-id. "TestAccounts" recursive.
 Environment division.
 Configuration section.
 Repository.
     Class Account         is "Account"
     Class CheckingAccount is "CheckingAccount"
     Class Check           is "Check".
 Data Division.
*  Working data is declared in Local-storage
*  so that each thread has its own copy:
 Local-storage section.
 01  anAccount        usage object reference Account.
 01  aCheckingAccount usage object reference CheckingAccount.
 01  aCheck           usage object reference Check.
 01  payee            usage object reference Account.
*
 Procedure division.
 Test-Account-section.
     Display "Test Account class"
*  Create account 123456 with 0 balance:
     Invoke Account "createAccount"
       using by value 123456
       returning anAccount
```

```
*  Deposit 500 to the account:
     Invoke anAccount "credit" using by value 500
     Invoke anAccount "print"
     Display space
*
     Display "Test CheckingAccount class"
*  Create checking account 777777 with balance of 300:
     Invoke CheckingAccount "createCheckingAccount"
       using by value 777777 300
       returning aCheckingAccount
*  Set account 123456 as the payee:
     Set payee to anAccount
*  Initialize check for 125 to be paid by account 777777 to payee:
     Invoke Check New
       using by value aCheckingAccount, payee, 125
       returning aCheck
*  Debit the payer, and credit the payee:
     Invoke aCheckingAccount "processCheck"
       using by value aCheck
     Invoke aCheckingAccount "print"
     Invoke anAccount "print"
*
     Stop Run.
 End program "TestAccounts".
```

TestAccounts クライアント・プログラムが生成する出力

```
Test Account class
 Account:       123456
 Balance:         $500

Test CheckingAccount class
 Account:       777777
 Balance:         $174
 Check fee:         $1
 Account:       123456
 Balance:         $625
```

関連タスク 641 ページの『クラスのインスタンスの作成および初期化』 650 ページの『ファクトリー・メソッドの定義』 652 ページの『ファクトリー・メソッドまたは静的メソッドの呼び出し』 329 ページの『第 16 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行』

プロシージャー指向 COBOL プログラムのラッピング

ラッパー は、オブジェクト指向コードとプロシージャー指向コードの間のインターフェースを提供するクラスです。ファクトリー・メソッドは、既存のプロシージャー型 COBOL コード用にラッパーを書き込み、Java プログラムからアクセス可能にするときの便利な方法を提供します。

COBOL コードをラップするためには、以下の手順を実行します。

1. FACTORY 段落を含む単純 COBOL クラスを作成する。
2. FACTORY 段落で、CALL ステートメントを使用してプロシージャー型プログラムを呼び出すファクトリー・メソッドをコーディングする。

Java プログラムは、静的メソッドの呼び出しの式を使用して、すなわち、COBOL プロシージャー型プログラムを呼び出して、ファクトリー・メソッドを呼び出すことができます。

関連タスク
617 ページの『クラスの定義』
648 ページの『ファクトリー・セクションの定義』
650 ページの『ファクトリー・メソッドの定義』

OO アプリケーションの構造化

次の 3 つの方法のいずれかで、オブジェクト指向 COBOL 構文を使用するアプリケーションを構造化することができます。

オブジェクト指向アプリケーションは、次のいずれかで始めることができます。

- COBOL プログラム。名前は何でも構いません。

  UNIX のもとでは、リンクされたモジュールの名前 (プログラム名と一致していなければなりません) をコマンド・プロンプトで指定することで、アプリケーションを実行できます。また、PDSE のモジュールとしてプログラムをバインドし、EXEC PGM ステートメントを使用して JCL で実行することもできます。
- main という名前のメソッドを含む Java クラス定義。main は、単一の String[] 型パラメーターを持つ public、static、および void として宣言します。

  main を含むクラスの名前を指定し、0 以上のストリングをコマンド行引数として渡すことで、java コマンドでアプリケーションを実行できます。
- main という名前のファクトリー・メソッドを含む COBOL クラス定義。main は、RETURNING 句を指定せず、java.lang.String 型のエレメントの配列であるクラスへのオブジェクト参照である単一の USING パラメーターを指定して宣言します。つまり、main は、事実上、String[] 型のパラメーターを 1 つ持つ、public、static、および void です。

  main を含むクラスの名前を指定し、0 以上のストリングをコマンド行引数として渡すことで、java コマンドでアプリケーションを実行できます。

  以下の場合には、この方法でオブジェクト指向アプリケーションを構成します。
  - java コマンドを使用してアプリケーションを実行する。
  - アプリケーションが Java クラス・ファイルの main メソッドで開始しなければならない環境 (IMS Java 従属領域など) でアプリケーションを実行する。
  - 標準的な Java プログラミング方式に従う。

  660 ページの『例: java コマンドを使用して実行される COBOL アプリケーション』

関連タスク 329 ページの『第 16 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行』 650 ページの『ファクトリー・メソッドの定義』 669 ページの『Java 用の配列およびストリングの宣言』 481 ページの『第 22 章 COBOL プログラムの開発 (IMS の場合)』

例: java コマンドを使用して実行される COBOL アプリケーション

以下の例では、main という名前のファクトリー・メソッドを含む COBOL クラス定義を示します。

いずれの場合も、main には RETURNING 句がなく、java.lang.String 型のエレメントの配列であるクラスへのオブジェクト参照である単一の USING パラメーターがあります。これらのアプリケーションは、java コマンドを使用して実行できます。

メッセージの表示

```
 cbl dll,thread
 Identification Division.
 Class-id. CBLmain inherits Base.
 Environment Division.
 Configuration section.
 Repository.
     Class Base is "java.lang.Object"
     Class stringArray is "jobjectArray:java.lang.String"
     Class CBLmain is "CBLmain".
*
 Identification Division.
 Factory.
  Procedure division.
*
   Identification Division.
   Method-id. "main".
   Data division.
   Linkage section.
   01 SA usage object reference stringArray.
   Procedure division using by value SA.
     Display " >> COBOL main method entered"
     .
   End method "main".
 End factory.
 End class CBLmain.
```

入力ストリングのエコー

```
 cbl dll,thread,lib,pgmname(longmixed),ssrange
 Identification Division.
 Class-id. Echo inherits Base.
 Environment Division.
 Configuration section.
 Repository.
     Class Base is "java.lang.Object"
     Class stringArray is "jobjectArray:java.lang.String"
     Class jstring is "java.lang.String"
     Class Echo is "Echo".
*
 Identification Division.
 Factory.
  Procedure division.
*
   Identification Division.
   Method-id. "main".
   Data division.
   Local-storage section.
   01 SAlen        pic s9(9) binary.
   01 I            pic s9(9) binary.
   01 SAelement    object reference jstring.
   01 SAelementlen pic s9(9) binary.
```

```
  01 Sbuffer      pic X(65535).
  01 P            pointer.
  Linkage section.
  01 SA           object reference stringArray.
  Copy "JNI.cpy" suppress.
  Procedure division using by value SA.
    Set address of JNIEnv to JNIEnvPtr
    Set address of JNINativeInterface to JNIEnv
    Call GetArrayLength using by value JNIEnvPtr SA
      returning SAlen
    Display "Input string array length: " SAlen
    Display "Input strings:"
    Perform varying I from 0 by 1 until I = SAlen
      Call GetObjectArrayElement
        using by value JNIEnvPtr SA I
        returning SAelement
      Call "GetStringPlatformLength"
        using by value JNIEnvPtr
                       SAelement
                       address of SAelementlen
                       0
      Call "GetStringPlatform"
        using by value JNIEnvPtr
                       SAelement
                       address of Sbuffer
                       length of Sbuffer
                       0
      Display Sbuffer(1:SAelementlen)
    End-perform
    .
  End method "main".
End factory.
End class Echo.
```

関連タスク 329 ページの『第 16 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行』 650 ページの『ファクトリー・メソッドの定義』
663 ページの『第 31 章 Java メソッドとの通信』

第 31 章 Java メソッドとの通信

Java との言語間インターオペラビリティーを達成するには、Java Native Interface (JNI) でのサービスの使用、データ型のコーディング、および COBOL プログラムのコンパイルに関する特定の規則および指針に従う必要があります。

Java で書き込まれたメソッドを COBOL プログラムから呼び出したり、COBOL で書き込まれたメソッドを Java プログラムから呼び出したりすることができます。Java の基本オブジェクト機能に対応するには、COBOL オブジェクト指向言語をコーディングする必要があります。追加の Java 機能に対応するには、JNI サービスを呼び出すことができます。

Java プログラムはマルチスレッド化され、非同期シグナルを用いる場合があります。したがって、THREAD オプションを使用して COBOL プログラムをコンパイルしてください。

676 ページの『例: COBOL で書かれた J2EE クライアント』

関連タスク 139 ページの『COBOL での国別データ (Unicode) の使用』 『JNI サービスへのアクセス』 668 ページの『Java とのデータ共用』 613 ページの『第 30 章 オブジェクト指向プログラムの作成』 329 ページの『第 16 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行』 549 ページの『第 27 章 マルチスレッド化のための COBOL プログラムの準備』

関連参照
Java 2 Enterprise Edition Developer's Guide

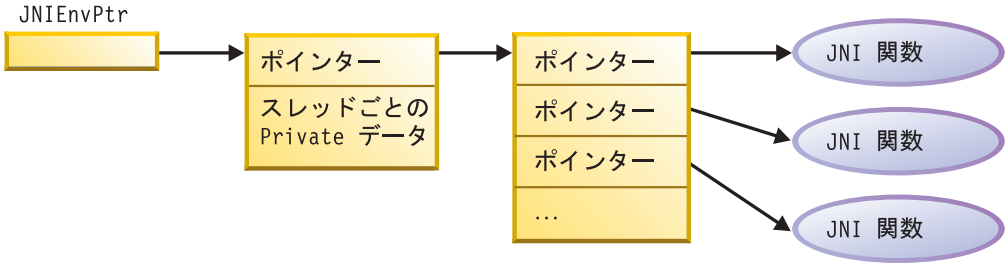
JNI サービスへのアクセス

Java Native Interface (JNI) は、COBOL と Java を併用するアプリケーションを開発する際に使用できる多くの呼び出し可能サービスを提供します。このサービスへのアクセスを円滑に行うには、COBOL プログラムの LINKAGE SECTION に JNI.cpy をコピーします。

この JNI.cpy コピーブックには、以下の定義が含まれています。

- Java JNI タイプに対応する COBOL データ定義
- JNINativeInterface、呼び出し可能サービス関数にアクセスするための関数ポインターが含まれている JNI 環境構造

次の図に示すように、JNI 環境ポインターからの 2 つのレベルの間接化技法によって JNI 環境構造を取得します。



特殊レジスター JNIEnvPtr を使用して、JNI 環境ポインターを参照し、JNI 環境構造のアドレスを取得します。JNIEnvPtr は、USAGE POINTER として暗黙的に定義されます。それを受取データ項目として使用することはできません。JNI 環境構造の内容を参照する前に、以下のステートメントをコーディングして、そのアドレス可能度を確立する必要があります。

```
Linkage section.
COPY JNI
. . .
 Procedure division.
           Set address of JNIEnv to JNIEnvPtr
           Set address of JNINativeInterface to JNIEnv
    . . .
```

上記コードは、以下の項目のアドレスを設定します。

- JNIEnv。JNI.cpy が提供するポインター・データ項目です。JNIEnvPtr は、環境ポインターが含まれている COBOL 特殊レジスターです。
- JNINativeInterface。JNI.cpy に含まれている COBOL グループ構造です。この構造には、JNI 呼び出し可能サービスの関数ポインターの配列が含まれている JNI 環境構造がマップされています。

上記のステートメントをコーディングした後に、関数ポインターを参照する CALL ステートメントを使用して、JNI 呼び出し可能サービスにアクセスすることができます。次の例に示すように、環境ポインターを必要とするサービスに、最初の引数として JNIEnvPtr 特殊レジスターを渡すことができます。

```
01 InputArrayObj usage object reference jlongArray.
01 ArrayLen pic S9(9) comp-5.
. . .
        Call GetArrayLength using by value JNIEnvPtr InputArrayObj
            returning ArrayLen
```

重要: すべての引数を値によって JNI 呼び出し可能サービスに渡します。

一部の JNI 呼び出し可能サービスは、Java クラス・オブジェクト参照を引数として必要とします。クラスに関連付けられたクラス・オブジェクトへの参照を取得するには、以下の JNI 呼び出し可能サービスのどちらかを使用します。

- GetObjectClass
- FindClass

制限: JNI 環境ポインターはスレッド固有のものです。スレッドから別のスレッドに渡すことはできません。

関連タスク 666 ページの『ローカル参照とグローバル参照の管理』 『Java 例外の処理』 669 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』 631 ページの『クライアントの定義』

関連参照
803 ページの『付録 F. JNI.cpy』
The Java Native Interface

Java 例外の処理

JNI サービスを使用して、Java 例外を throw したり、catch したりします。

例外のスロー: COBOL メソッドから Java 例外をスローするには、次のいずれかのサービスを使用します。

- Throw
- ThrowNew

throw したオブジェクトは、java.lang.Throwable のサブクラスのインスタンスにする必要があります。

Java 仮想マシン (JVM) は、呼び出しを含んでいるメソッドが完了して JVM に戻るまで、throw された例外を認識して処理することは行いません。

例外のキャッチ: Java 例外をスローした可能性があるメソッドを呼び出してから、次の手順を行うことができます。

1. 例外が発生したかどうかをテストします。
2. 例外が発生した場合は、その例外を処理します。
3. 例外をクリアします (クリアが適切な場合)。

次の JNI サービスを使用します。

- ExceptionOccurred
- ExceptionCheck
- ExceptionDescribe
- ExceptionClear

エラー分析を行うには、戻された例外オブジェクトによってサポートされるメソッドを使用します。このオブジェクトは、java.lang.Throwable クラスのインスタンスです。

『例: Java 例外の処理』

例: Java 例外の処理

次の例は、Java からの例外を catch するための JNI サービスの使用、およびエラー分析を行うための java.lang.Throwable の PrintStackTrace メソッドの使用を示しています。

```
Repository.
    Class JavaException is "java.lang.Exception".
. . .
Local-storage section.
```

```
01 ex usage object reference JavaException.
Linkage section.
COPY "JNI.cpy".
. . .
Procedure division.
    Set address of JNIEnv to JNIEnvPtr
    Set address of JNINativeInterface to JNIEnv
    . . .
    Invoke anObj "someMethod"
    Perform ErrorCheck
. . .
ErrorCheck.
    Call ExceptionOccurred
        using by value JNIEnvPtr
        returning ex
    If ex not = null then
        Call ExceptionClear using by value JNIEnvPtr
        Display "Caught an unexpected exception"
        Invoke ex "printStackTrace"
        Stop run
    End-if
```

ローカル参照とグローバル参照の管理

Java 仮想マシンは、ネイティブ・メソッド (COBOL メソッドなど) で使用されるオブジェクト参照を追跡します。この追跡によって、ガーベッジ・コレクションの際、まだ使用中のオブジェクトが解放されないようにします。

オブジェクト参照には、以下の 2 つのクラスがあります。

ローカル参照
ローカル参照は、呼び出したメソッドが稼働している間のみ有効です。ネイティブ・メソッドが戻ると、ローカル参照の自動解放が実行されます。

グローバル参照
グローバル参照は、明示的に削除するまで有効です。グローバル参照は、JNI サービス NewGlobalRef を使用して、ローカル参照から作成することができます。

以下のオブジェクト参照は常にローカルです。

- メソッド・パラメーターとして受け取られるオブジェクト参照
- メソッドの RETURNING 値としてメソッドの起動から戻されるオブジェクト参照
- JNI 関数への呼び出しによって戻されるオブジェクト参照
- INVOKE . . . NEW ステートメントを使用して作成するオブジェクト参照

ローカル参照またはグローバル参照のいずれかをオブジェクト参照引数として JNI サービスに渡すことができます。

RETURNING 値としてローカル参照またはグローバル参照のいずれかを戻すメソッドをコーディングできます。ただし、いずれの場合も、呼び出すプログラムが受け取る参照はローカル参照です。

メソッドの起動で USING 引数としてローカル参照またはグローバル参照のいずれかを渡すことができます。ただし、いずれの場合も、呼び出されたメソッドが受け取る参照はローカル参照です。

ローカル参照は、それが作成されたスレッド内でのみ有効です。ローカル参照をスレッドから別のスレッドに渡すことはできません。

関連タスク
663 ページの『JNI サービスへのアクセス』
『ローカル参照の削除、保管、および解放』

ローカル参照の削除、保管、および解放

ローカル参照は、メソッド内で、随時に手動で削除できます。ローカル参照は、メソッドの LOCAL-STORAGE SECTION に定義したオブジェクト参照内にのみ保管します。

次のいずれかのデータ項目で参照を保管する場合にローカル参照をグローバル参照に変換するには、SET ステートメントを使用します。

- オブジェクト・インスタンス変数
- ファクトリー変数
- メソッドの WORKING-STORAGE SECTION 内のデータ項目

そうしないと、エラーが発生します。メソッドが戻ったときにこれらのストレージ域は保持されるので、ローカル参照は無効になります。

ほとんどのケースにおいて、メソッドが戻るときに発生するローカル参照の自動解放に依存することができます。ただし、一部のケースにおいては、JNI サービス DeleteLocalRef を使用して、メソッド内のローカル参照を明示的に解放する必要があります。以下に、明示的解放が適切な 2 つの状態を示します。

- メソッドにおいて、ラージ・オブジェクトにアクセスすることで、オブジェクトへのローカル参照を作成します。膨大な計算を行った後で、メソッドが戻ります。このラージ・オブジェクトを別の計算に必要としない場合には、このオブジェクトを解放してください。このローカル参照は、ガーベッジ・コレクションの間にオブジェクトを解放する妨げになるからです。
- メソッドに多数のローカル参照を作成しますが、それらのすべてのローカル参照を同時には使用しません。Java 仮想マシンは、各ローカル参照を追跡するためのスペースが必要であるため、不要になったローカル参照を解放してください。ローカル参照を解放することによって、システムがメモリー不足になるのを防ぐことができます。

  例えば、COBOL メソッドにおいて、大規模な配列のオブジェクトをループし、エレメントをローカル参照として検索し、それぞれの反復ごとに 1 つのエレメントを操作します。それぞれの反復後に、配列エレメントへのローカル参照を解放することができます。

ローカル参照およびグローバル参照を管理するには、以下の呼び出し可能サービスを使用してください。

表 82. **ローカルおよびグローバル参照の JNI サービス**

サービス	入力引数	戻り値	目的
NewGlobalRef	• JNI 環境ポインター • ローカルまたはグローバル・オブジェクト参照	グローバル参照、またはシステムがメモリー不足のときは NULL	入力オブジェクト参照が参照するオブジェクトに新規グローバル参照を作成する
DeleteGlobalRef	• JNI 環境ポインター • グローバル・オブジェクト参照	なし	入力オブジェクト参照が参照するオブジェクトへのグローバル参照を削除する
DeleteLocalRef	• JNI 環境ポインター • ローカル・オブジェクト参照	なし	入力オブジェクト参照が参照するオブジェクトへのローカル参照を削除する

関連タスク
663 ページの『JNI サービスへのアクセス』

Java アクセス制御

Java アクセス修飾子 protected および private は、Java Native Interface を使用すると、強制されません。したがって、COBOL プログラムは、Java クライアントからは呼び出し不可能な protected または private Java メソッドを呼び出すことができます。この使用法はお勧めできません。

Java とのデータ共用

Java データ型と同じものを持つ COBOL データ型を共有することができます。(COBOL データ型には、Java 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。)

次の方法で Java とデータ項目を共有します。

- INVOKE ステートメントの USING 句に引数として渡します。
- Java メソッドから、USING 句のパラメーターとして受け取ります。
- INVOKE ステートメントの RETURNING 値として受け取ります。
- COBOL メソッドの PROCEDURE DIVISION ヘッダーの RETURNING 句の値として戻します。

配列およびストリングを渡したり受け取ったりするには、以下のように、それらをオブジェクト参照として宣言します。

- 特殊配列クラスのうちの 1 つのインスタンスが含まれているオブジェクト参照として、配列を宣言します。
- jstring クラスのインスタンスが含まれているオブジェクト参照として、ストリングを宣言します。

関連タスク 669 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』 669 ページの『Java 用の配列およびストリングの宣言』 671 ページの

の『Java 配列の取り扱い』 673 ページの『Java ストリングの取り扱い』 636 ページの『メソッドの呼び出し (INVOKE)』 521 ページの『第 25 章 データの共用』

COBOL および Java での相互運用可能なデータ型のコーディング

Java との通信時には、COBOL プログラムは、特定のデータ型しか使用できません。

表 83. **COBOL および Java で相互運用可能なデータ型**

Java の基本データ型	対応する COBOL データ型
boolean[1]	PIC X の後に以下の形式とまったく同じ 2 つの条件名を記述する。 *level-number* *data-name* PIC X. 88 *data-name*-false value X'00'. 88 *data-name*-true value X'01' through X'FF'.
byte[1]	1 バイト英数字: PIC X または PIC A
short	USAGE BINARY、COMP、COMP-4、または COMP-5、形式 S9(*n*) の PICTURE 節付き。ここで、1<=*n*<=4
int	USAGE BINARY、COMP、COMP-4、または COMP-5、形式 S9(*n*) の PICTURE 節付き。ここで、5<=*n*<=9
long	USAGE BINARY、COMP、COMP-4、または COMP-5、形式 S9(*n*) の PICTURE 節。ここで、10<=*n*<=18
float[2]	USAGE COMP-1
double[2]	USAGE COMP-2
char	1 文字基本国別: PIC N USAGE NATIONAL. (国別グループは不可です。)
クラス型 (オブジェクト参照)	USAGE OBJECT REFERENCE *class-name*

1. boolean 型と byte 型はいずれも PIC X に対応しているため、この 2 つは区別する必要があります。 PIC X は、前述の 2 つの条件名を指定して引数またはパラメーターを定義した場合にのみ、boolean 型として解釈されます。それ以外の場合、PIC X データ項目は、Java の byte 型として解釈されます。
2. Java 浮動小数点データは、IEEE 浮動小数点として表現されます。Enterprise COBOLただし、16 進浮動小数点表記を使用します。 INVOKE ステートメントを使用して浮動小数点引数を渡すとき、または、浮動小数点データを Java メソッドから受け取るとき、引数およびデータは必要に応じて、自動的に変換されます。

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

Java 用の配列およびストリングの宣言

Java と通信する場合には、特別な配列クラスを使用して配列を宣言し、jstring を使用してストリングを宣言してください。下のテーブルで示された COBOL データ型をコーディングしてください。

表 84. **COBOL および Java で相互運用可能な配列およびストリング**

Java データ型	対応する COBOL データ型
boolean[ ]	オブジェクト参照 jbooleanArray
byte[ ]	オブジェクト参照 jbyteArray
short[ ]	オブジェクト参照 jshortArray
int[ ]	オブジェクト参照 jintArray
long[ ]	オブジェクト参照 jlongArray
char[ ]	オブジェクト参照 jcharArray
Object[ ]	オブジェクト参照 jobjectArray
String	オブジェクト参照 jstring

Java とのインターオペラビリティーのためにこれらのクラスのいずれかを使用するには、 REPOSITORY 段落で項目をコーディングする必要があります。以下に例を示します。

```
Configuration section.
Repository.
    Class jbooleanArray is "jbooleanArray".
```

オブジェクト配列型に対する REPOSITORY 段落記入項目では、以下のいずれかの形式の外部クラス名を指定しなければなりません。

```
"jobjectArray"
"jobjectArray:external-classname-2"
```

最初のケースでは、REPOSITORY 記入項目は、配列エレメントが java.lang.Object 型のオブジェクトである配列クラスを指定しています。 2 番目のケースでは、REPOSITORY 記入項目は、配列のエレメントが *external-classname-2* 型のオブジェクトである配列クラスを指定しています。jobjectArray 型の指定と、配列のエレメントの外部クラス名の間の分離文字として、コロンをコーディングします。

次の例は、両方のケースを示しています。この例では、oa は、java.lang.Object 型のオブジェクトであるエレメントの配列を定義しています。また、aDepartment は、com.acme.Employee 型のオブジェクトであるエレメントの配列を定義しています。

```
Environment Division.
Configuration Section.
Repository.
    Class jobjectArray is "jobjectArray"
    Class Employee     is "com.acme.Employee"
    Class Department   is "jobjectArray:com.acme.Employee".
. . .
Linkage section.
01 oa          usage object reference jobjectArray.
01 aDepartment usage object reference Department.
. . .
Procedure division using by value aDepartment.
. . .
```

660 ページの『例: java コマンドを使用して実行される COBOL アプリケーション』

以下の Java 配列型は現在、COBOL プログラムとの相互協調処理にはサポートされていません。

表 85. **COBOL および Java で相互運用可能でない配列型**

Java データ型	対応する COBOL データ型
float[ ]	オブジェクト参照 jfloatArray
double[ ]	オブジェクト参照 jdoubleArray

関連タスク

619 ページの『クラス定義用の REPOSITORY 段落』

Java 配列の取り扱い

COBOL プログラムで配列を表すには、その配列の Java タイプに対応するデータ・タイプの単一基本項目が含まれているグループ項目をコーディングします。その配列に適した OCCURS または OCCURS DEPENDING ON 節を指定します。

例えば、次のコードは、jlongArray オブジェクトから 500 以下の整数値を受け取る構造を指定します。

```
01  longArray.
       02 X pic S9(10) comp-5 occurs 1 to 500 times depending on N.
```

特殊な Java 配列クラスのオブジェクトを操作するには、JNI が提供するサービスを呼び出します。サービスを使用して、配列の個々のエレメントにアクセスして設定し、呼び出したサービスを使用して、以下を行います。

表 86. **JNI 配列サービス**

サービス	入力引数	戻り値	目的
GetArrayLength	• JNI 環境ポインター • 配列オブジェクトの参照	2 進数フルワード整数としての配列の長さ	Java 配列オブジェクト内のエレメント数を取得する
NewBooleanArray、 NewByteArray、 NewCharArray、 NewShortArray、 NewIntArray、 NewLongArray	• JNI 環境ポインター • 2 進数フルワード整数としての、配列内のエレメントの数	配列オブジェクトの参照、または配列を構成できない場合は NULL	新しい Java 配列オブジェクトを作成する
GetBooleanArrayElements、 GetByteArrayElements、 GetCharArrayElements、 GetShortArrayElements、 GetIntArrayElements、 GetLongArrayElements	• JNI 環境ポインター • 配列オブジェクトの参照 • ブール項目へのポインターポインターが NULL でない場合は、配列エレメントのコピーが作成されたときは、ブール項目は true に設定される。コピーが作成された場合、変更を配列オブジェクトに書き戻す必要がある場合は、対応する Release*xxx*ArrayElements サービスを呼び出さなければならない。	ストレージ・バッファーへのポインター	配列エレメントを Java 配列からストレージ・バッファーに抽出する。サービスにより、ポインターがストレージ・バッファーに戻される。ポインターは、LINKAGE SECTION に定義される COBOL グループ・データ項目のアドレスとして使用することができる。

表 86. **JNI 配列サービス** *(続き)*

サービス	入力引数	戻り値	目的
ReleaseBooleanArrayElements、 ReleaseByteArrayElements、 ReleaseCharArrayElements、 ReleaseShortArrayElements、 ReleaseIntArrayElements、 ReleaseLongArrayElements	• JNI 環境ポインター • 配列オブジェクトの参照 • ストレージ・バッファーへのポインター • 2 進数フルワード整数としてのリリース・モード。詳細については、Java JNI の資料を参照。 (推奨: 配列の内容をコピーして戻し、ストレージ・バッファーを解放するには、0 を指定する。)	なし。配列のストレージは解放される。	Java 配列から抽出したエレメントが含まれているストレージ・バッファーを解放し、条件によっては、更新された配列値を配列オブジェクトにマップして戻す。
NewObjectArray	• JNI 環境ポインター • 2 進数フルワード整数としての、配列内のエレメントの数 • 配列エレメント・クラスに対するオブジェクト参照 • 最初のエレメント値に対するオブジェクト参照。すべての配列エレメントにはこの値が設定されます。	配列オブジェクトの参照、または配列を構成できない場合は NULL[1]	新しい Java オブジェクト配列を作成する。
GetObjectArrayElement	• JNI 環境ポインター • 配列オブジェクトの参照 • 2 進数フルワード整数としての、起点 0 の配列エレメント索引	オブジェクト参照 [2]	オブジェクト配列内の特定の索引のエレメントを返す。
SetObjectArrayElement	• JNI 環境ポインター • 配列オブジェクトの参照 • 2 進数フルワード整数としての、起点 0 の配列エレメント索引 • 新しい値に対するオブジェクト参照	なし [3]	オブジェクト配列内のエレメントを設定する。

1. システムがメモリー不足の場合、NewObjectArray は例外を throw します。
2. 索引が有効でない場合、GetObjectArrayElement は例外を throw します。
3. 索引が有効でない場合、または新しい値が配列のエレメント・クラスのサブクラスでない場合、SetObjectArrayElement は例外を throw します。

660 ページの『例: java コマンドを使用して実行される COBOL アプリケーション』 673 ページの『例: Java int 配列の処理』

関連タスク 669 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』 669 ページの『Java 用の配列およびストリングの宣言』 663 ページの『JNI サービスへのアクセス』

例: Java int 配列の処理

次の例は、Java 配列クラスと JNI サービスを使用した、COBOL での Java 配列の処理を示しています。

```
 cbl lib,thread,dll
 Identification division.
 Class-id. OOARRAY inherits Base.
 Environment division.
 Configuration section.
 Repository.
     Class Base is "java.lang.Object"
     Class jintArray is "jintArray".
 Identification division.
 Object.
 Procedure division.
   Identification division.
   Method-id. "ProcessArray".
   Data Division.
   Local-storage section.
   01 intArrayPtr pointer.
   01 intArrayLen pic S9(9) comp-5.
   Linkage section.
      COPY JNI.
   01 inIntArrayObj usage object reference jintArray.
   01 intArrayGroup.
      02 X pic S9(9) comp-5
          occurs 1 to 1000 times depending on intArrayLen.
   Procedure division using by value inIntArrayObj.
       Set address of JNIEnv to JNIEnvPtr
       Set address of JNINativeInterface to JNIEnv

       Call GetArrayLength
         using by value JNIEnvPtr inIntArrayObj
         returning intArrayLen
       Call GetIntArrayElements
         using by value JNIEnvPtr inIntArrayObj 0
         returning IntArrayPtr
       Set address of intArrayGroup to intArrayPtr

*  . . . process the array elements X(I) . . .

       Call ReleaseIntArrayElements
         using by value JNIEnvPtr inIntArrayObj intArrayPtr 0.
   End method "ProcessArray".
 End Object.
 End class OOARRAY.
```

Java ストリングの取り扱い

COBOL は、Java ストリング・データを Unicode で表します。 Java ストリングを COBOL プログラムで表すには、jstring クラスのオブジェクト参照としてストリングを宣言してください。続いて、JNI サービスを使用して、COBOL 英数字または国別 (Unicode) データを設定するか、オブジェクトから抽出します。

Unicode 用のサービス: jstring オブジェクト参照と COBOL USAGE NATIONAL データ項目 これらのサービスへのアクセスは、JNINativeInterface 環境構造の関数ポインターを使用して行います。

表 87. **jstring 参照と国別データ間の変換サービス**

サービス	入力引数	戻り値
NewString[1]	• JNI 環境ポインター • COBOL 国別データ項目などの、Unicode ストリングへのポインター • ストリングの文字数。2 進数フルワード	jstring オブジェクト参照。
GetStringLength	• JNI 環境ポインター • jstring オブジェクト参照	jstring オブジェクト参照の Unicode 文字数。2 進数フルワード。
GetStringChars[1]	• JNI 環境ポインター • jstring オブジェクト参照 • ブール・データ項目を指すポインター、または NULL	• jstring オブジェクトから抜き出された Unicode 文字の配列を指すポインター、または NULL (操作が失敗した場合)。ポインターは、ReleaseStringChars を使用して解放されるまで有効。 • ブール・データ項目へのポインターが NULL でないとき、ブール値は、ストリングのコピーが作成される場合には true に、コピーが作成されない場合には false に設定される。
ReleaseStringChars	• JNI 環境ポインター • jstring オブジェクト参照 • GetStringChars から戻された Unicode 文字の配列へのポインター	なし。配列のストレージは解放される。
1. システムがメモリー不足の場合、このサービスは例外を throw します。		

EBCDIC 用のサービス: jstring オブジェクト参照と COBOL 英数字データ (PIC X(*n*)) との間の変換を行うには、以下の z/OS サービス (JNI の拡張) を使用してください。これらのサービスへのアクセスは、JNI 環境構造 JNINativeInterface の関数ポインターを使用して行います。

表 88. **jstring 参照と英数字データ間の変換サービス**

サービス	入力引数	戻り値
NewStringPlatform	• JNI 環境ポインター • jstring オブジェクトに変換するヌル終了 EBCDIC 文字ストリングへのポインター • 結果のための jstring オブジェクト参照へのポインター • ストリングの Java エンコード名へのポインター。ヌル終了 EBCDIC 文字ストリング [1] として表す	2 進数フルワード整数としての戻りコード: **0** 正常。 **-1** 誤った形式の入力データまたは正しくない入力文字。 **-2** 非サポート・エンコード。jstring オブジェクト参照ポインターは NULL に設定されます。

表 88. **jstring 参照と英数字データ間の変換サービス** *(続き)*

サービス	入力引数	戻り値
GetStringPlatformLength	• JNI 環境ポインター • 長さのための jstring オブジェクト参照 • 結果のための 2 進数フルワード値へのポインター • ストリングの Java エンコード名へのポインター。ヌル終了 EBCDIC 文字ストリング [1] として表す	2 進数フルワード整数としての戻りコード: **0** 正常。 **-1** 誤った形式の入力データまたは正しくない入力文字。 **-2** 非サポート・エンコード。jstring オブジェクト参照ポインターは NULL に設定されます。 2 番目の引数が参照するヌル終了バイトなど、変換した Java ストリングを保持するために必要な出力バッファーの長さ (バイト単位) を、3 番目の引数に戻します。
GetStringPlatform	• JNI 環境ポインター • ヌル終了ストリングに変換する jstring オブジェクト参照 • 変換済みストリングのための出力バッファーへのポインター • 2 進数フルワード整数としての出力バッファーの長さ • ストリングの Java エンコード名へのポインター。ヌル終了 EBCDIC 文字ストリング [1] として表す	2 進数フルワード整数としての戻りコード: **0** 正常。 **-1** 誤った形式の入力データまたは正しくない入力文字。 **-2** 非サポート・エンコード。出力ストリングはヌル・ストリングに設定されます。 **-3** 変換バッファーがいっぱいです。

1. ポインターが NULL の場合、Java file.encoding プロパティーからのエンコードが使用されます。

これらの EBCDIC サービスは、IBM Java 2 Software Development Kit の一部である DLL としてパッケージされています。サービスの詳細については、IBM Java 2 Software Development Kit の jni_convert.h を参照してください。

サービスを呼び出すには、CALL *literal* ステートメントを使用してください。呼び出しは、libjvm.x DLL サイド・ファイルを介して解決されます。このファイルは、オブジェクト指向言語を使用する COBOL プログラムのリンク手順に含める必要があります。

例えば、次のコードは、EBCDIC ストリング 'MyConverter' から Java ストリング・オブジェクトを作成します。 (このコード・フラグメントは、 676 ページの『例: COBOL で書かれた J2EE クライアント』で詳細に示す、J2EE クライアント・プログラムからのものです。)

```
Move z"MyConverter" to stringBuf
Call "NewStringPlatform"
  using by value JNIEnvPtr
```

```
                address of stringBuf
                address of jstring1
                0
  returning rc
```

EBCDIC サービスが、COBOL プログラムから呼び出す唯一の JNI サービスである場合には、JNI.cpy コピーブックをコピーする必要はありません。また、JNI 環境ポインターとのアドレス可能度を設定する必要もありません。

UTF-8 用のサービス: Java ネイティブ・インターフェースでは、jstring オブジェクト参照と UTF-8 ストリングとの間の変換用のサービスも提供しています。これらのサービスは、COBOL プログラムでの使用にはお勧めしません。 z/OS プラットフォームで UTF-8 文字ストリングを取り扱うことは難しいためです。

関連タスク 663 ページの『JNI サービスへのアクセス』 669 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』 669 ページの『Java 用の配列およびストリングの宣言』 139 ページの『COBOL での国別データ (Unicode) の使用』 329 ページの『第 16 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行』

例: COBOL で書かれた J2EE クライアント

次の例では、J2EE 準拠 EJB サーバー上で実行する Enterprise Bean にアクセスできる COBOL クライアント・プログラムを示します。

COBOL クライアントは、「*Java 2 Enterprise Edition Developer's Guide*」の『Getting Started』の章にある J2EE クライアント・プログラムと同等です。インプリメンテーションを比較する上でのユーザーの便宜のために、2 番目の例では上記資料に記載されている同等の Java クライアントを示します。 (エンタープライズ Bean は単純な通貨コンバーター エンタープライズ Bean の Java インプリメンテーションで、同資料に記載されています。)

COBOL クライアント (ConverterClient.cbl)

```
 Process pgmname(longmixed),lib,dll,thread
*****************************************************************
* Demo J2EE client written in COBOL.                            *
*                                                               *
* Based on the sample J2EE client written in Java, which is     *
* given in the "Getting Started" chapter of "The Java(TM) 2     *
* Enterprise Edition Developer's Guide."                        *
*                                                               *
* The client:                                                   *
*   - Locates the home interface of a session enterprise bean   *
*       (a simple currency converter bean)                      *
*   - Creates an enterprise bean instance                       *
*   - Invokes a business method (currency conversion)           *
*****************************************************************
 Identification division.
 Program-id. "ConverterClient" is recursive.
 Environment Division.
 Configuration section.
 Repository.
     Class InitialContext is "javax.naming.InitialContext"
     Class PortableRemoteObject
                          is "javax.rmi.PortableRemoteObject"
     Class JavaObject     is "java.lang.Object"
```

```
     Class JavaClass      is "java.lang.Class"
     Class JavaException  is "java.lang.Exception"
     Class jstring        is "jstring"
     Class Converter      is "Converter"
     Class ConverterHome  is "ConverterHome".
 Data division.
 Working-storage section.
 01 initialCtx        object reference InitialContext.
 01 obj               object reference JavaObject.
 01 classObj          object reference JavaClass.
 01 ex                object reference JavaException.
 01 currencyConverter object reference Converter.
 01 home              object reference ConverterHome.
 01 homeObject redefines home object reference JavaObject.
 01 jstring1          object reference jstring.
 01 stringBuf         pic X(500) usage display.
 01 len               pic s9(9) comp-5.
 01 rc                pic s9(9) comp-5.
 01 amount            comp-2.
 Linkage section.
     Copy JNI.
 Procedure division.
     Set address of JNIenv to JNIEnvPtr
     Set address of JNINativeInterface to JNIenv

*****************************************************************
* Create JNDI naming context.                                   *
*****************************************************************
     Invoke InitialContext New returning initialCtx
     Perform JavaExceptionCheck

*****************************************************************
* Create a jstring object for the string "MyConverter" for use  *
* as argument to the lookup method.                             *
*****************************************************************
     Move z"MyConverter" to stringBuf
     Call "NewStringPlatform"
       using by value JNIEnvPtr
                      address of stringBuf
                      address of jstring1
                      0
       returning rc
     If rc not = zero then
        Display "Error occurred creating jstring object"
        Stop run
     End-if

*****************************************************************
* Use the lookup method to obtain a reference to the home       *
* object bound to the name "MyConverter".  (This is the JNDI    *
* name specified when deploying the J2EE application.)          *
*****************************************************************
     Invoke initialCtx "lookup" using by value jstring1
       returning obj
     Perform JavaExceptionCheck

*****************************************************************
* Narrow the home object to be of type ConverterHome.           *
* First obtain class object for the ConverterHome class, by     *
* passing the null-terminated ASCII string "ConverterHome" to   *
* the FindClass API.  Then use this class object as the         *
* argument to the static method "narrow".                       *
*****************************************************************
     Move z"ConverterHome" to stringBuf
     Call "__etoa"
       using by value address of stringBuf
       returning len
```

```
     If len = -1 then
        Display "Error occurred on ASCII conversion"
        Stop run
     End-if
     Call FindClass
       using by value JNIEnvPtr
                      address of stringBuf
       returning classObj
     If classObj = null
        Display "Error occurred locating ConverterHome class"
        Stop run
     End-if
     Invoke PortableRemoteObject "narrow"
       using by value obj
                      classObj
       returning homeObject
     Perform JavaExceptionCheck

*****************************************************************
* Create the ConverterEJB instance and obtain local object      *
* reference for its remote interface                            *
*****************************************************************
     Invoke home "create" returning currencyConverter
     Perform JavaExceptionCheck

*****************************************************************
* Invoke business methods                                       *
*****************************************************************
     Invoke currencyConverter "dollarToYen"
       using by value +100.00E+0
       returning amount
     Perform JavaExceptionCheck

     Display amount

     Invoke currencyConverter "yenToEuro"
       using by value +100.00E+0
       returning amount
     Perform JavaExceptionCheck

     Display amount

*****************************************************************
* Remove the object and return.                                 *
*****************************************************************
     Invoke currencyConverter "remove"
     Perform JavaExceptionCheck

     Goback
     .

*****************************************************************
* Check for thrown Java exceptions                              *
*****************************************************************
 JavaExceptionCheck.
     Call ExceptionOccurred using by value JNIEnvPtr
       returning ex
     If ex not = null then
        Call ExceptionClear using by value JNIEnvPtr
        Display "Caught an unexpected exception"
        Invoke ex "PrintStackTrace"
        Stop run
     End-if
     .
 End program "ConverterClient".
```

Java クライアント (ConverterClient.java)

```
/*
 *
 * Copyright 2000 Sun Microsystems, Inc. All Rights Reserved.
 *
 * This software is the proprietary information of Sun Microsystems, Inc.
 * Use is subject to license terms.
 *
 */

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;

import Converter;
import ConverterHome;

public class ConverterClient {

   public static void main(String[] args) {
       try {
           Context initial = new InitialContext();
           Object objref = initial.lookup("MyConverter");

           ConverterHome home =
               (ConverterHome)PortableRemoteObject.narrow(objref,
                                          ConverterHome.class);

           Converter currencyConverter = home.create();

           double amount = currencyConverter.dollarToYen(100.00);
           System.out.println(String.valueOf(amount));
           amount = currencyConverter.yenToEuro(100.00);
           System.out.println(String.valueOf(amount));

           currencyConverter.remove();

       } catch (Exception ex) {
           System.err.println("Caught an unexpected exception!");
           ex.printStackTrace();
       }
   }
}
```

関連タスク 329 ページの『第 16 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行』 *WebSphere for z/OS: Applications*

関連参照

Java 2 Enterprise Edition Developer's Guide

第 7 部 特殊処理

第 32 章 割り込みおよびチェックポイント・リスタート

時間を延長してプログラムを実行していると、ジョブ終了になる前に、割り込みによって処理が停止することがあります。z/OS のチェックポイント・リスタート機能を使用すれば、割り込みを受けたプログラムを、ジョブ・ステップの先頭からまたは設定したチェックポイントから再始動することができます。

チェックポイント・リスタート機能は、多くの余分な処理を引き起こすので、マシン誤動作、入出力エラー、またはオペレーターの意図的な介入が原因の割り込みであると予想される場合にだけ使用するようにしてください。

チェックポイント・ルーチンは、プログラムが入っている COBOL ロード・モジュールから開始されます。プログラムの実行中に、チェックポイント・ルーチンは、COBOL RERUN 節を使用して指定されたポイントでレコードを作成します。チェックポイント・レコードには、プログラムがそのチェックポイントに達したときにレジスターおよび主記憶域の中に保管されていた情報のスナップショットが入れられます。

再始動ルーチンにより、割り込まれたプログラムが再始動されます。プログラムが割り込みを受けた後、いつでも再始動することができます。すなわち、即時に再始動すること (自動再始動) も、後で再始動すること (据え置き再始動) もできます。

関連タスク
『チェックポイントの設定』
687 ページの『プログラムの再始動』
690 ページの『再始動用のジョブの再実行依頼』
z/OS DFSMS: Checkpoint/Restart

関連参照 685 ページの『チェックポイント・データ・セットの定義用の DD ステートメント』 686 ページの『チェックポイント時に生成されるメッセージ』 689 ページの『据え置き再始動の要求用の形式』

チェックポイントの設定

チェックポイントを設定するには、ジョブ制御ステートメントを使用し、ENVIRONMENT DIVISION の RERUN 節を使用してください。それぞれの RERUN 節を特定の COBOL ファイルと関連付ける必要があります。

RERUN 節は、COBOL ファイル内の特定数のレコードが処理されるたびに、または END OF VOLUME に達したときに、チェックポイント・レコードをチェックポイント・データ・セットに書き込むことを指示します。RERUN 節は、EXTERNAL 属性で定義されたファイルには使用できません。

複数の COBOL ファイルからのチェックポイント・レコードを 1 つのチェックポイント・データ・セットに書きこむことができますが、チェックポイント・レコード専用の別個のデータ・セットを使用する必要があります。チェックポイント・レコードをプログラム・データ・セットの 1 つに組み込むことはできません。

制約事項: チェックポイント・データ・セットは順次編成でなければなりません。チェックポイントを、VSAM データ・セット、または拡張形式 QSAM データ・セットに割り振られたデータ・セットに書き込むことはできません。また、実行単位内のいずれかのプログラムが拡張形式 QSAM データ・セットをオープンしている場合にも、チェックポイントを取ることはできません。

チェックポイント・レコードは、DD ステートメントで定義されたチェックポイント・データ・セットに書き込まれます。 DD ステートメントでは、次のチェックポイント方式も選択します。

単一 (単一のチェックポイントを保管します)
ある時点では、チェックポイント・レコードは 1 つだけ存在します。最初のチェックポイント・レコードが作成されると、それ以降のチェックポイント・レコードは前のレコードにオーバーレイします。

この方式は、ほとんどのプログラムで受け入れられます。チェックポイント・データ・セット上のスペースを節約し、最新のチェックポイントでプログラムを再始動することができます。

複数 (複数の連続するチェックポイントを保管します)
チェックポイントは順次記録され、番号が付けられます。それぞれのチェックポイントが保管されます。

最後に取られたチェックポイント以外のチェックポイントでプログラムを再始動したい場合に、この方式を使用します。

標準 COBOL 85 に完全に準拠するには、複数チェックポイント方式を使用する必要があります。

ソート操作時のチェックポイントの場合、以下の要件があります。

- ソート操作中にチェックポイントを設定する場合は、実行用のジョブ制御プロシージャーの中に SORTCKPT の DD ステートメントを追加します。
- ASCII で照合されるソートでチェックポイント・レコードを設定することができますが、チェックポイント・データ・セットを示す *system-name* に ASCII ファイルを指定してはなりません。

関連タスク 260 ページの『DFSORT によるチェックポイント・リスタートの使用』 『チェックポイントの設計』 685 ページの『チェックポイントが成功したかどうかのテスト』

関連参照 685 ページの『チェックポイント・データ・セットの定義用の DD ステートメント』

チェックポイントの設計

データを容易に再構成できるように、プログラム内の重要ポイントにチェックポイントを設計してください。チェックポイントから再始動までの間に、ファイルの内容を変更してはなりません。

ディスク・ファイルを使用するプログラムの場合、前に処理されたレコードを識別できるようにプログラムを設計してください。例えば、支払利息が周期的に更新される貸付レコードを含んでいるディスク・ファイルを考えてみましょう。チェック

ポイントが取られ、レコードが更新された後に、プログラムが割り込まれた場合、最後のチェックポイントの後に更新されたレコードが、プログラムの再始動時に前と同様の更新が行われていないことをテストしたいと思います。これを行うために、各レコードに日付フィールドをセットアップし、レコードが処理されるたびに日付を更新します。次に再始動された後、日付フィールドをテストし、レコードが前に処理されたかどうかを判別します。

印刷ファイルを効率よく位置変更するためには、ページの最後の行を印刷した後にファイル上でチェックポイントを取ってください。

チェックポイントが成功したかどうかのテスト

チェックポイントを出す入力ステートメントまたは出力ステートメントが実行されるたびに、RETURN-CODE 特殊レジスターがチェックポイント・ルーチンからの戻りコードで更新されます。したがって、チェックポイントが成功したかどうかをテストして、再始動に関する条件が適切かどうかを判別できます。

戻りコードが 4 より大きい場合は、チェックポイントに関するエラーが発生しています。戻りコードを調べ、誤った出力が作成される原因となるような再始動を回避してください。

関連参照
z/OS DFSMS: Checkpoint/Restart (戻りコード)

チェックポイント・データ・セットの定義用の DD ステートメント

チェックポイント・データ・セットを定義するには、DD ステートメントを使用します。

テープの場合:

```
//ddname  DD DSNAME=data-set-name,
//           [VOLUME=SER=volser,]UNIT=device-type,
//           DISP=({NEW|MOD},PASS)
```

直接アクセス装置の場合:

```
//ddname  DD DSNAME=data-set-name,
//           [VOLUME=(PRIVATE,RETAIN,SER=volser),]
//           UNIT=device-type,SPACE=(subparms),
//           DISP=({NEW|MOD},PASS,KEEP)
```

ddname
DD ステートメントへのリンクを提供します。COBOL RERUN 節で使用される *assignment-name* の ddname 部分と同じ。

data-set-name
再始動プロシージャーに対してチェックポイント・データ・セットを識別します。チェックポイント・レコードを記録するために使用される、データ・セットに与えられる名前。

volser 通し番号でボリュームを識別します。

device-type
装置を識別します。

subparms
データ・セットに必要なトラック・スペースの大きさを指定します。

MOD 複数連続チェックポイント・メソッドを指定します。

NEW 単一チェックポイント・メソッドを指定します。

PASS ジョブの最後のジョブ・ステップを除き、ジョブ・ステップの正常完了時に、データ・セットの削除を妨げます。最後のステップであれば、データ・セットは削除されます。

KEEP ジョブ・ステップが異常終了した場合に、データ・セットを保管します。

『例: チェックポイント・データ・セットの定義』

例: チェックポイント・データ・セットの定義

以下の例は、チェックポイント・データ・セットの定義に使用できる、JCL および COBOL コーディングを示しています。

テープを使用した、単一チェックポイント・レコードの作成:

```
//CHECKPT  DD DSNAME=CHECK1,VOLUME=SER=ND0003,
//            UNIT=TAPE,DISP=(NEW,KEEP),LABEL=(,NL)
            . . .
       ENVIRONMENT DIVISION.
            . . .
            RERUN ON CHECKPT EVERY
              5000 RECORDS OF ACCT-FILE.
```

ディスクを使用した、単一チェックポイント・レコードの作成:

```
//CHEK     DD DSNAME=CHECK2,
//            VOLUME=(PRIVATE,RETAIN,SER=DB0030),
//            UNIT=3380,DISP=(NEW,KEEP),SPACE=(CYL,5)
            . . .
       ENVIRONMENT DIVISION.
            . . .
            RERUN ON CHEK EVERY
              20000 RECORDS OF PAYCODE.
            RERUN ON CHEK EVERY
              30000 RECORDS OF IN-FILE.
```

テープを使用した、複数の連続チェックポイント・レコードの作成:

```
//CHEKPT   DD DSNAME=CHECK3,VOLUME=SER=111111,
//            UNIT=TAPE,DISP=(MOD,PASS),LABEL=(,NL)
            . . .
       ENVIRONMENT DIVISION.
            . . .
            RERUN ON CHEKPT EVERY
              10000 RECORDS OF PAY-FILE.
```

チェックポイント時に生成されるメッセージ

システム・チェックポイント・ルーチンは、コンソールに通知メッセージを表示することにより、取られたチェックポイントの状況をオペレーターに知らせます。

チェックポイントが正常に完了するたびに、ジョブ名 (*ddname*、*unit*、*volser*) を、取られたチェックポイント (*checkid*) に関連付けるメッセージが表示されます。

制御プログラムは、*checkid* を 8 文字の文字ストリングとして割り当てます。先頭文字は文字 *C* で、その後にチェックポイントを示す 10 進数が続きます。例えば、次のメッセージは、ジョブ・ステップで設定された 4 番目のチェックポイントを示します。

```
checkid C0000004
```

プログラムの再始動

システム再始動ルーチンは、チェックポイント・レコードに記録された情報を取得し、主ストレージおよびすべてのレジスターの内容を復元し、プログラムを再始動します。

再始動ルーチンは、次のいずれかの方法で開始することができます。

- 割り込みによってプログラムが停止した時に、自動的に
- 後で据え置き再始動として

ジョブ制御言語の RD パラメーターは、再始動のタイプを決定します。 RD パラメーターは、JOB または EXEC ステートメントのいずれかに指定することができます。 JOB ステートメントにコーディングされた場合、このパラメーターは、EXEC ステートメントの RD パラメーターをオーバーライドします。

再始動とチェックポイント書き込みの両方を抑止する場合は、RD=NC をコーディングします。

制約事項: SORT または MERGE 操作時に COBOL プログラムによって取られたチェックポイントで再始動しようとすると、エラー・メッセージが出され、再始動は取り消されます。 DFSORT によって取られたチェックポイントだけが有効です。

データ・セットの DD ステートメントに SYSOUT パラメーターが指定されていると、そのデータ・セットは、再始動のタイプに応じてさまざまな方法で処理されます。

チェックポイント・データ・セットがマルチボリュームの場合は、チェックポイント項目が書き込まれたボリュームのシーケンス番号を、VOLUME パラメーターに含めなければなりません。チェックポイント・データ・セットが、標準外ラベル付きまたはラベルなしの 7 トラック・テープに置かれている場合は、SYSCHK DD ステートメントに DCB=(TRTCH=C,. . .) が含まれなくてはなりません。

関連タスク 260 ページの『DFSORT によるチェックポイント・リスタートの使用』 『自動再始動の要求』 688 ページの『据え置き再始動の要求』

自動再始動の要求

自動再始動が行われるのは、最新のチェックポイントが取られたときだけです。割り込みの前にチェックポイントが取られなかった場合、自動再始動は、ジョブ・ステップの先頭から行われます。

自動再始動が行われる時は、システムはユニット・レコード装置を除くすべての装置の位置変更をします。

自動再始動を行いたい場合は、RD=R または RD=RNC をコーディングします。

- RD=R は、最新のチェックポイントで再始動を行うことを示します。チェックポイントを記録するためには、プログラム中の少なくとも 1 つのデータ・セットに対して RERUN 節をコーディングしてください。割り込みの前にチェックポイントが取られなかった場合、自動再始動は、ジョブ・ステップの先頭から行われます。
- RD=RNC は、チェックポイントを書き込まないこと、およびどの再始動もジョブ・ステップの先頭から行うことを示します。この場合、RERUN 節は不要です。もしあっても、無視されます。

RD パラメーターを省略すると、CHKPT マクロ命令がアクティブのままとなり、処理中にチェックポイントが取られることがあります。最初のチェックポイントの後で割り込みが行われる場合、自動再始動が行われます。

自動的に再始動するためには、プログラムが次の条件を満たしていなければなりません。

- プログラムでは、RD パラメーターを使用するか、またはチェックポイントを取ることによって、再始動を要求しなければなりません。
- ジョブを中止させた異常終了は、再始動を可能にするコードを戻さなければなりません。
- オペレーターが、再始動を許可しなければなりません。

690 ページの『例: ステップ再始動の要求』

据え置き再始動の要求

据え置き再始動は、取られたチェックポイントが必ずしも最新のものでなくても、任意のチェックポイントで行うことが可能です。ジョブ・ステップの先頭以外のチェックポイントからプログラムを再始動することができます。

据え置き再始動が正常に完了すると、システムは、ジョブが再始動したことを示すメッセージをコンソールに表示します。それから、制御権がプログラムに与えられます。

据え置き再始動を行いたい場合は、RD パラメーターを RD=NR としてコーディングします。この形式のパラメーターは自動再始動を抑止しますが、RERUN 節がコーディングされていれば、チェックポイント・レコードの書き込みを許可します。

据え置き再始動は、JOB カードで RESTART パラメーターを指定し、チェックポイント・データ・セットを識別する SYSCHK DD ステートメントを使用して、要求してください。SYSCHK DD ステートメントがジョブの中にあっても、JOB ステートメントに RESTART パラメーターが含まれていないと、その SYSCHK DD ステートメントは無視されます。CHECKID サブパラメーターの指定なしの RESTART パラメーターをジョブに含める場合は、ジョブの最初の EXEC ステートメントの前に SYSCHK DD ステートメントがあってはなりません。

690 ページの『例: 特定チェックポイント・ステップでのジョブの再始動』

関連参照
『据え置き再始動の要求用の形式』

据え置き再始動の要求用の形式

JOB ステートメントおよび SYSCHK DD ステートメントの RESTART パラメーターの形式は次のとおりです。

```
//jobname  JOB MSGLEVEL=1,RESTART=(request[,checkid])
//SYSCHK   DD  DSNAME=data-set-name,
//             DISP=OLD[,UNIT=device-type,
//             VOLUME=SER=volser]
```

MSGLEVEL=1 (または MSGLEVEL=(1,*y*))
MSGLEVEL は必須です。

RESTART=(*request*,[*checkid*])
再始動が行われる特定のチェックポイントを識別します。

request 次のいずれかの形式を取ります。

***** ジョブの先頭から再始動することを示します。

stepname
ジョブ・ステップの先頭から再始動することを示します。

stepname.procstep
ジョブ・ステップ内のプロシージャー・ステップから再始動することを示します。

checkid
再始動が行われるチェックポイントを指示します。

SYSCHK チェックポイント・データ・セットを制御プログラムに識別するために使用される DD 名。SYSCHK DD ステートメントは、再実行依頼されたジョブの最初の EXEC ステートメントの直前で、かつ JOBLIB ステートメントの後になければなりません。

data-set-name
チェックポイント・データ・セットを識別します。チェックポイントが取られた時に使用された名前と同じでなければなりません。

device-type* および *volser
チェックポイント・データ・セットが入っているボリュームの装置タイプおよび通し番号を識別します。

『例: 据え置き再始動の要求』

例: 据え置き再始動の要求

この例は、チェックポイント ID (CHECKID) C0000003 で IGYWCLG プロシージャーの GO ステップを再始動する JCL を示しています。

```
//jobname  JOB MSGLEVEL=1,RESTART=(stepname.GO,C0000003)
//SYSCHK   DD DSNAME=CHEKPT,
//            DISP=OLD[,UNIT=3380,VOLUME=SER=111111]
           . . .
```

再始動用のジョブの再実行依頼

再始動のためにジョブを再実行依頼する場合、再始動されるジョブ・ステップの実行に影響する可能性のある、どの DD ステートメントについても注意してください。再始動ルーチンは、再入力されたジョブの DD ステートメントの情報を使用して、再始動後に使用するファイルをリセットします。

ジョブ・ステップ終了時にデータ・セットを削除したい場合は、(DELETE ではなく) PASS または KEEP の条件付き後処理を指定してください。この後処理を指定すると、割り込みによって強制的に再始動が実行される場合にそのデータ・セットが使えるようになります。ステップの先頭からジョブを再始動したい場合は、前回の実行で作成したデータ・セット (DD ステートメントで NEW と定義された) をすべて廃棄するか、またはデータ・セットに OLD のマークを付けるように DD ステートメントを変更しなければなりません。

テープまたはディスク上の入力データ・セットは、システムによって自動的に再配置されます。

691 ページの『例: ステップ再始動のためのジョブの再実行依頼』
691 ページの『例: チェックポイント・リスタートのためのジョブの再実行依頼』

例: 特定チェックポイント・ステップでのジョブの再始動

この例は、特定ステップでジョブを再始動するための一連のジョブ制御ステートメントを示しています。

```
//PAYROLL  JOB  MSGLEVEL=1,REGION=80K,
//              RESTART=(STEP1,CHECKPT4)
//JOBLIB   DD   DSNAME=PRIV.LIB3,DISP=OLD
//SYSCHK   DD   DSNAME=CHKPTLIB,
//              [UNIT=TAPE,VOL=SER=456789,]
//              DISP=(OLD,KEEP)
//STEP1    EXEC PGM=PROG4,TIME=5
```

例: ステップ再始動の要求

この例は、異常終了したジョブ・ステップのステップ再始動を要求する RD パラメーターの使用法を示しています。

```
//J1234    JOB  386,SMITH,MSGLEVEL=1,RD=R
//S1       EXEC PGM=MYPROG
//INDATA   DD   DSNAME=INVENT[,UNIT=TAPE],DISP=OLD,
//              [VOLUME=SER=91468,]
//              LABEL=RETPD=14
//REPORT   DD   SYSOUT=A
//WORK     DD   DSNAME=T91468,DISP=(,,KEEP),
//              UNIT=SYSDA,SPACE=(3000,(5000,500)),
//              VOLUME=(PRIVATE,RETAIN,,6)
//DDCKPNT  DD   UNIT=TAPE,DISP=(MOD,PASS,CATLG),
//              DSNAME=C91468,LABEL=(,NL)
```

DDCKPNT DD ステートメントはチェックポイント・データ・セットを定義します。このステップの場合、RERUN 節の実行後、CHKPT 取り消しが出されなければ、自動チェックポイント・リスタートを行うことができます。

例: ステップ再始動のためのジョブの再実行依頼

次の例は、ステップ再始動のためのジョブを再実行依頼する前に JCL に対して行うことができる変更を示しています。

```
//J3412    JOB  386,SMITH,MSGLEVEL=1,RD=R,RESTART=*
//S1       EXEC PGM=MYPROG
//INDATA   DD   DSNAME=INVENT[,UNIT=TAPE],DISP=OLD,
//              [VOLUME=SER=91468,]LABEL=RETPD=14
//REPORT   DD   SYSOUT=A
//WORK     DD   DSNAME=S91468,
//              DISP=(,,KEEP),UNIT=SYSDA,
//              SPACE=(3000,(5000,500)),
//              VOLUME=(PRIVATE,RETAIN,,6)
//DDCHKPNT DD   UNIT=TAPE,DISP=(MOD,PASS,CATLG),
//              DSNAME=R91468,LABEL=(,NL)
```

上記の例では以下の変更が行われました。

- 元のジョブと再始動されたジョブを区別するために、ジョブ名が変更されました (J1234 から J3412 へ)。
- RESTART パラメーターが JOB ステートメントに追加されて、最初のジョブ・ステップから再始動を開始するように指示します。
- WORK DD ステートメントには、このデータ・セットに対して KEEP という条件付き後処理が最初に割り当てられていました。
  - 前回ジョブを実行した際にこのステップが正常終了した場合には、データ・セットは削除されているので、このステートメントに変更を加える必要はありません。
  - ステップが異常終了した場合、データ・セットは保管されています。その場合は、新しいデータ・セット (示されているように T91468 ではなく S91468) を定義するか、またはジョブを再実行依頼する前にデータ・セットの状況を OLD に変更しなければなりません。
- 新しいデータ・セット (C91468 ではなく R91468) も、チェックポイント・データ・セットとして定義されました。

690 ページの『例: ステップ再始動の要求』

例: チェックポイント・リスタートのためのジョブの再実行依頼

次の例は、チェックポイント・リスタートのためのジョブを再実行依頼する前に JCL に対して行うことができる変更を示しています。

```
//J3412    JOB  386,SMITH,MSGLEVEL=1,RD=R,
//              RESTART=(*,C0000002)
//SYSCHK   DD   DSNAME=C91468,DISP=OLD
//S1       EXEC PGM=MYPROG
//INDATA   DD   DSNAME=INVENT,UNIT=TAPE,DISP=OLD,
//              VOLUME=SER=91468,LABEL=RETPD=14
//REPORT   DD   SYSOUT=A
//WORK     DD   DSNAME=T91468,DISP=(,,KEEP),
//              UNIT=SYSDA,SPACE=(3000,(5000,500)),
//              VOLUME=(PRIVATE,RETAIN,,6)
//DDCKPNT  DD   UNIT=TAPE,DISP=(MOD,KEEP,CATLG),
//              DSNAME=C91468,LABEL=(,NL)
```

上記の例では以下の変更が行われました。

- 元のジョブと再始動されたジョブを区別するために、ジョブ名が変更されました (J1234 から J3412 へ)。
- RESTART パラメーターが JOB ステートメントに追加されて、C0000002 という名前のチェックポイント項目で最初のステップから再始動を開始するように指示します。
- DD ステートメントの DDCKPNT では、チェックポイント・データ・セットに CATLG という条件付き後処理が最初に割り当てられました。
  - 前回ジョブを実行した際にこのステップが正常終了した場合には、データ・セットは保管されています。その場合、SYSCHK DD ステートメントには、チェックポイント・データ・セットの検索に必要なすべての情報が含まれていなければなりません。
  - ジョブが異常終了した場合、データ・セットはカタログされています。その場合は、示されているように、SYSCHK DD ステートメントに必要なパラメーターは DSNAME と DISP だけです。

V=R が指定された場合は、実行されているジョブでチェックポイントが取られても、十分なページング不能な動的ストレージが使用可能になるまで、そのジョブは再始動することはできません。

第 33 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
694 ページの『2000 年言語拡張 (MLE)』

関連タスク
696 ページの『日付に関連したロジック問題の解決』
702 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』
705 ページの『リテラルを日付として操作する』
708 ページの『トリガーと限界の設定』
710 ページの『日付別のソートおよびマージ』
711 ページの『日付フィールドに対する算術の実行』
714 ページの『日付処理の明示的制御』
716 ページの『日付関連診断メッセージの分析および回避』
718 ページの『日付処理上の問題の回避』

関連参照
355 ページの『DATEPROC』

404 ページの『YEARWINDOW』
DATE FORMAT 節 (*Enterprise COBOL 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 Enterprise COBOL 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。この節は、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ために、DATA DIVISION 内の項目に追加されます。

  DATE FORMAT 節には幾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。詳細については、以下の関連参照を参照してください。
- 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
695 ページの『この拡張の原則と目標』

関連参照
355 ページの『DATEPROC』
404 ページの『YEARWINDOW』
日付フィールドの使用に関する制約事項 (*Enterprise COBOL 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 年言語拡張は幾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、3 つのアプローチ (世紀ウィンドウの使用、内部ブリッジング、または全フィールド拡張) のいずれか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 89. **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 89. **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。

698 ページの『例: 世紀ウィンドウ』
699 ページの『例: 内部ブリッジング』
700 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
『世紀ウィンドウの使用』
698 ページの『内部ブリッジングの使用』
699 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 2007 年に稼働中である場合、世紀ウィンドウは 1957 から 2056 であり、2008 年には自動的に 1958 から 2057 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。

698 ページの『例: 世紀ウィンドウ』

関連参照
355 ページの『DATEPROC』
404 ページの『YEARWINDOW』
DATE FORMAT 節 (*Enterprise COBOL 言語解説書*)
日付フィールドの使用に関する制約事項 (*Enterprise COBOL 言語解説書*)

例: 世紀ウィンドウ

次の例は、DATE FORMAT 節によってプログラムを変更して自動日付ウィンドウ操作の機能を使用する方法を (太字で) 示しています。

```
CBL LIB,QUOTE,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 070102 (2007 年 1 月 2 日) を含み、Date-Returned が 061231 (2006 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 (より以前なので)、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。

『例: 内部ブリッジング』

関連タスク
697 ページの『世紀ウィンドウの使用』
711 ページの『日付フィールドに対する算術の実行』
『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。

『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  LIB,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*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。

使用している DFSORT (または等価の製品) のバージョンに適切な機能が備わっている場合、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サポートされます。ソートおよびマージの操作以外で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。

703 ページの『例: 年先行型日付フィールドの比較』

関連概念
『互換性のある日付』

関連タスク 710 ページの『日付別のソートおよびマージ』
704 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

次の例は、拡張日付フィールドと比較されるウィンドウ表示日付フィールドを示しています。

```
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 230107 (2007 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 230108 (2008 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。しかし、Last-Review-Date が 230199 の場合に 1 を加算すると 230200 になり、意図した結果にはなりません。

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 081220.
```

世紀ウィンドウが 1950-2049 で、Date-Due の内容が 081220 (2008 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 = 081220
```

リテラル 081220 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、1908 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、それ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 (081220 "YYXXXX")
If Date-Due = Function DATEVAL (20081220 "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。以下に例を示します。

```
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(-60) 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

706 ページの『仮定による世紀ウィンドウ』
707 ページの『非日付の処理』

関連タスク
714 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。ウィンドウ表示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、非日付をウィンドウ表示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、これは意図した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。以下に例を示します。

```
If Date-Complete Less than
    Function Dateval (Date-Target "YYXXXX") . . .
```

関連タスク

714 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。

トリガーと限界の設定

トリガーと限界は、その値が非数値であるため、またはその値の中の年以外の部分が実際の日付であり得ないような値であるため、有効な日付には決して適合しない特殊値です。トリガーと限界は、日付フィールドにおいて認識されますが、日付フィールドと組み合わせて使われる非日付データにおいても認識されます。

日付フィールドのタイプ	特殊値
英数字のウィンドウ表示日付または年フィールド	HIGH-VALUE、LOW-VALUE、および SPACE
DATE FORMAT 節で少なくとも 1 つの X が指定された英数字および数字のウィンドウ表示日付フィールド (すなわち、年号だけではない日付フィールド)	すべて 9 またはすべて 0

トリガーと限界の相違は値にあるのではなく、その使用方法にあります。トリガーまたは限界としては、任意の特殊値を使用できます。

特殊値をトリガーとして使用すると、「日付が未初期化」または「会計が決済日超過」などの特定の条件を示すことができます。特殊値を限界として使用すると、それは有効などの日付よりも前または後の日付として機能することになります。LOW-VALUE、SPACE、および 0 は下限であり、HIGH-VALUE および全桁 9 は上限です。

トリガーと限界のサポートを有効にするには、DATEPROC コンパイラー・オプションの TRIG サブオプションを指定します。DATEPROC(TRIG) コンパイラー・オプションが有効であると、ウィンドウ表示日付フィールドの自動拡張において (比較や算術などでオペランドとして使われる前)、それらの特殊値が認識されます。

DATEPROC(TRIG) オプションを使用すると、ウィンドウ表示日付の比較の際に実行速度が遅くなります。DATEPROC(NOTRIG) オプションは、すべてのウィンドウ表示日付フィールドの有効な日付値を想定するパフォーマンス・オプションです。

実ウィンドウ表示日付フィールドまたは仮定ウィンドウ表示日付フィールドの内容がトリガーである場合は、通常のウィンドウ操作の場合のように世紀の値が 19 または 20 と推測されるのではなく、日付の拡張結果の世紀部分に値が波及するかのようにしてコンパイラーによって拡張されます。それによって、アプリケーションで特殊値をテストしたり、上限または下限として特殊値を使用したりできます。また、DATEPROC(TRIG) を指定すると、SORT および MERGE ステートメントによる DFSORT 特殊標識のサポートも使用可能になります。これらは、トリガーと限界に対応します。

709 ページの『例: 限界の使用』

関連タスク
『符号条件の使用』

関連参照
355 ページの『DATEPROC』

例: 限界の使用

この例は、どうすれば有効期限フィールドを使用して通常の有効期限 (さもなければ「無期限の」予約購読を許容する上限) を保持できるかを示しています。

例えば、予約購読の期限が切れたかどうかをアプリケーションで検査する際に、一部の予約購読は無期限扱いにしたいという場合を考えてみましょう。次のコード・フラグメントを見てください。

```
Process Dateproc(Flag,Trig). . .
. . .
01 SubscriptionRecord.
   03 ExpirationDate  PIC 9(6) Date Format yyxxxx.
. . .
77 TodaysDate  Pic 9(6) Date Format yyxxxx.
. . .
   If TodaysDate >= ExpirationDate
      Perform SubscriptionExpired
```

アプリケーションが次の値を検出したとしましょう。

- 今日の日付は 2007 年 1 月 4 日 (TodaysDate では 070104 と表されます) です。
- 予約申し込みレコードには、通常の有効期限である 1999 年 12 月 31 日 (ExpirationDate では 991231 と表されます) が含まれています。
- 別の予約申し込みレコードには、特別の有効期限 (ExpirationDate では 999999 とコーディングされます) が含まれています。

どちらの日付もウィンドウ表示されているので、最初の予約申し込みは、20070104 が 19991231 と比較される場合のようにテストされ、そのテストは正常に行われます。しかし、コンパイラーが特殊値を検出すると、ウィンドウ操作ではなく拡張が引き起こされます。したがって、20070104 が 99999999 と比較される場合のようにテストが行われます。このテストは常に失敗します。

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、DATEPROC コンパイラー・オプションの NOTRIG サブオプションを指定してコンパイルしている場合、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 組み込み関数または DATEPROC コンパイラー・オプションの TRIG サブオプションを使用しても、同じ結果が得られます。

関連概念
707 ページの『非日付の処理』

関連タスク 708 ページの『トリガーと限界の設定』
714 ページの『日付処理の明示的制御』

関連参照
355 ページの『DATEPROC』

日付別のソートおよびマージ

ご使用のソート製品が Y2PAST オプションとウィンドウ表示西暦年 ID (Y2B、Y2C、Y2D、Y2S、および Y2Z) をサポートしている場合は、ウィンドウ表示日付フィールドをソート・キーとして使用してソートおよびマージ操作を実行することができます。実際のところ DATE FORMAT 節を使用して指定できるすべての日付フィールドがサポートされますが、それには 2 進年フィールドと年末尾型日付フィールドも含まれます。

フィールドは、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に従って、ウィンドウ表示西暦年順序でソートされます。ご使用のソート製品が Y2T、Y2U、Y2W、Y2X、および Y2Y の日付フィールド ID もサポートする場合は、DATEPROC コンパイラー・オプションの TRIG サブオプションを使用することができます。

DFSORT が認識する特殊標識は、COBOL がサポートする次のものと厳密に合致します。英数字の日付または年フィールド用の LOW-VALUE、HIGH-VALUE、および SPACE、ならびに少なくとも 1 桁の年以外の桁を持つ、数値および英数字の日付フィールド用の「すべてゼロ」および「すべて 9」。

DFSORT は、ソートおよびマージを行う IBM ライセンス・プログラムです。DFSORT に言及している場所では、任意の同等のソートまたはマージ製品を使用することができます。

『例: 日時別のソート』

関連タスク
251 ページの『ウィンドウ表示日付フィールドに基づいたソート』
DFSORT アプリケーション・プログラミング・ガイド (OPTION 制御ステートメント: Y2PAST)

関連参照
355 ページの『DATEPROC』
404 ページの『YEARWINDOW』
日付フィールドの使用に関する制約事項 (*Enterprise COBOL 言語解説書*)

例: 日時別のソート

次の例は、口座番号内の日時に従ってソートされるトランザクション・レコードを持つトランザクション・ファイルを示しています。Trans-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。

```
SD  Transaction-File
    Record Contains 29 Characters
    Data Record is Transaction-Record
01  Transaction-Record.
    05  Trans-Account  PIC 9(8).
    05  Trans-Type     PIC X.
    05  Trans-Date     PIC 9(5) Date Format yyxxx.
    05  Trans-Time     PIC 9(6).
    05  Trans-Amount   PIC 9(7)V99.
. . .
    Sort Transaction-File
         On Ascending Key  Trans-Account
                           Trans-Date
                           Trans-Time
         Using Input-File
         Giving Sorted-File.
```

COBOL は DFSORT に関連情報を渡し、DFSORT がソート操作を正しく行えるようにします。COBOL が常に DFSORT に渡す情報のほか、COBOL は以下の情報 (DFSORT も使用します) も渡します。

- Y2PAST ソート・オプションとしての世紀ウィンドウ
- ウィンドウ表示西暦年フィールドおよび Trans-Date の日付形式

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
『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』
713 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。以下に例を示します。

```
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。

また、DATEPROC(TRIG) コンパイラー・オプションが有効なら、結果に 0 または 9 のトリガー値が含まれている場合、その結果の年部分 (それぞれ、00 または 99) が世紀ウィンドウの範囲内であっても SIZE ERROR 条件が発生します。

関連タスク
698 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。以下に例を示します。

```
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表示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。

- VSAM 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
『DATEVAL の使用』
715 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.
```

『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。

『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。以下に例を示します。

```
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 (*Enterprise COBOL 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。以下に例を示します。

```
01  Invoice-Record.
        03  Invoice-Date    Pic x(5) Date Format yyxxx.
. . .
    If FUNCTION UNDATE(Invoice-Date) Equal "00999" . . .
```

関連参照
UNDATE (*Enterprise COBOL 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
714 ページの『日付処理の明示的制御』
COBOL Millennium Language Extensions Guide (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 (*Enterprise COBOL 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
『パック 10 進数フィールドの問題の回避』
719 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 規則に従いません。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。以下に例を示します。

```
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 8 部 パフォーマンスおよび生産性の向上

第 34 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。制御フローが混乱したプログラムは、理解や維持が困難です。また、制御フローが混乱していると、コードの最適化も禁止されます。

このため、パフォーマンスの向上を直接試みる前に、プログラムのいくつかの局面を評価する必要があります。

1. プログラムの基礎アルゴリズムを調べる。最高のパフォーマンスを得るためには、適切なアルゴリズムが不可欠です。例えば、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。
2. データ構造を調べる。データ構造はアルゴリズムに適したものにする必要があります。プログラムが頻繁にデータにアクセスする場合は、可能であれば、データにアクセスするために必要なステップの数を減らします。
3. アルゴリズムとデータ構造を改善したら、パフォーマンスに影響を与える COBOL ソース・コードのその他の詳細を調べる。

より優れたコード・シーケンスを生成し、システム・サービスをより活用するようなプログラムを作成することができます。プログラムのパフォーマンスに影響を与えるのは、次の分野です。

- コーディング技法。これには、最適化プログラムを援助するプログラミング・スタイルの使用、効率的なデータ型の選択、およびテーブルの効率的な処理が含まれます。
- 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。特定のコンパイラー・オプションおよび言語は、プログラムの効率に影響を与えます。
- ランタイム環境。どのランタイム・オプションを選択するかについて、またコンパイルされたプログラムの実行方法を制御するその他のランタイム考慮事項について、注意深く考慮してください。
- CICS、IMS、または使用している VSAM のもとでの実行。これらのプログラムを効率的に実行するのに役立つさまざまなヒントがあります。

関連概念

732 ページの『最適化』

Enterprise COBOL Version 3 Performance Tuning

関連タスク

724 ページの『最適なプログラミング・スタイルの使用』
726 ページの『効率的なデータ型の選択』
728 ページの『テーブルの効率的処理』
731 ページの『コードの最適化』
735 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』

740 ページの『CICS、IMS、または VSAM での効率的な実行』
言語環境プログラム・プログラミング・ガイド (ランタイム・オプションの指定)

関連参照
735 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』
言語環境プログラム・プログラミング・ガイド (ストレージのパフォーマンスに関する考慮事項)

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、式の因数処理、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
『構造化プログラミングの使用』
『一括表示表現』
725 ページの『シンボリック定数の使用』
725 ページの『定数計算のグループ化』
725 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。その結果、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。

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
54 ページの『数値データの形式』

関連タスク
『効率的な計算データ項目の選択』
727 ページの『一貫性のあるデータ型の使用』
727 ページの『算術式の効率化』
728 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が大きな精度 (有効数字) の中間結果を持つ場合、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 データ項目で生成されるコードは、場合によっては (特にステートメントが複雑であるか、丸めを指定している場合は) BINARY データ項目で生成されるコードと同じ速さになることがあります。

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)。
- 乗算および除算にライブラリー・ルーチンを使用しないようにするには、PICTURE 指定を 15 桁以下にする。

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) として宣言します。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。特に、テーブル処理演算がアプリケーションの主要部分を占めているときなどは、努力の成果が十分報われる可能性があります。

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行われる不要な変換を避けるには、OCCURS . . . DEPENDING ON オブジェクトに BINARY を指定してください。
- 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
730 ページの『テーブル参照の最適化』

関連タスク
79 ページの『テーブル内の項目の参照』
726 ページの『効率的なデータ型の選択』

関連参照
390 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
732 ページの『最適化』

関連タスク
728 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。

再コンパイルされずにプログラムが頻繁に使用される場合、開発時にこのコンパイラー・オプションを使用することもできます。しかし、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、再コンパイルを頻繁に行うと、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上でどのように機能するかを見るためには、OPTIMIZE オプションを指定した場合としない場合でのコンパイルを行い、そのうえで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
『最適化』

関連参照
369 ページの『LIST』
378 ページの『OPTIMIZE』

最適化

生成されたコードの効率を上げるため、OPTIMIZE コンパイラー・オプションを使用することができます。

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。
- 含まれている (ネストされた) プログラムに対する PERFORM ステートメントおよび CALL ステートメントのコンパイル済みコードを単純化します。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。プロシージャー統合を行えない場合、最適化プログラムは、できるだけ単純なリンケージ (2 つ程度の命令) を使用して、呼び出し先プログラムとの間を往復します。
- 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。
- 実行できないコードをプログラムから削除し、警告メッセージでそのコードを示します (到達不能コードの除去)。

- 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。)

含まれているプログラム・プロシージャーの統合

含まれているプログラム・プロシージャーの統合では、含まれているプログラム・コードが、含まれているプログラムへの CALL と置き換わります。結果として生じるプログラムは、CALL リンケージのオーバーヘッドもなく、より線形の制御フローとなり、より速く実行されます。

プログラム・サイズ: 含まれているプログラムを複数の CALL ステートメントが呼び出している場合、それぞれのプログラムがこのような各ステートメントと置き換わるのであれば、収容プログラムが相当大きくなる可能性があります。最適化プログラムはこの増加を 50 % 以内に制限し、それ以降はプログラムを統合しません。これにより最適化プログラムはその CALL ステートメントに、次善の最適化を選択します。リンケージのオーバーヘッドをわずか 2 つの命令にすることができます。

到達不能コード: この統合の結果として、含まれている 1 つのプログラムが何回も繰り返されることがあります。その後の最適化はプログラムの各コピーで進められるため、コードのコピー先のコンテキストによっては、到達不能な部分が検出されることがあります。

関連概念
730 ページの『テーブル参照の最適化』
『PERFORM プロシージャー統合』

関連参照
378 ページの『OPTIMIZE』

PERFORM プロシージャー統合

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。この利点は、結果として生じるプログラムが、PERFORM リンケージのオーバーヘッドがなく、より線形になった制御フローで、より速く実行される点です。

プログラム・サイズ: 実行されたプロシージャーが複数のステートメントによって呼び出され、それぞれが PERFORM ステートメントを置き換えると、プログラムが大きくなる可能性があります。最適化プログラムはこの増加を 50 % 以内に制限し、それ以降はプロシージャーを統合しません。プログラム・サイズが問題である場合は、セクション名の優先順位番号を使用することによって、特定のインスタンスのプロシージャー統合を避けることができます。

ある PERFORM ステートメントを、それによって実行されるプロシージャーで置換したくない場合は、PERFORM ステートメントをあるセクションに置き、実行済みプロシージャーを、異なる優先順位番号を持つ別のセクションに置きます。これにより最適化プログラムはその PERFORM ステートメントに、次善の最適化を選択します。リンケージのオーバーヘッドを命令 2 つほどにすることができます。

到達不能コード: プロシージャー統合のせいで、1 つの PERFORM プロシージャーが何度か繰り返されることがあります。その後の最適化はプロシージャーの各コピーで進められるため、コードのコピー先のコンテキストによっては、到達不能な部分が検出されることがあります。

『例: PERFORM プロシージャー統合』

例: PERFORM プロシージャー統合

次の例は、プロシージャー統合によって変換されるコードを示しています。

次のプログラムにあるすべての PERFORM ステートメントが変換されます。

```
1   SECTION 5.
11. PERFORM 12
    STOP RUN.
12. PERFORM 21
    PERFORM 21.
2   SECTION 5.
21. IF A < 5 THEN
      ADD 1 TO A
      DISPLAY A
    END-IF.
```

このプログラムは、最初から次のように作成されていたかのようにコンパイルされます。

```
1   SECTION 5.
11.
12. IF A < 5 THEN
      ADD 1 TO A
      DISPLAY A
    END-IF.
    IF A < 5 THEN
      ADD 1 TO A
      DISPLAY A
    END-IF.
    STOP RUN.
```

それに対して、次のプログラムでは、最初の PERFORM ステートメントである PERFORM 12 だけが、プロシージャー統合によって最適化されます。

```
1   SECTION.
11. PERFORM 12
    STOP RUN.
12. PERFORM 21
    PERFORM 21.
2   SECTION 5.
21. IF A < 5 THEN
      ADD 1 TO A
      DISPLAY A
    END-IF.
```

関連概念

730 ページの『テーブル参照の最適化』

関連タスク

731 ページの『コードの最適化』
723 ページの『第 34 章 プログラムのチューニング』

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. システム・プログラマーと一緒に、どのオプションがインストール先のオーバーライド不能オプションとして固定されているかを調べる。
3. インストール先で固定されていないオプションについては、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。

COBOL ではセグメンテーション言語を使用できますが、それを使用してもストレージ割り振りは改善されません。COBOL はオーバーレイを実行しないからです。

関連概念
732 ページの『最適化』

関連タスク
731 ページの『コードの最適化』
422 ページの『リストの入手』

関連参照
『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 90. **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) (346 ページの『ARITH』 を参照)	10 進数で許可される最大桁数を増やします	一般には、ありません。	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。
347 ページの『AWO』	バッファーおよび装置スペースの最適使用を入手する	このオプションを使用すると、パフォーマンス上の節約になります。これは、入出力を処理するためにデータ管理サービスを呼び出す回数が減るからです。	一般には、ありません。	AWO を使用すると、APPLY WRITE-ONLY 節は、V モード・レコードを持つ物理順次のプログラム内において、すべてのファイルに適用されます。
DATA(31) (354 ページの『DATA』 を参照)	DFSMS に、16MB 境界より上の QSAM バッファーを割り振らせる (RENT および DATA(31) コンパイラー・オプションを使用して)	拡張形式 QSAM データ・セットは大量のバッファーを必要とする可能性があるので、制限のないストレージでバッファーを割り振ると、仮想記憶域の制約問題を回避することができます。	一般には、ありません。	DFSMS が備わった z/OS システムでは、アプリケーションがストライプ拡張形式 QSAM データ・セットを使用する場合には、RENT および DATA(31) コンパイラー・オプションを使用して、16MB 境界より上のストレージから、QSAM ファイル用の入出力バッファーを割り振らせるようにしてください。
361 ページの『DYNAM』	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。	呼び出しは言語環境プログラム・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。
363 ページの『FASTSRT』	IBM DFSORT プロダクト (または同等の製品) がすべての入出力を処理することを指定する	各レコードの処理後に Enterprise COBOL に戻るというオーバーヘッドを排除します。	なし	直接作業ファイルをソート作業ファイルとして使用する場合は、FASTSRT の使用をお勧めします。すべてのソートがこのオプションに適格とは限りません。

表 90. **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
NUMPROC(PFD) (375 ページの『NUMPROC』を参照)	数値演算の無効な符号処理をバイパスさせる	数値比較に関してはかなり効率のよいコードを生成します。	COMP-3 および DISPLAY 数値データ項目へのほとんどの参照では、NUMPROC(PFD) を使用すると、符号を「調整」するための余分のコードの生成が禁止されます。この余分のコードは、他のタイプの最適化も妨げる可能性があります。NUMPROC(MIG) および NUMPROC(NOPFD) を使用した場合は、余分のコードが生成されます。	NUMPROC(PFD) を使用すると、コンパイラーは、データには正しい符号があるので、データは符号「修正」プロセスを迂回すると想定します。すべての外部データ・ファイルに COMP-3 または DISPLAY 符号付き数値データの適切な符号が入っているとは限らないので、NUMPROC(PFD) が不適切なプログラムもあります。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、NUMPROC(PFD) の使用をお勧めします。
OPTIMIZE(STD) (378 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。	NOOPTIMIZE は通常、頻繁なコンパイルが必要となるプログラム開発の段階で使用されます。これにより、シンボリック・デバッグも可能になります。実稼働用には、OPTIMIZE の使用をお勧めします。
OPTIMIZE(FULL) (378 ページの『OPTIMIZE』を参照)	生成されたコードをパフォーマンスがよくなるように最適化し、さらに DATA DIVISION も最適化する	一般に、もっと効率的な実行時コードが得られ、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。
383 ページの『RENT』	再入可能プログラムを生成する	プログラムを共用ストレージ (LPA/ELPA) に入れ、実行をより高速化することができます。	プログラムを再入可能にするために追加のコードが生成されます。	
RMODE(ANY) (385 ページの『RMODE』を参照)	プログラムをどこにでもロードできるようにする	RMODE(ANY) と NORENT を使用すると、プログラムとその WORKING-STORAGE を 16MB 境界より上に置き、16MB 境界より下のストレージを解放することができます。	一般には、ありません。	

表 90. **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
NOSSRANGE (390 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。	なし	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。
TEST(NOHOOK) または NOTEST (392 ページの『TEST』 を参照)	デバッグ・ツールをフルに活用するために作成される追加のオブジェクト・コードを回避するには、TEST(NOHOOK) または NOTEST を使用する。TEST(NOHOOK) の場合は、SEP サブオプションを使用して、オブジェクト・コードのサイズをさらに縮小することができる。	TEST(HOOK) を使用した場合は追加のコードが生成されるので、実動環境で使用すると、パフォーマンスが大きく低下する可能性があります。	なし	サブオプション NOHOOK を含まないTEST は、コンパイラー・オプション NOOPT を強制的に有効にします。実稼働の場合は、NOTEST または TEST(NOHOOK) を使用することをお勧めします。その際、SEP サブオプションの指定は任意です。この結果、コンパイルされたフックではなく、オーバーレイ・フックとなります。 実稼働時、プログラムが異常終了したときにデータ項目のシンボリック・ダンプを定様式ダンプで取得したい場合は、TEST(NOHOOK) を使用してコンパイルします。その際、SEP サブオプションの指定は任意です。
396 ページの『THREAD』	複数の POSIX スレッドまたは PL/I タスクを含む言語環境プログラムのエンクレーブで、プログラムを実行できるようにする	なし	シリアライゼーション・ロジックのオーバーヘッドがあるため、ちょっとしたパフォーマンス・ペナルティーがあります。	これはスレッド化または非スレッド化環境に当てはまります。

表 90. **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
TRUNC(OPT) (397 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、そのパフォーマンスについては COBOL for OS/390 & VM で改善されたとはいえ、上記のオプション中では最も低速のオプションです。	TRUNC(STD) は標準 COBOL 85 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
732 ページの『最適化』
45 ページの『ストレージとそのアドレス可能度』

関連タスク
315 ページの『コンパイル・エラー・メッセージのリストの生成』
『パフォーマンスの評価』
13 ページの『バッファーおよび装置スペースの最適化』
735 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』
253 ページの『FASTSRT を使用してのソートのパフォーマンスの向上』
191 ページの『ストライプ拡張形式 QSAM データ・セットの使用』
728 ページの『テーブルの効率的処理』

関連参照
60 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』
192 ページの『QSAM ファイル用のバッファーの割り振り』
341 ページの『第 17 章 コンパイラー・オプション』
344 ページの『矛盾するコンパイラー・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 91. **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?
AWO	可能な場合は AWO オプションを使用していますか。	
DATA	QSAM ストライプ・データ・セットを使用するときは、RENT および DATA(31) オプションを使用していますか。ロード・モジュールは、 AMODE 31 ですか。 ALL31(ON) で実行していますか。	
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。	
FASTSRT	直接作業ファイルをソート作業ファイルとして使用するとき、FASTSRT オプションを使用しましたか。	
NUMPROC	可能な場合、NUMPROC(PFD) を使用していますか。	
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。	
RENT	RENT と NORENT のパフォーマンスのトレードオフを検討してください。	
RMODE(ANY)	NORENT プログラムで RMODE(ANY) を使用していますか。パフォーマンスのトレードオフとストレージ使用とを比較検討してください。	
SSRANGE	NOSSRANGE を実稼働に使用していますか。	
TEST	実稼働に NOTEST、TEST(NOHOOK)、または TEST(NOHOOK,SEP) を使用しますか。	
TRUNC	可能な場合、TRUNC(OPT) を使用していますか。	

関連概念
45 ページの『ストレージとそのアドレス可能度』

関連タスク
735 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
735 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』

CICS、IMS、または VSAM での効率的な実行

以下のヒントに従うなら、CICS または IMS のもとで実行されるオンライン・プログラム、あるいは VSAM を使用するプログラムのパフォーマンスを向上させることができます。

CICS: アプリケーションが CICS のもとで実行される場合、EXEC CICS LINK コマンドを CALL ステートメントに変換してトランザクション応答時間を改善します。

IMS: アプリケーションが IMS のもとで実行される場合は、アプリケーション・プログラムとライブラリー・ルーチンをプリロードすれば、ロードおよび検索のオーバーヘッドを削減するのに役立ちます。入出力活動も減少する可能性があります。

システムのパフォーマンスを向上させるためには、RENT コンパイラー・オプションを使用して、可能な限りアプリケーションとライブラリー・ルーチンをプリロードします。また、言語環境プログラムのライブラリー・ルーチン保存 (LRR) 機能を使用すると、IMS/TM 領域でのパフォーマンスを向上することができます。

VSAM: VSAM ファイルを使用する場合、順次アクセスの場合にはデータ・バッファーの数、ランダム・アクセスの場合には索引バッファーの数を増やしてください。また、アプリケーションに適した制御インターバル・サイズ (CISZ) を選択してください。 CISZ が小さいと、ランダム処理での検索は速くなりますが、挿入が犠牲になります。 CISZ が大きいと、順次処理の場合に効率がよくなります。

パフォーマンスを向上させるためには、レコードを順次アクセスし、複数の代替索引の使用を可能な限り避けます。代替索引を使用する場合、アクセス方式サービス・プログラムにより、 AIXBLD ランタイム・オプションよりもさらに効率的に索引が作成されます。

関連タスク

454 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』
481 ページの『第 22 章 COBOL プログラムの開発 (IMS の場合)』
227 ページの『VSAM パフォーマンスの向上』
言語環境プログラム・カスタマイズ

関連参照

言語環境プログラム・プログラミング・ガイド (ランタイム・オプションの指定)

第 35 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 言語環境プログラム呼び出し可能サービスを使用することによって、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 1 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。

言語環境プログラム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
64 ページの『数字組み込み関数の使用』
65 ページの『数学用の呼び出し可能サービスの使用』
67 ページの『データ呼び出し可能サービスの使用』
『反復コーディングの除去』
123 ページの『データ項目の変換 (組み込み関数)』
126 ページの『データ項目の評価 (組み込み関数)』
745 ページの『言語環境プログラム呼び出し可能サービスの使用』

反復コーディングの除去

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・レベルで使用することで、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ことができます。 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・ライブラリーを使用するには、複数のシステム定義を使用するか、複数の定義と IN/OF 句 (IN/OF *library-name*) の組み合わせを使用します。

z/OS バッチ
JCL を使用して、SYSLIB DD ステートメント内のデータ・セットを連結します。あるいは、複数の DD ステートメントを定義し、COPY ステートメントの IN/OF 句を使用します。

TSO ALLOCATE コマンドを使用して、SYSLIB のデータ・セットを連結します。あるいは、複数の ALLOCATE ステートメントを発行し、COPY ステートメントの IN/OF 句を使用します。

UNIX SYSLIB 環境変数を使用して、コピーブックへの複数のパスを定義します。あるいは、複数の環境変数を使用し、COPY ステートメントの IN/OF 句を使用します。

以下に例を示します。

```
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。

『例: COPY ステートメントの使用』

関連参照
407 ページの『第 18 章 コンパイラー指示ステートメント』

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
743 ページの『反復コーディングの除去』

関連参照
407 ページの『第 18 章 コンパイラー指示ステートメント』

言語環境プログラム呼び出し可能サービスの使用

言語環境プログラム呼び出し可能サービスにより、数多くのタイプのプログラミング作業が容易になります。 CALL ステートメントを使用して呼び出します。

言語環境プログラムは、以下のタスクに役立てることができます。

- 条件の処理

  言語環境プログラムの条件処理機能により、COBOL アプリケーションは予期しないエラーに反応できるようになります。言語構造体またはランタイム・オプションを使用して、それぞれの条件を処理するレベルを選択できます。例えば、COBOL プログラム内の特定のエラーを処理し、それを言語環境プログラムに処理させるか、またはオペレーティング・システムに処理させることができます。

  言語環境プログラムの条件処理のサポートでは、COBOL はプロシージャー・ポインター・データ項目を提供します。
- 動的ストレージの管理

  これらのサービスにより、ストレージの取得、解放、および再割り振りを行うことができます。また、自分自身のストレージ・プールも作成できます。
- 日時の計算

  日時サービスを使用すると、幾つかの形式で現在の地方時および日付を取得でき、日時の変換を実行できます。 2 つの呼び出し可能サービス CEEQCEN および CEESCEN により、2 桁年 (1991 を表す 91 または 2007 を表す 07 など) を処理する予測可能な方法が提供されます。
- 数学計算

数学呼び出し可能サービスで容易に実行される計算としては、対数関数、指数関数、三角関数、平方根関数、および整数関数があります。

COBOL では組み込み関数もサポートされます。この組み込み関数の中には、呼び出し可能サービスによって提供される数学関数および日付関数と同じ働きをする関数もいくつかあります。言語環境プログラム呼び出し可能サービスと組み込み関数は、多少の例外はありますが、同じ結果を戻します。これらの相違点を理解したうえで、どちらを使用するかを決定してください。

- メッセージの処理

  メッセージ処理サービスには、メッセージの取得、ディスパッチング、およびフォーマット設定のサービスがあります。非 CICS アプリケーション用のメッセージは、ファイルまたはプリンターに送信できます。 CICS メッセージは、CICS 一時データ・キューに送信されます。言語環境プログラムは、メッセージを分割して、宛先のレコード長を収容できるようにし、日本語や英語のような正しい各国語でメッセージを表示します。

- 各国語のサポート

  これらのサービスにより、アプリケーションは、アプリケーション・ユーザーが必要とする言語をサポートしやすくなります。言語と国を設定すると、デフォルトの日付、時刻、番号、および通貨形式を入手することができます。例えば、日付を「23 June 07」または「6,23,07」のように表示することができます。

- デバッグ・ツール の始動および言語環境プログラム定様式ダンプの取得

  デバッグ・ツール は COBOL-CICS プログラムのバッチ式デバッグおよび対話式デバッグの両方をはじめ、COBOL アプリケーションに高度なデバッグ機能を提供します。デバッグ・ツール により、ホストから、または Rational Developer for System z のデバッグ・パースペクティブ を併用して Windows ベースのワークステーションから、COBOL アプリケーションをデバッグすることができます。

  選択したオプションに応じて、言語環境プログラムの定様式ダンプには、データ項目の名前と値のほか、状態、プログラム・トレースバック、制御ブロック、ストレージ、およびファイルに関する情報が入れられます。すべての言語環境プログラム・ダンプには、共通の十分にラベル付けされた読みやすいフォーマットがあります。

748 ページの『例: 言語環境プログラムの呼び出し可能サービス』

関連概念

747 ページの『言語環境プログラムの呼び出し可能サービスのサンプル・リスト』

関連タスク

64 ページの『数字組み込み関数の使用』
65 ページの『数学用の呼び出し可能サービスの使用』
67 ページの『データ呼び出し可能サービスの使用』
747 ページの『言語環境プログラム・サービスの呼び出し』
516 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』

言語環境プログラムの呼び出し可能サービスのサンプル・リスト

次の表は、言語環境プログラムで使用可能な呼び出し可能サービスの例を示しています。リストされているサービスよりもっと多くのサービスを使用できます。

表 92. **言語環境プログラム呼び出し可能サービス (callable services)**

関数型	サービス	目的
条件処理	CEEHDLR	ユーザー条件処理ルーチンを登録する
	CEESGL	条件を発生させる、または通知する
	CEEMRCR	条件処理ルーチンが終了した後でプログラムが実行を再開する場所を指示する
動的ストレージ	CEEGTST	ストレージを獲得する
	CEECZST	これまでに割り振られたストレージ・ブロックのサイズを変更する
	CEEFRST	ストレージを解放する
日付および時刻	CEECBLDY	日付を表すストリングを COBOL 整数日付形式に変換する (日付を 1600 年 12 月 31 日以降の日数として表す)
	CEEQCEN、CEESCEN	言語環境プログラムの世紀ウィンドウ (年を表すのに 2 桁を使用するプログラムの場合に有用) を照会および設定する
	CEEGMTO	ローカル・システム時間とグリニッジ標準時の差を計算する
	CEELOCT	3 つの形式を選択して現在の地方時を入手する
数学	CEESIABS	整数の絶対値を計算する
	CEESSNWN	単精度浮動小数点数に最も近い整数を計算する
	CEESSCOS	角の余弦を計算する
メッセージ処理	CEEMOUT	メッセージをディスパッチする
	CEEMGET	メッセージを検索する
各国語サポート	CEE3LNG	現在の各国語を変更または照会する
	CEE3CTY	現在の国を変更または照会する
	CEE3MCS	特定の国のデフォルトの通貨記号を入手する
一般	CEE3DMP	言語環境プログラム定様式ダンプを取得する
	CEETEST	デバッグ・ツールなどのデバッグ・ツールを開始する

関連参照

言語環境プログラム・プログラミング・リファレンス

言語環境プログラム・サービスの呼び出し

言語環境プログラムサービスを呼び出すには、そ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。 CALL ステートメントの変数は、そのサービスに必要な定義を指定して、DATA DIVISION に定義します。

```
77  argument          comp-1.
77  feedback-code     pic  x(12)  display.
77  result            comp-1.. . .
CALL "CEESSSQT" using argument, feedback-code, result
```

上の例では、言語環境プログラム・サービス CEESSSQT は、変数 argument の平方根の値を計算し、この値を変数 result に入れて戻します。

フィードバック・コード・パラメーターを指定するかどうかを選択できます。指定した場合、feedback-code に戻される値は、サービスが正しく完了されたかどうかを示します。フィードバック・コードではなく OMITTED を指定したときに、サービスが失敗すると、言語環境プログラム条件が自動的に 言語環境プログラム 条件マネージャーに通知されます。このような条件は、ユーザー作成条件処理ルーチンに組み込まれたリカバリー・ロジックによって処理するか、または処理されない条件については、デフォルトの言語環境プログラム処理を許可することができます。いずれの場合も、ロジックを作成して呼び出しを終えるたびにフィードバック・コードをチェックする必要がなくなります。

言語環境プログラムの呼び出し可能サービスを呼び出し、OMITTED をフィードバック・コードに指定すると、サービスが成功した場合には RETURN-CODE 特殊レジスターが 0 に設定されます。サービスが失敗した場合には変更されません。フィードバック・コードに OMITTED を指定しない場合は、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。

『例: 言語環境プログラムの呼び出し可能サービス』

関連概念
言語環境プログラム・プログラミング・ガイド (一般呼び出し可能サービス)

関連参照
言語環境プログラム・プログラミング・リファレンス (一般呼び出し可能サービス)
CALL ステートメント (*Enterprise COBOL 言語解説書*)

例: 言語環境プログラムの呼び出し可能サービス

この例で示している COBOL プログラムは、言語環境プログラム・サービスの CEEDAYS および CEEDATE を使用して、COBOL ACCEPT ステートメントの結果として生じる日付をフォーマットし、表示します。

CEEDAYS および CEEDATE を使用すると、言語環境プログラムがないと必要になるようなコーディングが少なくて済みます。

```
 ID DIVISION.
 PROGRAM-ID. HOHOHO.
*************************************************************
* FUNCTION:  DISPLAY TODAY'S DATE IN THE FOLLOWING FORMAT: *
*            WWWWWWWWW, MMMMMMMM DD, YYYY                  *
*                                                          *
*            For example: TUESDAY, SEPTEMBER 18, 2007      *
*                                                          *
*************************************************************
 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 LANGUAGE ENVIRONMENT 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.
```

第 9 部 付録

付録 A. 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。

755 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、COBOL for OS/390 & VM バージョン 2 リリース 2 より前の IBM COBOL のリリース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
54 ページの『数値データの形式』
70 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
755 ページの『固定小数点データと中間結果』
761 ページの『浮動小数点データと中間結果』
762 ページの『非算術ステートメントの算術式』
346 ページの『ARITH』

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 (*Enterprise COBOL 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
63 ページの『算術式の使用』

関連参照
754 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
>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。

i + *d* の値	その他の条件	取られるアクション
=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。

758 ページの『例: 固定小数点の算術での指数』

関連参照

754 ページの『中間結果用の用語』
758 ページの『中間結果での切り捨て』
758 ページの『バイナリー・データと中間結果』
761 ページの『浮動小数点データと中間結果』
759 ページの『固定小数点算術で評価される組み込み関数』
346 ページの『ARITH』
SIZE ERROR 句 (*Enterprise COBOL 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
754 ページの『中間結果用の用語』
755 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
54 ページの『数値データの形式』

関連参照
755 ページの『固定小数点データと中間結果』
346 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
755 ページの『固定小数点データと中間結果』
346 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

754 ページの『中間結果用の用語』
755 ページの『固定小数点データと中間結果』
761 ページの『浮動小数点データと中間結果』
346 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。
- 指数に小数部の桁が含まれている。
- 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
754 ページの『中間結果用の用語』
346 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。
  - 指数に小数部の桁が含まれている。
  - 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。

  以下に例を示します。

  ```
  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、その例を示します。

```
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

70 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

754 ページの『中間結果用の用語』
755 ページの『固定小数点データと中間結果』
761 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 (*Enterprise COBOL 言語解説書*)
EVALUATE ステートメント (*Enterprise COBOL 言語解説書*)
条件式 (*Enterprise COBOL 言語解説書*)

付録 B. 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、標準 COBOL 85 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。

『例: 複合 ODO』

関連タスク 767 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』 768 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
766 ページの『ODO オブジェクト値の変更の影響』
OCCURS DEPENDING ON 節 (*Enterprise COBOL 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 句を指定した 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、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。

以下に例を示します。

- 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 『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』 768 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 (765 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 (*Enterprise COBOL 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、テーブルにエレメントを追加すると、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可能性があります。

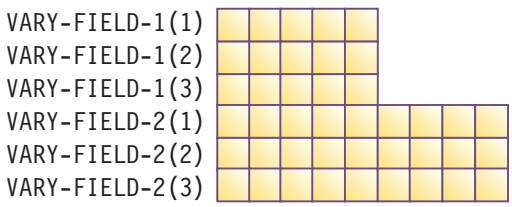
この種のエラーを回避するためには、次の手順を行ってください。

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).
```

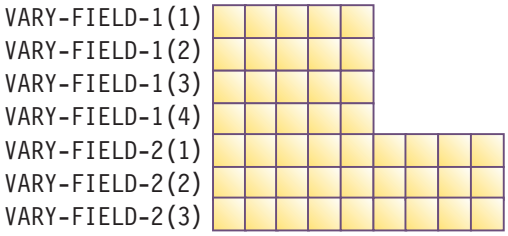
VARY-FIELD-1 の各エレメントは 5 バイトで、VARY-FIELD-2 の各エレメントは 9 バイトです。CONTROL-1 と CONTROL-2 の両方に値 3 が含まれている場合は、VARY-FIELD-1 および VARY-FIELD-2 のストレージは次のように示すことができます。



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。

付録 C. 2 バイト文字セット (DBCS) データの変換

言語環境プログラムのサービス・ルーチン IGZCA2D および IGZCD2A は、DBCS データを含んでいる英数字データ項目を、純 DBCS データ項目との間で変換するためのものでしたが、これには STRING、UNSTRING、および参照変更などの操作を確実に実行するという目的がありました。

互換性を保たせるため、これらのサービス・ルーチンは引き続き提供されます。しかし、互換性の目的の場合は、現在、代わりに国別データ項目および国別変換操作を使用することをお勧めしています。

これらのサービス・ルーチンは、コード・ページ引数をサポートしないので、CODEPAGE コンパイラー・オプションで指定されたコード・ページの影響を受けません。 DBCS コンパイラー・オプションは操作に影響を与えません。

関連タスク
147 ページの『国別 (Unicode) 表現との間の変換』
157 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
『DBCS 表記』
『英数字から DBCS データへの変換 (IGZCA2D)』
774 ページの『DBCS から英数字データへの変換 (IGZCD2A)』
350 ページの『CODEPAGE』

DBCS 表記

DBCS データ変換例では、DBCS 項目を記述するために以下の記号が使用されています。

記号	意味
< および >	それぞれ、シフトアウト (SO) およびシフトイン (SI)
D0, D1, D2, . . ., D*n*	1 バイト EBCDIC 文字に対応する、2 バイト EBCDIC 文字を除く任意の DBCS 文字
.A、.B、.C、. . .	1 バイト EBCDIC 文字に対応する、任意の 2 バイトの EBCDIC 文字。ピリオド (.) は値 X'42' を表します。
A、B、または s などの単一の文字	任意の 1 バイトの EBCDIC 文字

英数字から DBCS データへの変換 (IGZCA2D)

言語環境プログラムの IGZCA2D サービス・ルーチンは、2 バイト文字を含んでいる英数字データを純 DBCS データに変換します。

IGZCA2D の構文

IGZCA2D サービス・ルーチンを使用するには、CALL ステートメントを使用して次の 4 つのパラメーターをこのルーチンに渡します。

parameter-1
変換の送信フィールドで、英数字データ項目として処理されます。

parameter-2
変換の受信フィールドで、DBCS データ項目として処理されます。

parameter-2 で参照変更を使用することはできません。

parameter-3
変換される *parameter-1* の中のバイトの数。

これは、*parameter-1* の LENGTH OF 特殊レジスターにするか、または変換される *parameter-1* のバイト数を含む 4 バイトの USAGE IS BINARY データ項目にすることができます。シフト・コードはそれぞれ 1 バイトとしてカウントされます。

parameter-4
変換されたデータを受け取る *parameter-2* の中のバイトの数。

これは、*parameter-2* の LENGTH OF 特殊レジスターにするか、または変換されるデータを受け取る *parameter-2* のバイト数を含む 4 バイトの USAGE IS BINARY データ項目にすることができます。

使用上の注意

- *parameter-1*、*parameter-3*、および *parameter-4* はルーチン BY REFERENCE または BY CONTENT に渡すことができますが、*parameter-2* は BY REFERENCE に渡さなければなりません。
- コンパイラーは、これらのパラメーターの構文検査を行いません。パラメーターを正しく設定し、CALL ステートメントを使用して、変換ルーチンに渡すようにしてください。そうでない場合、結果は予測できません。
- *parameter-1* から *parameter-2* を作成する際、IGZCA2D は以下の変更を行います。
  - DBCS データはそのまま変更しないで、シフト・コードを除去します。
  - 単一バイト (非スペース) EBCDIC 文字 X'*nn*' を X'42*nn*' で表す文字に変換します。
  - 単一バイト・スペース (X'40') を、X'4240' ではなく DBCS スペース (X'4040') に変換します。
- IGZCA2D は、*parameter-1*、*parameter-3*、または *parameter-4* の内容を変更しません。
- *parameter-3* の内容および *parameter-4* の内容の有効範囲は、1 から 134,217,727 です。

773 ページの『例: IGZCA2D』

関連参照
773 ページの『IGZCA2D の戻りコード』

IGZCA2D の戻りコード

IGZCA2D は、変換の状況を反映するように RETURN-CODE 特殊レジスターを設定します。

表 93. **IGZCA2D の戻りコード**

戻りコード	解説
0	*parameter-1* が変換され、その結果が *parameter-2* に入れられました。
2	*parameter-1* が変換され、その結果が *parameter-2* に入れられました。*parameter-2* は、右側に DBCS スペースが埋め込まれました。
4	*parameter-1* が変換され、その結果が *parameter-2* に入れられました。*parameter-2* に入れられた DBCS データは、右側が切り捨てられました。
6	*parameter-1* が変換され、その結果が *parameter-2* に入れられました。X'00' から X'3F' の範囲または X'FF' の 1 バイト文字が検出されました。有効な 1 バイト文字が範囲外の DBCS 文字に変換されました。
8	*parameter-1* が変換され、その結果が *parameter-2* に入れられました。X'00' から X'3F' の範囲または X'FF' の 1 バイト文字が検出されました。有効な 1 バイト文字が範囲外の DBCS 文字に変換されました。 *parameter-2* は、右側に DBCS スペースが埋め込まれました。
10	*parameter-1* が変換され、その結果が *parameter-2* に入れられました。X'00' から X'3F' の範囲または X'FF' の 1 バイト文字が検出されました。有効な 1 バイト文字が範囲外の DBCS 文字に変換されました。 *parameter-2* の中の DBCS データは、右側が切り捨てられました。
12	*parameter-1* の中の対になったシフト・コードの間で奇数のバイトが検出されました。変換は行われませんでした。
13	*parameter-1* の中で、対になっていないかまたはネストされたシフト・コードが検出されました。変換は行われませんでした。
14	*parameter-1* と *parameter-2* がオーバーラップしました。変換は行われませんでした。
15	*parameter-3* または *parameter-4* に与えられた値が範囲外でした。変換は行われませんでした。
16	*parameter-4* の中で奇数のバイトがコーディングされました。変換は行われませんでした。

例: IGZCA2D

この CALL ステートメント例は、alpha-item 内の英数字データを DBCS データに変換します。変換の結果は dbcs-item に入れられます。

```
CALL "IGZCA2D" USING BY REFERENCE alpha-item dbcs-item
    BY CONTENT LENGTH OF alpha-item LENGTH OF dbcs-item
```

変換前の alpha-item および dbcs-item の内容と長さが次のものであるとします。

```
alpha-item = AB<D1D2D3>CD
dbcs-item  = D4D5D6D7D8D9D0

LENGTH OF alpha-item = 12
LENGTH OF dbcs-item  = 14
```

変換後、alpha-item および dbcs-item には次のものが入ります。

```
alpha-item = AB<D1D2D3>CD
dbcs-item  = .A.BD1D2D3.C.D
```

RETURN-CODE レジスターの内容は 0 です。

関連参照
771 ページの『DBCS 表記』

DBCS から英数字データへの変換 (IGZCD2A)

言語環境プログラムの IGZCD2A ルーチンは、純 DBCS データを、2 バイト文字を含むことのある英数字データに変換します。

IGZCD2A の構文

IGZCD2A サービス・ルーチンを使用するには、CALL ステートメントを使用して次の 4 つのパラメーターをこのルーチンに渡します。

parameter-1
変換の送信フィールドで、DBCS データ項目として処理されます。

parameter-2
変換の受信フィールドで、英数字データ項目として処理されます。

parameter-3
変換される *parameter-1* の中のバイトの数。

これは、*parameter-1* の LENGTH OF 特殊レジスターにするか、または変換される *parameter-1* のバイト数を含む 4 バイトの USAGE IS BINARY データ項目にすることができます。

parameter-4
変換されたデータを受け取る *parameter-2* の中のバイトの数。

これは、*parameter-2* の LENGTH OF 特殊レジスターにするか、または変換されるデータを受け取る *parameter-2* のバイト数を含む 4 バイトの USAGE IS BINARY データ項目にすることができます。シフト・コードはそれぞれ 1 バイトとしてカウントされます。

使用上の注意

- *parameter-1*、*parameter-3*、および *parameter-4* はルーチン BY REFERENCE または BY CONTENT に渡すことができますが、*parameter-2* は BY REFERENCE に渡さなければなりません。
- コンパイラーは、これらのパラメーターの構文検査を行いません。パラメーターを正しく設定し、変換ルーチンに渡すようにしてください。そうでない場合、結果は予測できません。
- *parameter-1* から *parameter-2* を作成する際、IGZCD2A は以下の変更を行います。
  - 1 バイト EBCDIC 文字に対応しない DBCS 文字の前後にシフト・コードを挿入します。

  - DBCS 文字が 1 バイト EBCDIC 文字に対応する場合は、DBCS 文字を 1 バイト文字に変換します。
  - DBCS スペース (X'4040') を 単一バイト・スペース (X'40') に変換します。
- IGZCD2A は、*parameter-1*、*parameter-3*、または *parameter-4* の内容を変更しません。
- 変換されたデータに 2 バイト文字が含まれている場合、シフト・コードは *parameter-2* の長さに含まれます。
- *parameter-3* の内容および *parameter-4* の内容の有効範囲は、1 から 134,217,727 です。

『例: IGZCD2A』

関連参照
『IGZCD2A の戻りコード』

IGZCD2A の戻りコード

IGZCD2A は、変換の状況を反映するように RETURN-CODE 特殊レジスターを設定します。

表 94. **IGZCD2A の戻りコード**

戻りコード	解説
0	*parameter-1* が変換され、その結果が *parameter-2* に入れられました。
2	*parameter-1* が変換され、その結果が *parameter-2* に入れられました。*parameter-2* は、右側に単一バイト・スペースが埋め込まれました。
4	*parameter-1* が変換され、その結果が *parameter-2* に入れられました。*parameter-2* は、右側で切り捨てられました。[1]
14	*parameter-1* と *parameter-2* がオーバーラップしました。変換は行われませんでした。
15	*parameter-3* または *parameter-4* の値が範囲外でした。変換は行われませんでした。
16	*parameter-3* の中で奇数のバイトがコーディングされました。変換は行われませんでした。

1. 切り捨てが DBCS 文字の中で行われる場合、その切り捨ては偶数バイト境界で行われ、シフトイン (SI) が挿入されます。必要な場合には、シフトインの後で、英数字データに単一バイト・スペースが埋め込まれます。

例: IGZCD2A

この CALL ステートメント例は、dbcs-item 内の DBCS データを 2 バイト文字を含む英数字データに変換します。変換の結果は alpha-item に入れられます。

```
CALL "IGZCD2A" USING BY REFERENCE dbcs-item alpha-item
        BY CONTENT LENGTH OF dbcs-item LENGTH OF alpha-item
```

変換前の dbcs-item および alpha-item の内容と長さが次のものであるとします。

```
dbcs-item  = .A.BD1D2D3.C.D
alpha-item = sssssssssss
```

```
LENGTH OF dbcs-item  = 14
LENGTH OF alpha-item = 12
```

変換後、dbcs-item および alpha-item には次のものが入ります。

```
dbcs-item  = .A.BD1D2D3.C.D
alpha-item = AB<D1D2D3>CD
```

RETURN-CODE レジスターの内容は 0 です。

関連参照

771 ページの『DBCS 表記』

付録 D. XML 参照資料

ここでは、XML パーサーおよび XML GENERATE ステートメントが特殊レジスター XML-CODE に戻す、XML 例外コードについて記載します。

関連参照
『継続を許可する XML PARSE 例外』
782 ページの『継続を許可しない XML PARSE 例外』
785 ページの『XML GENERATE 例外』
XML 仕様

継続を許可する XML PARSE 例外

例外イベントが発生すると、パーサーは特殊レジスター XML-CODE を例外を識別する値に設定します。 XMLPARSE コンパイラー・オプションの設定および XML-CODE の値によっては、パーサーが処理を続行できる場合があります。

XMLPARSE(XMLSS) の場合

XMLPARSE(XMLSS) コンパイラー・オプションが有効である場合、パーサーは例外イベントが発生すると処理を続行しません。パーサーは、処理プロシージャーで XML-CODE 特殊レジスターに行った変更を無視します。 XML PARSE ステートメントの最後にある XML-CODE の値は、パーサーが設定した値であり、元の例外コードを示します。処理プロシージャーが例外イベント後にパーサーに戻ると、制御は、ON EXCEPTION 句で指定したステートメント、または XML PARSE ステートメントの最後 (ON EXCEPTION 句をコーディングしていない場合) に渡ります。
XMLPARSE(XMLSS) オプションが有効である場合に可能な例外の指定については、z/OS XML システム・サービスに関する下記の関連参照を参照してください。

XMLPARSE(COMPAT) の場合

XMLPARSE(COMPAT) コンパイラー・オプションが有効である場合、パーサーは例外イベント後も処理を続行できます。たとえば、関連例外コードが次の範囲のいずれかであれば、パーサーは処理を続行することができます。

- 1-99
- 100,001 から 165,535

778 ページの表 95 は、パーサーが処理を続行できる例外コードを示しています。

この表には、それぞれの例外と、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記述しています。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*

用語の定義については、以下の関連タスクの『XML 文書のエンコード方式についての理解』を参照してください。

表 95. **続行可能な XML PARSE 例外 (XMLPARSE(COMPAT) の場合)**

コード	説明	継続され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。	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。
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。

表 95. **続行可能な XML PARSE 例外 (XMLPARSE(COMPAT) の場合)** *(続き)*

コード	説明	継続されるパーサーのアクション
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。
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	処理命令ターゲット名が、小文字、大文字、または大/小文字混合の 'xml' 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。

表 95. **続行可能な XML PARSE 例外 (XMLPARSE(COMPAT) の場合)** *(続き)*

コード	説明	継続されるパーサーのアクション
13	パーサーで、16 進文字参照 (形式 *&#xdddd;* 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。

表 95. **続行可能な XML PARSE 例外 (XMLPARSE(COMPAT) の場合)** *(続き)*

コード	説明	継続されるパーサーのアクション
70	実際の文書エンコードは EBCDIC であり、 CODEPAGE コンパイラー・オプションではサポートされる EBCDIC コード・ページが指定されていましたが、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、CODEPAGE コンパイラー・オプションで指定されたエンコードを使用します。
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、CODEPAGE コンパイラー・オプションはサポートされる EBCDIC コード・ページを指定していませんでした。	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、 CODEPAGE コンパイラー・オプションではサポートされる EBCDIC コード・ページが指定されておらず、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、 CODEPAGE コンパイラー・オプション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。
82	実際の文書エンコードは ASCII ですが、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、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 ですが、文書エンコード宣言ではコード・ページ 813、819、および 920 のいずれも指定さ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。
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	CODEPAGE コンパイラー・オプション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、CODEPAGE コンパイラー・オプション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。

関連概念

568 ページの『XML-CODE』

関連タスク
582 ページの『XML 文書のエンコード方式についての理解』
588 ページの『XML PARSE の例外処理』

関連参照 401 ページの『XMLPARSE』
z/OS XML System Services User's Guide and Reference (SA23-1350)

継続を許可しない XML PARSE 例外

XMLPARSE(XMLSS) の場合

XMLPARSE(XMLSS) コンパイラー・オプションが有効である場合、パーサーは例外イベントが発生すると処理を強制終了します。処理プロシージャーがイベント後にパーサーに戻ると、パーサーは、ON EXCEPTION 句で指定したステートメント、または XML PARSE ステートメントの最後 (ON EXCEPTION 句をコーディングしていない場合) に制御を渡します。

XMLPARSE(COMPAT) の場合

XMLPARSE(COMPAT) コンパイラー・オプションが有効である場合、XML-CODE をゼロに設定していて例外を処理してから制御をパーサーに返しても、下表に示された例外についてこれ以上イベントはパーサーから返されません。パーサーは、ON EXCEPTION 句で指定したステートメント、または XML PARSE ステートメントの最後 (ON EXCEPTION 句をコーディングしていない場合) に制御を渡します。

表 96. **継続を許可しない XML PARSE 例外**

コード	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。

表 96. **継続を許可しない XML PARSE 例外** *(続き)*

コード	説明
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。
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。

表 96. **継続を許可しない XML PARSE 例外** *(続き)*

コード	説明
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。
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。
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。
500-599	内部エラーこのエラーをサービス担当者に報告してください。

関連概念

568 ページの『XML-CODE』

関連タスク
588 ページの『XML PARSE の例外処理』

関連参照 401 ページの『XMLPARSE』

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 97. **XML GENERATE 例外**

コード	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	DBCS 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、DBCS 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	CODEPAGE コンパイラー・オプションで指定された CCSID ページは、Unicode への変換ではサポートされません。
411	CODEPAGE コンパイラー・オプションで指定された CCSID ページは、サポート対象の 1 バイト EBCDIC CCSIDではありません。
414	XML 文書に指定された CCSID は無効であるか、サポートされていませんでした。
416	XML 名前空間 ID に無効な XML 文字が含まれていました。
417	エレメント文字コンテンツまたは属性値に 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。
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。

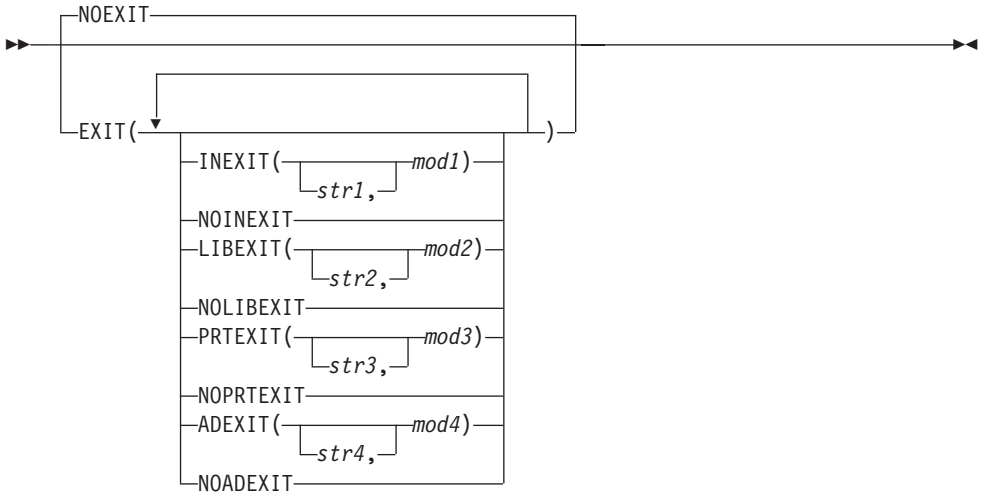
関連タスク
598 ページの『XML 出力生成時のエラーの処理』

付録 E. EXIT コンパイラー・オプション

EXIT オプションは、SYSIN、SYSLIB (またはコピー・ライブラリー)、および SYSPRINT の代わりにユーザー提供モジュールをコンパイラーが受け入れることができるようにする場合に使用します。

SYSADATA の場合、ADEXIT サブオプションは、SYSADATA レコードごとに、そのレコードがファイルに書き込まれた直後に呼び出されることになるモジュールを提供します。

EXIT オプションの構文



デフォルト: NOEXIT

省略形: EX(INX|NOINX、LIBX|NOLIBX、PRTX|NOPRTX、ADX|NOADX)

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と、NOEXIT が実施されます。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することができます。サブオプションの肯定形式と否定形式 (INEXIT|NOINEXT、LIBEXIT|NOLIBEXIT、PRTEXIT|NOPRTEXIT、または ADEXIT|NOADEXIT) 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。

EXIT オプションは、呼び出し時に JCL PARM フィールドで (TSO/E のもとではコマンド引数で)、またはインストール時にのみ指定できます。 EXIT オプションを PROCESS (CBL) ステートメントに指定してはなりません。

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。

ADEXIT(['*str4*',]*mod4*)
コンパイラーは、SYSADATA 出力を、ユーザー提供のロード・モジュール (*mod4* はモジュール名) に渡します。

モジュール名 *mod1*、*mod2*、*mod3*、および *mod4* 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*、および *str4* 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、単一引用符で囲む必要があります。任意の文字を使用できますが、組み込む単一引用符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。

str1、*str2*、*str3*、または *str4* のいずれかが指定された場合、そのストリングは次の形式で、適切なユーザー出口モジュールに渡されます。

LL	ストリング

LL は、そのストリングの長さを含む (ハーフワード境界上の) ハーフワードです。

800 ページの『例: INEXIT ユーザー出口』

関連タスク 789 ページの『ユーザー出口作業域の使用』 789 ページの『出口モジュールからの呼び出し』 798 ページの『CICS および SQL ステートメントに関連して EXIT コンパイラー・オプションを使用する』

関連参照
790 ページの『INEXIT の処理』
791 ページの『LIBEXIT の処理』
795 ページの『PRTEXIT の処理』
796 ページの『ADEXIT の処理』
797 ページの『出口モジュールでのエラー処理』

ユーザー出口作業域の使用

ユーザーが出口を使用するとき、コンパイラーはユーザー出口作業域を提供します。その作業域には、出口モジュールが取得した GETMAIN ストレージのアドレスを保管できます。この作業域を使用して、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は、フルワード境界に常駐する 4 フルワードです。このフルワードは、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ます。作業域のアドレスは、パラメーター・リストの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。

コンパイル時に複数の出口がアクティブになる場合は、作業域を使用するための独自の規則を確立する必要があります。例えば、INEXIT モジュールは作業域の最初のワードを使用し、LIBEXIT モジュールは 2 番目のワードを使用し、PRTEXIT モジュールは 3 番目のワードを使用し、ADEXIT モジュールは 4 番目のワードを使用します。

関連参照
790 ページの『INEXIT の処理』
791 ページの『LIBEXIT の処理』
795 ページの『PRTEXIT の処理』
796 ページの『ADEXIT の処理』

出口モジュールからの呼び出し

出口モジュールで COBOL 標準リンケージを使用して、COBOL プログラムまたはライブラリー・ルーチンを呼び出します。呼び出しチェーンを正しくトレースするためには、これらの規則を知る必要があります。

プログラムまたはルーチンに呼び出しを行うと、レジスターは次のようにセットアップされます。

R1 呼び出されるプログラムまたはライブラリー・ルーチンに渡されるパラメーター・リストを指す

R13 呼び出し側プログラムまたはルーチンが提供するレジスター保管域を指す

R14 呼び出し側プログラムまたはルーチンの戻りアドレスを保持する

R15 呼び出されるプログラムまたはルーチンのアドレスを保持する

出口モジュールは、24 の RMODE 属性と ANY の AMODE 属性を持っていなければなりません。

関連概念
45 ページの『ストレージとそのアドレス可能度』

INEXIT の処理

SYSIN の代わりにユーザー提供ロード・モジュールからソース・コードを読み取る場合に、出口モジュールを使用します。

表 98. **INEXIT 処理**

コンパイラーのアクション	出口モジュールによる結果としてのアクション
初期化時に出口モジュール (*mod1*) 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	処理するためのソースを準備します。OPEN 要求の状況をコンパイラーに戻します。
ソース・ステートメントが必要になったときは、GET 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	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) データの終わり標識のいずれかを戻します
データの終わりが存在するときに、CLOSE 命令コードを使用して出口モジュールを呼び出します	その出力に関連のあるすべてのリソースを解放します

INEXIT パラメーター

コンパイラーはパラメーター・リストを使用して、出口モジュールと連絡します。パラメーター・リストはアドレスが入っている 10 個のフルワードで構成され、レジスター 1 にはパラメーター・リストのアドレスが入ります。戻りコード、データ長、およびデータ・パラメーターは、出口モジュールによってコンパイラーに戻され、他の項目はコンパイラーから出口モジュールに渡されます。次の表に、パラメーター・リストの内容を説明します。

表 99. **INEXIT パラメーター**

Offset	次のもののアドレスが入る	項目の説明
00	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
04	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
08	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。 0=操作は成功しました; 4=データの終わり; 12=操作は失敗しました
12	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
16	データ長	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の長さを指定します (80 でなければなりません)

表 99. **INEXIT パラメーター** *(続き)*

Offset	次のもののアドレスが入る	項目の説明
20	データまたは *str1*	出口モジュールが設定するフルワードであり、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が入ります。 *str1* 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。
24	未使用	(LIBEXIT の専用)
28	未使用	(LIBEXIT の専用)
32	未使用	(LIBEXIT の専用)
36	未使用	(LIBEXIT の専用)

800 ページの『例: INEXIT ユーザー出口』

関連タスク 798 ページの『CICS および SQL ステートメントに関連して EXIT コンパイラー・オプションを使用する』

関連参照
『LIBEXIT の処理』

LIBEXIT の処理

SYSLIB (または *library-name*) データ・セットの代わりに出口モジュールが使用されます。 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。

LIBEXIT を指定する場合は、LIB コンパイラー・オプションが有効でなければなりません。

表 100. **LIBEXIT 処理**

コンパイラーのアクション	出口モジュールによる結果としてのアクション
初期化時に出口モジュール (*mod2*) 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	指定された *library-name* を処理のために準備します。OPEN 要求の状況をコンパイラーに渡します。
library-name が正常にオープンされた場合に、FIND 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	指定された *library-name* 内の要求された *text-name* (または *basis-name*) のところに、位置決めを行います。この場所がアクティブ・コピーブックになります。位置決めが完了したときに、適切な戻りコードをコンパイラーに渡します。

表 100. **LIBEXIT 処理** *(続き)*

コンパイラーのアクション	出口モジュールによる結果としてのアクション
GET 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	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、またはデータの終わり標識のいずれかをコンパイラーに渡します
データの終わりが存在するときに、CLOSE 命令コードを使用して出口モジュールを呼び出します	その入力に関連のあるすべてのリソースを解放します

ネストされた COPY ステートメントによる LIBEXIT の処理

アクティブ・コピーブックのレコードは COPY ステートメントを含むことができます。 (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。)

コンパイラーは、*text-name* への再帰呼び出しを許可しません。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。

次の表は、ネストされていない 1 つ以上の有効な COPY ステートメントがあるときに、LIBEXIT の処理がどのように変わるかを示しています。

表 101. **ネストなしの COPY ステートメントによる LIBEXIT の処理**

コンパイラーのアクション	出口モジュールによる結果としてのアクション
初期化時に出口モジュール (*mod2*) 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	指定された *library-name* を処理のために準備します。OPEN 要求の状況をコンパイラーに渡します。
library-name が正常にオープンされた場合に、FIND 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	指定された *library-name* 内の要求された *text-name* (または *basis-name*) のところに、位置決めを行います。この場所がアクティブ・コピーブックになります。位置決めが完了したときに、適切な戻りコードをコンパイラーに渡します。
library-name が正常にオープンされた場合に、FIND 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	前のアクティブ・コピーブックのところに、位置決めを設定し直します。位置決めが完了したときに、適切な戻りコードをコンパイラーに渡します。
GET 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。 同じレコードが渡されたかどうかを検査します。	このコピーブックから前に渡されたものと同じレコードをコンパイラーに渡します。検査後に、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、またはデータの終わり標識のいずれかをコンパイラーに渡します。

表 101. **ネストなしの COPY ステートメントによる LIBEXIT の処理** *(続き)*

コンパイラーのアクション	出口モジュールによる結果としてのアクション
データの終わりが存在するときに、CLOSE 命令コードを使用して出口モジュールを呼び出します	その入力に関連のあるすべてのリソースを解放します

次の表は、ネストされた 1 つの有効な COPY ステートメントをコンパイラーが検出すると、LIBEXIT の処理がどのように変わるかを示しています。

表 102. **ネストされた COPY ステートメントによる LIBEXIT の処理**

コンパイラーのアクション	出口モジュールによる結果としてのアクション
ネストされた *COPY* ステートメントからの要求された library-name が前にオープンされていなければ、OPEN 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	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押し入れます。要求されたアクション (OPEN) を完了させます。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。
要求された新しい *text-name*	のために FIND 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押し入れます。要求されたアクション (FIND) を完了させます。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。
GET 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	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、またはデータの終わり標識のいずれかをコンパイラーに渡します。データの終わりに、スタックからその制御情報をポップします。

LIBEXIT パラメーター

コンパイラーはパラメーター・リストを使用して、出口モジュールと連絡します。パラメーター・リストはアドレスが入っている 10 個のフルワードで構成され、レジスター 1 にはパラメーター・リストのアドレスが入ります。戻りコード、データ長、およびデータ・パラメーターは、出口モジュールによってコンパイラーに戻され、他の項目はコンパイラーから出口モジュールに渡されます。

表 103. **LIBEXIT パラメーター**

Offset	次のもののアドレスが入る	項目の説明
00	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。 2=LIBEXIT

表 103. **LIBEXIT パラメーター** *(続き)*

Offset	次のもののアドレスが入る	項目の説明
04	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; 4=FIND
08	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。 0=操作は成功しました; 4=データの終わり; 12=操作は失敗しました
12	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
16	データ長	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の長さを指定します (80 でなければなりません)
20	データまたは *str2*	出口モジュールが設定するフルワードであり、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が入ります。 *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。
24	システム *library-name*	COPY ステートメントの *library-name* が入れられる 8 文字の区域。プログラム名の処理規則および変換規則が適用されます。必要に応じて、ブランクが埋め込まれます。OPEN、CLOSE、および FIND に適用されます。
28	システム *text-name*	COPY ステートメントの *text-name* (BASIS ステートメントの *basis-name*) が入れられる 8 文字の区域。*program name* の処理規則と変換規則が適用されます。必要に応じて、ブランクが埋め込まれます。FIND にのみ適用されます。
32	ライブラリー名	COPY ステートメントからの完全な *library-name* が入れられる 30 文字の区域。必要な場合にはブランクが埋め込まれ、そのまま (英大文字に変換されず) で使用されます。OPEN、CLOSE、および FIND に適用されます。
36	テキスト名	COPY ステートメントからの完全な *text-name* が入れられる 30 文字の区域。必要な場合にはブランクが埋め込まれ、そのまま (英大文字に変換されず) で使用されます。FIND にのみ適用されます。

関連タスク 798 ページの『CICS および SQL ステートメントに関連して EXIT コンパイラー・オプションを使用する』

PRTEXIT の処理

SYSPRINT データ・セットの代わりに出口モジュールを使用します。

表 104. **PRTEXIT 処理**

コンパイラーのアクション	出口モジュールによる結果としてのアクション
初期化時に出口モジュール (*mod3*) 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	処理のためにその出力宛先を準備します。OPEN 要求の状況をコンパイラーに渡します。
行が印刷されるときに、印刷されるレコードのアドレスと長さを与え、PUT 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	PUT 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。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。
データの終わりが存在するときに、CLOSE 命令コードを使用して出口モジュールを呼び出します	その出力宛先に関連のあるすべてのリソースを解放します

PRTEXIT パラメーター

コンパイラーはパラメーター・リストを使用して、出口モジュールと連絡します。パラメーター・リストはアドレスが入っている 10 個のフルワードで構成され、レジスター 1 にはパラメーター・リストのアドレスが入ります。戻りコード、データ長、およびデータ・バッファー・パラメーターは、出口モジュールによってコンパイラーに戻され、他の項目はコンパイラーから出口モジュールに渡されます。

表 105. **PRTEXIT パラメーター**

Offset	次のもののアドレスが入る	項目の説明
00	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。 3=PRTEXIT
04	命令コード	操作のタイプを示すハーフワード。 0=OPEN; 1=CLOSE; 3=PUT
08	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。 0=操作は成功しました; 12=操作は失敗しました
12	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
16	データ長	PUT 操作によって提供されるレコードの長さを指定するフルワード (コンパイラーはこの値を 133 に設定します)

表 105. **PRTEXIT パラメーター** *(続き)*

Offset	次のもののアドレスが入る	項目の説明
20	データ・バッファーまたは *str3*	コンパイラーが PUT 操作によって印刷されるレコードを入れたデータ・バッファーのアドレスが含まれているフルワード。 *str3* 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。
24	未使用	(LIBEXIT の専用)
28	未使用	(LIBEXIT の専用)
32	未使用	(LIBEXIT の専用)
36	未使用	(LIBEXIT の専用)

関連タスク 798 ページの『CICS および SQL ステートメントに関連して EXIT コンパイラー・オプションを使用する』

関連参照
791 ページの『LIBEXIT の処理』

ADEXIT の処理

ADEXIT モジュールを使用するには、SYSADATA 出力を作成するコンパイラー・オプション ADATA と、DD ステートメント SYSADATA が必要です。

表 106. **ADEXIT 処理**

コンパイラーのアクション	出口モジュールによる結果としてのアクション
初期化時に出口モジュール (*mod4*) 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	処理のためにその出力宛先を準備します。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 SYSADATA レコードを書き込んだときに、その SYSADATA のアドレスと長さを与え、PUT 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	PUT 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
データの終わりが存在するときに、CLOSE 命令コードを使用して出口モジュールを呼び出します	すべてのリソースを解放します

ADEXIT パラメーター

コンパイラーはパラメーター・リストを使用して、出口モジュールと連絡します。パラメーター・リストはアドレスが入っている 10 個のフルワードで構成され、レジスター 1 にはパラメーター・リストのアドレスが入ります。戻りコード、データ

長、およびデータ・バッファー・パラメーターは、出口モジュールによってコンパイラーに戻され、他の項目はコンパイラーから出口モジュールに渡されます。

表 107. **ADEXIT パラメーター**

Offset	次のもののアドレスが入る	項目の説明
00	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。 4=ADEXIT
04	命令コード	操作のタイプを示すハーフワード。 0=OPEN; 1=CLOSE; 3=PUT
08	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。 0=操作は成功しました; 12=操作は失敗しました
12	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
16	データ長	PUT 操作によって提供されるレコードの長さを指定するフルワード
20	データ・バッファーまたは *str4*	コンパイラーが PUT 操作によって印刷されるレコードを入れたデータ・バッファーのアドレスが含まれているフルワード。 *str4* 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。
24	未使用	(LIBEXIT の専用)
28	未使用	(LIBEXIT の専用)
32	未使用	(LIBEXIT の専用)
36	未使用	(LIBEXIT の専用)

関連タスク 798 ページの『CICS および SQL ステートメントに関連して EXIT コンパイラー・オプションを使用する』

関連参照
791 ページの『LIBEXIT の処理』

出口モジュールでのエラー処理

出口モジュールをロードできないときや、出口モジュールが「operation failed (操作失敗)」メッセージまたはゼロ以外の戻りコードを戻すたびに、コンパイラーはエラー・メッセージを報告します。

次のいずれかのイベントが発生すると、メッセージ IGYSI5008 がオペレーターに出され、コンパイラーは戻りコード 16 で終了します。

- 出口モジュールをロードできない。
- OPEN 要求のときに、INEXIT からゼロ以外の戻りコードを受け取った。

• OPEN 要求のときに、PRTEXIT からゼロ以外の戻りコードを受け取った。

出口タイプと操作 (OPEN または LOAD) がメッセージで識別されます。INEXIT または PRTEXIT からその他のエラーが戻されると、コンパイラーは終了します。

コンパイラーは、以下の条件を検出し、報告します。

5203 SYSPRINT 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。

5204 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。

5205 SYSIN 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。

5206 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。

CICS および SQL ステートメントに関連して EXIT コンパイラー・オプションを使用する

EXIT コンパイラー・オプションのサブオプションを使用してコンパイルし、CICS または SQL ステートメントを変換する必要がある場合、出口モジュールで実行できるアクションは、別個の CICS 変換プログラムおよび DB2 プリコンパイラーを使用するか、組み込まれた CICS 変換プログラムおよび DB2 coprocessor を使用するかによって異なります。

組み込みの変換プログラムを使用するときには、出口モジュールにおいて、EXEC CICS および EXEC SQL ステートメントを処理することができます。以下のテーブルは、4 つの出口モジュールの選択を示します。

表 108. **出口モジュールで CICS および SQL ステートメントに許可された操作**

サブオプションを使用したコンパイル	組み込みの CICS 変換プログラムおよび DB2 coprocessor を使用した変換	別個の CICS 変換プログラムおよび DB2 coprocessor を使用した変換	モジュールで許可された操作	コメント
INEXIT	はい	いいえ	INEXIT モジュールでは、EXEC CICS および EXEC SQL ステートメントを処理できます	INEXIT モジュールでは、EXEC ステートメントに生成された COBOL ステートメントの制御が得られません。
	いいえ	はい	INEXIT モジュールでは、EXEC ステートメントに生成された COBOL ステートメントを処理できます	生成されたステートメントは INEXIT モジュールで変更することができます。ただし、この変更について、IBM ではサポートしません。

表 108. **出口モジュールで CICS および SQL ステートメントに許可された操作** *(続き)*

サブオプションを使用したコンパイル	組み込みの CICS 変換プログラムおよび DB2 coprocessor を使用した変換	別個の CICS 変換プログラムおよび DB2 coprocessor を使用した変換	モジュールで許可された操作	コメント
LIBEXIT	はい	いいえ	EXEC SQL INCLUDE ステートメントによってもたらされたステートメントを LIBEXIT モジュールで処理できます。 LIBEXIT モジュールで EXEC CICS ソース・ステートメントを処理できます。	EXEC SQL INCLUDE ステートメントは、COBOL COPY ステートメントと同じように処理されます。
	いいえ	はい	LIBEXIT モジュールでは、EXEC CICS ステートメントに生成された COBOL ステートメントを処理できます	EXEC SQL INCLUDE ステートメントによってもたらされた入力ステートメントを処理できるのは、INEXIT サブオプションを使用した場合のみです。
PRTEXIT	はい	いいえ	PRTEXIT モジュールの SOURCE リストから、EXEC CICS および EXEC SQL ソース・ステートメントを処理できます	PRTEXIT モジュールは、生成された COBOL ソース・ステートメントにアクセスできません。
	いいえ	はい	PRTEXIT モジュールでは、EXEC ステートメントに生成された COBOL SOURCE リスト・ステートメントを処理できます	
ADEXIT	はい	いいえ	ADEXIT モジュールでは、EXEC CICS および EXEC SQL ソース・ステートメントを処理できます	ADEXIT モジュールは、生成された COBOL ソース・ステートメントにアクセスできません。
	いいえ	はい	ADEXIT モジュールでは、EXEC ステートメントに生成された COBOL SYSADATA ソース・ステートメントを処理できます	

関連概念
467 ページの『DB2 coprocessor』
460 ページの『組み込みの CICS 変換プログラム』

関連タスク
472 ページの『SQL オプションを使用したコンパイル』
458 ページの『CICS オプションを使用したコンパイル』

関連参照
790 ページの『INEXIT の処理』
791 ページの『LIBEXIT の処理』
795 ページの『PRTEXIT の処理』
796 ページの『ADEXIT の処理』

例: INEXIT ユーザー出口

次の例は、COBOL で書かれた INEXIT ユーザー出口モジュールを示しています。

```
*************************************************************
*                                                           *
* Name:  SKELINX                                            *
*                                                           *
* Function:  Example of an INEXIT user exit written         *
*            in the COBOL language.                         *
*                                                           *
*************************************************************

 Identification Division.
    Program-ID.  Skelinx.

 Environment Division.

 Data Division.

   WORKING-STORAGE Section.

*   *******************************************************
*   *                                                     *
*   *  Local variables.                                   *
*   *                                                     *
*   *******************************************************

    01  Record-Variable    Pic X(80).

*   *******************************************************
*   *                                                     *
*   *  Definition of the User-Exit Parameter List, which  *
*   *  is passed from the COBOL compiler to the user exit *
*   *  module.                                            *
*   *                                                     *
*   *******************************************************

   Linkage Section.
    01  Exit-Type         Pic 9(4)   Binary.
    01  Exit-Operation    Pic 9(4)   Binary.
    01  Exit-ReturnCode   Pic 9(9)   Binary.
    01  Exit-WorkArea.
        05  INEXIT-Slot   Pic 9(9)   Binary.
        05  LIBEXIT-Slot  Pic 9(9)   Binary.
        05  PRTEXIT-Slot  Pic 9(9)   Binary.
        05  Reserved-Slot Pic 9(9)   Binary.
    01  Exit-DataLength   Pic 9(9)   Binary.
    01  Exit-DataArea     Pointer.
    01  Exit-Open-Parm    Redefines  Exit-DataArea.
        05  String-Len    Pic 9(4)   Binary.
        05  Open-String   Pic X(64).
    01  Exit-Print-Line   Redefines  Exit-DataArea  Pic X(133).
    01  Exit-LIBEXIT      Pic X(8).
    01  Exit-Systext      Pic X(8).
    01  Exit-CBLLibrary   Pic X(30).
    01  Exit-CBLText      Pic X(30).

*************************************************************
*                                                           *
*  Begin PROCEDURE DIVISION                                 *
*                                                           *
*  Invoke the section to handle the exit.                   *
*                                                           *
*************************************************************
```

```
 Procedure Division Using Exit-Type       Exit-Operation
                          Exit-ReturnCode Exit-WorkArea
                          Exit-DataLength Exit-DataArea
                          Exit-LIBEXIT    Exit-Systext
                          Exit-CBLLibrary Exit-CBLText.

     Evaluate Exit-type
       When (1) Perform Handle-INEXIT
       When (2) Perform Handle-LIBEXIT
       When (3) Perform Handle-PRTEXIT
     End-Evaluate
     Move 16 To Exit-ReturnCode
     Goback.

************************************************
*    I N E X I T   E X I T   P R O C E S S O R *
************************************************
 Handle-INEXIT.

     Evaluate Exit-Operation
       When (0) Perform INEXIT-Open
       When (1) Perform INEXIT-Close
       When (2) Perform INEXIT-Get
     End-Evaluate

     Move 16 To Exit-ReturnCode
     Goback.

 INEXIT-Open.
*    -------------------------------------------------------
*    Prepare for reading source
*    -------------------------------------------------------
     Goback.

 INEXIT-Close.
*    -------------------------------------------------------
*    Release resources
*    -------------------------------------------------------
     Goback.

 INEXIT-Get.
*    -------------------------------------------------------
*    Retrieve next source record
*    -------------------------------------------------------

*    -------------------------------------------------------
*    Return the address of the record to the compiler.
*    -------------------------------------------------------
     Set Exit-DataArea to Address of Record-Variable

*    -------------------------------------------------------
*    Set length of record in User-Exit Parameter List
*    -------------------------------------------------------
     Move 80 To Exit-DataLength

     Goback.

************************************************
*    L I B E X I T   P R O C E S S O R         *
************************************************
 Handle-LIBEXIT.
     Display "**** This module for INEXIT only"
     Move 16 To Exit-ReturnCode
     Goback.

************************************************
*    P R I N T  E X I T  P R O C E S S O R     *
```

```
*****************************************************
 Handle-PRTEXIT.
     Display "**** This module for INEXIT only"
     Move 16 To Exit-ReturnCode
     Goback.
*****************************************************

 End Program Skelinx.
```

付録 F. JNI.cpy

このリストはコピーブック JNI.cpy を示しています。これを使用すると、COBOL プログラムから Java Native Interface (JNI) サービスにアクセスすることができます。

JNI.cpy には、Java JNI タイプに対応するサンプル COBOL データ定義と、JNI 呼び出し可能サービスにアクセスするための関数ポインターを含む JNI 環境構造である JNINativeInterface が入っています。

JNI.cpy は、COBOL インストール・ディレクトリーの include サブディレクトリー (通常は /usr/lpp/cobol/include) にあります。の HFS にあります。 JNI.cpy は、C プログラマーが JNI にアクセスするために使用するヘッダー・ファイル jni.h に類似しています。

```
*****************************************************************
* COBOL declarations for Java native method interoperation      *
*                                                               *
* To use the Java Native Interface callable services from a     *
* COBOL program:                                                *
* 1) Use a COPY statement to include this file into the         *
*    the Linkage Section of the program, e.g.                   *
*      Linkage Section.                                         *
*      Copy JNI                                                 *
* 2) Code the following statements at the beginning of the      *
*    Procedure Division:                                        *
*      Set address of JNIEnv to JNIEnvPtr                       *
*      Set address of JNINativeInterface to JNIEnv              *
*****************************************************************
*
* Sample JNI type definitions in COBOL
*
*01 jboolean1 pic X.
*  88 jboolean1-true  value X'01' through X'FF'.
*  88 jboolean1-false value X'00'.
*
*01 jbyte1 pic X.
*
*01 jchar1 pic N usage national.
*
*01 jshort1 pic s9(4)  comp-5.
*01 jint1   pic s9(9)  comp-5.
*01 jlong1  pic s9(18) comp-5.
*
*01 jfloat1  comp-1.
*01 jdouble1 comp-2.
*
*01 jobject1 object reference.
*01 jclass1  object reference.
*01 jstring1 object reference jstring.
*01 jarray1  object reference jarray.
*
*01 jbooleanArray1 object reference jbooleanArray.
*01 jbyteArray1    object reference jbyteArray.
*01 jcharArray1    object reference jcharArray.
*01 jshortArray1   object reference jshortArray.
*01 jintArray1     object reference jintArray.
*01 jlongArray1    object reference jlongArray.
*01 floatArray1    object reference floatArray.
```

```
*01 jdoubleArray1  object reference jdoubleArray.
*01 jobjectArray1  object reference jobjectArray.


* Possible return values for JNI functions.
 01 JNI-RC pic S9(9) comp-5.
* success
   88 JNI-OK        value  0.
* unknown error
   88 JNI-ERR       value -1.
* thread detached from the VM
   88 JNI-EDETACHED value -2.
* JNI version error
   88 JNI-EVERSION  value -3.
* not enough memory
   88 JNI-ENOMEM    value -4.
* VM already created
   88 JNI-EEXIST    value -5.
* invalid arguments
   88 JNI-EINVAL    value -6.

* Used in ReleaseScalarArrayElements
 01 releaseMode pic s9(9) comp-5.
   88 JNI-COMMIT value 1.
   88 JNI-ABORT  value 2.

 01 JNIenv pointer.

* JNI Native Method Interface - environment structure.
 01 JNINativeInterface.
   02 pointer.
   02 pointer.
   02 pointer.
   02 pointer.
   02 GetVersion                      function-pointer.
   02 DefineClass                     function-pointer.
   02 FindClass                       function-pointer.
   02 FromReflectedMethod             function-pointer.
   02 FromReflectedField              function-pointer.
   02 ToReflectedMethod               function-pointer.
   02 GetSuperclass                   function-pointer.
   02 IsAssignableFrom                function-pointer.
   02 ToReflectedField                function-pointer.
   02 Throw                           function-pointer.
   02 ThrowNew                        function-pointer.
   02 ExceptionOccurred               function-pointer.
   02 ExceptionDescribe               function-pointer.
   02 ExceptionClear                  function-pointer.
   02 FatalError                      function-pointer.
   02 PushLocalFrame                  function-pointer.
   02 PopLocalFrame                   function-pointer.
   02 NewGlobalRef                    function-pointer.
   02 DeleteGlobalRef                 function-pointer.
   02 DeleteLocalRef                  function-pointer.
   02 IsSameObject                    function-pointer.
   02 NewLocalRef                     function-pointer.
   02 EnsureLocalCapacity             function-pointer.
   02 AllocObject                     function-pointer.
   02 NewObject                       function-pointer.
   02 NewObjectV                      function-pointer.
   02 NewObjectA                      function-pointer.
   02 GetObjectClass                  function-pointer.
   02 IsInstanceOf                    function-pointer.
   02 GetMethodID                     function-pointer.
   02 CallObjectMethod                function-pointer.
   02 CallObjectMethodV               function-pointer.
   02 CallObjectMethodA               function-pointer.
```

```
02 CallBooleanMethod               function-pointer.
02 CallBooleanMethodV              function-pointer.
02 CallBooleanMethodA              function-pointer.
02 CallByteMethod                  function-pointer.
02 CallByteMethodV                 function-pointer.
02 CallByteMethodA                 function-pointer.
02 CallCharMethod                  function-pointer.
02 CallCharMethodV                 function-pointer.
02 CallCharMethodA                 function-pointer.
02 CallShortMethod                 function-pointer.
02 CallShortMethodV                function-pointer.
02 CallShortMethodA                function-pointer.
02 CallIntMethod                   function-pointer.
02 CallIntMethodV                  function-pointer.
02 CallIntMethodA                  function-pointer.
02 CallLongMethod                  function-pointer.
02 CallLongMethodV                 function-pointer.
02 CallLongMethodA                 function-pointer.
02 CallFloatMethod                 function-pointer.
02 CallFloatMethodV                function-pointer.
02 CallFloatMethodA                function-pointer.
02 CallDoubleMethod                function-pointer.
02 CallDoubleMethodV               function-pointer.
02 CallDoubleMethodA               function-pointer.
02 CallVoidMethod                  function-pointer.
02 CallVoidMethodV                 function-pointer.
02 CallVoidMethodA                 function-pointer.
02 CallNonvirtualObjectMethod      function-pointer.
02 CallNonvirtualObjectMethodV     function-pointer.
02 CallNonvirtualObjectMethodA     function-pointer.
02 CallNonvirtualBooleanMethod     function-pointer.
02 CallNonvirtualBooleanMethodV    function-pointer.
02 CallNonvirtualBooleanMethodA    function-pointer.
02 CallNonvirtualByteMethod        function-pointer.
02 CallNonvirtualByteMethodV       function-pointer.
02 CallNonvirtualByteMethodA       function-pointer.
02 CallNonvirtualCharMethod        function-pointer.
02 CallNonvirtualCharMethodV       function-pointer.
02 CallNonvirtualCharMethodA       function-pointer.
02 CallNonvirtualShortMethod       function-pointer.
02 CallNonvirtualShortMethodV      function-pointer.
02 CallNonvirtualShortMethodA      function-pointer.
02 CallNonvirtualIntMethod         function-pointer.
02 CallNonvirtualIntMethodV        function-pointer.
02 CallNonvirtualIntMethodA        function-pointer.
02 CallNonvirtualLongMethod        function-pointer.
02 CallNonvirtualLongMethodV       function-pointer.
02 CallNonvirtualLongMethodA       function-pointer.
02 CallNonvirtualFloatMethod       function-pointer.
02 CallNonvirtualFloatMethodV      function-pointer.
02 CallNonvirtualFloatMethodA      function-pointer.
02 CallNonvirtualDoubleMethod      function-pointer.
02 CallNonvirtualDoubleMethodV     function-pointer.
02 CallNonvirtualDoubleMethodA     function-pointer.
02 CallNonvirtualVoidMethod        function-pointer.
02 CallNonvirtualVoidMethodV       function-pointer.
02 CallNonvirtualVoidMethodA       function-pointer.
02 GetFieldID                      function-pointer.
02 GetObjectField                  function-pointer.
02 GetBooleanField                 function-pointer.
02 GetByteField                    function-pointer.
02 GetCharField                    function-pointer.
02 GetShortField                   function-pointer.
02 GetIntField                     function-pointer.
02 GetLongField                    function-pointer.
02 GetFloatField                   function-pointer.
02 GetDoubleField                  function-pointer.
```

```
02 SetObjectField                 function-pointer.
02 SetBooleanField                function-pointer.
02 SetByteField                   function-pointer.
02 SetCharField                   function-pointer.
02 SetShortField                  function-pointer.
02 SetIntField                    function-pointer.
02 SetLongField                   function-pointer.
02 SetFloatField                  function-pointer.
02 SetDoubleField                 function-pointer.
02 GetStaticMethodID              function-pointer.
02 CallStaticObjectMethod         function-pointer.
02 CallStaticObjectMethodV        function-pointer.
02 CallStaticObjectMethodA        function-pointer.
02 CallStaticBooleanMethod        function-pointer.
02 CallStaticBooleanMethodV       function-pointer.
02 CallStaticBooleanMethodA       function-pointer.
02 CallStaticByteMethod           function-pointer.
02 CallStaticByteMethodV          function-pointer.
02 CallStaticByteMethodA          function-pointer.
02 CallStaticCharMethod           function-pointer.
02 CallStaticCharMethodV          function-pointer.
02 CallStaticCharMethodA          function-pointer.
02 CallStaticShortMethod          function-pointer.
02 CallStaticShortMethodV         function-pointer.
02 CallStaticShortMethodA         function-pointer.
02 CallStaticIntMethod            function-pointer.
02 CallStaticIntMethodV           function-pointer.
02 CallStaticIntMethodA           function-pointer.
02 CallStaticLongMethod           function-pointer.
02 CallStaticLongMethodV          function-pointer.
02 CallStaticLongMethodA          function-pointer.
02 CallStaticFloatMethod          function-pointer.
02 CallStaticFloatMethodV         function-pointer.
02 CallStaticFloatMethodA         function-pointer.
02 CallStaticDoubleMethod         function-pointer.
02 CallStaticDoubleMethodV        function-pointer.
02 CallStaticDoubleMethodA        function-pointer.
02 CallStaticVoidMethod           function-pointer.
02 CallStaticVoidMethodV          function-pointer.
02 CallStaticVoidMethodA          function-pointer.
02 GetStaticFieldID               function-pointer.
02 GetStaticObjectField           function-pointer.
02 GetStaticBooleanField          function-pointer.
02 GetStaticByteField             function-pointer.
02 GetStaticCharField             function-pointer.
02 GetStaticShortField            function-pointer.
02 GetStaticIntField              function-pointer.
02 GetStaticLongField             function-pointer.
02 GetStaticFloatField            function-pointer.
02 GetStaticDoubleField           function-pointer.
02 SetStaticObjectField           function-pointer.
02 SetStaticBooleanField          function-pointer.
02 SetStaticByteField             function-pointer.
02 SetStaticCharField             function-pointer.
02 SetStaticShortField            function-pointer.
02 SetStaticIntField              function-pointer.
02 SetStaticLongField             function-pointer.
02 SetStaticFloatField            function-pointer.
02 SetStaticDoubleField           function-pointer.
02 NewString                      function-pointer.
02 GetStringLength                function-pointer.
02 GetStringChars                 function-pointer.
02 ReleaseStringChars             function-pointer.
02 NewStringUTF                   function-pointer.
02 GetStringUTFLength             function-pointer.
02 GetStringUTFChars              function-pointer.
02 ReleaseStringUTFChars          function-pointer.
```

```
02 GetArrayLength                   function-pointer.
02 NewObjectArray                   function-pointer.
02 GetObjectArrayElement            function-pointer.
02 SetObjectArrayElement            function-pointer.
02 NewBooleanArray                  function-pointer.
02 NewByteArray                     function-pointer.
02 NewCharArray                     function-pointer.
02 NewShortArray                    function-pointer.
02 NewIntArray                      function-pointer.
02 NewLongArray                     function-pointer.
02 NewFloatArray                    function-pointer.
02 NewDoubleArray                   function-pointer.
02 GetBooleanArrayElements          function-pointer.
02 GetByteArrayElements             function-pointer.
02 GetCharArrayElements             function-pointer.
02 GetShortArrayElements            function-pointer.
02 GetIntArrayElements              function-pointer.
02 GetLongArrayElements             function-pointer.
02 GetFloatArrayElements            function-pointer.
02 GetDoubleArrayElements           function-pointer.
02 ReleaseBooleanArrayElements      function-pointer.
02 ReleaseByteArrayElements         function-pointer.
02 ReleaseCharArrayElements         function-pointer.
02 ReleaseShortArrayElements        function-pointer.
02 ReleaseIntArrayElements          function-pointer.
02 ReleaseLongArrayElements         function-pointer.
02 ReleaseFloatArrayElements        function-pointer.
02 ReleaseDoubleArrayElements       function-pointer.
02 GetBooleanArrayRegion            function-pointer.
02 GetByteArrayRegion               function-pointer.
02 GetCharArrayRegion               function-pointer.
02 GetShortArrayRegion              function-pointer.
02 GetIntArrayRegion                function-pointer.
02 GetLongArrayRegion               function-pointer.
02 GetFloatArrayRegion              function-pointer.
02 GetDoubleArrayRegion             function-pointer.
02 SetBooleanArrayRegion            function-pointer.
02 SetByteArrayRegion               function-pointer.
02 SetCharArrayRegion               function-pointer.
02 SetShortArrayRegion              function-pointer.
02 SetIntArrayRegion                function-pointer.
02 SetLongArrayRegion               function-pointer.
02 SetFloatArrayRegion              function-pointer.
02 SetDoubleArrayRegion             function-pointer.
02 RegisterNatives                  function-pointer.
02 UnregisterNatives                function-pointer.
02 MonitorEnter                     function-pointer.
02 MonitorExit                      function-pointer.
02 GetJavaVM                        function-pointer.
02 GetStringRegion                  function-pointer.
02 GetStringUTFRegion               function-pointer.
02 GetPrimitiveArrayCritical        function-pointer.
02 ReleasePrimitiveArrayCritical    function-pointer.
02 GetStringCritical                function-pointer.
02 ReleaseStringCritical            function-pointer.
02 NewWeakGlobalRef                 function-pointer.
02 DeleteWeakGlobalRef              function-pointer.
02 ExceptionCheck                   function-pointer.
```

関連タスク 329 ページの『UNIX のもとでのオブジェクト指向アプリケーションのコンパイル』 663 ページの『JNI サービスへのアクセス』

付録 G. COBOL SYSADATA ファイルの内容

ADATA コンパイラー・オプションを使用すると、コンパイラーは、プログラム・データを含んだファイルを生成します。コンパイラー・リストではなく、このファイルを使用して、プログラムに関する情報を取り出すことができます。例えば、シンボリック・デバッグ・ツールや相互参照ツールに対応したプログラムに関する情報を取り出すことができます。

811 ページの『例: SYSADATA』

関連参照

345 ページの『ADATA』
『SYSADATA ファイルに影響する既存のコンパイラー・オプション』
810 ページの『SYSADATA レコード・タイプ』
812 ページの『SYSADATA レコード記述』

SYSADATA ファイルに影響する既存のコンパイラー・オプション

いくつかのコンパイラー・オプションは、SYSADATA ファイルの内容に影響を与えることがあります。

COMPILE

NOCOMPILE(W|E|S) を使用すると、コンパイルが実行途中で停止され、その結果、特定のメッセージが失われる可能性があります。

EXIT INEXIT はコンパイル・ソース・ファイルの識別を禁止します。

LANGUAGE

LANGUAGE は、メッセージ・テキスト (大文字英語、英大 / 小文字混合、または日本語) を制御します。日本語を選択すると、そのために DBCS 文字がエラー識別レコードに書き込まれることになる場合があります。

TEST TEST を使用すると、SYSADATA ファイルの内容にも影響を与える、追加のオブジェクト・テキスト・レコードが作成されます。

NUM NUM を使用すると、コンパイラーは、生成されたシーケンス番号ではなく、ソース・レコードのカラム 1 から 6 の内容を行番号に使用します。無効 (非数値)、または順不同の番号は、直前のレコードより 1 だけ大きな数値で置き換えられます。

以下の SYSADATA フィールドには、NUM|NONUM 設定によってその内容が異なる、行番号が含まれています。

タイプ	フィールド	レコード
0020	AE_LINE	外部シンボル・レコード
0030	ATOK_LINE	トークン・レコード
0032	AF_STMT	ソース・エラー・レコード
0038	AS_STMT	ソース・レコード

タイプ	フィールド	レコード
0039	AS_REP_EXP_SLIN	COPY REPLACING レコード
0039	AS_REP_EXP_ELIN	COPY REPLACING レコード
0042	ASY_STMT	記号レコード
0044	AX_DEFN	記号相互参照レコード
0044	AX_STMT	記号相互参照レコード
0046	AN_STMT	ネストされたプログラム・レコード

タイプ 0038 ソース・レコードには、行番号とレコード番号に関連する 2 つのフィールドが含まれています。

- AS_STMT には、NUM および NONUM の両方に、コンパイラー行番号が含まれています。
- AS_CUR_REC# には、物理ソース・レコード番号が含まれています。

上記の 2 つのフィールドは常に、上記フィールドのすべてにおいて使われるコンパイラー行番号と物理ソース・レコード番号を相関させるために使用されます。

残りのコンパイラー・オプションは、SYSADATA ファイルに直接的な影響は与えませんが、FLAGSAA、FLAGSTD、SSRANGE など、特定のオプションに関連付けられた、別のエラー・メッセージの生成をトリガーする可能性があります。

811 ページの『例: SYSADATA』

関連参照
『SYSADATA レコード・タイプ』
352 ページの『COMPILE』
367 ページの『LANGUAGE』
374 ページの『NUMBER』
392 ページの『TEST』

SYSADATA レコード・タイプ

SYSADATA ファイルは、種々のレコード・タイプに分類されるレコードを含みます。各レコード・タイプには、コンパイルされる COBOL プログラムに関する情報が提供されます。

各レコードは、以下の 2 つのセクションで構成されます。

- 全レコード・タイプに対して同一の構造を有し、レコード・タイプを識別するレコード・コードを含む、12 バイトのヘッダー・セクション
- レコード・タイプによって異なる、可変長データ・セクション

表 109. **SYSADATA レコード・タイプ**

レコード・タイプ	アクション
815 ページの『ジョブ識別レコード: X'0000'』	ソース・データの処理に使用する環境に関する情報を記述します

表 109. **SYSADATA レコード・タイプ** *(続き)*

レコード・タイプ	アクション
815 ページの『ADATA 識別レコード: X'0001'』	SYSADATA ファイルのレコードに関する共通情報を記述します
816 ページの『コンパイル単位の開始/終了レコード: X'0002'』	ソース・ファイル内のコンパイル単位の開始と終了のマーク付けを行います
816 ページの『オプション・レコード: X'0010'』	コンパイルに使用するコンパイラー・オプションを記述します
827 ページの『外部シンボル・レコード: X'0020'』	プログラム内のすべての外部名、定義、および参照を記述します
828 ページの『構文解析ツリー・レコード: X'0024'』	プログラムの構文解析ツリーにノードを定義します
842 ページの『トークン・レコード: X'0030'』	ソース・トークンを定義します
855 ページの『ソース・エラー・レコード: X'0032'』	ソース・プログラム・ステートメントのエラーを記述します
856 ページの『ソース・レコード: X'0038'』	単一のソース行を記述します
857 ページの『COPY REPLACING レコード: X'0039'』	コピーブック内のテキストとの、COPY.?.?.REPLACING *operand-1* の突き合わせの結果として、テキスト置換のインスタンスを記述します
857 ページの『記号レコード: X'0042'』	プログラムに定義される、単一の記号を記述します。プログラムに定義される、それぞれの記号ごとに 1 つの記号レコードがあります。
870 ページの『記号相互参照レコード: X'0044'』	単一の記号への参照を記述します
871 ページの『ネストされたプログラム・レコード: X'0046'』	プログラムの名前とネスト・レベルを記述します
872 ページの『ライブラリー・レコード: X'0060'』	各ライブラリーで使用されるライブラリー・ファイルとメンバーを記述します
873 ページの『統計レコード: X'0090'』	コンパイルに関する統計を記述します
873 ページの『EVENTS レコード: X'0120'』	EVENTS レコードは、COBOL/370™ との互換性を提供します。レコード形式は、COBOL/370 と同一ですが、レコードの先頭に置かれる標準 ADATA ヘッダー、および EVENTS レコード・データの長さを示すフィールドが追加されます。

例: SYSADATA

以下の例は、COBOL プログラムのリストの一部を示しています。この COBOL プログラムを ADATA オプションを使用してコンパイルした場合には、関連データ・ファイルに生成されるレコードは、次の表に示す順序で記述されます。

```
000001        IDENTIFICATION DIVISION.                                  AD000020
000002        PROGRAM-ID. AD04202.                                      AD000030
000003        ENVIRONMENT DIVISION.                                     AD000040
000004        DATA DIVISION.                                            AD000050
000005        WORKING-STORAGE SECTION.                                  AD000060
000006        77   COMP3-FLD2  pic  S9(3)v9.                           AD000070
000007        PROCEDURE DIVISION.                                       AD000080
000008            STOP RUN.
```

タイプ	説明
X'0120'	EVENTS タイム・スタンプ・レコード
X'0120'	EVENTS プロセッサー・レコード
X'0120'	EVENTS ファイル ID レコード
X'0120'	EVENTS プログラム・レコード
X'0001'	ADATA 識別レコード
X'0000'	ジョブ識別レコード
X'0010'	オプション・レコード
X'0038'	ステートメント 1 のソース・レコード
X'0038'	ステートメント 2 のソース・レコード
X'0038'	ステートメント 3 のソース・レコード
X'0038'	ステートメント 4 のソース・レコード
X'0038'	ステートメント 5 のソース・レコード
X'0038'	ステートメント 6 のソース・レコード
X'0038'	ステートメント 7 のソース・レコード
X'0038'	ステートメント 8 のソース・レコード
X'0020'	AD04202 の外部シンボル・レコード
X'0044'	STOP の記号相互参照レコード
X'0044'	COMP3-FLD2 の記号相互参照レコード
X'0044'	AD04202 の記号相互参照レコード
X'0042'	AD04202 の記号レコード
X'0042'	COMP3-FLD2 の記号レコード
X'0090'	統計レコード
X'0120'	EVENTS ファイル終わりレコード

関連参照
『SYSADATA レコード記述』

SYSADATA レコード記述

関連データ・ファイルに書き込まれるレコードのフォーマットは、以下の関連参照に示されています。

それぞれのレコード・タイプに記述されたフィールドは、以下のシンボルで表されます。

C 文字 (EBCDIC または ASCII) データを表す

H 2 バイトの 2 進整数データを表す

F 4 バイトの 2 進整数データを表す

A 4 バイトの 2 進整数アドレスとオフセット・データを表す

X 16 進数 (ビット) データまたは 1 バイトの 2 進整数データを表す

データ型には、境界合わせは一切含まれていません。したがって、上記の暗黙の長さは、長さ指標 (L*n*) を含めることで変更される可能性があります。すべての整数データは、ヘッダー・フラグ・バイトの指標ビットによって、ビッグ・エンディアン形式、またはリトル・エンディアン形式になっています。ビッグ・エンディアン 形式では、ビット 0 が常に最上位ビットで、ビット *n* が最下位ビットであることを意味します。*リトル・エンディアン* は、Intel® プロセッサー上で見られる「バイト反転」整数に適用されます。

未定義フィールドおよび未使用値はすべて、予約済みです。

関連参照
『共通ヘッダー・セクション』
815 ページの『ジョブ識別レコード: X'0000'』
815 ページの『ADATA 識別レコード: X'0001'』
816 ページの『コンパイル単位の開始|終了レコード: X'0002'』
816 ページの『オプション・レコード: X'0010'』
827 ページの『外部シンボル・レコード: X'0020'』
828 ページの『構文解析ツリー・レコード: X'0024'』
842 ページの『トークン・レコード: X'0030'』
855 ページの『ソース・エラー・レコード: X'0032'』
856 ページの『ソース・レコード: X'0038'』
857 ページの『COPY REPLACING レコード: X'0039'』
857 ページの『記号レコード: X'0042'』
870 ページの『記号相互参照レコード: X'0044'』
871 ページの『ネストされたプログラム・レコード: X'0046'』
872 ページの『ライブラリー・レコード: X'0060'』
873 ページの『統計レコード: X'0090'』
873 ページの『EVENTS レコード: X'0120'』

共通ヘッダー・セクション

次の表は、すべてのレコード・タイプに共通するヘッダー・セクションの形式を示しています。 MVS および VSE の場合、各レコードの前に 4 バイトの RDW (レコード記述子ワード) が置かれます。この RDW は通常、アクセス方式によってのみ使用され、ダウンロード・ユーティリティーによって、ストリップされます。

表 110. **SYSADATA 共通ヘッダー・セクション**

フィールド	サイズ	説明
言語コード	XL1	**16** 高水準アセンブラー **17** すべてのプラットフォームの COBOL **40** サポートされるプラットフォームの PL/I

表 110. **SYSADATA 共通ヘッダー・セクション** *(続き)*

フィールド	サイズ	説明
レコード・タイプ	HL2	以下のいずれかのレコード・タイプ。 **X'0000'** ジョブ識別レコード [1] **X'0001'** ADATA 識別レコード **X'0002'** コンパイル単位の開始/終了レコード **X'0010'** オプション・レコード [1] **X'0020'** 外部シンボル・レコード **X'0024'** 構文解析ツリー・レコード **X'0030'** トークン・レコード **X'0032'** ソース・エラー・レコード **X'0038'** ソース・レコード **X'0039'** COPY REPLACING レコード **X'0042'** 記号レコード **X'0044'** 記号相互参照レコード **X'0046'** ネストされたプログラム・レコード **X'0060'** ライブラリー・レコード **X'0090'** 統計レコード [1] **X'0120'** EVENTS レコード
関連データ・アーキテクチャー・レベル	XL1	**3** ヘッダー構造の定義レベル
フラグ	XL1	**.... ..1.** ADATA レコード整数は、リトル・エンディアン (Intel) 形式です **.... ...1** このレコードは、次のレコードに続行されます **1111 11..** 将来の利用のために予約済み
関連データ・レコード・エディション・レベル	XL1	特定のレコード・タイプの新規形式を表すために使用され、通常は 0 です
予約済み	CL4	将来の利用のために予約済み
関連データ・フィールド長	HL2	ヘッダーの後に置かれるデータの長さ (バイト単位)

1. バッチ・コンパイル (一連のプログラム) が ADATA オプションで実行されるときには、それぞれのコンパイルごとに、複数ジョブ識別、オプション、および統計レコードがあります。

12 バイト・ヘッダーのマッピングには、MVS および VSE のアクセス方式が必要とする、可変長レコード記述子ワードに使用される領域は組み込まれません。

ジョブ識別レコード: X'0000'

次の表に、ジョブ識別レコードの内容を示します。

表 111. **SYSADATA ジョブ識別レコード**

フィールド	サイズ	説明
日付	CL8	YYYYMMDD 形式のコンパイルの日付
時刻	CL4	HHMM 形式のコンパイルの時刻
プロダクト番号	CL8	関連データ・ファイルを生成したコンパイラーのプロダクト番号
プロダクト・バージョン	CL8	V.R.M 形式の、関連データ・ファイルを生成したプロダクトのバージョン番号
PTF レベル	CL8	関連データ・ファイルを生成したプロダクトの PTF レベル番号 (PTF 番号が利用不可の場合、このフィールドはブランクです。)
システム ID	CL24	コンパイルが実行されたシステムのシステム識別
ジョブ名	CL8	コンパイル・ジョブの MVS ジョブ名
ステップ名	CL8	コンパイル・ステップの MVS ステップ名
PROC ステップ	CL8	コンパイル・プロシージャーの MVS プロシージャー・ステップ名
入力ファイル数 [1]	HL2	このレコードに記録した入力ファイルの数 以下の、7 つのフィールドのグループでは、このフィールドの値に従って、*n* 回発生します。
...入力ファイル番号	HL2	ファイルの割り当てシーケンス番号
...入力ファイル名長	HL2	次の入力ファイル名の長さ
...ボリューム通し番号長	HL2	ボリューム通し番号の長さ
...メンバー名長	HL2	メンバー名の長さ
...入力ファイル名	CL(*n*)	コンパイルの入力ファイルの名前
...ボリューム通し番号	CL(*n*)	入力ファイルが常駐する (最初の) ボリュームのボリューム通し番号
...メンバー名	CL(*n*)	該当する場合には、入力ファイル内のメンバーの名前

1. 入力ファイル数が、関連データ・ファイルのレコード・サイズを超える場合、レコードは次のレコードに続行されます。入力ファイルの現行数 (そのレコードの) はレコードに保管され、レコードは関連データ・ファイルに書き込まれます。入力ファイルの残りは次のレコードに入れられます。入力ファイルの数のカウントは、現行レコードのカウントです。

ADATA 識別レコード: X'0001'

次の表に、ADATA 識別レコードの内容を示します。

表 112. **ADATA 識別レコード**

フィールド	サイズ	説明
時刻 (2 進数)	XL8	世界時 (UT) は、真夜中のグリニッジ標準時を基準としたマイクロ秒数を表す 2 進数で、下位ビットは 1 マイクロ秒を表します。この時刻は、時間帯から独立したタイム・スタンプとして使用することができます。 Windows および AIX システムでは、フィールドのバイト 5 から 8 のみが、時刻を入れるフルワード・バイナリー・フィールドとして使用されます。
CCSID[1]	XL2	コード化文字セット ID
文字セット・フラグ	XL1	**X'80'** EBCDIC (IBM-037) **X'40'** ASCII (IBM-1252)
コード・ページ名長	XL2	後に続く、コード・ページ名の長さ
コード・ページ名	CL(*n*)	コード・ページの名前
1. 適切な CCS フラグが常に設定されます。CCSID がゼロ以外に設定されると、コード・ページ名の長さはゼロとなります。CCSID がゼロに設定されると、コード・ページ名の長さはゼロ以外の値となり、コード・ページ名が表示されます。		

コンパイル単位の開始|終了レコード: X'0002'

次の表に、コンパイル単位の開始|終了レコードの内容を示します。

表 113. **SYSADATA コンパイル単位の開始|終了レコード**

フィールド	サイズ	説明
タイプ	HL2	コンパイル単位タイプ。以下のどちらかが使用できます。 **X'0000'** 開始コンパイル単位 **X'0001'** 終了コンパイル単位
予約済み	CL2	将来の利用のために予約済み
予約済み	FL4	将来の利用のために予約済み

オプション・レコード: X'0010'

次の表に、オプション・レコードの内容を示します。

表 114. **SYSADATA オプション・レコード**

フィールド	サイズ	説明
オプション・バイト 0	XL1	**1111 1111** 将来の利用のために予約済み

表 114. **SYSADATA オプション・レコード** *(続き)*

フィールド	サイズ	説明
オプション・バイト 1	XL1	**1... ....** ビット 1 = DECK、ビット 0 = NODECK **.1.. ....** ビット 1 = ADATA、ビット 0 = NOADATA **..1. ....** ビット 1 = COLLSEQ(EBCDIC)、ビット 0 = COLLSEQ(LOCALE|BINARY) (Windows および AIX のみ) **...1 ....** ビット 1 = SEPOBJ、ビット 0 = NOSEPOBJ (Windows および AIX のみ) **.... 1...** ビット 1 = NAME、ビット 0 = NONAME **.... .1..** ビット 1 = OBJECT、ビット 0 = NOOBJECT **.... ..1.** ビット 1 = SQL、ビット 0 = NOSQL **.... ...1** ビット 1 = CICS、ビット 0 = NOCICS
オプション・バイト 2	XL1	**1... ....** ビット 1 = OFFSET、ビット 0 = NOOFFSET **.1.. ....** ビット 1 = MAP、ビット 0 = NOMAP **..1. ....** ビット 1 = LIST、ビット 0 = NOLIST **...1 ....** ビット 1 = DBCSXREF、ビット 0 = NODBCSXREF **.... 1...** ビット 1 = XREF(SHORT)、ビット 0 = XREF(SHORT) ではない。このフラグは、ビット 7 のフラグと組み合わせて使用します。このフラグによって、XREF(FULL) はオフ状態で示され、ビット 7 のフラグはオン状態で示されます。 **.... .1..** ビット 1 = SOURCE、ビット 0 = NOSOURCE **.... ..1.** ビット 1 = VBREF、ビット 0 = NOVBREF **.... ...1** ビット 1 = XREF、ビット 0 = XREF ではない。ビット 4 以降のフラグも参照。

表 *114.* **SYSADATA オプション・レコード** *(続き)*

フィールド	サイズ	説明
オプション・バイト 3	XL1	**1... ....** ビット 1 = 指定される FLAG 組み込み診断レベル (FLAG(*x*,*y*) のように値 *y* が指定されます) **.1.. ....** ビット 1 = FLAGSTD、ビット 0 = NOFLAGSTD **..1. ....** ビット 1 = NUM、ビット 0 = NONUM **...1 ....** ビット 1 = SEQUENCE、ビット 0 = NOSEQUENCE **.... 1...** ビット 1 = SOSI、ビット 0 = NOSOSI (Windows および AIX のみ) **.... .1..** ビット 1 = NSYMBOL(NATIONAL)、ビット 0 = NSYMBOL(DBCS) **.... ..1.** ビット 1 = PROFILE、ビット 0 = NOPROFILE (AIX のみ) **.... ...1** ビット 1 = WORD、ビット 0 = NOWORD
オプション・バイト 4	XL1	**1... ....** ビット 1 = ADV、ビット 0 = NOADV **.1.. ....** ビット 1 = APOST、ビット 0 = QUOTE **..1. ....** ビット 1 = DYNAM、ビット 0 = NODYNAM **...1 ....** ビット 1 = AWO、ビット 0 = NOAWO **.... 1...** ビット 1 = 指定済み RMODE、ビット 0 = RMODE(AUTO) **.... .1..** ビット 1 = RENT、ビット 0 = NORENT **.... ..1.** ビット 1 = RES: COBOL の場合、このフラグは常にオンに設定されます。 **.... ...1** ビット 1 = RMODE(24)、ビット 0 = RMODE(ANY)

表 114. **SYSADATA オプション・レコード** *(続き)*

フィールド	サイズ	説明
オプション・バイト 5	XL1	**1... ....** ビット 1 = SQLCCSID、ビット 0 = NOSQLCCSID **.1.. ....** ビット 1 = OPT、ビット 0 = NOOPT **..1. ....** ビット 1 = LIB、ビット 0 = NOLIB **...1 ....** ビット 1 = DBCS、ビット 0 = NODBCS **.... 1...** ビット 1 = OPT(FULL)、ビット 0 = OPT(FULL) ではない **.... .1..** ビット 1 = SSRANGE、ビット 0 = NOSSRANGE **.... ..1.** ビット 1 = TEST、ビット 0 = NOTEST **.... ...1** ビット 1 = PROBE、ビット 0 = NOPROBE (Windows のみ)
オプション・バイト 6	XL1	**..1. ....** ビット 1 = NUMPROC(PFD)、ビット 0 = NUMPROC(NOPFD) **...1 ....** ビット 1 = NUMCLS(ALT)、ビット 0 = NUMCLS(PRIM) **.... .1..** ビット 1 = BINARY(S390)、ビット 0 = BINARY(NATIVE) (Windows および AIX のみ) **.... ..1.** ビット 1 = TRUNC(STD)、ビット 0 = TRUNC(OPT) **.... ...1** ビット 1 = ZWB、ビット 0 = NOZWB **11.. 1...** 将来の利用のために予約済み

表 *114.* **SYSADATA オプション・レコード** *(続き)*

フィールド	サイズ	説明
オプション・バイト 7	XL1	**1... ....** ビット 1 = ALOWCBL、ビット 0 = NOALOWCBL **.1.. ....** ビット 1 = TERM、ビット 0 = NOTERM **..1. ....** 1 = DUMP、ビット 0 = NODUMP **.... ..1.** ビット 1 = CURRENCY、ビット 0 = NOCURRENCY **...1 11.1** 将来の利用のために予約済み
オプション・バイト 8	XL1	**1... ....** ビット 1 = XMLPARSE(XMLSS)、ビット 0 = XMLPARSE(COMPAT) **.1.. ....** ビット 1 = OPTFILE、ビット 0 = OPTFILE 以外 **..11 1111** 将来の利用のために予約済み
オプション・バイト 9	XL1	**1... ....** ビット 1 = DATA(24)、ビット 0 = DATA(31) **.1.. ....** ビット 1 = FASTSRT、ビット 0 = NOFASTSRT **..1. ....** ビット 1 = SIZE(MAX)、ビット 0 = SIZE(*nnnn*) または SIZE(*nnnn*K) **.... .1..** ビット 1 = THREAD、ビット 0 = NOTHREAD **...1 1.11** 将来の利用のために予約済み
オプション・バイト A	XL1	**1111 1111** 将来の利用のために予約済み
オプション・バイト B	XL1	**1111 1111** 将来の利用のために予約済み

表 114. **SYSADATA オプション・レコード** *(続き)*

フィールド	サイズ	説明
オプション・バイト C	XL1	**1... ....** ビット 1 = NCOLLSEQ(LOCALE) (Windows および AIX のみ) **.1.. ....** 将来の利用のために予約済み **..1. ....** ビット 1 = INTDATE(LILIAN)、ビット 0 = INTDATE(ANSI) **...1 ....** ビット 1 = NCOLLSEQ(BINARY) (Windows および AIX のみ) **.... 1...** ビット 1 = CHAR(EBCDIC)、ビット 0 = CHAR(NATIVE) (Windows および AIX のみ) **.... .1..** ビット 1 = FLOAT(HEX)、ビット 0 = FLOAT(NATIVE) (Windows および AIX のみ) **.... ..1.** ビット 1 = COLLSEQ(BINARY) (Windows および AIX のみ) **.... ...1** ビット 1 = COLLSEQ(LOCALE) (Windows および AIX のみ)

表 114. **SYSADATA オプション・レコード** *(続き)*

フィールド	サイズ	説明
オプション・バイト D	XL1	**1... ....** ビット 1 = DLL、ビット 0 = NODLL **.1.. ....** ビット 1 = EXPORTALL、ビット 0 = NOEXPORTALL **..1. ....** ビット 1 = CODEPAGE **...1 ....** ビット 1 = DATEPROC、ビット 0 = NODATEPROC **.... 1...** ビット 1 = DATEPROC(FLAG)、ビット 0 = DATEPROC(NOFLAG) **.... .1..** ビット 1 = YEARWINDOW **.... ..1.** ビット 1 = WSCLEAR、ビット 0 = NOWSCLEAR (Windows および AIX のみ) **.... ...1** ビット 1 = BEOPT、ビット 0 = NOBEOPT (Windows および AIX のみ)
オプション・バイト E	XL1	**1... ....** ビット 1 = DATEPROC(TRIG)、ビット 0 = DATEPROC(NOTRIG) **.1.. ....** ビット 1 = DIAGTRUNC、ビット 0 = NODIAGTRUNC **.... .1..** ビット 1 = LSTFILE(UTF-8)、ビット 0 = LSTFILE(LOCALE) (Windows および AIX のみ) **.... ..1.** ビット 1 = MDECK、ビット 0 = NOMDECK **.... ...1** ビット 1 = MDECK(NOCOMPILE) **..11 1...** 将来の利用のために予約済み
オプション・バイト F	XL1	**1111 1111** 将来の利用のために予約済み

表 114. **SYSADATA オプション・レコード** *(続き)*

フィールド	サイズ	説明
フラグ・レベル	XL1	**X'00'** フラグ (I) **X'04'** フラグ (W) **X'08'** フラグ (E) **X'0C'** フラグ (S) **X'10'** フラグ (U) **X'FF'** Noflag
組み込み診断レベル	XL1	**X'00'** フラグ (I) **X'04'** フラグ (W) **X'08'** フラグ (E) **X'0C'** フラグ (S) **X'10'** フラグ (U) **X'FF'** Noflag
FLAGSTD (FIPS) 指定	XL1	**1... ....** 最小 **.1.. ....** 中間 **..1. ....** 高 **...1 ....** IBM 拡張 **.... 1...** レベル 1 セグメンテーション **.... .1..** レベル 2 セグメンテーション **.... ..1.** デバッグ **.... ...1** 廃止
フラグ用に予約済み	XL1	**1111 1111** 将来の利用のために予約済み
コンパイラー・モード	XL1	**X'00'** 無条件 Nocompile、Nocompile(I) **X'04'** Nocompile(W) **X'08'** Nocompile(E) **X'0C'** Nocompile(S) **X'FF'** コンパイル
スペース値	CL1	

表 *114.* **SYSADATA オプション・レコード** *(続き)*

フィールド	サイズ	説明
3 値オプション用のデータ	XL1	**1... ....** NAME(ALIAS) 指定 **.1.. ....** NUMPROC(MIG) 指定 **..1. ....** TRUNC(BIN) 指定 **...1 1111** 将来の利用のために予約済み
TEST サブオプション	XL1	**1... ....** TEST(HOOK) **.1.. ....** TEST(SEP) **..1. ....** TEST(EJPD) **...1 1111** TEST サブオプションのために予約済み
OUTDD 名長	HL2	OUTDD 名の長さ
RWT ID 長	HL2	予約語テーブル ID の長さ
LVLINFO	CL4	ユーザー指定 LVLINFO データ
PGMNAME サブオプション	XL1	**1... ....** ビット 1 = PGMNAME(COMPAT) **.1.. ....** ビット 1 = PGMNAME(LONGUPPER) **..1. ....** ビット 1 = PGMNAME(LONGMIXED) **...1 1111** 将来の利用のために予約済み
記入項目インターフェース・サブオプション	XL1	**1... ....** ビット 1 = EntryInterface(System) (Windows のみ) **.1.. ....** ビット 1 = EntryInterface(OptLink) (Windows のみ) **..11 1111** 将来の利用のために予約済み

表 114. **SYSADATA オプション・レコード** *(続き)*

フィールド	サイズ	説明
CallInterface サブオプション	XL1	**1... ....** ビット 1 = CallInterface(System) (Windows および AIX のみ) **.1.. ....** ビット 1 = CallInterface(OptLink) (Windows のみ) **...1 ....** ビット 1 = CallInterface(Cdecl) (Windows のみ) **.... 1...** ビット 1 = CallInterface(System(Desc)) (Windows および AIX のみ) **..1. .111** 将来の利用のために予約済み
ARITH サブオプション	XL1	**1... ....** ビット 1 = ARITH(COMPAT) **.1.. ....** ビット 1 = ARITH(EXTEND) **11 1111** 将来の利用のために予約済み
DBCS 要件	FL4	DBCS XREF ストレージ要件
DBCS ORDPGM 長	HL2	DBCS 配列プログラムの名前の長さ
DBCS ENCTBL 長	HL2	DBCS エンコード・テーブルの名前の長さ
DBCS ORD TYPE	CL2	DBCS 配列型
予約済み	CL6	将来の利用のために予約済み
変換済み SO	CL1	変換済み SO 16 進数値
変換済み SI	CL1	変換済み SI 16 進数値
言語 ID	CL2	このフィールドには、LANGUAGE オプションからの 2 文字の省略形 (EN、UE、JA、または JP のいずれか) が保持されます。
予約済み	CL8	将来の利用のために予約済み
INEXIT 名長	HL2	SYSIN ユーザー出口名の長さ
PRTEXIT 名長	HL2	SYSPRINT ユーザー出口名の長さ
LIBEXIT 名長	HL2	'Library' ユーザー出口名の長さ
ADEXIT 名長	HL2	ADATA ユーザー出口名の長さ
CURROPT	CL5	CURRENCY オプション値
予約済み	CL1	将来の利用のために予約済み
YEARWINDOW	HL2	YEARWINDOW オプション値
CODEPAGE	HL2	CODEPAGE CCSID オプション値
予約済み	CL50	将来の利用のために予約済み
LINECNT	HL2	LINECOUNT 値
予約済み	CL2	将来の利用のために予約済み

表 114. **SYSADATA オプション・レコード** *(続き)*

フィールド	サイズ	説明
BUFSIZE	FL4	BUFSIZE オプション値
サイズ値	FL4	SIZE オプション値
予約済み	FL4	将来の利用のために予約済み
フェーズ常駐ビット バイト 1	XL1	**1... ....** ビット 1 = ユーザー領域内の IGYCLIBR **.1.. ....** ビット 1 = ユーザー領域内の IGYCSCAN **..1. ....** ビット 1 = ユーザー領域内の IGYCDSCN **...1 ....** ビット 1 = ユーザー領域内の IGYCGROU **.... 1...** ビット 1 = ユーザー領域内の IGYCPSCN **.... .1..** ビット 1 = ユーザー領域内の IGYCPANA **.... ..1.** ビット 1 = ユーザー領域内の IGYCFGEN **.... ...1** ビット 1 = ユーザー領域内の IGYCPGEN
フェーズ常駐ビット バイト 2	XL1	**1... ....** ビット 1 = ユーザー領域内の IGYCOPTM **.1.. ....** ビット 1 = ユーザー領域内の IGYCLSTR **..1. ....** ビット 1 = ユーザー領域内の IGYCXREF **...1 ....** ビット 1 = ユーザー領域内の IGYCDMAP **.... 1...** ビット 1 = ユーザー領域内の IGYCASM1 **.... .1..** ビット 1 = ユーザー領域内の IGYCASM2 **.... ..1.** ビット 1 = ユーザー領域内の IGYCDIAG **.... ...1** 将来の利用のために予約済み
フェーズ常駐ビット バイト 3 および 4	XL2	予約済み
予約済み	CL8	将来の利用のために予約済み
OUTDD 名	CL(*n*)	OUTDD 名
RWT	CL(*n*)	予約語テーブル ID
DBCS ORDPGM	CL(*n*)	DBCS 配列プログラム名

表 114. **SYSADATA オプション・レコード** *(続き)*

フィールド	サイズ	説明
DBCS ENCTBL	CL(*n*)	DBCS エンコード・テーブル名
INEXIT 名	CL(*n*)	SYSIN ユーザー出口名
PRTEXIT 名	CL(*n*)	SYSPRINT ユーザー出口名
LIBEXIT 名	CL(*n*)	'Library' ユーザー出口名
ADEXIT 名	CL(*n*)	ADATA ユーザー出口名

外部シンボル・レコード: X'0020'

次の表に、外部シンボル・レコードの内容を示します。

表 115. **SYSADATA 外部シンボル・レコード**

フィールド	サイズ	説明
セクション・タイプ	XL1	**X'00'** PROGRAM-ID 名 (メインエントリー・ポイント名) **X'01'** ENTRY 名 (2 次エントリー・ポイント名) **X'02'** 外部参照 (参照先の外部エントリー・ポイント) **X'04'** COBOL の場合は適用外 **X'05'** COBOL の場合は適用外 **X'06'** COBOL の場合は適用外 **X'0A'** COBOL の場合は適用外 **X'12'** 内部参照 (参照先の内部サブプログラム) **X'C0'** 外部クラス名 (OO COBOL クラス定義) **X'C1'** METHOD-ID 名 (OO COBOL メソッド定義) **X'C6'** メソッド参照 (OO COBOL メソッド参照) **X'FF'** COBOL の場合は適用外 タイプ X'12'、X'C0'、X'C1'、および X'C6' は、COBOL のみ対応です。
フラグ	XL1	COBOL の場合は適用外
予約済み	HL2	将来の利用のために予約済み
記号 ID	FL4	参照を含むプログラムの記号 ID (タイプ x'02' および x'12' の場合のみ)
行番号	FL4	参照を含むステートメントの行番号 (タイプ x'02' および x'12' の場合のみ)
セクション長	FL4	COBOL の場合は適用外
LD ID	FL4	COBOL の場合は適用外
予約済み	CL8	将来の利用のために予約済み
外部名の長さ	HL2	外部名の文字数
別名の長さ	HL2	COBOL の場合は適用外
外部名	CL(*n*)	外部名

表 115. **SYSADATA 外部シンボル・レコード** *(続き)*

フィールド	サイズ	説明
別名セクション名	CL(*n*)	COBOL の場合は適用外

構文解析ツリー・レコード: X'0024'

次の表に、構文解析ツリー・レコードの内容を示します。

表 116. **SYSADATA 構文解析ツリー・レコード**

フィールド	サイズ	説明
ノード番号	FL4	コンパイラーが生成するノード番号。1 から開始
ノード・タイプ	HL2	ノードのタイプ: **001** プログラム **002** クラス **003** メソッド
		101 見出し部 **102** 環境部 **103** データ部 **104** 手続き部 **105** 終了プログラム/メソッド/クラス
		201 宣言本文 **202** 非宣言本文
		301 セクション **302** プロシージャー・セクション
		401 段落 **402** 手順段落
		501 文 **502** ファイル定義 **503** ソート・ファイル定義 **504** プログラム名 **505** プログラム属性 **508** ENVIRONMENT DIVISION 節 **509** CLASS 属性 **510** METHOD 属性 **511** USE ステートメント

表 116. **SYSADATA 構文解析ツリー・レコード** *(続き)*

フィールド	サイズ	説明
		601 ステートメント **602** データ記述節 **603** データ入力項目 **604** ファイル記述節 **605** データ入力項目名 **606** データ入力項目レベル **607** EXEC 記入項目
		701 EVALUATE サブジェクト句 **702** EVALUATE WHEN 句 **703** EVALUATE WHEN OTHER 句 **704** SEARCH WHEN 句 **705** INSPECT CONVERTING 句 **706** INSPECT REPLACING 句 **707** INSPECT TALLYING 句 **708** PERFORM UNTIL 句 **709** PERFORM VARYING 句 **710** PERFORM AFTER 句 **711** ステートメント・ブロック **712** 範囲終了符号 **713** INITIALIZE REPLACING 句 **714** EXEC CICS コマンド **720** DATA DIVISION 句
		801 句 **802** ON 句 **803** NOT 句 **804** THEN 句 **805** ELSE 句 **806** 条件 **807** 式 **808** 相対索引付け **809** EXEC CICS オプション **810** 予約語 **811** INITIALIZE REPLACING カテゴリー

表 116. **SYSADATA 構文解析ツリー・レコード** *(続き)*

フィールド	サイズ	説明
		901 セクションまたは段落名 **902** ID **903** 英字名 **904** クラス名 **905** 条件名 **906** ファイル名 **907** 指標名 **908** 簡略名 **910** シンボリック文字 **911** リテラル **912** 関数 ID **913** データ名 **914** 特殊レジスター **915** プロシージャー参照 **916** 算術演算子 **917** 全プロシージャー **918** INITIALIZE リテラル (トークンなし) **919** ALL リテラルまたは形象定数 **920** キーワード・クラス・テスト名 **921** ID レベルの予約語 **922** 単項演算子 **923** 関係演算子
		1001 添え字 **1002** 参照変更
ノード・サブタイプ	HL2	ノードのサブタイプ。 セクション・タイプの場合: **0001** CONFIGURATION セクション **0002** INPUT-OUTPUT セクション **0003** FILE セクション **0004** WORKING-STORAGE セクション **0005** LINKAGE セクション **0006** LOCAL-STORAGE セクション **0007** REPOSITORY セクション

表 116. **SYSADATA 構文解析ツリー・レコード** *(続き)*

フィールド	サイズ	説明
		段落タイプの場合: **0001** PROGRAM-ID 段落 **0002** AUTHOR 段落 **0003** INSTALLATION 段落 **0004** DATE-WRITTEN 段落 **0005** SECURITY 段落 **0006** SOURCE-COMPUTER 段落 **0007** OBJECT-COMPUTER 段落 **0008** SPECIAL-NAMES 段落 **0009** FILE-CONTROL 段落 **0010** I-O-CONTROL 段落 **0011** DATE-COMPILED 段落 **0012** CLASS-ID 段落 **0013** METHOD-ID 段落 **0014** REPOSITORY 段落
		環境部節タイプの場合: **0001** WITH DEBUGGING MODE **0002** MEMORY-SIZE **0003** SEGMENT-LIMIT **0004** CURRENCY-SIGN **0005** DECIMAL POINT **0006** PROGRAM COLLATING SEQUENCE **0007** ALPHABET **0008** SYMBOLIC-CHARACTER **0009** CLASS **0010** ENVIRONMENT NAME **0011** SELECT

表 116. **SYSADATA 構文解析ツリー・レコード** *(続き)*

フィールド	サイズ	説明
		データ記述節タイプの場合: **0001** BLANK WHEN ZERO **0002** DATA-NAME OR FILLER **0003** JUSTIFIED **0004** OCCURS **0005** PICTURE **0006** REDEFINES **0007** RENAMES **0008** SIGN **0009** SYNCHRONIZED **0010** USAGE **0011** VALUE **0023** GLOBAL **0024** EXTERNAL

表 116. **SYSADATA 構文解析ツリー・レコード** *(続き)*

フィールド	サイズ	説明
		ファイル記述節タイプの場合: **0001** FILE STATUS **0002** ORGANIZATION **0003** ACCESS MODE **0004** RECORD KEY **0005** ASSIGN **0006** RELATIVE KEY **0007** PASSWORD **0008** PROCESSING MODE **0009** RECORD DELIMITER **0010** PADDING CHARACTER **0011** BLOCK CONTAINS **0012** RECORD CONTAINS **0013** LABEL RECORDS **0014** VALUE OF **0015** DATA RECORDS **0016** LINAGE **0017** ALTERNATE KEY **0018** LINES AT TOP **0019** LINES AT BOTTOM **0020** CODE-SET **0021** RECORDING MODE **0022** RESERVE **0023** GLOBAL **0024** EXTERNAL **0025** LOCK

表 116. **SYSADATA 構文解析ツリー・レコード** *(続き)*

フィールド	サイズ	説明
		ステートメント・タイプの場合: **0002** NEXT SENTENCE **0003** ACCEPT **0004** ADD **0005** ALTER **0006** CALL **0007** CANCEL **0008** CLOSE **0009** COMPUTE **0010** CONTINUE **0011** DELETE **0012** DISPLAY **0013** DIVIDE (INTO) **0113** DIVIDE (BY) **0014** ENTER **0015** ENTRY **0016** EVALUATE **0017** EXIT **0018** GO **0019** GOBACK **0020** IF **0021** INITIALIZE **0022** INSPECT

表 116. **SYSADATA 構文解析ツリー・レコード** *(続き)*

フィールド	サイズ	説明
		0023 INVOKE **0024** MERGE **0025** MOVE **0026** MULTIPLY **0027** OPEN **0028** PERFORM **0029** READ **0030** READY **0031** RELEASE **0032** RESET **0033** RETURN **0034** REWRITE **0035** SEARCH **0036** SERVICE **0037** SET **0038** SORT **0039** START **0040** STOP **0041** STRING **0042** SUBTRACT **0043** UNSTRING **0044** EXEC SQL **0144** EXEC CICS **0045** WRITE **0046** XML

表 116. **SYSADATA 構文解析ツリー・レコード** *(続き)*

フィールド	サイズ	説明
		句タイプの場合: **0001** INTO **0002** DELIMITED **0003** INITIALIZE. . .REPLACING **0004** INSPECT. . .ALL **0005** INSPECT. . .LEADING **0006** SET. . .TO **0007** SET. . .UP **0008** SET. . .DOWN **0009** PERFORM. . .TIMES **0010** DIVIDE. . .REMAINDER **0011** INSPECT. . .FIRST **0012** SEARCH. . .VARYING **0013** MORE-LABELS **0014** SEARCH ALL **0015** SEARCH. . .AT END **0016** SEARCH. . .TEST INDEX **0017** GLOBAL **0018** LABEL **0019** DEBUGGING **0020** SEQUENCE **0021** 将来の利用のために予約済み **0022** 将来の利用のために予約済み **0023** 将来の利用のために予約済み **0024** TALLYING **0025** 将来の利用のために予約済み **0026** ON SIZE ERROR **0027** ON OVERFLOW **0028** ON ERROR **0029** AT END **0030** INVALID KEY

表 116. **SYSADATA 構文解析ツリー・レコード** *(続き)*

フィールド	サイズ	説明
		0031 END-OF-PAGE
		0032 USING
		0033 BEFORE
		0034 AFTER
		0035 EXCEPTION
		0036 CORRESPONDING
		0037 将来の利用のために予約済み
		0038 RETURNING
		0039 GIVING
		0040 THROUGH
		0041 KEY
		0042 DELIMITER
		0043 POINTER
		0044 COUNT
		0045 METHOD
		0046 PROGRAM
		0047 INPUT
		0048 OUTPUT
		0049 I-O
		0050 EXTEND
		0051 RELOAD
		0052 ASCENDING
		0053 DESCENDING
		0054 DUPLICATES
		0055 NATIVE (USAGE)
		0056 INDEXED
		0057 FROM
		0058 FOOTING
		0059 LINES AT BOTTOM
		0060 LINES AT TOP
		0061 XML ENCODING
		0062 XML GENERATE XML-DECLARATION
		0063 XML GENERATE ATTRIBUTES
		0064 XML GENERATE NAMESPACE
		0065 XML PARSE PROCESSING

表 116. **SYSADATA 構文解析ツリー・レコード** *(続き)*

フィールド	サイズ	説明
		関数 ID タイプの場合: **0001** COS **0002** LOG **0003** MAX **0004** MIN **0005** MOD **0006** ORD **0007** REM **0008** SIN **0009** SUM **0010** TAN **0011** ACOS **0012** ASIN **0013** ATAN **0014** CHAR **0015** MEAN **0016** SQRT **0017** LOG10 **0018** RANGE **0019** LENGTH **0020** MEDIAN **0021** NUMVAL **0022** RANDOM **0023** ANNUITY **0024** INTEGER **0025** ORD-MAX **0026** ORD-MIN **0027** REVERSE **0028** MIDRANGE **0029** NUMVAL-C **0030** VARIANCE **0031** FACTORIAL **0032** LOWER-CASE

表 116. **SYSADATA 構文解析ツリー・レコード** *(続き)*

フィールド	サイズ	説明
		0033 UPPER-CASE **0034** CURRENT-DATE **0035** INTEGER-PART **0036** PRESENT-VALUE **0037** WHEN-COMPILED **0038** DAY-OF-INTEGER **0039** INTEGER-OF-DAY **0040** DATE-OF-INTEGER **0041** INTEGER-OF-DATE **0042** STANDARD-DEVIATION **0043** YEAR-TO-YYYY **0044** DAY-TO-YYYYDDD **0045** DATE-TO-YYYYMMDD **0046** UNDATE **0047** DATEVAL **0048** YEARWINDOW **0049** DISPLAY-OF **0050** NATIONAL-OF
		特殊レジスター・タイプの場合: **0001** ADDRESS OF **0002** LENGTH OF
		キーワード・クラス・テスト名タイプの場合: **0001** ALPHABETIC **0002** ALPHABETIC-LOWER **0003** ALPHABETIC-UPPER **0004** DBCS **0005** KANJI **0006** NUMERIC **0007** NEGATIVE **0008** POSITIVE **0009** ZERO

表 116. **SYSADATA 構文解析ツリー・レコード** *(続き)*

フィールド	サイズ	説明
		予約語タイプの場合: **0001** TRUE **0002** FALSE **0003** ANY **0004** THRU
		ID、データ名、索引名、条件名、または簡略名タイプの場合: **0001** REFERENCED **0002** CHANGED **0003** REFERENCED & CHANGED
		初期化リテラル・タイプの場合: **0001** ALPHABETIC **0002** ALPHANUMERIC **0003** NUMERIC **0004** ALPHANUMERIC-EDITED **0005** NUMERIC-EDITED **0006** DBCS/EGCS **0007** NATIONAL **0008** NATIONAL-EDITED
		プロシージャー名タイプの場合: **0001** SECTION **0002** PARAGRAPH
		ID レベルの予約語タイプの場合: **0001** ROUNDED **0002** TRUE **0003** ON **0004** オフ **0005** SIZE **0006** DATE **0007** DAY **0008** DAY-OF-WEEK **0009** TIME **0010** WHEN-COMPILED **0011** PAGE **0012** DATE YYYYMMDD **0013** DAY YYYYDDD

表 116. **SYSADATA 構文解析ツリー・レコード** *(続き)*

フィールド	サイズ	説明
		算術演算子タイプの場合: **0001** PLUS **0002** MINUS **0003** TIMES **0004** DIVIDE **0005** DIVIDE REMAINDER **0006** EXPONENTIATE **0007** NEGATE
		関係演算子タイプの場合: **0008** LESS **0009** LESS OR EQUAL **0010** EQUAL **0011** NOT EQUAL **0012** GREATER **0013** GREATER OR EQUAL **0014** AND **0015** OR **0016** CLASS CONDITION **0017** NOT CLASS CONDITION
親ノード番号	FL4	ノードの親のノード番号
左方兄弟ノード番号	FL4	ノードの左方兄弟のノード番号 (ある場合)。なしの場合、値はゼロ。
記号 ID	FL4	以下のタイプのいずれかのユーザー名の場合は、ノードの記号 ID。 • データ入力項目 • ID • ファイル名 • 指標名 • プロシージャー名。 • 条件名 • 簡略名 段落 ID に対応するプロシージャー名を除き、この値は記号 (タイプ 42) レコード内の記号 ID に対応します。 他のすべてのノード・タイプの場合、この値はゼロです。

表 116. **SYSADATA 構文解析ツリー・レコード** *(続き)*

フィールド	サイズ	説明
セクション記号 ID	FL4	修飾段落名参照の場合は、ノードが含まれているセクションの記号 ID。この値は記号 (タイプ 42) レコード内のセクション ID に対応します。 他のすべてのノード・タイプの場合、この値はゼロです。
最初のトークン番号	FL4	ノードに関連付けられた最初のトークンの番号
最後のトークン番号	FL4	ノードに関連付けられた最後のトークンの番号
予約済み	FL4	将来の利用のために予約済み
フラグ	CL1	ノードに関する情報は、以下を参照してください。 **X'80'** 予約済み **X'40'** 生成されたノード、トークンなし
予約済み	CL3	将来の利用のために予約済み

トークン・レコード: X'0030'

コンパイラーは、コメント行として扱われる行のトークン・レコードを生成しません。そのような行としては、以下にリストするものがありますが、それらに限定されません。

- コメント行。これは、桁 7 にアスタリスク (*) またはスラッシュ (/) が指定されている行です。
- 以下のコンパイラー指示ステートメント:
  - *CBL (*CONTROL)
  - BASIS
  - COPY
  - DELETE
  - EJECT
  - INSERT
  - REPLACE
  - SKIP1
  - SKIP2
  - SKIP3
  - TITLE
- デバッグ行。これは桁 7 に D が指定されている行です (WITH DEBUGGING MODE が指定されていない場合)。

表 117. **SYSADATA トークン・レコード**

フィールド	サイズ	説明
トークン番号	FL4	コンパイラーが生成するソース・ファイル内のトークン番号。1 から開始。ソースにはコピーブックを組み込み済みです。

表 117. **SYSADATA トークン・レコード** *(続き)*

フィールド	サイズ	説明
トークン・コード	HL2	トークンの型 (ユーザー名、リテラル、予約語など)。 予約語の場合は、コンパイラー予約語テーブル値が使用されます。 PICTURE ストリングの場合は、特別コード 0000 が使用されます。 継続されるトークンの各ピース (最後のピース以外) の場合は、特殊コード 3333 が使用されます。 その他の場合は、以下のコードが使用されます。 **0001** ACCEPT **0002** ADD **0003** ALTER **0004** CALL **0005** CANCEL **0007** CLOSE **0009** COMPUTE **0011** DELETE **0013** DISPLAY **0014** DIVIDE **0017** READY **0018** END-PERFORM **0019** ENTER **0020** ENTRY **0021** EXIT **0022** EXEC EXECUTE **0023** GO **0024** IF **0025** INITIALIZE **0026** INVOKE **0027** INSPECT **0028** MERGE **0029** MOVE

表 117. **SYSADATA トークン・レコード** *(続き)*

フィールド	サイズ	説明
		0030 MULTIPLY
		0031 OPEN
		0032 PERFORM
		0033 READ
		0035 RELEASE
		0036 RETURN
		0037 REWRITE
		0038 SEARCH
		0040 SET
		0041 SORT
		0042 START
		0043 STOP
		0044 STRING
		0045 SUBTRACT
		0048 UNSTRING
		0049 USE
		0050 WRITE
		0051 CONTINUE
		0052 END-ADD
		0053 END-CALL
		0054 END-COMPUTE
		0055 END-DELETE
		0056 END-DIVIDE
		0057 END-EVALUATE
		0058 END-IF
		0059 END-MULTIPLY
		0060 END-READ
		0061 END-RETURN
		0062 END-REWRITE
		0063 END-SEARCH
		0064 END-START
		0065 END-STRING
		0066 END-SUBTRACT
		0067 END-UNSTRING
		0068 END-WRITE
		0069 GOBACK

表 117. **SYSADATA トークン・レコード** *(続き)*

フィールド	サイズ	説明
		0070 EVALUATE
		0071 RESET
		0072 SERVICE
		0073 END-INVOKE
		0074 END-EXEC
		0075 XML
		0076 END-XML
		0099 FOREIGN-VERB
		0101 DATA-NAME
		0105 DASHED-NUM
		0106 DECIMAL
		0107 DIV-SIGN
		0108 EQ
		0109 EXPONENTIATION
		0110 GT
		0111 INTEGER
		0112 LT
		0113 LPAREN
		0114 MINUS-SIGN
		0115 MULT-SIGN
		0116 NONUMLIT
		0117 PERIOD
		0118 PLUS-SIGN
		0121 RPAREN
		0122 SIGNED-INTEGER
		0123 QUID
		0124 COLON
		0125 IEOF
		0126 EGCS-LIT
		0127 COMMA-SPACE
		0128 SEMICOLON-SPACE
		0129 PROCEDURE-NAME
		0130 FLT-POINT-LIT
		0131 言語環境プログラム

表 117. **SYSADATA トークン・レコード** *(続き)*

フィールド	サイズ	説明
		0132 GE
		0133 IDREF
		0134 EXPREF
		0136 CICS
		0137 NEW
		0138 NATIONAL-LIT
		0200 ADDRESS
		0201 ADVANCING
		0202 AFTER
		0203 ALL
		0204 ALPHABETIC
		0205 ALPHANUMERIC
		0206 ANY
		0207 AND
		0208 ALPHANUMERIC-EDITED
		0209 BEFORE
		0210 BEGINNING
		0211 FUNCTION
		0212 CONTENT
		0213 CORR
		CORRESPONDING
		0214 DAY
		0215 DATE
		0216 DEBUG-CONTENTS
		0217 DEBUG-ITEM
		0218 DEBUG-LINE
		0219 DEBUG-NAME
		0220 DEBUG-SUB-1
		0221 DEBUG-SUB-2
		0222 DEBUG-SUB-3
		0223 DELIMITED
		0224 DELIMITER
		0225 DOWN

表 117. **SYSADATA トークン・レコード** *(続き)*

フィールド	サイズ	説明
		0226 NUMERIC-EDITED
		0227 XML-EVENT
		0228 END-OF-PAGE
		EOP
		0229 EQUAL
		0230 ERROR
		0231 XML-NTEXT
		0232 EXCEPTION
		0233 EXTEND
		0234 FIRST
		0235 FROM
		0236 GIVING
		0237 GREATER
		0238 I-O
		0239 IN
		0240 INITIAL
		0241 INTO
		0242 INVALID
		0243 SQL
		0244 LESS
		0245 LINAGE-COUNTER
		0246 XML-TEXT
		0247 LOCK
		0248 GENERATE
		0249 NEGATIVE
		0250 NEXT
		0251 NO
		0252 NOT
		0253 NUMERIC
		0254 KANJI
		0255 OR
		0256 OTHER
		0257 OVERFLOW
		0258 PAGE
		0259 CONVERTING

表 117. **SYSADATA トークン・レコード** *(続き)*

フィールド	サイズ	説明
		0260 POINTER
		0261 POSITIVE
		0262 DBCS
		0263 PROCEDURES
		0264 PROCEED
		0265 REFERENCES
		0266 DAY-OF-WEEK
		0267 REMAINDER
		0268 REMOVAL
		0269 REPLACING
		0270 REVERSED
		0271 REWIND
		0272 ROUNDED
		0273 RUN
		0274 SENTENCE
		0275 STANDARD
		0276 RETURN-CODE
		SORT-CORE-SIZE
		SORT-FILE-SIZE
		SORT-MESSAGE
		SORT-MODE-SIZE
		SORT-RETURN
		TALLY
		XML-CODE
		0277 TALLYING
		0278 SUM
		0279 TEST
		0280 THAN
		0281 UNTIL
		0282 UP
		0283 UPON
		0284 VARYING
		0285 RELOAD
		0286 TRUE

表 117. **SYSADATA トークン・レコード** *(続き)*

フィールド	サイズ	説明
		0287 THEN
		0288 RETURNING
		0289 ELSE
		0290 SELF
		0291 SUPER
		0292 WHEN-COMPILED
		0293 ENDING
		0294 FALSE
		0295 REFERENCE
		0296 NATIONAL-EDITED
		0297 COM-REG
		0298 ALPHABETIC-LOWER
		0299 ALPHABETIC-UPPER
		0301 REDEFINES
		0302 OCCURS
		0303 SYNC
		SYNCHRONIZED
		0304 MORE-LABELS
		0305 JUST
		JUSTIFIED
		0306 SHIFT-IN
		0307 BLANK
		0308 VALUE
		0309 COMP
		COMPUTATIONAL
		0310 COMP-1
		COMPUTATIONAL-1
		0311 COMP-3
		COMPUTATIONAL-3
		0312 COMP-2
		COMPUTATIONAL-2
		0313 COMP-4
		COMPUTATIONAL-4
		0314 DISPLAY-1
		0315 SHIFT-OUT

表 117. **SYSADATA トークン・レコード** *(続き)*

フィールド	サイズ	説明	
		0316	INDEX
		0317	USAGE
		0318	SIGN
		0319	LEADING
		0320	SEPARATE
		0321	INDEXED
		0322	LEFT
		0323	RIGHT
		0324	PIC
			PICTURE
		0325	VALUES
		0326	GLOBAL
		0327	EXTERNAL
		0328	BINARY
		0329	PACKED-DECIMAL
		0330	EGCS
		0331	PROCEDURE-POINTER
		0332	COMP-5
			COMPUTATIONAL-5
		0333	FUNCTION-POINTER
		0334	TYPE
		0335	JNIENVPTR
		0336	NATIONAL
		0337	GROUP-USAGE
		0401	HIGH-VALUE
			HIGH-VALUES
		0402	LOW-VALUE
			LOW-VALUES
		0403	QUOTE
			QUOTES
		0404	SPACE
			SPACES
		0405	ZERO

表 117. **SYSADATA トークン・レコード** *(続き)*

フィールド	サイズ	説明	
		0406	ZEROES
			ZEROS
		0407	NULL
			NULLS
		0501	BLOCK
		0502	BOTTOM
		0505	CHARACTER
		0506	CODE
		0507	CODE-SET
		0514	FILLER
		0516	FOOTING
		0520	LABEL
		0521	LENGTH
		0524	LINAGE
		0526	OMITTED
		0531	RENAMES
		0543	TOP
		0545	TRAILING
		0549	RECORDING
		0601	INHERITS
		0603	RECURSIVE
		0701	ACCESS
		0702	ALSO
		0703	ALTERNATE
		0704	AREA
			AREAS
		0705	ASSIGN
		0707	COLLATING
		0708	COMMA
		0709	CURRENCY
		0710	CLASS
		0711	DECIMAL-POINT
		0712	DUPLICATES
		0713	DYNAMIC
		0714	EVERY

表 117. **SYSADATA トークン・レコード** *(続き)*

フィールド	サイズ	説明
		0716 MEMORY
		0717 MODE
		0718 MODULES
		0719 MULTIPLE
		0720 NATIVE
		0721 オフ
		0722 OPTIONAL
		0723 ORGANIZATION
		0724 POSITION
		0725 PROGRAM
		0726 RANDOM
		0727 RELATIVE
		0728 RERUN
		0729 RESERVE
		0730 SAME
		0731 SEGMENT-LIMIT
		0732 SELECT
		0733 SEQUENCE
		0734 SEQUENTIAL
		0736 SORT-MERGE
		0737 STANDARD-1
		0738 TAPE
		0739 WORDS
		0740 PROCESSING
		0741 APPLY
		0742 WRITE-ONLY
		0743 COMMON
		0744 ALPHABET
		0745 PADDING
		0746 SYMBOLIC
		0747 STANDARD-2
		0748 OVERRIDE
		0750 PASSWORD

表 117. **SYSADATA トークン・レコード** *(続き)*

フィールド	サイズ	説明
		0801 ARE
		IS
		0802 ASCENDING
		0803 AT
		0804 BY
		0805 CHARACTERS
		0806 CONTAINS
		0808 COUNT
		0809 DEBUGGING
		0810 DEPENDING
		0811 DESCENDING
		0812 DIVISION
		0814 FOR
		0815 ORDER
		0816 INPUT
		0817 REPLACE
		0818 KEY
		0819 LINE
		LINES
		0821 OF
		0822 ON
		0823 OUTPUT
		0825 RECORD
		0826 RECORDS
		0827 REEL
		0828 SECTION
		0829 SIZE
		0830 STATUS
		0831 THROUGH
		THRU
		0832 TIME
		0833 TIMES
		0834 TO
		0836 UNIT

表 117. **SYSADATA トークン・レコード** *(続き)*

フィールド	サイズ	説明
		0837 USING
		0838 WHEN
		0839 WITH
		0901 PROCEDURE
		0902 DECLARATIVES
		0903 END
		1001 DATA
		1002 FILE
		1003 FD
		1004 SD
		1005 WORKING-STORAGE
		1006 LOCAL-STORAGE
		1007 LINKAGE
		1101 ENVIRONMENT
		1102 CONFIGURATION
		1103 SOURCE-COMPUTER
		1104 OBJECT-COMPUTER
		1105 SPECIAL-NAMES
		1106 REPOSITORY
		1107 INPUT-OUTPUT
		1108 FILE-CONTROL
		1109 I-O-CONTROL
		1201 ID
		IDENTIFICATION
		1202 PROGRAM-ID
		1203 AUTHOR
		1204 INSTALLATION
		1205 DATE-WRITTEN
		1206 DATE-COMPILED
		1207 SECURITY
		1208 CLASS-ID
		1209 METHOD-ID
		1210 METHOD
		1211 FACTORY

表 117. **SYSADATA トークン・レコード** *(続き)*

フィールド	サイズ	説明
		1212 OBJECT **2020** TRACE **3000** DATADEF **3001** F-NAME **3002** UPSI-SWITCH **3003** CONDNAME **3004** CONDVAR **3005** BLOB **3006** CLOB **3007** DBCLOB **3008** BLOB-LOCATOR **3009** CLOB-LOCATOR **3010** DBCLOB-LOCATOR **3011** BLOB-FILE **3012** CLOB-FILE **3013** DBCLOB-FILE **3014** DFHRESP **5001** PARSE **5002** AUTOMATIC **5003** PREVIOUS **9999** COBOL
トークン長	HL2	トークンの長さ
トークン列	FL4	ソース・リスト内のトークンの開始列番号
トークン行	FL4	ソース・リスト内のトークンの行番号
フラグ	CL1	トークンに関する情報は、以下を参照してください。 **X'80'** トークンは継続 **X'40'** 継続されるトークンの最後のピース PICTURE ストリングの場合は、ソース・トークンが継続されても、1 つのトークン・レコードしか生成されません。トークン・コード 0000、最初のピースのトークン列と行、完全ストリングの長さ、継続なしフラグ・セット、完全ストリングのトークン・テキストが与えられます。
予約済み	CL7	将来の利用のために予約済み
トークン・テキスト	CL(*n*)	実トークン・ストリング

ソース・エラー・レコード: X'0032'

次の表に、ソース・エラー・レコードの内容を示します。

表 118. **SYSADATA ソース・エラー・レコード**

フィールド	サイズ	説明
ステートメント番号	FL4	エラーのあるステートメントのステートメント番号
エラー ID	CL16	エラー・メッセージ ID (左寄せ、ブランク埋め込み)
エラーの重大度	HL2	エラーの重大度
エラー・メッセージ長	HL2	エラー・メッセージ・テキストの長さ
行位置	XL1	FIPS メッセージに指定される行位置標識
予約済み	CL7	将来の利用のために予約済み
エラー・メッセージ	CL(*n*)	エラー・メッセージ・テキスト

ソース・レコード: X'0038'

次の表に、ソース・レコードの内容を示します。

表 119. **SYSADATA ソース・レコード**

フィールド	サイズ	説明
行番号	FL4	ソース・レコードのリスト行番号
入力レコード番号	FL4	現行入力ファイル内の入力ソース・レコード番号
1 次ファイル番号	HL2	このレコードが基本入力ファイルからの場合は、入力ファイルの割り当てシーケンス番号 (ジョブ識別レコード内の入力ファイル *n* フィールドを参照してください)。
ライブラリー・ファイル番号	HL2	このレコードが COPYIBASIS 入力ファイルからの場合は、ライブラリー入力ファイルの割り当てシーケンス番号。(ライブラリー・レコード内のメンバー・ファイル ID *n* フィールドを参照してください。)
予約済み	CL8	将来の利用のために予約済み
親レコード番号	FL4	親ソース・レコード番号。COPYIBASIS ステートメントのレコード番号となります。
親 1 次ファイル番号	HL2	このレコードの親が 1 次入力ファイルからの場合は、親ファイルの割り当てシーケンス番号 (ジョブ識別レコード内の入力ファイル *n* フィールドを参照してください)。
親ライブラリー割り当てファイル番号	HL2	このレコードの親が COPYIBASIS 入力ファイルからの場合は、親ライブラリー・ファイルの割り当てシーケンス番号。 (ライブラリー・レコード内の COPY/BASIS メンバー・ファイル ID *n* フィールドを参照してください。)
予約済み	CL8	将来の利用のために予約済み
ソース・レコードの長さ	HL2	後に続く実ソース・レコードの長さ。
予約済み	CL10	将来の利用のために予約済み
ソース・レコード	CL(*n*)	

COPY REPLACING レコード: X'0039'

REPLACING アクションが実行されるごとに、1 つの COPY REPLACING タイプのレコードが出力されます。すなわち、REPLACING 句の *operand-1* がコピーブックのテキストと一致するごとに、COPY REPLACING TEXT レコードが書き込まれます。

次の表に、COPY REPLACING レコードの内容を示します。

表 120. **SYSADATA COPY REPLACING レコード**

フィールド	サイズ	説明
置き換えられるストリングの開始行番号	FL4	REPLACING の結果であるテキストの開始のリスト行番号
置き換えられるストリングの開始桁番号	FL4	REPLACING の結果であるテキストの開始のリスト桁番号
置き換えられるストリングの終了行番号	FL4	REPLACING の結果であるテキストの終了のリスト行番号
置き換えられるストリングの終了桁番号	FL4	REPLACING の結果であるテキストの終了のリスト桁番号
オリジナル・ストリングの開始行番号	FL4	REPLACING により変更されたテキストの開始のソース・ファイル行番号
オリジナル・ストリングの開始列番号	FL4	REPLACING により変更されたテキストの開始のソース・ファイル桁番号
オリジナル・ストリングの終了行番号	FL4	REPLACING により変更されたテキストの終了のソース・ファイル行番号
オリジナル・ストリングの終了列番号	FL4	REPLACING により変更されたテキストの終了のソース・ファイル桁番号

記号レコード: X'0042'

次の表に、記号レコードの内容を示します。

表 121. **SYSADATA 記号レコード**

フィールド	サイズ	説明
記号 ID	FL4	固有の記号 ID
行番号	FL4	記号が定義または宣言されるソース・レコードのリスト行番号
レベル	XL1	記号の真のレベル番号 (または構造内のデータ項目の相対レベル番号)。 COBOL の場合、これは、01 から 49 の範囲、66 (RENAMES 項目の場合)、77、または 88 (条件項目の場合) になります。
修飾標識	XL1	**X'00'** 固有の名前。修飾は不要。 **X'01'** このデータ項目は修飾が必須。名前はプログラム内で固有ではありません。このフィールドが適用されるのは、このデータ項目がレベル 01 名ではない 場合のみです。

表 121. **SYSADATA 記号レコード** *(続き)*

フィールド	サイズ	説明
記号タイプ	XL1	**X'68'** クラス名 (クラス ID) **X'58'** メソッド名 **X'40'** データ名 **X'20'** プロシージャー名。 **X'10'** 簡略名 **X'08'** プログラム名 **X'81'** 予約済み 以下は、上記のタイプの ORed です (該当する場合)。 **X'04'** 外部 **X'02'** グローバル
記号属性	XL1	**X'01'** 数値 **X'02'** 次のいずれかのクラスの基本文字: • 英字 • 英数字 • DBCS • 国別 **X'03'** グループ **X'04'** ポインター **X'05'** 指標データ項目 **X'06'** 指標名 **X'07'** 条件 **X'0F'** ファイル **X'10'** ソート・ファイル **X'17'** クラス名 (リポジトリー) **X'18'** オブジェクト参照

表 121. **SYSADATA 記号レコード** *(続き)*

フィールド	サイズ	説明
節	XL1	記号定義で指定されている節。 数値 (X'01')、基本文字 (X'02')、グループ (X'03')、ポインター (X'04')、指標データ項目 (X'05')、またはオブジェクト参照 (X'18') の記号属性を持つ記号の場合: **1... ....** 値 **.1.. ....** 索引付き **..1. ....** 再定義 **...1 ....** 名前変更 **.... 1...** 発生 **.... .1..** Occurs キーあり **.... ..1.** ケースにより発生 **.... ...1** 親で発生 両方のファイル・タイプの場合: **1... ....** 選択 **.1.. ....** 割り当て **..1. ....** 再実行 **...1 ....** 同一領域 **.... 1...** 同一レコード域 **.... .1..** 記録モード **.... ..1.** 予約済み **.... ...1** レコード

表 121. **SYSADATA 記号レコード** *(続き)*

フィールド	サイズ	説明
		簡略名記号の場合: **01** CSP **02** C01 **03** C02 **04** C03 **05** C04 **06** C05 **07** C06 **08** C07 **09** C08 **10** C09 **11** C10 **12** C11 **13** C12 **14** S01 **15** S02 **16** S03 **17** S04 **18** S05 **19** CONSOLE **20** SYSIN|SYSIPT **22** SYSOUT|SYSLST|SYSLIST **24** SYSPUNCH|SYSPCH **26** UPSI-0 **27** UPSI-1 **28** UPSI-2 **29** UPSI-3 **30** UPSI-4 **31** UPSI-5 **32** UPSI-6 **33** UPSI-7 **34** AFP-5A

表 121. **SYSADATA 記号レコード** *(続き)*

フィールド	サイズ	説明
データ・フラグ 1	XL1	両方のファイル・タイプの場合、および数値 (X'01')、基本文字 (X'02')、グループ (X'03')、ポインター (X'04')、指標データ項目 (X'05')、またはオブジェクト参照 (X'18') の記号属性を持つ記号の場合: **1... ....** 再定義 **.1.. ....** 名前変更 **..1. ....** 同期化 **...1 ....** 暗黙的に再定義 **.... 1...** 日付フィールド **.... .1..** 暗黙の再定義 **.... ..1.** FILLER **.... ...1** レベル 77

表 121. **SYSADATA 記号レコード** *(続き)*

フィールド	サイズ	説明
データ・フラグ 2	XL1	数値 (X'01') の記号属性を持つ記号の場合: **1... ....** 2 進数 **.1.. ....** 外部浮動小数点 (USAGE DISPLAY または USAGE NATIONAL の) **..1. ....** 内部浮動小数点 **...1 ....** 圧縮 **.... 1...** 外部 10 進数 (USAGE DISPLAY または USAGE NATIONAL の) **.... .1..** 負の位取り **.... ..1.** 数字編集 (USAGE DISPLAY または USAGE NATIONAL の) **.... ...1** 将来の利用のために予約済み 基本文字 (X'02') またはグループ (X'03') の記号属性を持つ記号の場合: **1... ....** 英字 **.1.. ....** 英数字 **..1. ....** 編集英数字 **...1 ....** グループは独自の ODO オブジェクトを含む **.... 1...** DBCS 項目 **.... .1..** グループ可変長 **.... ..1.** EGCS 項目 **.... ...1** 編集 EGCS

表 121. **SYSADATA 記号レコード** *(続き)*

フィールド	サイズ	説明
		両方のファイル・タイプの場合: **1... ....** レコード内の ODO のオブジェクト **.1.. ....** レコード内の ODO のサブジェクト **..1. ....** 順次アクセス **...1 ....** ランダム・アクセス **.... 1...** 動的アクセス **.... .1..** 位置指定モード **.... ..1.** レコード域 **.... ...1** 将来の利用のために予約済み 他のすべてのデータ型の場合、フィールドはゼロです。
データ・フラグ 3	XL1	両方のファイル・タイプの場合: **1... ....** すべてのレコードが同じ長さ **.1.. ....** 固定長 **..1. ....** 可変長 **...1 ....** 未定義 **.... 1...** スパン **.... .1..** ブロック **.... ..1.** 書き込みのみ適用 **.... ...1** 同一ソート・マージ領域 他のすべてのデータ型の場合、フィールドはゼロです。

表 121. **SYSADATA 記号レコード** *(続き)*

フィールド	サイズ	説明
ファイル編成	XL1	両方のファイル・タイプの場合: **1... ....** QSAM **.1.. ....** ASCII **..1. ....** 標準ラベル **...1 ....** ユーザー・ラベル **.... 1...** VSAM 順次 **.... .1..** VSAM 索引 **.... ..1.** VSAM 相対 **.... ...1** 行順次 他のすべてのデータ型の場合、フィールドはゼロです。
USAGE 節	FL1	**X'00'** USAGE IS DISPLAY **X'01'** USAGE IS COMP-1 **X'02'** USAGE IS COMP-2 **X'03'** USAGE IS PACKED-DECIMAL または USAGE IS COMP-3 **X'04'** USAGE IS BINARY、USAGE IS COMP、または USAGE IS COMP-4 **X'05'** USAGE IS DISPLAY-1 **X'06'** USAGE IS POINTER **X'07'** USAGE IS INDEX **X'08'** USAGE IS PROCEDURE-POINTER **X'09'** USAGE IS OBJECT-REFERENCE **X'0B'** NATIONAL **X'0A'** FUNCTION-POINTER
記号節	FL1	**X'00'** SIGN 節なし **X'01'** SIGN IS LEADING **X'02'** SIGN IS LEADING SEPARATE CHARACTER **X'03'** SIGN IS TRAILING **X'04'** SIGN IS TRAILING SEPARATE CHARACTER

表 121. **SYSADATA 記号レコード** *(続き)*

フィールド	サイズ	説明
標識	FL1	**X'01'** JUSTIFIED 節あり。右寄せ属性が有効。 **X'02'** BLANK WHEN ZERO 節あり。
サイズ	FL4	このデータ項目のサイズ。この項目がストレージで占有する実バイト数。DBCS 項目の場合、この数は、文字数ではなく、バイト数で示されます。可変長項目の場合、このフィールドには、コンパイラーによってこの項目に予約されるストレージの最大サイズが反映されます。「長さ属性」とも呼ばれます。
精度	FL1	固定データ項目または浮動データ項目の精度
スケール	FL1	固定データ項目のスケール係数。小数点の右方の桁数。

表 121. **SYSADATA 記号レコード** *(続き)*

フィールド	サイズ	説明
ベース・ロケーター・タイプ	FL1	ホストの場合: **01** ベース・ロケーター・ファイル **02** ベース・ロケーター作業用ストレージ **03** ベース・ロケーター・リンケージ・セクション **05** ベース・ロケーター特殊レジスター **07** 変数別索引付き **09** COMREG 特殊レジスター **10** UPSI スイッチ **13** Varloc 項目のベース・ロケーター **14** 外部データのベース・ロケーター **15** ベース・ロケーター英数字 FUNC **16** ベース・ロケーター英数字 EVAL **17** オブジェクト・データのベース・ロケーター **19** Local-Storage のベース・ロケーター **20** ファクトリー・データ **21** XML-TEXT および XML-NTEXT Windows および AIX の場合: **01** ベース・ロケーター・ファイル **02** ベース・ロケーター・リンケージ・セクション **03** Varloc 項目のベース・ロケーター **04** 外部データのベース・ロケーター **05** オブジェクト・データのベース・ロケーター **06** XML-TEXT および XML-NTEXT **10** ベース・ロケーター作業用ストレージ **11** ベース・ロケーター特殊レジスター **12** ベース・ロケーター英数字 FUNC **13** ベース・ロケーター英数字 EVAL **14** 変数別索引付き **16** COMREG 特殊レジスター **17** UPSI スイッチ **18** ファクトリー・データ **22** Local-Storage のベース・ロケーター

表 121. **SYSADATA 記号レコード** *(続き)*

フィールド	サイズ	説明
日付形式	FL1	日付形式は以下のとおりです。 **01** YY **02** YYXX **03** YYXXXX **04** YYXXX **05** YYYY **06** YYYYXX **07** YYYYXXXX **08** YYYYXXX **09** YYX **10** YYYYX **22** XXYY **23** XXXXYY **24** XXXYY **26** XXYYYY **27** XXXXYYYY **28** XXXYYYY **29** XYY **30** XYYYY
データ・フラグ 4	XL1	数値 (X'01') の記号属性を持つ記号の場合: **1... ....** 数値国別 基本文字 (X'02') の記号属性を持つ記号の場合: **1... ....** 国別 **.1.. ....** 国別編集 グループ (X'03') の記号属性を持つ記号の場合: **1... ....** 国別グループ (Group-Usage National)
予約済み	FL3	将来の利用のために予約済み

表 121. **SYSADATA 記号レコード** *(続き)*

フィールド	サイズ	説明
住所情報	FL4	ホストの場合、ベース・ロケーター番号および変位は以下のとおりです。 **ビット 0 から 4** 未使用 **ビット 5 から 19** ベース・ロケーター (BL) 番号 **ビット 20 から 31** ベース・ロケーターの変位 Windows および AIX の場合、W コード SymId。
構造変位	AL4	構造内の記号のオフセット。このオフセットは変数位置指定項目に 0 を設定します。
親変位	AL4	定義中の項目の即時親からのバイト・オフセット。
親 ID	FL4	定義中の項目の即時親の記号 ID。
再定義 ID	FL4	この項目が再定義するデータ項目の記号 ID (該当する場合)。
開始名前変更 ID	FL4	この項目がレベル 66 項目の場合は、この項目が名前変更した開始 COBOL データ項目の記号 ID。レベル 66 項目でない場合、このフィールドは 0 に設定されます。
終了名前変更 ID	FL4	この項目がレベル 66 項目の場合は、この項目が名前変更した終了 COBOL データ項目の記号 ID。レベル 66 項目でない場合、このフィールドは 0 に設定されます。
プログラム名記号 ID	FL4	プログラムのプログラム名の ID またはこの記号が定義されているクラスのクラス名。
OCCURS 最小 段落 ID	FL4	OCCURS の最小値 段落名の PROC 名 ID
OCCURS 最大 セクション ID	FL4	OCCURS の最大値 セクション名の PROC 名 ID
寸法	FL4	寸法の数
大/小文字ビット・ベクトル	XL4	シンボル名の文字の大/小文字は、1 文字につき 1 ビットで表現されます。それぞれのビットには次の意味があります。 **0** 大文字 **1** 小文字 ビット 0 は、先頭文字の大/小文字を示します。ビット 1 は、2 番目の文字の大/小文字を示します。ビット 3 以降も同様です。
予約済み	CL8	将来の利用のために予約済み
値の組のカウント	HL2	値の組のカウント
記号名の長さ	HL2	記号名の文字数

表 121. **SYSADATA 記号レコード** *(続き)*

フィールド	サイズ	説明
データ名のピクチャー・データ長 または ファイル名の割り当て名の長さ	HL2	ピクチャー・データ内の文字の数: 関連した PICTURE 節を記号が持たない場合はゼロ。 (PICTURE フィールドの長さ。) 長さは、ソース入力内で検出されたときのフィールドを表します。この長さは、複製係数を含んでいる PICTURE 項目の拡張フィールドを表すものではありません。PICTURE ストリングの最大 COBOL 長は、50 バイトです。このフィールドがゼロの場合は、PICTURE が指定されていないことを示します。 これがファイル名の場合は、外部ファイル名の文字数。これは、割り当て名の DD 名部分です。ファイル名および ASSIGN USING が指定されている場合は、ゼロ。
データ名の初期値の長さ CLASS-ID の外部クラス名の長さ	HL2	記号値の文字の数。記号に初期値がない場合はゼロ。 CLASS-ID の外部クラス名の文字の数
データ名の ODO 記号名 ID ファイル名の場合、ASSIGN データ名の ID	FL4	データ名の場合、ODO 記号名の ID。ODO が指定されていない場合はゼロ。 ファイル名の場合、ASSIGN USING データ名の記号 ID。ASSIGN TO が指定されている場合はゼロ。
キー・カウント	HL2	定義されたキーの数
索引カウント	HL2	索引記号 ID のカウント。指定がない場合は、ゼロ
記号名	CL(*n*)	
データ名のピクチャー・データ・ストリング または ファイル名の割り当て名	CL(*n*)	ユーザーが入力したとおりの PICTURE 文字ストリング。文字ストリングには、全記号、括弧、およびレプリカの生成係数が含まれます。 これがファイル名の場合は外部ファイル名。これは、割り当て名の DD 名部分です。
索引 ID リスト	(*n*)FL4	各索引記号名の ID
キー	(*n*)XL8	このフィールドには、配列に対して指定するキーを記述するデータが含まれます。以下の 3 つのフィールドは、「キー・カウント」フィールドに指定されている回数だけ繰り返されます。
...キー・シーケンス	FL1	昇順または降順の標識。 **X'00'** DESCENDING **X'01'** ASCENDING
...充てん文字	CL3	予約済み
...キー ID	FL4	配列内のキー・フィールドであるデータ項目の記号 ID

表 121. **SYSADATA 記号レコード** *(続き)*

フィールド	サイズ	説明
データ名の初期値データ CLASS-ID の外部クラス名	CL(*n*)	このフィールドには、この記号の INITIAL VALUE 節で指定するデータが含まれます。以下の 4 つの適切なサブフィールドは、「値の組カウント」フィールド内のカウントに応じて繰り返されます。このフィールドのデータの全長は、「初期値の長さ」フィールドに入れられます。 CLASS-ID の外部クラス名。
...第 1 値の長さ	HL2	最初の値の長さ
...第 1 値のデータ	CL(*n*)	最初の値。 このフィールドには、ソース・ファイルの VALUE 節に指定されたとおりに、リテラル (または形象定数) が入ります。このフィールドには、開始/終了区切り文字、組み込み引用符、および SHIFT IN/SHIFT OUT 文字が入ります。リテラルが複数行にわたる場合には、行は 1 つの長いストリングに連結されます。形象定数が指定された場合には、このフィールドには、そのワードに関連付けられた値ではなく、実際の予約語が入ります。
...第 2 値の長さ	HL2	2 番目の値の長さ — THRU 値の組でない場合は、ゼロ
...第 2 値のデータ	CL(*n*)	2 番目の値。 このフィールドには、ソース・ファイルの VALUE 節に指定されたとおりに、リテラル (または形象定数) が入ります。このフィールドには、開始/終了区切り文字、組み込み引用符、および SHIFT IN/SHIFT OUT 文字が入ります。リテラルが複数行にわたる場合には、行は 1 つの長いストリングに連結されます。形象定数が指定された場合には、このフィールドには、そのワードに関連付けられた値ではなく、実際の予約語が入ります。

記号相互参照レコード: X'0044'

次の表に、記号相互参照レコードの内容を示します。

表 122. **SYSADATA 記号相互参照レコード**

フィールド	サイズ	説明
記号長	HL2	記号の長さ
ステートメント定義	FL4	記号が定義または宣言されているステートメント番号 **VERB XREF の場合のみ:** 動詞カウント。この動詞への参照の合計数。
参照の数 [1]	HL2	このレコード内の、後に続く記号への参照の数

表 122. **SYSADATA 記号相互参照レコード** *(続き)*

フィールド	サイズ	説明
相互参照タイプ	XL1	**X'01'** プログラム **X'02'** プロシージャー **X'03'** 動詞。 **X'04'** 記号またはデータ名 **X'05'** メソッド **X'06'** クラス
予約済み	CL7	将来の利用のために予約済み
記号名	CL(*n*)	記号。可変長。
...参照フラグ	CL1	記号またはデータ名参照の場合: **C' '** ブランクは参照のみの意味 **C'M'** 修正参照フラグ プロシージャー型記号参照の場合: **C'A'** `ALTER` (procedure-name) **C'D'** `GO TO` (procedure-name) `DEPENDING ON` **C'E'** (`PERFORM`) から (procedure-name) までの範囲の終わり **C'G'** `GO TO` (procedure-name) **C'P'** `PERFORM` (procedure-name) **C'T'** (`ALTER`) `TO PROCEED TO` (procedure-name) **C'U'** デバッグ用に使用 (procedure-name)
...ステートメント番号	XL4	記号または動詞が参照されているステートメント番号

1. 参照フラグ・フィールドおよびステートメント番号フィールドは、参照フィールドで指示される回数分だけ、発生します。例えば、参照フィールドの数に 10 の値が入っているとき、データ名、プロシージャー、またはプログラム式シンボルの場合には、参照フラグおよびステートメント番号ペアは 10 回発生し、動詞の場合には、ステートメント番号が 10 回発生します。

   参照数が、SYSADATA ファイルのレコード・サイズを超える場合、レコードは次のレコードに続行されます。レコードの共通ヘッダー・セクションに、継続フラグが設定されます。

ネストされたプログラム・レコード: X'0046'

次の表に、ネストされたプログラム・レコードの内容を示します。

表 123. **SYSADATA ネストされたプログラム・レコード**

フィールド	サイズ	説明
ステートメント定義	FL4	記号が定義または宣言されているステートメント番号
ネスト・レベル	XL1	プログラム・ネスト・レベル

表 123. **SYSADATA ネストされたプログラム・レコード** *(続き)*

フィールド	サイズ	説明
プログラム属性	XL1	**1... ....** 初期 **.1.. ....** 共通 **..1. ....** PROCEDURE DIVISION using **...1 1111** 将来の利用のために予約済み
予約済み	XL1	将来の利用のために予約済み
プログラム名の長さ	XL1	次のフィールドの長さ
プログラム名	CL(*n*)	プログラム名

ライブラリー・レコード: X'0060'

次の表に、SYSADATA ライブラリー・レコードの内容を示します。

表 124. **SYSADATA ライブラリー・レコード**

フィールド	サイズ	説明
メンバーの数 [1]	HL2	このレコードに記述される COPY/INCLUDE コード・メンバーの数のカウント
ライブラリー名長	HL2	ライブラリー名の長さ
ライブラリー・ボリューム長	HL2	ライブラリー・ボリューム ID の長さ
連結番号	XL2	ライブラリーの連結番号
ライブラリー DD 名長	HL2	ライブラリー DD 名の長さ
予約済み	CL4	将来の利用のために予約済み
ライブラリー名	CL(*n*)	COPY/INCLUDE メンバーが取り出された元のライブラリーの名前
ライブラリー・ボリューム	CL(*n*)	ライブラリーが常駐するボリュームのボリューム識別
ライブラリー DD 名	CL(*n*)	このライブラリーに使用される DD 名 (または同義)
...COPY/BASIS メンバー・ファイル ID[2]	HL2	... の後の名前のライブラリー・ファイル ID
...COPY/BASIS 名長	HL2	... の後の名前の長さ
...COPY/BASIS 名	CL(*n*)	使用されてきた COPY/BASIS メンバーの名前

表 124. **SYSADATA ライブラリー・レコード** *(続き)*

フィールド	サイズ	説明
1. ライブラリーから COPY メンバーが 10 メンバー取り出された場合には、「メンバーの数」フィールドに 10 が入り、「COPY/BASIS メンバー・ファイル ID」フィールド、「COPY/BASIS 名長」フィールド、および「COPY/BASIS 名」フィールドのオカレンスが 10 回出現します。 2. COPY/BASIS メンバーが別のライブラリーから取り出された場合には、それぞれの一意のライブラリーごとに、ライブラリー・レコードは SYSADATA ファイルに書き込まれます。		

統計レコード: X'0090'

次の表に、統計レコードの内容を示します。

表 125. **SYSADATA 統計レコード**

フィールド	サイズ	説明
ソース・レコード	FL4	処理されたソース・レコードの数
DATA DIVISION ステートメント	FL4	処理された DATA DIVISION ステートメントの数
PROCEDURE DIVISION ステートメント	FL4	処理された PROCEDURE DIVISION ステートメントの数
コンパイル番号	HL2	バッチ・コンパイル番号
エラーの重大度	XL1	最高のエラー・メッセージ重大度
フラグ	XL1	**1... ....** ジョブ終了標識 **.1.. ....** クラス定義標識 **..11 1111** 将来の利用のために予約済み
EOJ 重大度	XL1	コンパイル・ジョブの最大戻りコード
プログラム名の長さ	XL1	プログラム名の長さ
プログラム名	CL(*n*)	プログラム名

EVENTS レコード: X'0120'

以前のレベルのコンパイラーとの互換性を持たせるために、イベント・レコードが ADATA ファイルに含められます。

イベント・レコードには次のタイプがあります。

- タイム・スタンプ
- プロセッサー
- ファイル終了
- プログラム

- ファイル ID
- エラー

表 126. **SYSADATA EVENTS TIMESTAMP レコードのレイアウト**

フィールド	サイズ	説明
ヘッダー	CL12	標準 ADATA レコード・ヘッダー
レコード長	HL2	続く EVENTS レコード・データの長さ (このハーフワードは除く)
EVENTS レコード・タイプ TIMESTAMP のレコード	CL12	C'TIMESTAMP'
ブランク・セパレーター	CL1	
改訂レベル	XL1	
ブランク・セパレーター	CL1	
日付	XL8	YYYYMMDD
時間	XL2	HH
分	XL2	MI
秒	XL2	SS

表 127. **SYSADATA EVENTS PROCESSOR レコードのレイアウト**

フィールド	サイズ	説明
ヘッダー	CL12	標準 ADATA レコード・ヘッダー
レコード長	HL2	続く EVENTS レコード・データの長さ (このハーフワードは除く)
EVENTS レコード・タイプ PROCESSOR のレコード	CL9	C'PROCESSOR'
ブランク・セパレーター	CL1	
改訂レベル	XL1	
ブランク・セパレーター	CL1	
出力ファイル ID	XL1	
ブランク・セパレーター	CL1	
行クラス標識	XL1	

表 128. **SYSADATA EVENTS FILE END レコードのレイアウト**

フィールド	サイズ	説明
ヘッダー	CL12	標準 ADATA レコード・ヘッダー
レコード長	HL2	続く EVENTS レコード・データの長さ (このハーフワードは除く)

表 128. **SYSADATA EVENTS FILE END レコードのレイアウト** *(続き)*

フィールド	サイズ	説明
EVENTS レコード・タイプ FILE END のレコード	CL7	C'FILEEND'
ブランク・セパレーター	CL1	
改訂レベル	XL1	
ブランク・セパレーター	CL1	
入力ファイル ID	XL1	
ブランク・セパレーター	CL1	
拡張標識	XL1	

表 129. **SYSADATA EVENTS PROGRAM レコードのレイアウト**

フィールド	サイズ	説明
ヘッダー	CL12	標準 ADATA レコード・ヘッダー
レコード長	HL2	続く EVENTS レコード・データの長さ (このハーフワードは除く)
EVENTS レコード・タイプ PROGRAM のレコード	CL7	C'PROGRAM'
ブランク・セパレーター	CL1	
改訂レベル	XL1	
ブランク・セパレーター	CL1	
出力ファイル ID	XL1	
ブランク・セパレーター	CL1	
プログラム入力レコード番号	XL1	

表 130. **SYSADATA EVENTS FILE ID レコードのレイアウト**

フィールド	サイズ	説明
ヘッダー	CL12	標準 ADATA レコード・ヘッダー
レコード長	HL2	続く EVENTS レコード・データの長さ (このハーフワードは除く)
EVENTS レコード・タイプ FILE ID のレコード	CL7	C'FILEID'
ブランク・セパレーター	CL1	
改訂レベル	XL1	

表 130. **SYSADATA EVENTS FILE ID レコードのレイアウト** *(続き)*

フィールド	サイズ	説明
ブランク・セパレーター	CL1	
入力ソース・ファイル ID	XL1	ソース・ファイルのファイル ID
ブランク・セパレーター	CL1	
参照標識	XL1	
ブランク・セパレーター	CL1	
ソース・ファイル名の長さ	H2	
ブランク・セパレーター	CL1	
ソース・ファイル名	CL(n)	

表 131. **SYSADATA EVENTS ERROR レコードのレイアウト**

フィールド	サイズ	説明
ヘッダー	CL12	標準 ADATA レコード・ヘッダー
レコード長	HL2	続く EVENTS レコード・データの長さ (このハーフワードは除く)
EVENTS レコード・タイプ ERROR のレコード	CL5	C'ERROR'
ブランク・セパレーター	CL1	
改訂レベル	XL1	
ブランク・セパレーター	CL1	
入力ソース・ファイル ID	XL1	ソース・ファイルのファイル ID
ブランク・セパレーター	CL1	
Annot クラス	XL1	Annot クラス・メッセージの配置
ブランク・セパレーター	CL1	
エラー入力レコード番号	XL10	
ブランク・セパレーター	CL1	
エラー開始行番号	XL10	
ブランク・セパレーター	CL1	

表 131. **SYSADATA EVENTS ERROR レコードのレイアウト** *(続き)*

フィールド	サイズ	説明
エラー・トークン開始番号	XL1	エラー・トークン開始の桁番号
ブランク・セパレーター	CL1	
エラー終了行番号	XL10	
ブランク・セパレーター	CL1	
エラー・トークン終了番号	XL1	エラー・トークン終了の桁番号
ブランク・セパレーター	CL1	
エラー・メッセージの ID 番号	XL9	
ブランク・セパレーター	CL1	
エラー・メッセージの重大度コード	XL1	
ブランク・セパレーター	CL1	
エラー・メッセージの重大度レベル番号	XL2	
ブランク・セパレーター	CL1	
エラー・メッセージ長	HL3	
ブランク・セパレーター	CL1	
エラー・メッセージ・テキスト	CL(n)	

付録 H. サンプル・プログラムの使用

サンプル・プログラムはお手持ちのプロダクト・テープに収録されており、COBOL の多数の言語エレメントおよび概念を実例によって説明します。

この情報には以下の項目が含まれています。

- プログラムの概要 (2 つのサンプルに関するプログラム図表を含む)
- 入力データの形式とサンプル
- 作成される報告書のサンプル
- プログラムの実行方法についての情報
- 例示されている言語エレメントと概念のリスト

これらのプログラムに関する疑似コードとその他のコメントは、プログラムのプロローグに収められており、プログラム・リストで入手できます。

次の 3 つのサンプル・プログラムがあります。

- IGYTCARA は、QSAM ファイルと VSAM 索引付きファイルの使用例を提供し、多くの COBOL 組み込み関数の使用方法を示します。
- IGYTCARB は、IBM 対話式システム生産性向上機能 (ISPF) を使用する例です。
- IGYTSALE は、言語環境プログラム呼び出し可能サービスの機能のいくつかの使用例です。

関連概念
『IGYTCARA: バッチ・アプリケーション』
883 ページの『IGYTCARB: 対話式プログラム』
886 ページの『IGYTSALE: ネストされたプログラム・アプリケーション』

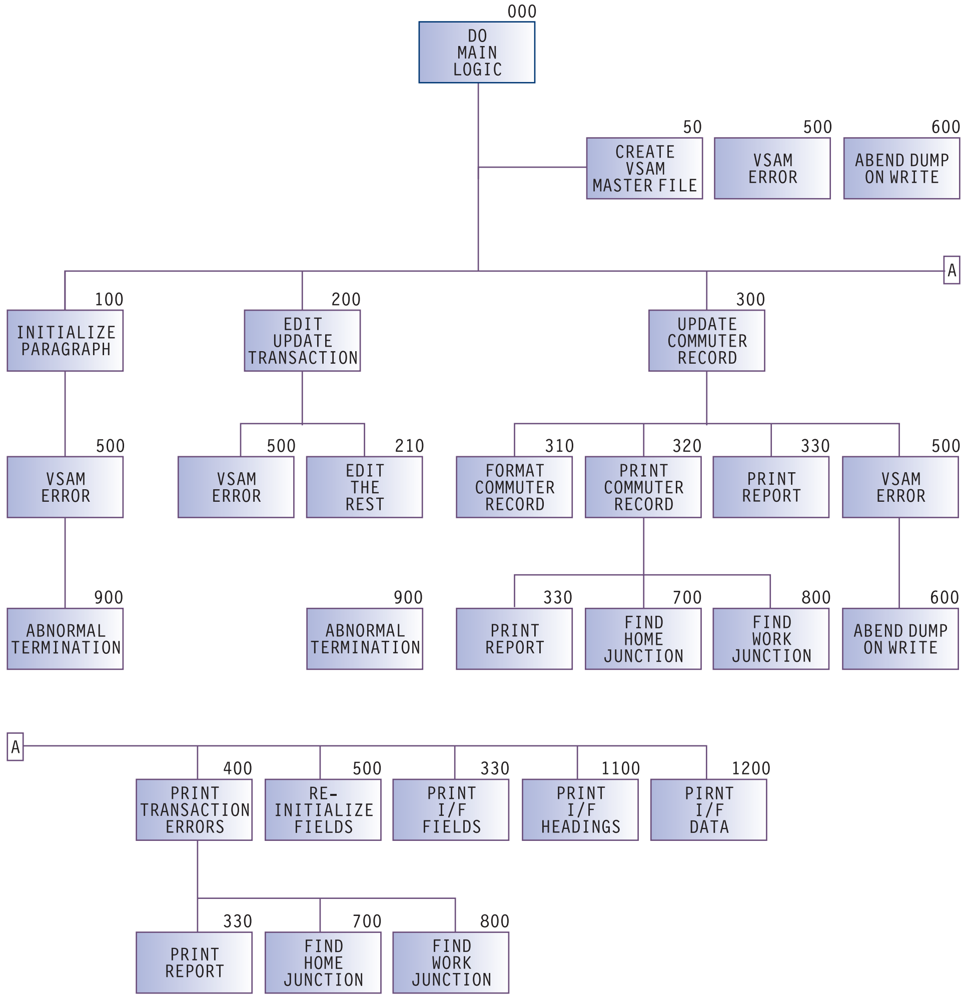
IGYTCARA: バッチ・アプリケーション

いくつかの地方事業所を持つ会社が従業員のための駐車場を作りたいとします。アプリケーション IGYTCARA は、トランザクション・ファイル項目を妥当性検査し (QSAM 順次ファイル処理)、マスター・ファイルを更新します (VSAM 索引付きファイル処理)。

このバッチ・アプリケーションでは、次の 2 つのタスクが実行されます。

- 同じ居住地域から同じ事業所まで同乗できる従業員の報告書を作成します
- 駐車場データを更新します
  - 新しい従業員のデータを追加します
  - 関係する従業員の情報を変更します
  - 従業員レコードを削除します
  - 無効な更新要求をリストします

次の図は、アプリケーションの各部分とそれらがどのように編成されているかを示しています。



関連タスク

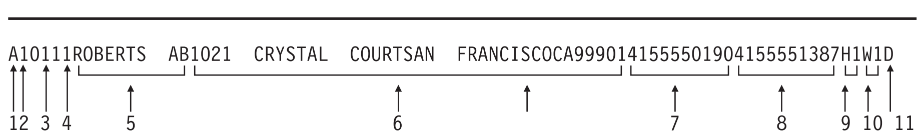
882 ページの『IGYTCARA の実行準備』

関連参照

『IGYTCARA の入力データ』
881 ページの『IGYTCARA によって作成される報告書』
894 ページの『示されている言語エレメントおよび概念』

IGYTCARA の入力データ

プログラムへの入力として、会社は関連のある従業員から情報を集め、その情報をコーディングして、入力ファイルを作成しました。以下に、入力ファイルの形式の例 (フィールド相互間のスペースは、入力ファイルの場合と同様、省かれています) を示し、その後に各項目を説明しています。



1. トランザクション・コード

2. シフト
3. ホーム・コード
4. 業務コード
5. 通勤者名
6. 自宅の住所
7. 自宅の電話番号
8. 勤務先電話
9. 居住地域コード
10. 勤務地域コード
11. 運転状況コード

以下は、入力ファイルのセクションの例です。

```
A10111ROBERTS  AB1021 CRYSTAL COURTSAN FRANCISCOCA99901415555019041555551387H1W1D
A20212KAHN     DE789 EMILY LANE    SAN FRANCISCOCA99921415555189041555552589H2W2D
P48899                                          99ASDFG0005557890123ASDFGHJ    T
R10111ROBERTS  AB1221 CRYSTAL COURTSAN FRANCISCOCA99901415555019041555551387H1W1D
A20212KAHN     DE789 EMILY LANE    SAN FRANCISCOCA99921415555189041555552589H2W2D
D20212KAHN     DE
D20212KAHN     DE
A20212KAHN     DE789 EMILY LANE    SAN FRANCISCOCA99921415555189041555552589H2W2D
A10111BONNICK  FD1025 FIFTH AVENUE SAN FRANCISCOCA99905415555959041555557895H8W3
A10111PETERSON SW435 THIRD AVENUE  SAN FRANCISCOCA99905415555546904155553717H3W4
   . . .
```

IGYTCARA によって作成される報告書

IGYTCARA によって作成される出力報告書の最初のページを、以下のサンプルに示しています。実際の出力は、システムによって形式が多少変わることがあります。

```
1REPORT  #: IGYTCAR1                          COMMUTER FILE UPDATE LIST                                    PAGE #:    1
 -PROGRAM #: IGYTCAR1       RUN TIME: 01:40                                          RUN DATE: 11/24/2003
 -=============================================================================================================================
       | RE-|  SHIFT |              |                      |               |                          |STA-|
 TRANS |CORD|HOME CODE|   COMMUTER   |        HOME          |  HOME PHONE   | HOME LOCATION JUNCTION   |TUS | TRANS. ERROR
 CODE  |TYPE|WORK CODE|    NAME      |       ADDRESS        |  WORK PHONE   | WORK LOCATION JUNCTION   |CODE|
 =============================================================================================================================
  A    NEW   1 01 11  ROBERTS    AB 1021 CRYSTAL COURT      (415) 555-0190 RODNEY/CRYSTAL            D
                                    SAN FRANCISCO   CA 99901 (415) 555-1387 BAYFAIR PLAZA
 -----------------------------------------------------------------------------------------------------------------------------
  A    NEW   2 02 12  KAHN       DE 789 EMILY LANE          (415) 555-1890 COYOTE                    D
                                    SAN FRANCISCO   CA 99921 (415) 555-2589 14TH STREET/166TH AVENUE
 -----------------------------------------------------------------------------------------------------------------------------
  P          4 88 99                                        (000) 555-7890 HOME CODE '  ' NOT FOUND.  T
                                                  99 ASDFG (123) ASD-FGHJ WORK CODE '  ' NOT FOUND.      TRANSACT. CODE
                                                                                                         SHIFT CODE
                                                                                                         HOME LOC. CODE
                                                                                                         WORK LOC. CODE
                                                                                                         LAST NAME
                                                                                                         INITIALS
                                                                                                         ADDRESS
                                                                                                         CITY
                                                                                                         STATE CODE
                                                                                                         ZIPCODE
                                                                                                         HOME PHONE
                                                                                                         WORK PHONE
                                                                                                         HOME JUNCTION
                                                                                                         WORK JUNCTION
                                                                                                         DRIVING STATUS
 -----------------------------------------------------------------------------------------------------------------------------
  R    OLD   1 01 11  ROBERTS    AB 1021 CRYSTAL COURT      (415) 555-0190 RODNEY/CRYSTAL            D
                                    SAN FRANCISCO   CA 99901 (415) 555-1387 BAYFAIR PLAZA
        NEW   1 01 11  ROBERTS    AB 1221 CRYSTAL COURT      (415) 555-0190 RODNEY/CRYSTAL              D
                                    SAN FRANCISCO   CA 99901 (415) 555-1387 BAYFAIR PLAZA
 -----------------------------------------------------------------------------------------------------------------------------
  A          2 02 12  KAHN       DE 789 EMILY LANE          (415) 555-1890 COYOTE                    D
                                    SAN FRANCISCO   CA 99921 (415) 555-2589 14TH STREET/166TH AVENUE      DUPLICATE REC.
 -----------------------------------------------------------------------------------------------------------------------------
  D    OLD   2 02 12  KAHN       DE 789 EMILY LANE          (415) 555-1890 COYOTE                    D
                                    SAN FRANCISCO   CA 99921 (415) 555-2589 14TH STREET/166TH AVENUE
 -----------------------------------------------------------------------------------------------------------------------------
  D          2 02 12  KAHN       DE                                                                   REC. NOT FOUND
 -----------------------------------------------------------------------------------------------------------------------------
  A    NEW   2 02 12  KAHN       DE 789 EMILY LANE          (415) 555-1890 COYOTE                    D
                                    SAN FRANCISCO   CA 99921 (415) 555-2589 14TH STREET/166TH AVENUE
 -----------------------------------------------------------------------------------------------------------------------------
```

```
A    NEW   1 01 11  BONNICK    FD 1025 FIFTH AVENUE        (415) 555-9590 RODNEY
                                  SAN FRANCISCO   CA 99905 (415) 555-7895 17TH FREEWAY SAN LEANDRO
-------------------------------------------------------------------------------------------------------------
A    NEW   1 01 11  PETERSON   SW 435 THIRD AVENUE         (415) 555-4690 RODNEY/THIRD AVENUE
```

IGYTCARA の実行準備

IGYTCARA プログラム (IGYTCARA、IGYTCODE、および IGYTRANX) に必要なすべてのファイルは、プロダクト・インストール・テープで提供されます。これらのファイルは、IGY.V4R1M0.SIGYSAMP データ・セット内にあります。

データ・セット名とプロシージャー名は、インストール時に変更することができます。これらの名前の妥当性について、システム・プログラマーにお問い合わせください。

IGYTCARA のソース・ファイルの中の、CBL ステートメントの以下のオプションは変更しないでください。

- NOADV
- NODYNAM
- NONAME
- NONUMBER
- QUOTE
- SEQUENCE

これらのオプションが有効な場合、プログラムは診断メッセージを出しません。ソース・ファイル内のシーケンス番号ストリングを使用すれば、使用されている言語エレメントを見つけることができます。

関連概念
879 ページの『IGYTCARA: バッチ・アプリケーション』

関連タスク 『IGYTCARA の実行』

関連参照
880 ページの『IGYTCARA の入力データ』
881 ページの『IGYTCARA によって作成される報告書』
894 ページの『示されている言語エレメントおよび概念』

IGYTCARA の実行

次のプロシージャーで、IGYTCARA プログラムのコンパイル、リンク・エディット、および実行が行われます。プログラムのコンパイルだけ、またはコンパイルとリンク・エディットだけを行う場合は、IGYWCLG カタログ式プロシージャーを変更しなければなりません。

z/OS のもとで IGYTCARA を実行するには、JCL を使用して VSAM クラスターを定義し、プログラムをコンパイルします。小文字で示されているフィールド (会計情報、ボリューム通し番号、装置名、クラスター接頭部) には、システムおよびインストール先固有の情報を挿入しなければなりません。以下の例では、名前 IGYTCAR.MASTFILE を使用しましたが、別の名前を使用しても構いません。

1. 次の JCL を使用して、必要な VSAM クラスターを作成します。

```
//CREATE   JOB (acct-info),'IGYTCAR CREATE VSAM',MSGLEVEL=(1,1),
// TIME=(0,29)
//CREATE   EXEC PGM=IDCAMS
//VOL1     DD VOL=SER=your-volume-serial,UNIT=your-unit,DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
 DELETE your-prefix.IGYTCAR.MASTFILE -
     FILE(VOL1) -
     PURGE
 DEFINE CLUSTER -
    (NAME(your-prefix.IGYTCAR.MASTFILE) -
     VOLUME(your-volume-serial) -
     FILE(VOL1) -
     INDEXED -
     RECSZ(80 80) -
     KEYS(16 0) -
     CYLINDERS(2))
/*
```

既存のクラスターを除去するために、VSAM クラスターの作成前に、DELETE が出されます。

2. 次の JCL を使用して、IGYTCARA プログラムをコンパイル、リンク・エディット、および実行します。

```
//IGYTCARA JOB (acct-info),'IGYTCAR',MSGLEVEL=(1,1),TIME=(0,29)
//TEST  EXEC IGYWCLG
//COBOL.SYSLIB  DD DSN=IGY.V4R1M0.SIGYSAMP,DISP=SHR
//COBOL.SYSIN   DD DSN=IGY.V4R1M0.SIGYSAMP(IGYTCARA),DISP=SHR
//GO.SYSOUT     DD SYSOUT=A
//GO.COMMUTR    DD DSN=your-prefix.IGYTCAR.MASTFILE,DISP=SHR
//GO.LOCCODE    DD DSN=IGY.V4R1M0.SIGYSAMP(IGYTCODE),DISP=SHR
//GO.UPDTRANS   DD DSN=IGY.V4R1M0.SIGYSAMP(IGYTRANX),DISP=SHR
//GO.UPDPRINT   DD SYSOUT=A,DCB=BLKSIZE=133
//
```

関連タスク

199 ページの『第 10 章 VSAM ファイルの処理』

関連参照 286 ページの『コンパイル、リンク・エディット、実行用のプロシージャー (IGYWCLG)』

IGYTCARB: 対話式プログラム

IGYTCARB には、IBM 対話式システム生産性向上機能 (ISPF) を使用してダイアログ・マネージャーと Enterprise COBOL を呼び出して駐車場データを入力するための対話式プログラムが入っています。 IGYTCARB は、IGYTCARA のような突き合わせプログラムまたは駐車場リストへの入力として使用できるファイルを作成します。

IGYTCARB の入力データは、IGYTCARA の場合と同じです。IGYTCARB により、ISPF パネルを介して入力ファイルに情報を追加することができます。IGYTCARB によって使用されるパネルの例を、以下に示します。

```
--------------------------- CARPOOL DATA ENTRY -------------------------------
             New Data Entry                                 Previous Entry
Type =======> -                                A, R, or D    A
Shift ======> -                                1, 2, or 3    1
Home Code ==> --                               2  Chars      01
Work Code ==> --                               2  Chars      11
Name =======> ---------                        9  Chars      POPOWICH
```

```
Initials ===> --                          2  Chars      AD
Address ====> ------------------          18 Chars      134 SIXTH AVENUE
City =======> -------------               13 Chars      SAN FRANCISCO
State ======> --                          2  Chars      CA
Zip Code ===> -----                       5  Chars      99903
Home Phone => ----------                  10 Chars      4155553390
Work Phone => ----------                  10 Chars      4155557855
Home Jnc code > --                        2  Chars      H3
Work Jnc Code > --                        2  Chars      W7
Commuter Stat > -                         D, R or blank
```

関連タスク
『IGYTCARB の実行準備』

IGYTCARB の実行準備

対話式システム生産性向上機能 (ISPF) のもとで、IGYTCARB プログラムを実行します。 IGYTCARB (IGYTCARB、IGYTRANB、および IGYTPNL) に必要なすべてのファイルは、IGY.V4R1M0.SIGYSAMP データ・セットのプロダクト・インストール・テープで提供されます。

データ・セット名とプロシージャー名は、インストール時に変更することができます。名前を確認するには、システム・プログラマーにお問い合わせください。

IGYTCARB のソース・ファイルの中の、CBL ステートメントの以下のオプションは変更しないでください。

- NONUMBER
- QUOTE
- SEQUENCE

これらのオプションが有効な場合、プログラムは診断メッセージを出しません。ソース・ファイル内のシーケンス番号ストリングを使用すれば、言語エレメントを見つけることができます。

関連概念
883 ページの『IGYTCARB: 対話式プログラム』

関連タスク 『IGYTCARB の実行』

関連参照
894 ページの『示されている言語エレメントおよび概念』

IGYTCARB の実行

次のプロシージャーで、IGYTCARB プログラムのコンパイル、リンク・エディット、および実行が行われます。プログラムのコンパイルだけ、またはコンパイルとリンク・エディットだけを行う場合は、このプロシージャーを変更しなければなりません。

z/OS で IGYTCARB を実行するには、次の手順を行います。

1. ISPF エディターを使用して、IGYTCARB 呼び出しを含めるよう、ISPF/PDF 基本オプション・パネル (ISR@PRIM) または他のパネルを変更します。パネル ISR@PRIM は、設置場所の PDF パネル・データ・セット (通常は ISRPLIB) に入っています。

   次の例は、2 つの指示された位置で IGYTCARB 呼び出しを含めるよう変更された ISR@PRIM パネルを示しています。パネル定義の上部にオプションを追加 (または変更) する場合は、パネルの下部にも対応する行を追加 (または変更) しなければなりません。

```
%----------------------  ISPF/PDF PRIMARY OPTION PANEL  -----------------------
%OPTION  ===>_ZCMD                                                             +
%                                                        +USERID   - &ZUSER
%   0 +ISPF PARMS  - Specify terminal and user parameters   +TIME     - &ZTIME
%   1 +BROWSE      - Display source data or output listings +TERMINAL - &ZTERM
%   2 +EDIT        - Create or change source data           +PF KEYS  - &ZKEYS
%   3 +UTILITIES   - Perform utility functions
%   4 +FOREGROUND  - Invoke language processors in foreground
%   5 +BATCH       - Submit to batch for language processing
%   6 +COMMAND     - Enter TSO or Workstation commands
%   7 +DIALOG TEST - Perform dialog testing
%   8 +LM UTILITIES- Perform library management utility functions
%   C +IGYTCARB    - Run IGYTCARB UPDATE TRANSACTION PROGRAM        (1)
%   T +TUTORIAL    - Display information about ISPF/PDF
%   X +EXIT        - Terminate using console, log, and list defaults
%
%
+Enter%END+command to terminate ISPF.
%
)INIT
  .HELP = ISR00003
  &ZPRIM = YES         /* ALWAYS A PRIMARY OPTION MENU */
  &ZHTOP = ISR00003    /* TUTORIAL TABLE OF CONTENTS   */
  &ZHINDEX = ISR91000  /* TUTORIAL INDEX - 1ST PAGE    */
  VPUT (ZHTOP,ZHINDEX) PROFILE
)PROC
  &Z1 = TRUNC(&ZCMD,1)
  IF (&Z1 &notsym.= '.')
    &ZSEL = TRANS( TRUNC (&ZCMD,'.')
                  0,'PANEL(ISPOPTA)'
                  1,'PGM(ISRBRO)  PARM(ISRBRO01)'
                  2,'PGM(ISREDIT) PARM(P,ISREDM01)'
                  3,'PANEL(ISRUTIL)'
                  4,'PANEL(ISRFPA)'
                  5,'PGM(ISRJB1) PARM(ISRJPA) NOCHECK'
                  6,'PGM(ISRPCC)'
                  7,'PGM(ISRYXDR) NOCHECK'
                  8,'PANEL(ISRLPRIM)'
                  C,'PGM(IGYTCARB)'                                (2)
                  T,'PGM(ISPTUTOR) PARM(ISR00000)'
                ' ',' '
                  X,'EXIT'
                  *,'?' )
    &ZTRAIL = .TRAIL
  IF (&Z1 = '.') .msg = ISPD141
)END
```

   この例の **(1)** で示されているように、以下を入力して、パネルの上部に IGYTCARB を追加します。

```
%   C +IGYTCARB    - Run IGYTCARB UPDATE TRANSACTION PROGRAM
```

(2) で示されているように、以下を入力して、パネルの下部に対応する行を追加します。

```
C,'PGM(IGYTCARB)'
```

2. ISR@PRIM (または他の変更したパネル) と IGYTPNL をライブラリーに入れ、このライブラリーを ISPPLIB 連結の最初のライブラリーにします。
3. IGYTCARB 内のシーケンス行 IB2200 にコメントを加え、シーケンス行 IB2210 のコメントを外します。 (z/OS のもとでは、OPEN EXTEND 動詞がサポートされます。)
4. IGYTCARB をコンパイルおよびリンク・エディットし、結果として生じたロード・モジュールを LOADLIB に入れます。
5. 次のコマンドを使用して、ISPLLIB を割り振ります。

   ```
   ALLOCATE FILE(ISPLLIB) DATASET(DSN1, SYS1.COBLIB, DSN2) SHR REUSE
   ```

   ここで、*DSN1* は、ステップ 4 の LOADLIB のライブラリー名です。*DSN2* は、インストールされた ISPLLIB です。
6. 次のコマンドを使用して、入力データ・セットと出力データ・セットを割り振ります。

   ```
   ALLOCATE FILE(UPDTRANS) DA('IGY.V4R1M0.SIGYSAMP(IGYTRANB)') SHR REUSE
   ```
7. 次のコマンドを使用して、ISPPLIB を割り振ります。

   ```
   ALLOCATE FILE(ISPPLIB) DATASET(DSN3, DSN4) SHR REUSE
   ```

   ここで、*DSN3* は、変更されたパネルが含まれているライブラリーです。*DSN4* は、ISPF パネル・ライブラリーです。
8. 変更されたパネルを使用して、IGYTCARB を呼び出します。

関連参照
対話式システム生産性向上機能 (ISPF) ダイアログ開発者ガイドとリファレンス

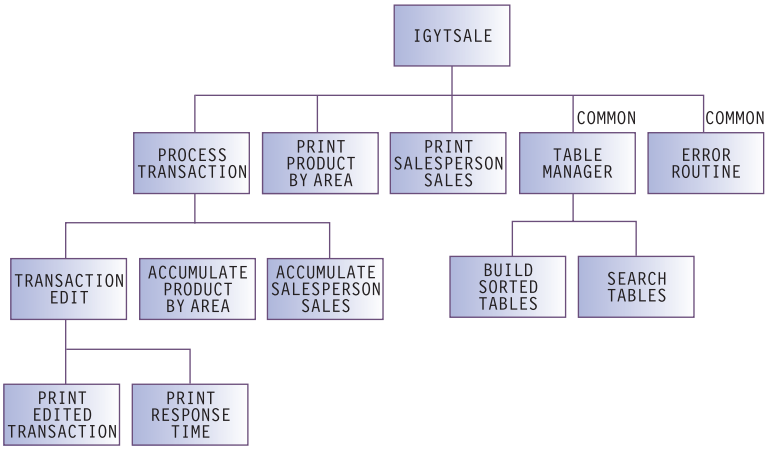
IGYTSALE: ネストされたプログラム・アプリケーション

アプリケーション IGYTSALE は、スポーツ用品販売店の商品売上と売上手数料を追跡します。

このネストされたプログラム・アプリケーションでは、次のタスクが実行されます。

1. 製品種目、顧客、および販売員の人数を記録します。このデータは、IGYTABLE と呼ばれるファイルに保管されます。
2. 有効なトランザクションとトランザクション・エラーを記録するファイルを保守します。無効なトランザクションにはすべてフラグが立てられ、結果は報告書に印刷されます。処理されるトランザクションは、IGYTRANA と呼ばれるファイルの中にあります。
3. トランザクションを処理して、地域ごとの売上高を報告します。
4. 各販売員の販売実績と手数料を記録して、その結果を報告書に印刷します。
5. 販売日と出荷日を地方時と UTC (世界協定時) で報告し、応答時間を計算します。

次の図は、アプリケーションの各部分を階層として示しています。



次の図は、各部分がどのようにネストされているかを示します。

```
IGYTSALE
  Process-transactions.
    Transaction-edit is Initial.
      Print-edited-Transactions and Response-Time.
      End Program Print-edited-transactions.
    End Program transaction-edit.
    Accumulate-product-by-area.
    End Program Accumulate-product-by-area.
    Accumulate-salesperson-sales.
    End Accumulate-salesperson-sales.
  End Program Process-transactions.
  Print-product-by-area.
  End Program Print-product-by-area.
  Print-salesperson-sales.
  End Program Print-salesperson-sales.
  Table-manager is common.
    Build-sorted-tables.
    End Program Build-sorted-tables.
    Search-tables.
    End Program Search-tables.
  End Program Table-manager.
  Error-routine is Common.
  End Program Error-routine is Common.
End Program IGYTSALE.
```

関連タスク

893 ページの『IGYTSALE の実行準備』

関連参照

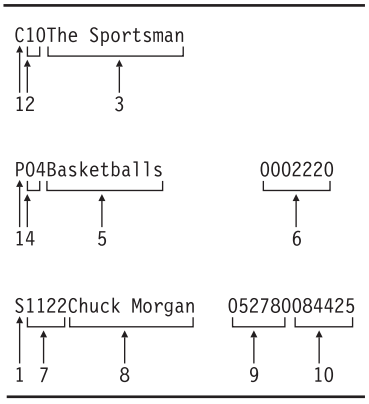
『IGYTSALE の入力データ』

889 ページの『IGYTSALE によって作成される報告書』
894 ページの『示されている言語エレメントおよび概念』

IGYTSALE の入力データ

プログラムへの入力として、この流通業者は、その顧客、販売員、および製品についての情報を集め、その情報をコーディングし、入力ファイルを作成しました。

IGYTABLE と呼ばれるこの入力ファイルは、トランザクション処理時に使用できるように 3 つの異なるテーブルにロードされます。以下に、このファイルの形式を示し、その後に各項目を説明しています。

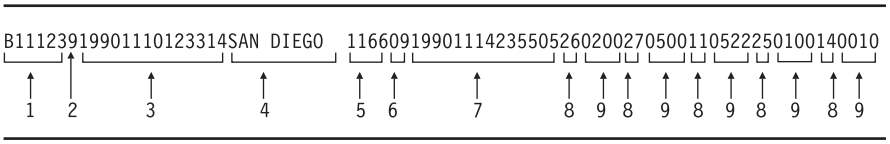


1. レコード・タイプ
2. 顧客コード
3. 顧客名
4. 製品コード
5. 製品説明
6. 製品の単価
7. 販売員の番号
8. 販売員名
9. 雇用日付
10. 手数料のレート

フィールド 1 の値 (C、P、または S) によって、入力レコードの形式が決まります。IGYTABLE のセクションの例を、以下に示します。

```
S1111Edyth Phillips 062484042327
S1122Chuck Morgan   052780084425
S1133Art Tung       022882061728
S1144Billy Jim Bob  010272121150
S1155Chris Preston  122083053377
S1166Al Willie Roz  111276100000
P01Footballs           0000620
P02Football Equipment  0032080
P03Football Uniform    0004910
P04Basketballs         0002220
P05Basketball Rim/Board0008830
P06Basketball Uniform  0004220
C01L. A. Sports
C02Gear Up
C03Play Outdoors
C04Sports 4 You
C05Sports R US
C06Stay Active
C07Sport Shop
C08Stay Sporty
C09Hot Sports
C10The Sportsman
C11Playing Ball
C12Sports Play
. . .
```

さらに、この流通業者では販売トランザクションに関する情報を集めました。各トランザクションは、特定の顧客に対する個々の販売員の売上高を示しています。顧客は、各トランザクション時に 1 から 5 個の品目を購入することができます。トランザクション情報がコーディングされ、IGYTRANA と呼ばれる入力ファイルに入れられます。以下に、このファイルの形式を示し、その後に各項目を説明しています。



1. 販売注文番号
2. 送り状を送った品目 (発注された各品目の数)
3. 販売の日付 (年月日時分秒)
4. 販売区域
5. 販売員の番号
6. 顧客コード
7. 出荷の日付 (年月日時分秒)
8. 製品コード
9. 売上数量

フィールド 8 および 9 は、発注された各品目の数 (フィールド 2) に応じて 1 から 8 回発生します。IGYTRANA のセクションの例を、以下に示します。

```
A000011199002270101O1CNTRL VALLEY11442019900228259999
A000041199003101005O0CNTRL VALLEY11441019900403150099
A000051199004182224O9CNTRL VALLEY11441219900419059900
A000061199005231510O0CNTRL VALLEY11442019900623250004
      419990324591515SAN DIEGO    11615        6020013220011052204510O
B111144199011110033O1SAN DIEGO   116615199011142602001322001105220411OO
A000071199011150032O5CNTRL VALLEY11332019901117120023
C001254199001181015O7SF BAY AREA 113315199001201602001122002505221451l1
B111164199012011320l3SF BAY AREA 113315199012030602001022001105220451O2
B111173199012010708l3SAN Diego   116566199012033302Oo132200120522041100
B111184199012211915445AN DIEGO   116614199012231602001422001305220403OO
B111194199012102115445AN DIEGO   112212199012140602001522001605220505OO
B111204199012120008l6SAN DIEGO   112204199012131502000522001605220401OO
B111214199012011315445AN DIEGO   113302199012031202001122001405222501OO
B111224199011120733l2SAN DIEGO   112210199011131002001622002605222501OO
B111239199011101233l4SAN DIEGO   116609199011142602002705001105222501001400l0
B111242199013135100O0SAN DIEGO   116611        1 0200042200120a22141100
B111254199012150125l0SAN DIEGO   116615199012161102001622001305221411l1
B111261199011110000345AN DIEGO   113316199011132600221
B111271199011101541O0SAN DIEGO   112212199011131220O0
B111284199011101750O1SAN DIEGO   116615199011132602001322001605210411O4
・・・
```

IGYTSALE によって作成される報告書

下記の図は、IGYTSALE 出力のサンプルです。

プログラムは以下のデータを報告書に記録します。

- トランザクション・エラー

- 製品および区域別の販売実績
- 個々の販売員の販売実績と歩合
- 販売日から、販売された製品が配送された日付までの応答時間

実際の出力は、システムによって形式が多少変わることがあります。

『例: IGYTSALE のトランザクション・エラー』
『例: IGYTSALE の製品別および区域別の販売分析』
891 ページの『例: IGYTSALE の販売と歩合』
892 ページの『例: IGYTSALE の販売から配送までの応答時間』

例: IGYTSALE のトランザクション・エラー

IGYTSALE 出力の次の例では、最後の欄にトランザクション・エラーがあることを示しています。

```
Day of Report: Tuesday            C O B O L   S P O R T S       11/24/2003    03:12    Page:   1
                                        Invalid Edited Transactions
  Sales  Inv.  Sales          Sales        Sales  Cust. Product And Quantity Sold       Ship
  Order  Items Time Stamp     Area         Pers   Code                                  Date Stamp
  -----  ----- -------------- -----------  -----  ----- --------------------------      ------------
            4 19990324591515  SAN DIEGO      116     15 6020013220011052204510O                Error Descriptions
                                                                                          -Sales order number is missing
                                                                                          -Date of sale time stamp is invalid
                                                                                          -Salesperson number not numeric
                                                                                          -Product code not in product-table
                                                                                          -Date of ship time stamp is invalid
  B11117    3 19901201070833  SAN Diego     1165     66 33020o132200120522041100       19901203 Error Descriptions
                                                                                          -Sales area not in area-table
                                                                                          -Salesperson not in sales-per-table
                                                                                          -Customer code not in customer-table
                                                                                          -Product code not in product-table
                                                                                          -Quantity sold not numeric
  B11123    9 19901110123314  SAN DIEGO     1166     09 26020027050011052225010014O010 19901114 Error Descriptions
                                                                                          -Invoiced items is invalid
                                                                                          -Product and quantity not checked
                                                                                          -Date of ship time stamp is invalid
  B11124    2 19901313510000  SAN DIEGO     1166     11 1 0200042200120a22141100               Error Descriptions
                                                                                          -Date of sale time stamp is invalid
                                                                                          -Product code is invalid
                                                                                          -Date of ship time stamp is invalid
     133          81119110000  LOS ANGELES   1166     10 040112110210160321251104               Error Descriptions
                                                                                          -Sales order number is invalid
                                                                                          -Invoiced items is invalid
                                                                                          -Date of sale time stamp is invalid
                                                                                          -Product and quantity not checked
                                                                                          -Date of ship time stamp is invalid
  C11133    4 1990111944                    1166     10 040112110210160321251104               Error Descriptions
                                                                                          -Date of sale time stamp is invalid
                                                                                          -Sales area is missing
                                                                                          -Date of ship time stamp is invalid
  C11138    4 19901117091530  LOS ANGELES   1155        113200102010260321250004       19901119 Error Descriptions
                                                                                          -Customer code is invalid
  D00009    9 19901201222222  CNTRL COAST   115      19 141  1131221                  19901202 Error Descriptions
                                                                                          -Invoiced items is invalid
```

例: IGYTSALE の製品別および区域別の販売分析

IGYTSALE 出力の次の例では、製品および区域別による販売成績を示しています。

```
Day of Report: Tuesday            C O B O L   S P O R T S       11/24/2003    03:12    Page:   1
                                        Sales Analysis By Product By Area
                                                  Areas of Sale
|                    | CNTRL COAST | CNTRL VALLEY | LOS ANGELES | NORTH COAST | SAN DIEGO   | SF BAY AREA |                |
|  Product   Codes   |             |              |             |             |             |             | Product Totals |
|====================|=============|==============|=============|=============|=============|=============|================|
|Product Number  04  |             |              |             |             |             |             |                |
|Basketballs         |             |              |             |             |             |             |                |
|   Units Sold       |             |              |         433 |             |        2604 |        5102 |           8139 |
|   Unit Price       |             |              |       22.20 |             |       22.20 |       22.20 |                |
|   Amount of Sale   |             |              |    $9,612.60|             |   $57,808.80|  $113,264.40|     $180,685.80|
|--------------------|-------------|--------------|-------------|-------------|-------------|-------------|----------------|
|Product Number  05  |             |              |             |             |             |             |                |
|Basketball Rim/Board|             |              |             |             |             |             |                |
|   Units Sold       |             |         9900 |        2120 |          11 |        2700 |             |          14731 |
|   Unit Price       |             |        88.30 |       88.30 |       88.30 |       88.30 |             |                |
|   Amount of Sale   |             | $874,170.00  | $187,196.00 |     $971.30 | $238,410.00 |             | $1,300,747.30  |
|--------------------|-------------|--------------|-------------|-------------|-------------|-------------|----------------|
|Product Number  06  |             |              |             |             |             |             |                |
|Basketball Uniform  |             |              |             |             |             |             |                |
|   Units Sold       |             |              |             |         990 |         200 |         200 |           1390 |
|   Unit Price       |             |              |             |       42.20 |       42.20 |       42.20 |                |
|   Amount of Sale   |             |              |             |  $41,778.00 |   $8,440.00 |   $8,440.00 |      $58,658.00|
|--------------------|-------------|--------------|-------------|-------------|-------------|-------------|----------------|
|Product Number  10  |             |              |             |             |             |             |                |
|Baseball Cage       |             |              |             |             |             |             |                |
```

Units Sold	45		3450	16	200	3320	7031
Unit Price	890.00		890.00	890.00	890.00	890.00	
Amount of Sale	$40,050.00		$3,070,500.00	$14,240.00	$178,000.00	$2,954,800.00	$6,257,590.00
Product Number 11 Baseball Uniform							
Units Sold	10003		3578		2922	2746	19249
Unit Price	45.70		45.70		45.70	45.70	
Amount of Sale	$457,137.10		$163,514.60		$133,535.40	$125,492.20	$879,679.30
Product Number 12 Softballs							
Units Sold	10	137	2564	13	2200	22	4946
Unit Price	1.40	1.40	1.40	1.40	1.40	1.40	
Amount of Sale	$14.00	$191.80	$3,589.60	$18.20	$3,080.00	$30.80	$6,924.40
Product Number 13 Softball Bats							
Units Sold	3227		3300	1998	5444	99	14068
Unit Price	12.60		12.60	12.60	12.60	12.60	
Amount of Sale	$40,660.20		$41,580.00	$25,174.80	$68,594.40	$1,247.40	$177,256.80
Product Number 14 Softball Gloves							
Units Sold	1155		136	3119	3833	5152	13395
Unit Price	12.00		12.00	12.00	12.00	12.00	
Amount of Sale	$13,860.00		$1,632.00	$37,428.00	$45,996.00	$61,824.00	$160,740.00
Product Number 15 Softball Cage							
Units Sold	997	99	2000		2400		5496
Unit Price	890.00	890.00	890.00		890.00		
Amount of Sale	$887,330.00	$88,110.00	$1,780,000.00		$2,136,000.00		$4,891,440.00
Product Number 16 Softball Uniform							
Units Sold	44		465	16	6165	200	6890
Unit Price	45.70		45.70	45.70	45.70	45.70	
Amount of Sale	$2,010.80		$21,250.50	$731.20	$281,740.50	$9,140.00	$314,873.00
Product Number 25 RacketBalls							
Units Sold	1001	10003	1108	8989	200	522	21823
Unit Price	0.60	0.60	0.60	0.60	0.60	0.60	
Amount of Sale	$600.60	$6,001.80	$664.80	$5,393.40	$120.00	$313.20	$13,093.80
Product Number 26 Racketball Rackets							
Units Sold	21		862	194	944	31	2052
Unit Price	12.70		12.70	12.70	12.70	12.70	
Amount of Sale	$266.70		$10,947.40	$2,463.80	$11,988.80	$393.70	$26,060.40
Total Units Sold	16503	20139	20016	15346	29812	17394 *	119210 *
Total Sales	$1,441,929.40	$968,473.60	$5,290,487.50	$128,198.70	$3,163,713.90	$3,274,945.70 *	$14,267,748.80 *

例: IGYTSALE の販売と歩合

IGYTSALE 出力の次の例では、販売員別の販売実績と歩合を示しています。

```
Day of Report: Tuesday             C O B O L   S P O R T S       11/24/2003    03:12    Page:   1
                                                Sales and Commission Report

  Salesperson: Billy Jim Bob
  Customers:            Number of    Products      Total for       Discount     Discount      Commission
                         Orders      Ordered         Order         (if any)      Amount         Earned
  -------------------- ---------    --------    -------------     --------    -----------    -----------
  Sports Stop                3         10117        $6,161.40      2.25%          $138.63        $746.45
  The Sportsman              1            99         $88,110.00      5.06%        $4,458.36     $10,674.52
  Sports Play                1          9900        $874,170.00      7.59%       $66,349.50    $105,905.69
                           ---------    --------    -------------                 -----------    -----------
  Totals:                    5         20116        $968,441.40                  $70,946.49    $117,326.66
  Salesperson: Willie Al Roz
  Customers:            Number of    Products      Total for       Discount     Discount      Commission
                         Orders      Ordered         Order         (if any)      Amount         Earned
  -------------------- ---------    --------    -------------     --------    -----------    -----------
  Winners Club               4         13998     $1,572,775.90      7.59%       $119,373.69    $157,277.59
  Winning Sports             1          3222        $48,777.20      3.38%         $1,648.66      $4,877.72
  The Sportsman              1          1747        $27,415.50      3.38%           $926.64      $2,741.55
  Play Outdoors              1          2510        $18,579.60      3.38%           $627.99      $1,857.96
                           ---------    --------    -------------                 -----------    -----------
  Totals:                    7         21477     $1,667,548.20                  $122,576.98    $166,754.82
  Salesperson: Art Tung
  Customers:            Number of    Products      Total for       Discount     Discount      Commission
                         Orders      Ordered         Order         (if any)      Amount         Earned
  -------------------- ---------    --------    -------------     --------    -----------    -----------
  Sports Stop                1            23           $32.20      2.25%             $.72           $1.98
  Winners Club               2         16057     $2,274,885.00      7.59%       $172,663.77    $140,424.10
  Gear Up                    1          3022        $107,144.00      7.59%        $8,132.22      $6,613.78
  Sports Club                1            22           $279.40      2.25%            $6.28          $17.24
  Sports Fans Shop           1          1044         $20,447.30      3.38%          $691.11      $1,262.17
  L. A. Sports               1          1163        $979,198.10      7.59%       $74,321.13     $60,443.94
                           ---------    --------    -------------                 -----------    -----------
  Totals:                    7         21331     $3,381,986.00                  $255,815.23    $208,763.21
  Salesperson: Chuck Morgan
  Customers:            Number of    Products      Total for       Discount     Discount      Commission
                         Orders      Ordered         Order         (if any)      Amount         Earned
```

```
-------------------- ---------    --------    --------------    --------    -----------    -----------
Sports Play                3          7422      $3,817,245.40      7.59%     $289,728.92    $322,270.94
Sports 4 You               1          3022        $398,335.40      7.59%      $30,233.65     $33,629.46
The Sportsman              1          3022        $285,229.40      7.59%      $21,648.91     $24,080.49
Sports 4 Winners           1          1100         $68,509.40      5.06%       $3,466.57      $5,783.90
Sports Club                1         12027      $1,324,256.10      7.59%     $100,511.03    $111,800.32
                     ---------    --------    --------------                 -----------    -----------
Totals:                    7         26593      $5,893,575.70                 $445,589.08    $497,565.11
Salesperson: Chris Preston
Customers:           Number of    Products       Total for       Discount      Discount      Commission
                      Orders      Ordered         Order         (if any)       Amount         Earned
-------------------- ---------    --------    --------------    --------    -----------    -----------
Playing Ball               1          5535      $1,939,219.10      7.59%     $147,186.72    $103,509.69
Play Sports                1          5675        $225,130.80      7.59%      $17,087.42     $12,016.80
Winners Club               1           631         $14,069.70      2.25%         $316.56        $750.99
The Jock Shop              1          2332         $28,716.60      3.38%         $970.62      $1,532.80
                     ---------    --------    --------------                 -----------    -----------
Totals:                    4         14173      $2,207,136.20                 $165,561.32    $117,810.28
Salesperson: Edyth Phillips
Customers:           Number of    Products       Total for       Discount      Discount      Commission
                      Orders      Ordered         Order         (if any)       Amount         Earned
-------------------- ---------    --------    --------------    --------    -----------    -----------
Sports Play                2          3575         $92,409.90      5.06%       $4,675.94      $3,911.43
Winning Sports             1         11945         $56,651.40      5.06%       $2,866.56      $2,397.88
                     ---------    --------    --------------                 -----------    -----------
Totals:                    3         15520        $149,061.30                   $7,542.50      $6,309.31
  Grand Totals:           33        119210     $14,267,748.80                $1,068,031.60  $1,114,529.39
```

例: IGYTSALE の販売から配送までの応答時間

IGYTSALE 出力の次の例では、米国での販売日から、販売された製品がヨーロッパに向けて配送された日付までの応答時間を示しています。

```
Day of Report: Monday   COBOL SPORTS       11/24/2003    03:12    Page:   1
                     Response Time from USA Sale to European Ship
  Prod    Units    Sale Date/Time(PST)     Ship Date     Ship  Response Time
  Code    Sold     YYYYMMDD  HHMMSS        YYYYMMDD      Day     Days
  ----    -----    --------  ------        --------      ----  -------------
  25      9999     19900226  010101        19900228      WED        .95
  15        99     19900310  100530        19900403      TUE      23.57
  05      9900     19900418  222409        19900419      THU        .06
  25         4     19900523  151010        19900623      SAT      30.36
  04      1100     19901110  003301        19901114      WED       2.97
  12        23     19901114  003205        19901117      SAT       1.97
  14      5111     19900118  101527        19900120      SAT       1.57
  04      5102     19901201  132013        19901203      MON       1.44
  04       300     19901221  191544        19901223      SUN       1.19
  05       500     19901210  211544        19901214      FRI       3.11
  04       100     19901211  000816        19901213      THU        .99
  25       100     19901201  131544        19901203      MON       1.44
  25       100     19901112  073312        19901113      TUE        .68
  14      1111     19901214  012510        19901216      SUN        .94
  26        22     19901110  000034        19901113      TUE       1.99
  12      2000     19901110  154100        19901113      TUE       2.34
  04      1104     19901110  175001        19901113      TUE       2.25
  12       114     19901229  115522        19901230      SUN        .50
  15      2000     19901110  190113        19901114      WED       3.20
  10      1440     19901112  001500        19901115      THU       1.98
  25      1104     19901118  120101        19901119      MON        .49
  25         4     19901118  110030        19901119      MON        .54
  12       144     19901114  010510        19901119      MON       3.95
  14       112     19901119  010101        19901122      THU       1.95
  26       321     19901117  173945        19901119      MON       1.26
  13      1221     19901101  135133        19901102      FRI        .42
  10        22     19901029  210000        19901030      TUE        .12
  14        35     19901130  160500        19901201      SAT        .32
  11      9005     19901211  050505        19901212      WED        .78
  06       990     19900511  214409        19900515      TUE       3.09
  13      1998     19900712  150100        19900716      MON       3.37
  26        31     19901010  185559        19901011      THU        .21
  14        30     19901210  195500        19901212      WED       1.17
```

IGYTSALE の実行準備

IGYTSALE プログラム
(IGYTSALE、IGYTCRC、IGYTPRC、IGYTSRC、IGYTABLE、および IGYTRANA)
によって必要とされるすべてのファイルは、IGY.V4R1M0.SIGYSAMP データ・セットのプロダクト・インストール・テープにあります。

データ・セット名およびプロシージャー名は、インストール時に変更できます。これらの名前の妥当性について、システム・プログラマーにお問い合わせください。

IGYTSALE のソース・ファイルの中の、CBL ステートメントの以下のオプションは変更しないでください。

- LIB
- NONUMBER
- SEQUENCE
- NONUMBER
- QUOTE

これらのオプションを適用すると、プログラムは診断メッセージを発行しなくなる可能性があります。ソース・ファイル内のシーケンス番号ストリングを使用すれば、使用されている言語エレメントを見つけることができます。

IGYTSALE を実行すると、次のメッセージが SYSOUT データ・セットに印刷されます。

```
Program IGYTSALE Begins
There were 00041 records processed in this program
Program IGYTSALE Normal End
```

関連概念
886 ページの『IGYTSALE: ネストされたプログラム・アプリケーション』

関連タスク 『IGYTSALE の実行』

関連参照
887 ページの『IGYTSALE の入力データ』
889 ページの『IGYTSALE によって作成される報告書』
894 ページの『示されている言語エレメントおよび概念』

IGYTSALE の実行

次の JCL を使用して、IGYTSALE プログラムをコンパイル、リンク・エディット、および実行します。プログラムのコンパイルだけ、またはコンパイルとリンク・エディットだけを行う場合は、IGYWCLG カタログ式プロシージャーを変更します。

小文字で示されているフィールド (会計情報) には、システムおよびインストール先固有の情報を挿入しなければなりません。

```
//IGYTSALE JOB (acct-info),'IGYTSALE',MSGLEVEL=(1,1),TIME=(0,29)
//TEST  EXEC IGYWCLG
//COBOL.SYSLIB  DD DSN=IGY.V4R1M0.SIGYSAMP,DISP=SHR
//COBOL.SYSIN   DD DSN=IGY.V4R1M0.SIGYSAMP(IGYTSALE),DISP=SHR
```

```
//GO.SYSOUT     DD SYSOUT=A
//GO.IGYTABLE   DD DSN=IGY.V4R1M0.SIGYSAMP(IGYTABLE),DISP=SHR
//GO.IGYTRANS   DD DSN=IGY.V4R1M0.SIGYSAMP(IGYTRANA),DISP=SHR
//GO.IGYPRINT   DD SYSOUT=A,DCB=BLKSIZE=133
//GO.IGYPRT2    DD SYSOUT=A,DCB=BLKSIZE=133
//
```

示されている言語エレメントおよび概念

サンプル・プログラムは、COBOL 言語のエレメントおよび概念を示しています。

サンプル・プログラムに適正な言語エレメントを見つけるためには、シーケンス・ストリングの欄でプログラムの省略形を探します。

サンプル・プログラム	省略形
IGYTCARA	IA
IGYTCARB	IB
IGYTSALE	IS

次の表に、サンプル・プログラムで例示されている言語エレメントとプログラミングの概念を示します。言語エレメントまたは概念を説明し、シーケンス・ストリングを示しています。シーケンス・ストリングとは、ソース・ファイルのシーケンス・フィールドに現れる特殊な文字ストリングのことを言います。このストリングを検索引数として使用して、リスト内のエレメントを見つけることができます。

言語エレメントまたは概念	シーケンス・ストリング
ACCEPT . . . FROM DAY-OF-WEEK	IS0900
ACCEPT . . . FROM DATE	IS0901
ACCEPT . . . FROM TIME	IS0902
ADD . . . TO	IS4550
AFTER ADVANCING	IS2700
AFTER PAGE	IS2600
ALL	IS4200
ASSIGN	IS1101
AUTHOR	IA0040
CALL	IS0800
呼び出し可能サービス (言語環境プログラム): 1. CEEDATM: 日付および時刻出力の形式設定 2. CEEDCOD: フィードバック・コード検査 3. CEEGMTO: 地方時からの UTC オフセット 4. CEELOCT: ローカル日付および時刻 5. CEESECS: タイム・スタンプの秒への変換	 1. IS0875, IS2575 2. IS0905 3. IS0904 4. IS0850 5. IS2350, IS2550
ファイルの CLOSE	IS1900
交換可能なコンマ、セミコロン、およびスペース	IS3500, IS3600
ネストされたプログラムの COMMON ステートメント	IS4600
複合 OCCURS DEPENDING ON	IS0700, IS3700

言語エレメントまたは概念	シーケンス・ストリング
COMPUTE	IS4501
COMPUTE ROUNDED	IS4500
CONFIGURATION SECTION	IA0970
CONFIGURATION SECTION (オプション)	IS0200
CONTINUE ステートメント	IA5310, IA5380
COPY ステートメント	IS0500
DATA DIVISION (オプション)	IS5100
データ妥当性検査	IA5130-6190
Do-until (PERFORM . . . TEST AFTER)	IA4900-5010, IA7690-7770
Do-while (PERFORM . . . TEST BEFORE)	IS1660
END-ADD	IS2900
END-COMPUTE	IS4510
END-EVALUATE	IA6590, IS2450
END-IF	IS1680
END-MULTIPLY	IS3100
END-PERFORM	IS1700
END PROGRAM	IA9990
END-READ	IS1800
END-SEARCH	IS3400
ENVIRONMENT DIVISION (オプション)	IS0200
エラー処理、プログラムの終了	IA4620, IA5080, IA7800-7980
EVALUATE ステートメント	IA6270-6590
EVALUATE . . . ALSO	IS2400
EXIT PROGRAM は段落内で唯一のステートメントである必要はない	IS2000
指数	IS4500
EXTERNAL 節	IS1200
順次ファイルの FILE-CONTROL 記入項目	IA1190-1300
VSAM 索引付きファイルの FILE-CONTROL 記入項目	IA1070-1180
FILE SECTION (オプション)	IS0200
FILE STATUS コード検査	IA4600-4630, IA4760-4790
FILLER (オプション)	IS0400
フラグ、レベル 88、定義	IA1730-1800, IA2440-2480, IA2710
フラグ、レベル 88、テスト	IA4430, IA5200-5250
FLOATING POINT	IS4400
GLOBAL ステートメント	IS0300
ネストされたプログラムの INITIAL ステートメント	IS2300
INITIALIZE	IS2500
DATA DIVISION 内でのテーブルの初期化	IA2920-4260
インライン PERFORM ステートメント	IA4410-4520
I-O-CONTROL 段落 (オプション)	IS0200

言語エレメントまたは概念	シーケンス・ストリング
INPUT-OUTPUT SECTION (オプション)	IS0200
組み込み関数	
1. CURRENT-DATE	1. IA9005
2. MAX	2. IA9235
3. MEAN	3. IA9215
4. MEDIAN	4. IA9220
5. MIN	5. IA9240
6. STANDARD-DEVIATION	6. IA9230
7. UPPER-CASE	7. IA9015
8. VARIANCE	8. IA9225
9. WHEN-COMPILED	9. IA9000
IS (すべての節でオプション)	IS0700
LABEL RECORDS (オプション)	IS1150
LINKAGE SECTION	IS4900
指標と添え字の混合	IS3500
簡略名	IA1000
MOVE	IS0903
MOVE CORRESPONDING ステートメント	IA4810, IA4830
MULTIPLY . . . GIVING	IS3000
END-IF を使用した、ネストされた IF ステートメント	IA5460-5830
ネストされたプログラム	IS1000
NEXT SENTENCE	IS4300
NOT AT END	IS1600
NULL	IS4800
OBJECT-COMPUTER (オプション)	IS0200
OCCURS DEPENDING ON	IS0710
ODO は項目を受け取るのに最大長を使用する	IS1550
OPEN EXTEND	IB2210
OPEN INPUT	IS1400
OPEN OUTPUT	IS1500
ORGANIZATION (オプション)	IS1100
ページ替え	IA7180-7210
簡略化された条件の中の括弧	IS4850
PERFORM . . . WITH TEST AFTER (Do-until)	IA4900-5010, IA7690-7770
PERFORM . . . WITH TEST BEFORE (Do-while)	IS1660
PERFORM . . . UNTIL	IS5000
PERFORM . . . VARYING ステートメント	IA7690-7770
POINTER 関数	IS4700
印刷ファイル FD 記入項目	IA1570-1620
印刷報告書	IA7100-7360
PROCEDURE DIVISION . . . USING	IB1320-IB1650

言語エレメントまたは概念	シーケンス・ストリング
PROGRAM-ID (30 文字まで許可される)	IS0120
READ . . . INTO . . . AT END	IS1550
REDEFINES ステートメント	IA1940, IA2060, IA2890, IA3320
参照変更	IS2425
関係演算子 <= (より小さい、または等しい)	IS4400
関係演算子 >= (より大きい、または等しい)	IS2425
相対添え字付け	IS4000
REPLACE	IS4100
SEARCH ステートメント	IS3300
SELECT	IS1100
順序番号には任意の文字を入れることができる	IA, IB, IS
順次ファイル処理	IA4480-4510, IA4840-4870
PERFORM を使用した、順次テーブル探索	IA7690-7770
SEARCH を使用した、順次テーブル探索	IA5270-5320, IA5340-5390
SET INDEX	IS3200
SET . . . TO TRUE ステートメント	IA4390, IA4500, IA4860, IA4980
SOURCE-COMPUTER (オプション)	IS0200
SPECIAL-NAMES 段落 (オプション)	IS0200
STRING ステートメント	IA6950, IA7050
小文字のサポート	IS0100
TALLY	IS1650
ネストされたプログラムの TITLE ステートメント	IS0100
通勤者のレコードの更新	IA6200-6610
トランザクション作業値スペースの更新	IB0790-IB1000
USAGE BINARY	IS1300
USAGE PACKED-DECIMAL	IS1301
妥当性検査エレメント	IB0810, IB0860, IB1000
OCCURS が指定された VALUE	IS0600
VALUE SPACE (S)	IS0601
VALUE ZERO (S) (ES)	IS0600
可変長テーブルの制御変数	IA5100
可変長テーブルの定義	IA2090-2210
可変長テーブルのロード	IA4840-4990
VSAM 索引付きファイルのキー定義	IA1170
VSAM 戻りコードの表示	IA7800-7900
WORKING-STORAGE SECTION	IS0250

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。

〒106-8711
東京都港区六本木 3-2-12
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 Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
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。

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。

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。

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. _年を入れる_. All rights reserved.

この情報をソフトコピーでご覧になっている場合は、写真やカラーの図表は表示されない場合があります。

商標

以下は、International Business Machines Corporation の米国およびその他の国における商標です。

IBM
IBM logo
ibm.com
AIX
BookManager
CICS
COBOL/370
DB2
DFSMS
DFSORT
IMS
IMS/ESA
Language Environment
MVS
MVS/ESA
MVS/XA
OS/390
RACF
Rational
REXX
System z
VTAM
WebSphere
z/Architecture
z/OS
zSeries

Intel は、Intel Corporation の米国およびその他の国における商標です。

Java およびすべての Java 関連の商標およびロゴは Sun Microsystems, Inc.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。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。

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。

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)
- 「*ANSI X3.172-2002, American National Standard Dictionary for Information Systems* 」

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。

この用語集には、Sun Microsystems, Inc が Java および J2EE の用語集用に作成した定義が含まれています。Sun による定義には、その旨が示されています。

ア

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。

遊びバイト (slack bytes)
一部の数値項目の位置合わせが正しく行われるように、データ項目相互間またはレコード相互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。コンパイラーによって挿入される場合もあれば、プログラマーが挿入する必要がある場合もある。SYNCHRONIZED 節は、正しいアライメントが必要な場合に遊びバイトを挿入するようにコンパイラーに指示する。レコード間遊びバイトは、プログラマーが挿入する。

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。

異常終了 (abend)
プログラムの異常終了。

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。

インスタンス・データ (instance data)
オブジェクトの状態を定義するデータ。クラスによって導入されるインスタンス・データは、クラス定義の OBJECT 段落の DATA DIVISION の WORKING-STORAGE SECTION に定義される。オブジェクトの状態には、クラスが導入した、現行クラスによって継承されているインスタンス変数の状態も含まれる。インスタンス・データの個々のコピーは、各オブジェクト・インスタンスごとに作成される。

隠蔽 (hide)
サブクラスのファクトリーまたは静的メソッド (親クラスから継承された) を再定義すること。

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。

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、07 は 2007 と解釈される。*世紀ウィンドウ (century window)* も参照。*拡張西暦年 (expanded year)* と比較。

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。

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。

英字 (* alphabetic character)
文字または空白文字。

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。

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。

エンクレーブ (enclave)
言語環境プログラムのもとで実行される場合、エンクレーブは実行単位に類似している。エンクレーブは、LINK および C の system() 関数の使用によって、他のエンクレーブを作成できる。

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。

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。

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。

オブジェクト (object)
状態 (そのデータ値) および演算 (そのメソッド) を持つエンティティー。オブジェクトは状態と動作をカプセル化する手段である。クラス内の各オブジェクトは、そのクラスの 1 つのインスタンスであると言われる。

オブジェクト・インスタンス (object instance)
オブジェクト (object) を参照。

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。

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。

オブジェクト・デック
リンケージ・エディターへの入力として適切なオブジェクト・プログラムの部分。「オブジェクト・モジュール (*object module*)」および「テキスト・デック (*text deck*)」と同義。

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・プログラムまたはクラス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・モジュール (object module)
オブジェクト・デック (object deck) または *テキスト・デック (text deck)* と同義。

オブジェクト参照 (object reference)
クラスのインスタンスを識別する値。クラスが指定されなかった場合、オブジェクト参照は一般的なものとなり、任意のクラスのインスタンスに適用できる。

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。

オブジェクト指向プログラミング (object-oriented programming)
カプセル化および継承の概念に基づいたプログラミング・アプローチ。プロシージャー型プログラミング技法とは異なり、オブジェクト指向プログラミングでは、何かが達成される方法ではなく、問題を含むデータ・オブジェクトとその操作方法に重点を置く。

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。

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。

カ

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のメモリーを、Java ランタイム・システムが自動的に解放すること。

下位終了 (* low-order end)
文字ストリングの右端の文字。

階層ファイル・システム (hierarchical file system)
階層構造で編成されたファイルとディレクトリーの集合であり、z/OS UNIX を使用してアクセスできる。

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。

外部コード・ページ (external code page)
XML 文書では、CODEPAGE コンパイラー・オプションによって指定された値。

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。

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・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。

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。

拡張 (extensions)
COBOL 85 標準で記述されるもの以外で、IBM コンパイラーでサポートされる COBOL 構文とセマンティクス。

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。

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。

型式化オブジェクト参照 (typed object reference)
指定されたクラスまたはそのサブクラスのオブジェクトだけを参照できるデータ名。

カタログ式プロシージャー (cataloged procedure)
プロシージャー・ライブラリー (SYS1.PROCLIB) と呼ばれる区分データ・セットに置かれた一連のジョブ制御ステートメント。カタログ式プロシージャーを使用すると、JCL をコーディングする時間を節約して、エラーを減らすことができる。

カプセル化 (encapsulation)
オブジェクト指向プログラミングでは、オブジェクトの固有の詳細を隠すのに使用される技法。オブジェクトは、基礎構造を露

出しなくても、データの照会と操作を行うインターフェースを提供する。「*情報隠蔽 (information hiding)*」と同義。

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)。

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。

環境節 (* environment clause)
ENVIRONMENT DIVISION 記入項目の一部として現れる節。

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。

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。

関係 (* relation)
関係演算子 (relational operator) または *比較条件 (relation condition)* を参照。

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)

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。

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。

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。

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。

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。

キーワード (* keyword)
予約語または関数名で、その語の現れる形式がソース・プログラムの中で使用されるときには必須である。

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 (==)。

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。

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。

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。

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。

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。

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。

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。

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。

キロバイト (KB) (kilobyte(KB))
1 キロバイトは 1024 バイトに相当する。

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。

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。

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。

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

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。

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。

国別文字位置 (national character position)
文字位置 (character position) を参照。

組み込み関数 (built-in function)
組み込み関数 (intrinsic function) を参照。

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。

クライアント (client)
オブジェクト指向プログラミングにおいて、クラス内の 1 つ以上のメソッドからサービスを要求するプログラムまたはメソッド。

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・オブジェクト (class object)
クラスを表す実行時オブジェクト。

クラス階層 (class hierarchy)
オブジェクト・クラス間の関係を示すツリーのような構造。最上部に 1 つのクラス

が置かれ、その下に 1 つ以上のクラスの層が置かれる。「*継承階層 (inheritance hierarchy)*」と同義。

クラス識別記入項目 (* class identification entry)
IDENTIFICATION DIVISION の CLASS-ID 段落内の記入項目であり、クラス名を指定する節と、選択した属性をクラス定義に割り当てる節を含む。

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。

クラス定義 (* class definition)
クラスを定義する COBOL ソース単位。

クラス名 (オブジェクト指向) (class-name (object-oriented))
オブジェクト指向 COBOL クラス定義の名前。

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。

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。

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。

グローバル参照 (global reference)
メソッドの有効範囲外にあるオブジェクトの参照。

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。

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。

継承 (inheritance)
クラスのインプリメンテーションを、別のクラスを基にして使用するメカニズム。定義により、継承するクラスは継承されるクラスに準拠する。Enterprise COBOL は *多重継承* をサポートしない。サブクラスは、必ず 1 つの即時スーパークラスを有する。

継承階層 (inheritance hierarchy)
クラス階層 (class hierarchy) を参照。

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。

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。

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。

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。

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。

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、アプリケーション開発者は、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。

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。

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。*コード化文字セット (coded character set)*。

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。

高位終了 (* high-order end)
文字ストリングの左端の文字。

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。

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。

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。

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。

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。

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。

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。

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。

コメント記入項目 (* comment-entry)
IDENTIFICATION DIVISION 内の記入項目であり、コンピューターの文字セットから任意の文字を組み合わせることができる。

コメント行 (* comment line)
行の標識区域ではアスタリスク (*)、およびその行の区域 A および B ではコンピューターの文字セットの任意の文字で表されるソース・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。

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。

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。

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。

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。

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、Sun Microsystems, Inc. のコンポーネント作成用アーキテクチャーである。

サ

再帰 (recursion)
それ自体を呼び出すプログラム、または、自分で呼び出したプログラム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)。

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。

再使用可能環境 (reusable environment)
事前初期設定用の古い COBOL インターフェース (関数 ILBOSTP0 と IGZERRE、および RTEREUS ランタイム・オプション)、または言語環境プログラム・インターフェース CEEPIPI のいずれかを使用して、アセンブラー・プログラムをメインプログラムとして設定するときに、再使用可能環境が作成されます。

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。

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。

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。

サブクラス (* subclass)
別のクラスから継承するクラス。継承関係にある 2 つのクラスをまとめて考える場合、継承する側、つまり継承先のクラスをサブクラスといい、継承される側、つまり継承元のクラスをスーパークラスという。

サブプログラム (* subprogram)
呼び出し先プログラム (called program) を参照。

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。

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

算術式 (* arithmetic expression)
数字基本項目の ID、数値リテラル、そのような ID とリテラルを算術演算子で区切ったもの、2 つの算術式を算術演算子で区切ったもの、または算術式を括弧で囲んだもの。

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。

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。

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。

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。

式 (* expression)
算術式または条件式。

シグニチャー (signature)
(1) ある操作とそのパラメーターの名前。
(2) あるメソッドの名前とその仮パラメーターの数と型。

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。

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。

事前初期設定 (preinitialization)
プログラム (特に非 COBOL プログラム) からの複数の呼び出しの準備としての COBOL ランタイム環境の初期設定。この環境は、明示的に終了されるまで終了されない。

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。

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。

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 または INVOKE 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。

実際の小数点 (* actual decimal point)
データ項目内の 10 進小数点位置の、10 進小数点文字のピリオド (.) またはコンマ (,) を使用した、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
- Unicode 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

失敗した実行 (* 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。

指定変更 (override)
サブクラスのインスタンス・メソッド (親クラスから継承された) を再定義すること。

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。

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。

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。

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。

従属領域 (dependent region)
IMS において、メッセージ・ドリブン・プログラム、バッチ・プログラム、またはオンライン・ユーティリティーを含む MVS 仮想記憶領域

終了クラス・マーカー (end class 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 クラス定義の終わりを示す。クラス終了マーカーは次のとおり。

```
END CLASS class-name.
```

終了メソッド・マーカー (end method 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 メソッド定義の終わりを示す。メソッド終了マーカーは次のとおり。

```
END METHOD method-name.
```

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。

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。

言語環境プログラム準拠 (Language Environment-conforming)
言語環境プログラムの規則に準拠したオブジェクト・コードを生成するコンパイラー製品 (Enterprise COBOL、COBOL for OS/390 & VM、COBOL for MVS & VM、C/C++ for MVS & VM、PL/I for MVS & VM など) の特性。

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。

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。

条件 (* condition)
真理値を判別できる、実行時のプログラムの状況。条件がこれらの言語仕様または一般形式の「条件」 (*condition-1*, *condition-2*,...) に関連して現れる場合は、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 (condition)
言語環境プログラムによって使用可能にされる、あるいは認識される例外。したがって、ユーザー条件処理ルーチンと言語条件処理ルーチンの活動化に適している。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。

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。

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。

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。

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。

ジョブ制御言語 (job control language (JCL))
ジョブをオペレーティング・システムに識別させ、ジョブの要件を記述するために使われる制御言語。

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。

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。

スーパークラス (* superclass)
別のクラスによって継承されるクラス。サブクラス *(subclass)* も参照。

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。

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。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。

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。

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)。

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。
- 言語環境プログラム呼び出し可能サービスについては、CEESCEN で世紀ウィンドウを指定する。

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。

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。

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。

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。

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。

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。

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。プログラムまたはクラス定義。*コンパイル単位* とも呼ばれる。

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。

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)。

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。

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。

相対ファイル (* relative file)
相対編成のファイル。

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。

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。

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。

タ

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。

ダイナミック・リンク・ライブラリー (DLL)
リンク時ではなく、ロード時または実行時にプログラムにバインドされる実行可能コードおよびデータが入ったファイル。複数のアプリケーションが DLL 内のコードおよびデータを同時に共用することができる。DLL はプログラムの実行可能ファイルの一部ではないが、実行可能ファイルを正しく実行するためには必要となる可能性がある。

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。

対話式システム生産性向上機能 (ISPF)
TSO または VM ユーザーに対してメニュー方式のインターフェースを提供する IBM ソフトウェア・プロダクト。ISPF には、ライブラリー・ユーティリティー、強力なエディター、およびダイアログ管理が組み込まれている。

多重定義 (overload)
同じクラスで使用可能な別のメソッドと同一の名前を使い (ただし、異なるシグニチャーを使用して)、メソッドを定義すること。シグニチャー *(signature)* も参照。

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。

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。

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。

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)
CLASS-ID. (Class IDENTIFICATION DIVISION)
METHOD-ID. (Method 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 or Class
   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。

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。

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。

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。

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。

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。

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。

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。

次の実行可能ステートメント (* 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。

次の実行可能な文 (* 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。

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。

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。

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。

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。

テキスト・デック (text deck)
オブジェクト・デック (object deck) または *オブジェクト・モジュール (object module)* と同義。

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。

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。

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。

デバッグ行 (* debugging line)
行の標識区域に文字 D がある行のこと。

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。

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。

動的 CALL (dynamic CALL)
DYNAM オプションおよび NODLL 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 NODLL 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 *identifier* ステートメント。

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。

動的ストレージ域 (dynamic storage area (DSA))
動的に獲得されるストレージであり、レジスター保管域、および動的ストレージ割り振りに使用可能な区域 (プログラム変数など) から構成される。 DSA は、プログラムまたは関数が呼び出されるときに割り振られ、呼び出しインスタンスの継続時間の間持続します。 DSA は通常、言語環境プログラムによって管理されるスタック・セグメント内に割り振られる。

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 (小数点)
;	セミコロン
.	ピリオド (小数点、終止符)
″	引用符
(	左括弧
)	右括弧
>	より大記号
<	より小記号
:	コロン

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。

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。

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。

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、ファイルまたはデータ・セットのラベル。(2) 「*ファイル終わりラベル (end-of-file label)*」の同義語。

ナ

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。

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。

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。

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。

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。

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。

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。

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。

ネイティブ・メソッド (native method)
COBOL などの別のプログラム言語で記述されたインプリメンテーションを備える Java メソッド。

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。

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。

ハ

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。

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。(Sun)

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。

ハイパースペース (hiperspace)
z/OS 環境で、プログラムがバッファーとして使用できる最大 2 GB までの連続する仮想記憶アドレス範囲。

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。

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。

パッケージ (package)
関連する Java クラスの集まり。個々に、または全体としてインポートすることができる。

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。

パラメーター (parameter)
(1) 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。(2) メソッド呼び出しの USING 句内のデータ・エレメント。引数によって、呼び出されたメソッドが要求された操作を実行するために使用できる追加情報を与える。

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。

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。

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。

汎用オブジェクト参照 (universal object reference)
どのクラスのオブジェクトでも参照できるデータ名。

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。

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

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 または INVOKE 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または起動されたメソッドに値を渡すのに使用されます。

ビジネス・メソッド (business method)
ビジネス・ロジックまたはアプリケーションのルールをインプリメントする Enterprise Bean のメソッド。 (Sun)

非制限ストレージ (unrestricted storage)
2 GB バーより下のストレージ。16MB 境界より上または下がある。16MB 境界より上では、31 ビット・モードでのみ、アドレス可能。

ビッグ・エンディアン (big-endian)
メインフレームおよび AIX ワークステーションが 2 進データおよび UTF-16 文字を保管する際に使用するデフォルトの形式。この形式では、2 進数データ項目の最下位バイトが最上位のアドレスになり、UTF-16 文字の最下位バイトが最上位のアドレスになる。*リトル・エンディアン (little-endian)* と比較。

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 (*Enterprise COBOL 言語解説書*) を参照してください。

日付フィールド (date field)
次のうちのいずれか。

- 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 (*Enterprise COBOL 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。

非日付 (nondate)
次のうちのいずれか。

- 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

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。

標準 COBOL 85 (Standard COBOL 85)
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*」および「 *ANSI INCITS 23b-1993, Programming Languages - Correction Amendment 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*」および「*ISO/IEC 1989/AMD2:1994, Programming*

languages - Correction and clarification amendment for COBOL」に改訂されました。

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。

ファイル (* file)
論理レコードの集合。

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。

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。

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。

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。

ファクトリー・データ (factory data)
いったんクラスに割り振られ、クラスのすべてのインスタンスに共用されるデータ。ファクトリー・データは、クラス定義の FACTORY 段落の DATA DIVISION の WORKING-STORAGE SECTION 内に宣言される。Java private 静的データと同義。

ファクトリー・メソッド (factory method)
オブジェクト・インスタンスとは無関係に、クラスによってサポートされるメソッド。ファクトリー・メソッドは、クラス定義の FACTORY 段落に宣言される。Java public 静的メソッドと同義。これらは通常オブジェクトの作成をカスタマイズすることに使用される。

フォーマット (* format)
一連のデータの特定の配置。

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。

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。

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。

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。

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。

物理レコード (* physical record)
ブロック (block) を参照。

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。

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.
```

古くなったエレメント (* obsolete element)
標準 COBOL 85 の COBOL 言語エレメントのうち、標準 COBOL 2002 から削除されたもの。

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.
```

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。

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 および言語環境プログラムのプログラムと通信するために使用される。

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。

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。

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。

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。

文書タイプ定義 (document type definition (DTD))
XML 文書のクラスの文法。*XML タイプ定義 (XML type definition)* を参照。

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 文字以上。

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 (;)。

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 (.)。

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。

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。

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。

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ファイル・ラベルまたはデータ・セット・ラベル。(2) 「*ファイル開始ラベル (beginning-of-file label)*」の同義語。

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。

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)

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。

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。

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。

ボリューム切り替え処理手順 (volume switch procedures)
ファイルの終わりに達する前にユニットまたはリールの終わりに達したとき、自動的に実行されるシステム固有の処理手順。

マ

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。

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。

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。

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。

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。

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。

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。

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。

メソッド (method)
オブジェクトによってサポートされる操作の 1 つを定義し、そのオブジェクトに対する INVOKE ステートメントによって実行されるプロシージャー・コード。

メソッド定義 (* method definition)
メソッドを定義する COBOL ソース・コード。

メソッドの起動 (method invocation)
あるオブジェクトから別のオブジェクトへの通信で、受信オブジェクトにメソッドを実行するように要求するもの。

メソッド見出し記入項目 (* method identification entry)
IDENTIFICATION DIVISION の METHOD-ID 段落内の記入項目。この記入項目には、メソッド名を指定する節が入っている。

メソッド名 (method-name)
オブジェクト指向操作の名前。メソッドを起動するのに使用する場合、名前は、英数字リテラル、国別リテラル、カテゴリー英数字データ項目、またはカテゴリー国別データ項目にすることができます。メソッドを定義する METHOD-ID 段落で使用する場合、名前は英数字リテラルまたは国別リテラルにする必要があります。

メッセージ・キュー (message queue)
メッセージがアプリケーション・プログラムによって処理されたり端末に送信されたりする前に、キューに入れられるデータ・セット。

メッセージ処理プログラム (MPP)
トランザクションによって駆動され、オンライン IMS データベースとメッセージ・キューにアクセスできる、IMS アプリケーション・プログラム。

文字 (* character)
言語のそれ以上分割できない基本単位。

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

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。

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。

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。

ヤ

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。

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。

ユニット (unit)
直接アクセスのモジュール。その大きさは IBM によって規定されている。

呼び出し可能サービス (callable services)
言語環境プログラムでは、従来の言語環境プログラム定義の呼び出しインターフェースを使用して COBOL プログラムが呼び出すことができる一連のサービス。言語環境プログラムの規則を共有するすべてのプログラムがこれらのサービスを使用できる。

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。

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。

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。

ラ

ライブラリー・テキスト (* library text)
COBOL ライブラリー内の一連のテキスト・ワード、コメント行、区切りスペース、または区切りの疑似テキスト区切り文字。

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アクセスしたりできるようになる。

ランタイム環境 (runtime environment)
COBOL プログラムが実行される環境。

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。

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。

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。

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 文字を保管する際に使用するデフォルトの形式。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。

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。

リンカー (linker)
z/OS リンケージ・エディターまたは z/OS バインダーのいずれかを指す。

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実行可能ファイルを作成すること。

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。言語環境プログラムでは、プロシージャー、関数、またはサブルーチンのいずれかを指す。

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。

レコード (* record)
論理レコード (logical record) を参照。

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。

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。

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。

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。

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。

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。

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。

ローカル参照 (local reference)
メソッドの有効範囲内にあるオブジェクトの参照。

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。

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。

ワ

ワークステーション (workstation)
エンド・ユーザーが使用する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。

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。

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。

数字

1 バイト文字セット (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。

16MB 境界より上 (above the 16-MB line)
いわゆる 16MB 境界より上で、2 GB バーより下のストレージ。このストレージは、31 ビット・モードでのみアドレス可能である。1980 年代に IBM が MVS/XA™ アーキテクチャーを導入する以前、プログラムの仮想ストレージは 16 MB に限定されていました。 24 ビット・モードでコンパイルしたプログラムは、あたかも架空のストレージ境界線の下に抑えられているように、16MB のスペースしかアドレスすることができない。VS COBOL II 以降、31 ビット・モードでコンパイルしたプログラムは、16MB 境界より上に配置することができる。

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。

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。

A

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。

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

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。

C

CCSID
コード化文字セット ID (coded character set identifier) を参照。

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、以下にリストする文字で構成される。

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
/	斜線 (スラッシュ)
=	等号
$	通貨記号
,	コンマ (小数点)
;	セミコロン
.	ピリオド (小数点、終止符)
″	引用符
(	左括弧
)	右括弧
>	より大記号
<	より小記号
:	コロン

COBOL ワード (* COBOL word)
ワード (word) を参照。

*** 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およびクラス定義を記述する。

CONSOLE
オペレーター・コンソールと関連する COBOL 環境名。

D

DATA DIVISION
COBOL プログラムまたはメソッドの 1

つの部。使用するファイルおよびファイルに含まれるレコード、必要となる内部作業用ストレージ・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を記述する。

DBCS
2 バイト文字セット (double-byte character set (DBCS)) 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。

DBCS 文字 (DBCS character)
IBM の 2 バイト文字セットで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。

DLL *ダイナミック・リンク・ライブラリー (DLL) (dynamic link library (DLL))* を参照。

DLL アプリケーション (DLL application)
インポートしたプログラム、関数、または変数を参照するアプリケーション。

DLL リンケージ (DLL linkage)
DLL および NODYNAM 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。CALL は、別個のモジュール内のエクスポートされた名前に解決されるか、または、別個のモジュールに定義されたメソッドの INVOKE に解決される。

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。

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。

EJB *Enterprise JavaBeans* を参照。

EJB コンテナー (EJB container)
J2EE アーキテクチャーの EJB コンポーネント契約を実装するコンテナー。この契約は、セキュリティー、並行性、ライフ・サイクル管理、トランザクション、デプロイメント、およびその他のサービスを含んでいるエンタープライズ Bean 用のランタイム環境を指定します。EJB コンテナーは、EJB サーバーまたは J2EE サーバーによって提供される。(Sun)

EJB サーバー (EJB server)
EJB コンテナーにサービスを提供するソフトウェア。EJB サーバーは、1 つ以上の EJB コンテナーをホストできる。(Sun)

Enterprise Bean
ビジネス・タスクを実装し、EJB コンテナーに存在するコンポーネント。(Sun)

Enterprise JavaBeans
エンタープライズ・レベルのオブジェクト指向分散アプリケーションの開発と配置のために Sun Microsystems, Inc. が定義したコンポーネント・アーキテクチャー。

ENVIRONMENT DIVISION
COBOL プログラム、クラス定義、またはメソッド定義の 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。

Extensible Markup Language
XML を参照。

F

*** 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。

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。

I

*** 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。

IBM COBOL 拡張部分 (IBM COBOL extension)
COBOL 85 標準で記述されるもの以外で、IBM コンパイラーでサポートされる COBOL 構文とセマンティクス。

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。

IDENTIFICATION DIVISION
COBOL プログラム、クラス定義、またはメソッド定義の 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、またはメソッドを識別する。 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。

IGZCBSO
Enterprise COBOL のブートストラップ・ルーチン。このルーチンは、Enterprise COBOL プログラムが含まれているモジュールとリンク・エディットしなければならない。

*** INPUT-OUTPUT SECTION**
ENVIRONMENT DIVISION のセクションであり、オブジェクト・プログラムまたはメソッド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。

is-a 継承階層におけるクラスおよびサブクラスの特徴を表す関係。あるクラスに対して is-a 関係を持つサブクラスは、そのクラスから継承する。

ISPF *対話式システム生産性向上機能 (ISPF) (Interactive System Productivity Facility (ISPF))* を参照。

J

J2EE *Java 2 Platform, Enterprise Edition (J2EE)* を参照。

Java 2 Platform, Enterprise Edition (J2EE)
Sun Microsystems, Inc. が定義する、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の環境。J2EE プラットフォームは、マルチスレッドの 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、一群のサービス、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る。(Sun)

Java Database Connectivity (JDBC)
Java プログラムのデータベースへのアクセスを可能にする API を定義する、Sun Microsystems の仕様。

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。

Java バッチ処理プログラム (JBP)
オンライン・データベースおよび出力メッセージ・キューにアクセスできる IMS バッチ処理プログラム。 JBP はオンラインで動作するが、バッチ環境のプログラムと同じように、JCL または TSO セッションで起動する。

Java バッチ処理領域 (Java batch-processing region)
Java バッチ処理プログラムだけがスケジュールされる IMS 従属領域。

Java メッセージ処理プログラム (JMP)
トランザクションによって駆動され、オンライン IMS データベースとメッセージ・キューにアクセスできる、IMS Java アプリケーション・プログラム。

Java メッセージ処理領域 (Java message-processing region)
Java メッセージ処理プログラムだけがスケジュールされる IMS 従属領域。

JavaBeans
移植可能で、プラットフォームに依存しない、再使用可能なコンポーネント・モデル。 (Sun)

JBP *Java バッチ処理プログラム (JBP) (Java batch-processing program (JBP))* を参照。

JDBC *Java Database Connectivity (JDBC)* を参照。

JMP *Java メッセージ処理プログラム (JMP) (Java message-processing program (JMP))* を参照。

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。

L

*** 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。

LINKAGE SECTION
呼び出し先のプログラムまたはメソッドの DATA DIVISION 内のセクションであり、呼び出し側プログラムまたはメソッド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またはメソッドおよび呼び出し先プログラムまたはメソッドの両方から参照できる。

*** 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。

M

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。

MPP *メッセージ処理プログラム (MPP) (message-processing program (MPP))* を参照。

O

*** 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。

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                   PICS9.
    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。

P

Persistent Reusable JVM
トランザクション間で JVM をリセットすることによりトランザクション処理用にシリアルに再利用できる JVM。リセット・フェーズでは、JVM が既知の初期状態に復元される。

private
ファクトリー・データまたはインスタンス・データに適用されるため、そのデータを定義するクラスのメソッドだけがアクセス可能である。

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。

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。

Q

QSAM (待機順次アクセス方式)(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。

S

SBCS *1 バイト文字セット (SBCS) (single-byte character set (SBCS))* を参照。

Session Bean
EJB において、クライアントによって作成され、通常は 1 つのクライアント/サーバー・セッションの期間だけ存在する Enterprise Bean。 (Sun)

*** 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。

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。

U

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。Enterprise COBOL では、国別データ・タイプの表記としてビッグ・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。

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 Enterprise COBOL でリソースを一意に指す文字のシーケンス、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。.

URI *URI* を参照。

V

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。

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。

*** WORKING-STORAGE SECTION**
独立項目または作業用ストレージ・レコードあるいはその両方から構成される、作業用ストレージ・データ項目を記述する DATA DIVISION。

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 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 タイプ定義 (XML type definition)
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。

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 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。

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。

資料名リスト

Enterprise COBOL for z/OS

コンパイラーおよびランタイム 移行ガイド、SC88-4746

カスタマイズ・ガイド、SC88-4743

言語解説書、SC88-4745

Licensed Program Specifications, GI11-7871

プログラミング・ガイド、SC88-4744

ソフトコピー資料

次のコレクション・キットには、Enterprise COBOL およびその他の製品資料が含まれます。

z/OS Software Products Collection, SK3T-4270

z/OS and Software Products DVD Collection, SK3T-4271

サポート

Performance Tuning, www.ibm.com/support/docview.wss?uid=swg27001475

Enterprise COBOL for z/OS のご使用の際に問題がある場合は、サイト: www.ibm.com/software/awdtools/cobol/zos/support/ を参照してください。そこでは最新のサポート情報が提供されています。

関連資料

CICS Transaction Server for z/OS

アプリケーション・プログラミング・ガイド、SC88-4370

アプリケーション・プログラミング・リファレンス、SC88-4371

Customization Guide, SC34-6814

External Interfaces Guide, SC34-6830

z/OS XL C/C++

プログラミング・ガイド、SC88-8849

ランタイム・ライブラリー・リファレンス、SA88-8515

DB2 for z/OS

Application Programming and SQL Guide, SC18-9841

コマンド解説書、SC88-4351

SQL Reference, SC18-9854

Debug Tool

リファレンスおよびメッセージ, GC88-4748

ユーザーズ・ガイド, SC88-4747

z/OS DFSMS

カタログのためのアクセス方式サービス・プログラム、SC88-9109

Checkpoint/Restart, SC26-7401

Macro Instructions for Data Sets, SC26-7408

データ・セットの使用法, SC88-9114

ユーティリティー、SC88-8979

DFSORT

アプリケーション・プログラミング・ガイド、SD88-6331

インストールおよびカスタマイズ、SD88-6332

IMS

アプリケーション・プログラミング: データベース・マネージャー、SD88-6547

アプリケーション・プログラミング: 設計の手引き、SD88-6548

アプリケーション・プログラミング: EXEC DLI コマンド (CICS および IMS)、SD88-6549

アプリケーション・プログラミング: トランザクション・マネージャー、SD88-6550

Connect 手引きおよび解説書、SD88-6568

Java 手引きおよび解説書、SD88-6557

z/OS ISPF

ダイアログ開発者 ガイドとリファレンス、SC88-8964

ユーザーズ・ガイド 第 1 巻、SC88-8965

ユーザーズ・ガイド 第 2 巻、SC88-8966

z/OS 言語環境プログラム

概念、SA88-8555

カスタマイズ、SA88-8552

デバッグのガイド、GA88-8548

プログラミング・ガイド、SA88-8549

プログラミング・リファレンス、SA88-8550

ランタイム・メッセージ、SA88-8554

ランタイム マイグレーション・ガイド、GA88-8553

ILC(言語間通信) アプリケーションの作成、SA88-8551

z/OS MVS

JCL 解説書、SA88-8569

JCL ユーザーズ・ガイド、SA88-8570

プログラム管理: ユーザーズ・ガイドおよび解説書、SA88-8688

システム・コマンド、SA88-8593

z/OS TSO/E

コマンド解説書、SA88-8628

入門、SA88-8632

ユーザーズ・ガイド、SA88-8638

z/OS UNIX システム・サービス

コマンド解説書、SA88-8641

プログラミング: アセンブラー呼び出し可能サービス 解説書、SA88-8642

ユーザーズ・ガイド、SA88-8640

z/Architecture(R)

z/Architecture 解説書、SA88-8773

ソフトコピー資料 (z/OS 版)

次のコレクション・キットには、z/OS および関連製品の文献が含まれています。

z/OS CD Collection Kit、SK3T-4269

Unicode および文字表現

Unicode、www.unicode.org/

Character Data Representation Architecture: Reference and Registry, SC09-2190

z/OS Support for Unicode: Unicode サービス の使用、SA88-8813

Java

The Java Language Specification, Second Edition (Gosling 他著)、 java.sun.com/docs/books/jls/second_edition/html/j.title.doc.html

The Java Native Interface、java.sun.com/j2se/1.3/docs/guide/jni/index.html

The Java 2 Enterprise Edition Developer's Guide, java.sun.com/j2ee/sdk_1.2.1/techdocs/guides/ejb/html/DevGuideTOC.html

Java 2 on z/OS, www.ibm.com/servers/eserver/zseries/software/java/

Persistent Reusable Java Virtual Machine User's Guide, SC34-6201

WebSphere Application Server for z/OS

Applications, SA22-7959

XML

Extensible Markup Language (XML), www.w3.org/XML/

Namespaces in XML 1.0, www.w3.org/TR/REC-xml-names/

Namespaces in XML 1.1, www.w3.org/TR/xml-names11/

XML specification, www.w3.org/TR/REC-xml/

z/OS XML System Services User's Guide and Reference, SA23-1350

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

H

I

J

K

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: 5655-S71

Printed in Japan

SC88-4744-00

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12