Net Express

システムリファレンス

Micro Focus NetExpress™

システムリファレンス

Micro Focus®

第 3 版

1998 年 10 月

Micro Focus は、このマニュアルの内容が公正かつ正確であるよう万全を期しておりますが、このマニュアルの内容は予告なしに随時変更されることがあります。

このマニュアルに述べられているソフトウェアはライセンスに基づいて提供され、その使用および複写は、ライセンス契約に基づいてのみ許可されます。特に、Micro Focus 社製品のいかなる用途への適合性も明示的に本契約から除外されており、Micro Focus はいかなる必然的損害に対しても一切責任を負いません。

Micro Focus® は登録商標です。Form Designer™ 、Micro Focus COBOL™ 、および NetExpress™ は Micro Focus Limited の商標です。

Microsoft® 、Windows® および Windows for Workgroups® は Microsoft Corporation の登録商標です。

Visual Basic™ および Windows NT™ は、Micorosft Corporation の商標です。

IBM® は、International Business Machines Corporation の登録商標です。

UNIX® は、X/Open Company Limited の登録商標です。

序文

本書、 ***システムリファレンス*** は、Micro Focus NetExpress 製品に添付のマニュアルのうちの一部です。

表記規則

本書では、次の書体や規則を使用します。

- 入力するテキストは、次のように表示します。

  `cat` *`script_name`* `| more`

  斜体テキストはコマンドの一部として入力する変数を表します。

- コマンド行やコード例でオプション入力するテキストは、角かっこ ([]) で囲みます。次の式では、オプションの単語 NOT を入力しない場合、*`column_name`* は `pattern_value` に設定されます。一方、NOT を入力した場合、*`column_name`* は *`pattern_value`* 以外に設定されます。

  `column_name` `[NOT] LIKE` *`pattern_value`*

- 特定のモデルやオペレーティング システムだけに適用する項や段落については、段落のすぐ前に太字斜体の項目名が記載されています。例えば、次のようになります。

  UNIX

  この段落は UNIX システムだけに適用されます。

目次

第1章： 概要

本書、システムリファレンスには NetExpressを使用するすべてのプログラマにとって必要な参照情報が述べてあります。製品に含まれる多くのコンポーネントの操作方法を解説するものであり、とくに、コンパイラ指令、システム提供サブルーチンライブラリの仕様、ならびに関連する必ずしも必須ではない情報も述べられています。

第2章： 統合型プリプロセッサ

2.1 概　説 - 統合化プリプロセッサインタフェース

言語プリプロセッサ（「プレコンパイラ」としても知られています）は、COBOLプログラム内のCOBOL標準以外のコードまたはCOBOL以外のコードをコンパイラが処理できる形式に変換するために使用されます。

非統合型プリプロセッサは、原始ファイルを入力としてそれを読み込みんで解析し、修正原始ファイルを作成します。このファイルは、COBOLコンパイラに入力として渡されます。非統合型プリプロセッサには次のような欠点があります。

- プログラムのコンパイル時に処理が2段階になります。
- コンパイラとエディタが統合化されていません。たとえば、コーディング中にエラーが発生した場合に、エディタがエラーのある原始行を強調表示するようなことはありえません。
- リスト出力とデバッグツールでは、元の原始プログラムではなく書き換えられた原始プログラムを表示します。

これらの欠点により、開発サイクルが長くなることがあります。

統合化プリプロセッサインタフェースにはこのような欠点はありません。それは、プリプロセッサが元の原始プログラムと書き換えられた原始プログラムとの関連性を記録しているからです。コンパイラは実際には書き換えられたCOBOLプログラムを処理しませんが、書き換えられる前のCOBOLプログラムを表示します。

統合化プリプロセッサインタフェースを使用すると、プリプロセッサは原始ファイルを読み込んで修正された原始行をコンパイラに渡します。これにより、インタフェースは完全に汎用化されます。ただし、このアプローチにも欠点があります。それは、プリプロセッサ自身が継続などのCOBOLの構成要素、COPYファイルの拡張、REPLACEおよびREPLACINGの結果を扱わなければならないということです。

Micro Focusは、CPプリプロセッサを提供しています。このプリプロセッサではREPLACEおよびREPLACINGを処理できます。ただし、COPY文が通常のCOBOL構文の規則に準拠している必要があります。これに比べると、コンパイラでは原始ファイルを読み込んで単一トークンをプリプロセッサに渡す場合には、トークンおよびCOPY（またはそれ同等の）文の構文の形式などといった制約があります。

実際にコンパイルされる潜在コードは、元のコードとはかなり違います。そのため、次の点に注意してください。

- デバッガは、データ項目が潜在コードにないときはデータ項目を問い合せることができません。
- デバッガは、プリプロセッサが修正した行をコンパイラが処理せずその内容も把握していないため、これらの行を参照しません。
- XREFコンパイラでは、元の原始行が同じデータ項目を引用していても正確な相互参照表を作成しません。これは、プリプロセッサが挿入した情報またはコードが表示されないためです。

- ある指令、特にFLAGおよびMSなどの言語スイッチにより、コンパイラは統合化されているかどうかに関わらず、プリプロセッサが作成したコードを受け付けなかったりそのコードにフラグをつけたりします。統合化プリプロセッサインタフェースは、次の方法により簡単にそのような問題を回避します。
  - プリプロセッサが挿入した行にはフラグをつけません。
  - プリプロセッサがコンパイラ指令の設定を問い合わせることができるようにします。

プリプロセッサは、当然指令の設定を変えるような行を生成することができます。しかし、これにより問題が発生することがあります。特にプリプロセッサが選択した言語の予約語と矛盾するようなデータ項目名をコードで使用している場合などです。

次の項では、統合化プリプロセッサインタフェースの使用方法について説明します。

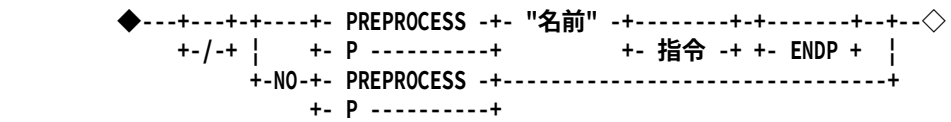
「2.2　プリプロセッサの呼び出し」
「2.3　プリプロセッサの作成」

プリプロセッサは、原始ファイル（COPYファイルなど）を読み込んで拡張するNetExpressにより提供されています。そしてREPLACEおよびREPLACINGの結果についての追加情報をスタック内のプリプロセッサに返します。詳細は、「2.4　CPプリプロセッサ」を参照してください。

2.2　プリプロセッサの呼び出し

プリプロセッサは、コンパイラのPREPROCESS指令により呼び出されます。この指令の一般的な構文は次のようになります。

構文:



プリプロセッサに渡すファイルをコンパイルするときにこの指令を指定します。

コマンド行上でPREPROCESS指令に続く指令は全てコマンド行の最後まで、または、END指令まで、検査なしにプリプロセッサへ渡されます。

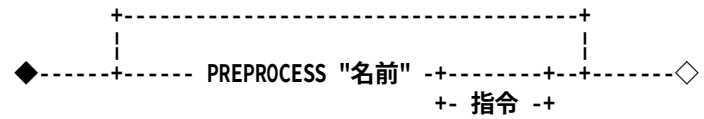
他のコンパイラ指令についても同様に、PREPROCESS指令は指令ファイルに格納するか、原始コード内の$SET文に設定します。PREPROCESS指令は、1箇所だけに設定します。$SET文に設定する場合は、$SET文を原始ファイルの最初の行とする必要があります。

複数プリプロセッサの呼出し

次のプリプロセッサを呼び出すようなPREPROCESS指令をプリプロセッサに渡すと複数のプリプロセッサが呼び出されます。2番目のプリプロセッサを呼び出すのは、最初のプリプロセッサの責任です。このスタック法により、3番目、4番目などのプリプロセッサを呼び出すことができます。スタック法を受け付けないプリプロセッサは、スタック内の最後のプリプロセッサとして使用されるだけです。

いくつかのプリプロセッサを呼び出すためのコマンド行は次のようになります。

構文:



スタック法を使用できるプリプロセッサを作成する場合の詳細については、「2.3.4　複数のプリプロセッサ」を参照してください。

プリプロセッサ名

1つ以上のオプションのプリプロセッサ・パラメータ

2.3　プリプロセッサの作成

本項は統合型プリプロセッサの書き方について説明し、プリプロセッサとコンパイラ間で情報をやりとりするためのインタフェースについて記述しています。

プリプロセッサはCOBOL以外の言語で書くことができますが、以下の説明ではCOBOLで書かれているものとしています。

統合化プリプロセッサインタフェースは、事前処理は編集の一形態であるという単純な概念に基づいて動作しています。プリプロセッサは原始コードの各行に次の意味を示すためにマークをつけます。

- 無変更
- 挿入（つまり、新しい行）
- 修正（つまり、コンパイルされていない行）

COBOLプログラムをコンパイルするとき、コンパイラは直接原始ファイルを読み込む代わりにプリプロセッサを呼び出し、プリプロセッサからコードを行単位で受け取ります。

デバッガの動作は、目的コードの各行の、原始コードの各行への写像に依ります。たとえ目的コードが原始コードに一致しなくても、上記の原始コードの各行に対するマーキングにより、この写像を有効とすることができます。

プリプロセッサの試験

プリプロセッサプログラムは、デバッグしようとする他のプログラムと同じようにコンパイルする必要があります。

プリプロセッサプログラムをアニメート用にコンパイルした後で、コンパイラそのものをアニメートするかのようにデバッガを使用します。コンパイラがアニメートされるのを見ることはありません。その理由は、コンパイラがデバッグ用にコンパイルされることはないからです。しかし、プリプロセッサのようにコンパイラが呼び出すプログラムについては、デバッグ用にコンパイルされるのを確認できます。

「2.3.2　プリプロセッサの返却コード」

「2.3.3　原始ファイルの修正」

「2.3.4　複数のプリプロセッサ」

2.3.1　コンパイラとプリプロセッサ間のインタフェースの定義

本項はコンパイラとプリプロセッサ間のインタフェースについて説明します。

プリプロセッサパラメータ

3個のパラメータが、インタフェースを介して渡されます。

mode-flag　制御情報を渡すのに使用されます。
buffer　原文情報（原始コード行とファイル名）に使用されます。
response　バッファ内の原始コード行の種類を示すのに使用されます。

これらのパラメータは以下のように定義されます。

```
01 mode-flag                    pic 9(2) comp-x.
01 buffer                       pic x(80).
01 response.
   03 response-status           pic 9(2) comp-x.
   03 response-code-1           pic 9(4) comp-x.
   03 filler redefines response-code-1.
      05 filler                 pic x.
      05 resp-main              pic 9(2) comp-x.
    03 response-code-2           pic 9(4) comp-x.
     03 filler redefines response-code-2.
      05 filler                 pic x.
      05 resp-more              pic 9(2) comp-x.
```

初期呼出し

コンパイラが通常に原始ファイルを開くときにプリプロセッサに対する初期呼出しが行われます。***mode-flag***パラメータは0に設定され、原始ファイルの名前（パス情報を含む）は***buffer***に設定されます。コンパイラが原始ファイルの場所を見つけることができた場合には、ファイル名にはフルパスと拡張子がふくまれています。見つけられなかった場合には、ファイル名はコマンド行で指定された名前になります。

さらに、初期呼出しにより、ハンドシェークプロセスが初期化されます。これにより、コンパイラおよびプリプロセッサはそれぞれのサポートレベルを決めることができます。また、新しい機能をプリプロセッサインタフェースに組み込むこともできます。この場合、コンパイラおよびプリプロセッサのサポートがなければその新しい機能は使用でません。

コンパイラは、プリプロセッサが処理できるような情報だけをプリプロセッサに渡します。同様に、プリプロセッサは、コンパイラが処理できるような要件だけをコンパイラに要求します。プロセスは、ハンドシェークプロセスを考慮しなくても従来のコンパイラおよびプリプロセッサが動作を続けられるように設計されています。

コンパイラがプリプロセッサを呼び出す場合、プリプロセッサにサポートレベルを知らせるためにはresponse-code-2を使用します。プリプロセッサがコンパイラに戻るときには、そのサポートレベル用のパラメータが使用されます。機能を追加すると、サポートレベルも上がります。そのレベルにはそれまでのレベルのサポート内容が含まれます。従来のコンパイラがサポートされるようにするには、値8224を設定します。この値は、パラメータが初期化されていないことを示し、ベースレベルであることを意味します。同様に、サポートレベルを設定していない従来のプリプロセッサは、パラメータの値を変更しません。そのため、32767以下の値はすべてベースレベルとして扱われます。

ベースレベルの機能については、次の項で説明します。ベースレベル以外の機能は、警告により強調表示されます。次のリストにその要約を示します。

コンパイラサポートレベル

response-code-2	**意味**
8224	ベースレベル
0	response-code-1にバッファの長さが含まれる
1	resp-main 14をサポートする
2	コンパイラはプリプロセッサのサポートレベルを把握し、プリプロセッサに中止するよう伝える
3	resp-main 17をサポートする

プリプロセッササポートレベル

response-code-2	**意味**
32767以下	ベースレベル
32768	コンパイラがプリプロセッサに中止するよう伝える
ベースレベル以外	コンパイラがバッファの長さをresponse-code-1に設定する。プリプロセッサはこの長さまでの原始行を返すことができる。response-code-2が8224の場合は、プロセッサは原始行の長さを80バイトと仮定する。

プリプロセッサはファイルを開き、処理が正常に行われた場合は***response-status***をゼロとし、失敗した場合には255として返します。

オペレーティングシステムコマンド行にはプリプロセッサに対する指令が入っており、スペースで終わっています。これらの指令はコンパイラコマンド行、指令ファイルまたは$SET文で指定され、PREPROCESS指令自身に続きます。

指令はコマンド行から統合型プリプロセッサへ情報を渡すのに使用され、統合型プリプロセッサの設計者により定義されます。プリプロセッサは、渡された指令がスペース１つだけで区切られていることは期待しません。コマンド行の形式はPREPROCESS指令の形式に合っていれば、各プリプロセッサ指令（最初の指令も含めて）の前に１つ以上のスペースを置くことができます。

後続呼出し

後続呼出しは、最後の行に達したことをプリプロセッサが示すまで、原始コードの行を要求します。

これらの呼出しでは、「2.3.2.4 COPY文の操作」の項に記述されている例外を除き、コンパイラは***mode-flag***パラメータを1に設定し、***response-status***パラメータを0に設定します。ただし、プリプロセッサは次の項で定義されているように情報を***buffer***、***resp-main***、および***resp-more***に設定して返します。

エラーがある場合、***response-status***はゼロ以外の値に設定されます（残りの項目は不定のままとされます）。

response-code-1と***response-code-2***の最初のバイトは将来の使用に取ってあるもので、いつもゼロに設定して返す必要があります。これを行うためのいちばん簡単な方法は、***resp-main***と***resp-more***を設定する前に***response-code-1***と***response-code-2***をゼロに設定します。

利用者の原始コードを修正する場合、元の原始コード行は常にそれらと置き換える行の前に戻して置く必要があることを、念頭に置いている必要があります。2行以上にまたがる原始コード行は単一行として処理されますので、その原始行全体（一部分ではなく）を修正のために渡す必要があります。

ベースレベル以外　コンパイラが***mode-flag***パラメータを2に設定した場合、プリプロセッサが原始プログラムの最後の行に達したことを示す前にコンパイラほとんど終了しています。この呼出しでは、プリプロセッサは必要なクリーンアップ操作（一時作業ファイルの削除など）を実行します。プリプロセッサがスタックされている場合には、他のプリプロセッサが呼び出され、同じパラメータ値により次のプリプロセッサを呼び出します。それからそのプリプロセッサを終了させます。

2.3.2　プリプロセッサの返却コード

プリプロセッサの初期呼出し後、後続呼出しにより次のいずれかが返されます。

- 原始行。次のマークが付けられています。
    - 無変更
    - 旧（注記を加えられているとして処理される）
    - 新

    - COPY文
- 次のような他の要件。
    - エラーカウントをインクリメントする
    - 指令設定を返す

resp-mainの値は、何が返されるかを示すために使用されます。追加情報は、***resp-more***または***buffer***に格納されます。***resp-main***の値は、以下のようになります。

値	説明
0	原始ファイルは完全に処理され、入力の最後となっています。***buffer***および***resp-more***は無視されます。
1	***buffer***には、元の原始コードにはなく、プリプロセッサにより付加された新しい行が入っています。***resp-more***には、オプションで置き換えられる元の原始行にある動詞の位置が入っています。プリプロセッサは注釈行を追加することはできません。
2	***buffer***には、元の原始コード内の行で、プリプロセッサにより注記を加えられているか、あるいは、置き換えられる行が入っています。***resp-more***は無視されます。
3	***buffer***には、元の原始コード内の行で、プリプロセッサにより拡張されようとしているCOPY文の先頭が入っている行が入っています。***resp-more***には行中の文の位置が入っています。
4	bufferには、COPY文の継続を含んだ元の原始コード内の行が入っています。resp-moreは無視されます。
5	bufferにはプリプロセッサが挿入した警告メッセージが入っています。注記行の形式（つまり、原始行の指標領域に値"*"が入る）とする必要があります。resp-moreは無視されます。
6	回復不可能なエラーが発生しました。これによりコンパイラは中止し、Micro Focusオプション製品の中には、COBOLエディタに入るものがあります。この場合、最大70文字のメッセージがbufferに書き出され、エディタの最下行に表示されます。resp-moreは無視されます。
7	エラーが発生しました。これによりコンパイラはそのエラーカウント数を増分します。全てのエラー種別はresp-moreを用いて指定できます。bufferの内容は無視されます。
8	この値は「2.4　CPプリプロセッサ」により生成されます。
9	この値は「2.4　CPプリプロセッサ」が使用されているときに使用されます。
10	次の11と同じ。
11	bufferには、プリプロセッサにより付加された新しい行で、プリプロセッサにより拡張されようとしているCOPY文の先頭が入っている行が入っています。COPY文が行において一意でないとき、または元の原始コードがCOBOL COPY文でない場合に使用されます。resp-moreは無視されます。
12	bufferには、COPY文の継続のあるプリプロセッサにより付加された新しい行が入っています。resp-moreは無視されます。

13	コンパイラは指令の設定に関する情報を返します。要求された指令は、オプションでbufferに格納されます。resp-moreは無視されます。
ベースレベル以外:14	この値は、元の原始コードに-INCまたは++INCLUDEが入っていることを示す、ということ以外は上記の10および11と同じです。
ベースレベル以外:17	エラーが検出されました。bufferにはCheckerが出力したメッセージが入っています。
32	bufferには、元の原始コードからの行で、プリプロセッサによって修正されていない行が入っています。
33 - 64	これらの値は、CPプリプロセッサにより生成されます。
128	COPYファイルの終わりに到達しました。bufferは空である必要があります。

「2.3.2.1　原始行の挿入」

「2.3.2.2　エラーメッセージの生成」

「2.3.2.3　指令の設定を問い合わせる」

「2.3.2.4　COPY文の操作」

2.3.2.1　原始行の挿入

resp-mainの値が1の場合、***resp-more***は、次のように置き換えられたCOBOL以外の動詞が元の原始ファイルのどこにあるかを示すために使用されます。

値	説明
0	文の置換は発生していません。

nn	現在行により置換されている非COBOL動詞の先頭文字があるカラムの番号。非COBOL文の入っている行は、***resp-main***に値2を設定して返すことにより、事前にマーキングされたと考えられます。また、2行以上であった場合は、その文は最初の行にあるものとされます。
	例えば、元の原始コードに次のような行が入っており、
	`exec abc`
	`do something useful`
	`end-exec`
	また、これらの3行が以下の行で置き換えられるとすると、
	`call abc_something_useful`
	nnの値はEXEC文の位置を示します。

2.3.2.2 エラーメッセージの生成

ベースレベル以外 プリプロセッサが原始コードを処理中にエラーを検出すると、プリプロセッサは出力するメッセージのテキストを返すことができます。

このメッセージはWARNINGおよびERRQ指令に従って他のコンパイラメッセージと同じように表示されます。また、原始行では、強調表示されます。

エラーを生成するには、プリプロセッサはエラーのある原始行を通らなければなりません。エラーメッセージは、次の設定で渡されます。

resp-main	17
resp-more	重要度を示す値 1　情報 2　警告 3　エラー 4　重大エラー
buffer	メッセージのテキスト。メッセージテキストが文字列OFFSETnnn（大文字小文字に注意してください）で始まっている場合には、nnnで指定された場所のエラーにフラグが付けられます。nnnには3バイトの数字を指定します。文字列は、メッセージの一部としては表示されません。

関数resp-main=17は、1行以上の原始行がプリプロセッサによってCheckerに返された場合に一度だけ使用できます。

注：この機能を使用する状況では、エラーが通常よりも遅く強調表示されることがあります。これは、コンパイラが継続などを探すために原始ファイルを先読みしているからです。コンパイラがプリプロセッサからのエラー要求を受け取って処理する場合には、ある行にエラーがあるということを先読みの前に決めます。そしてプリプロセッサによりフラグを付けられた行が強調表示されます。

従来のエラー状況を処理する技術もまだ使用できます。ただし、実際にエラーメッセージを出力することができるプリプロセッサが必要となるため、統合化という観点では劣っています。このような技術はお勧めしません。

プリプロセッサが原始コードの処理中にエラーを検出すると、プリプロセッサはこれをコンパイラに通信でき、そのエラーは構文エラーとして処理されます。これには次の2種類の方法があります。

- ***resp-main***を値5に設定し、注記行をバッファに入れます。これにより注記はリストファイルに挿入されます。
- ***resp-main***を値6に設定します。これによりコンパイラは終了します。

resp-moreの値はエラーが検出されたカラム番号を指定します。これはエディタに戻ったときカーソルを位置決めするのに使用されます。

上記2つの動作のどちらかに関連して、コンパイラに対してその内部エラーカウント数を増分させることも可能です。これを行うには***resp-main***を値7に設定し、どのエラーカウント数を***resp-more***で増加させるかを指定します。***resp-more***の値としては以下のものがあります。

値	重要度
1	回復不可能エラー
2	重大エラー
3	エラー
4	警告
5	情報
6	フラグ

回復不可能エラー数を増加させるとコンパイラを即座に中止させることになります。***buffer***の内容は無視されます。

コンパイラに対して動作を中止させたり、あるいは、エラーカウント数を増分させたりする前に、利用者にエラーメッセージを出力するのはプリプロセッサの役目です。情報提供の目的のみでリストファイルに挿入できるメッセージと混同しないようにする必要があります。

2.3.2.3 指令の設定を問い合わせる

resp-mainの値が13の場合、***buffer***にはスペースまたは要求されたある特定の指令の設定の名前が入っていなければなりません。

プロセッサが次に呼ばれるときに***buffer***には指令の設定値が入っています。最初に13が返された場合、コンパイラは指令の設定をすべてリストにし、その中から1つの値を返します。これ以降に返す値もこのリストの中の値です。

13以外の値がある時点で返された場合には、また13が返されたときにリストが再度作成されます。指令のリストはアルファベット順で作成され、すべての指令の設定が返されると、次の呼出しでは***buffer***にスペースが入れられます。***buffer***にスペースが入っている場合は、最初または2番目の指令の設定が返されます。***buffer***に指令名が入っている場合には、コンパイラはその指定された名前の指令までリストをとばして読みます。構文チェック段階の指令だけがこの方法で返されます。

2.3.2.4 COPY文の操作

プリプロセッサが、COPYファイルの内容を変更しない場合には、修正されていないCOPY文をコンパイラに渡すことができます。そこでCOPY文は拡張されます。

この方法で渡せることができるのは、有効なCOBOLの構成要素だけです。COBOL以外の構成要素は、コメントアウトされて有効なCOPY文に置き換えられます。置き換えられた文は。コンパイラによって拡張されます。

いずれの場合も、プリプロセッサはCOPYファイルそのものを読み込んだり処理したりする機会はありません。

コンパイラは、プリプロセッサがなくても拡張できるCOPY文をすべて拡張します。COPY文は、無変更行または修正行として戻されても拡張されます。

次の項目がサポートされています。

- 単純COPY文
- COPY...REPLACING文
- COPY 複写ファイル名 OF/IN 登録集名
- 複数行に渡るCOPY文
- COPY文の入れ子
- ++INCLUDE
- -INC

プリプロセッサは、COPYファイルの内容を検査する場合にはそれを拡張するかCPプリプロセッサを使用する必要があります。COPYファイルそのものの内容は主な原始ファイルの行と同じようにコンパイルに返されます。ただし、COPY文そのものは特別に処理されます。

最も簡単な例として、COPY文が1行以上の行で唯一の文であり完全なファイル名（必要なら拡張しおよびパスも含む）を指定している場合には、その最初の行がコンパイラに渡されます。このコンパイラでは、resp-mainには3が、resp-moreにはCOPY文の先頭のカラム番号が設定されています。そしてそれ以降の行はすべてresp-mainに4を設定して渡されます。

COPY文で指定されたファイルの位置は、ファイル名の拡張子またはパスを付けることによって確定されている場合、またはCOPY文がその行でユニークでないか規定のCOBOL COPY文でない場合には、そのCOPY文を含む行をすべてコメントアウト（resp-mainに2を設定）して挿入されたたときに通過する必要があります。プリプロセッサが拡張する挿入されたCOPY文はCOBOLの構文規則に準拠し、次のように設定する必要があります。

- resp-mainに11または14を設定します。
- 先頭行のresp-moreにnn（nnは、COPY文の先頭の位置を示します）を設定します。
- 2行目以降の行のresp-mainに12を設定します。

例えば、原始コードに次のような行が入っているとします。

```
01 ITEM-A. COPY "CPY-FIL .CPY".
```

これが置き換えられようとしている行であることを示すために***resp-main***を2に設定して、この行が最初に返されます。次の呼出しでは、プリプロセッサは下記の行を返します。

```
01 ITEM-A.
```

この場合、***resp-main***を1に設定してこれが置換行であることを示しています。次の呼出しでは、プリプロセッサは下記の行を返します。

```
COPY "COPY-FIL .CPY".
```

今度の場合は、***resp-main***は11に設定され、これがCOPY文だけの入った置換行であることを示しています。***resp-more***は20に設定され、元の原始行における語COPYの位置を示します。

COPY文がコンパイラに返される場合、プリプロセッサまたはコンパイラによって拡張されるかどうかに関わらず、コンパイラはそのCOPY文を解析してREPLACINGをチェックするということに注意してください。このREPLACINGは、このCOPYファイルおよび入れ子になったCOPYファイルのすべての行に影響します。このCOPY文で指定された名前は、アニメータが使用するために読み込まれ格納されます。大文字小文字を区別するプラットフォームにおいて名前を読み込む場合に不必要な大文字小文字の変換が発生しない様にこの名前を引用符で囲むことを強くお勧めします。そうしなければ、アニメータがファイルの位置を決めることができなくなるかもしれません。

ベースレベル以外　resp-main=14は、元の語がCOBOL COPY文でない（++INCLUDE、-INCなど）場合に使用されます。コンパイラ指令の中には、任意のCOBOL原始行の後ではなく前の＄SET文で指定できるものもあります。そのような文は、COPYではなく++INCLUDEが後に続きます。resp-main=14を使用すると、++INCLUDEが拡張され、同時に同様の指令が続いて設定されます。

値128は、プリプロセッサがCOPYファイルの拡張を終えたときに***resp-main***だけに使用できます。128が***resp-main***以外で使用されている場合には、コンパイラはそのコンパイルを中止します。値0は、主な原始ファイルの最後に使用できます。

見出し部の注釈記述項のCOPY文は、必ずしも拡張されるとは限りません。プリプロセッサが拡張を合図すると、COPY文が有効ではないときでもCOPYファイルを拡張しようとします（***resp-main***を3または11に設定します）。こととときコンパイラは、次にプリプロセッサを呼び出すときに***response-status***にゼロ以外を設定します。プリプロセッサは、COPYファイルを直ちに捨て、COPYファイルが空であったかのようにend-of-copyfileマーカーを送ります（***resp-main***には128を設定します）。

CPプリプロセッサは、他のプリプロセッサに代わってCOPYファイルを拡張します。CPプリプロセッサは、前述したように行を生成します。

2.3.3　原始ファイルの修正

原始ファイルを修正するためのCOBOLコマンドはたくさんあります。プリプロセッサがアクティブであると、REPLACE動詞（ANSI85動詞）はコンパイラによってサポートされず、プリプロセッサが処理する必要があります。

CPプリプロセッサは、REPLACEの効果に関する情報を他のプリプロセッサに返します。

2.3.4　複数のプリプロセッサ

同じ原始プログラムに対していくつかのプリプロセッサを同時にアクティブとすることは可能です。これらのプリプロセッサはスタック内に整列され、コンパイラはスタックのいちばん上のプリプロセッサを呼び出し、このプリプロセッサが次のプリプロセッサを呼び出すというふうに、原始コードを実際に読み込むスタックのいちばん下のプリプロセッサまで呼び出していきます。原始コードの各行は各プリプロセッサを介して順番に渡されて行き、最後にスタックのいちばん上のプリプロセッサに到達してコンパイラに渡されます。この動作を行うには、プリプロセッサはいくつかの追加規則に従う必要があります。

コンパイラはプリプロセッサに最初の呼出しをかけるときに、プリプロセッサ指令をコマンド行に書き出します。プリプロセッサはこのコマンド行を読み込んで、PREPROCESS指令を検出すると、そのPREPROCESS指令に指定されたプリプロセッサを呼び出す必要があります。プリプロセッサは、また、その指令に続く任意のパラメータをコマンド行に順次書き出すことによって、それらのパラメータを呼び出されたプリプロセッサへ渡す必要があります。そして本項で記述したハンドシェークプロセスを継続させます。

コンパイラに渡されたプリプロセッサ指令はENDP指令によって停止されるか行の最後に達すると停止します。そのため、複数のPREPROCESS指令はすべて指令ファイルの同じ行になければなりません。

2つのプリプロセッサ間のインタフェースは、コンパイラと上記指定のプリプロセッサ間のインタフェースと同じです。従って、スタック可能プリプロセッサをスタック・非スタック両方の状態で使用できます。更に、注目していただきたいのは、スタックするように設計されていないプリプロセッサが直接原始コードを読み込んでいる場合は、それをスタックの最後にスタックできます。

ほとんどの場合、プリプロセッサは個別組の構文に作用し、有効なCOBOL構文を生成します。従って、複数のプリプロセッサが、ある特定の原始行を処理しようとする事態は発生しません。しかし、コンパイラに至るスタック経路に沿って、原始コード行が異なるプリプロセッサにより何度も編集されるように、スタック内のプリプロセッサが言語レベルを表わすことは有り得ます。この場合、原始コードとコンパイラに最後に到達するコードの関係が維持されるように、応答フィールドの情報を読み込むように注意する必要があります。また、1つのプリプロセッサにより変更されたコードが、2番目のプリプロセッサにより引き続き修正されることがある場合は、スタック順序の選択には注意を払う必要があります。

2.3.5　プリプロセッサの作成での注意点

原始コードに対して行われた変更が、ユーザが選択したCOBOL指令またはデータ名と矛盾することがあります。注意すれば、どのような潜在的な問題も簡単に回避できます。

- ユーザが選択したCOBOL指令との矛盾

ユーザコードがある指令に従っていて、その指令をプリプロセッサが使用するのに必要な機能がサポートされていない場合、解決しなければならない問題が2つあります。1つ目は、その指令に従っていないコードにはフラグが付けられるということです。2つめは、必要な予約語が辞書にないということです。

フラグの問題を解決するには、コンパイラにプリプロセッサが挿入した行にはフラグを付けさせないことです。必要な予約語をすべて使用可能にするには、プロセッサがANS85指令などの行設定を作成することです。

ただし、これには好ましくない副作用があります。それはいくつかの文の動作が変更されてしまうことと、有効なデータ名が予約語になった代りに拒否されることがあること、この2つです。

実際にはどの語が必要なのかを判断し、ADDRSV指令を使用してこれらの必要な語だけを選択的に追加するのにもっと良い方法があります。たとえば、FUNCTIONという予約語だけが必要な場合、ADDRSV(FUNCTION)指令を使用してこの語だけを追加することができます。ただし、これでもまだ問題が発生する可能性があります。それはユーザコードがFUNCTIONをデータ名として宣言した場合です。あらゆる解決法の中で最も良い解決法は、ADDSYN指令を使用してユーザが使用しそうもないFUNCTIONに代わる語を選択することです。

たとえば、ADDSYN "FUNCTION" = "PREPGEN--FUNCTION"指令を生成して、その後コードを生成しているときにFUNCTIONの代りにPREPGEN--FUNCTIONを使用します。

注: ADDSYNを使用すると、その後に原始コードを処理するプリプロセッサを混乱させることがあります。

- ユーザが選択したCOBOL指令との矛盾

プリプロセッサが使用目的でデータを作成する必要がある場合、ユーザが選択した名前と矛盾しないような名前を選択する必要があります。そのような場合には、PREPGEN--USERIDのような名前を選択すると、ユーザが選択した名前と矛盾するようなことはありません。しかし、USERIDだけだとよく問題が起こります。

2.4 CPプリプロセッサ

CPプリプロセッサは、スタックされたプリプロセッサとして使用できるように設計されています。CPプリプロセッサは、原始ファイル（COPYファイルも含む）を読み込んで拡張したり、REPLACEおよびREPLACINGの結果に関する追加情報をスタック中の他のプリプロセッサに返します。したがって、これらの非常に複雑な他のプリプロセッサから取り除きます。

制限

CPプリプロセッサには次のような制約があります。

- 標準のCOBOLトークンおよびCOPY文しか受け付けない
- BASISの結果を表示しない
- 2バイト文字セット（DBCS）の文字を含む原始プログラムを受け付けるように設計されていない

「2.4.1　COPY展開」

「2.4.2　REPLACE通知」

「2.4.3　CPの指令」

「2.4.4　CPのエラーメッセージ」

2.4.1　COPY展開

CPプリプロセッサがCOPY文を見つけると、「2.3.2.4　COPY文の操作」の項に記述されているように原始行にマークを付け、COPYファイルを読み通します。コンパイラと同じパス名およびファイル拡張子を使用してCOPYファイルの場所を探します。

COPYファイルの場所が見つからない場合は、CPプリプロセッサはそのCOPY文をコメントアウトして適切なエラーメッセージを生成します。文の解析中にエラーが発生した場合は、プリプロセッサはその文を無視してエラーの時点からトークンの解析を続けます。他のプリプロセッサが修正されていないCOPY文を受け取った場合には、CPプリプロセッサがそのCOPY文を処理できなかったものと仮定し、そのままにしてコンパイラにエラーメッセージを作らせます。

2.4.2　REPLACE通知

他のプリプロセッサがコンパイラのようにREPLACEおよびREPLACINGの実行後に原始ファイルの概要を把握できるように、プリプロセッサはここで記述するような追加行を返します。ただし、コンパイラが以下に注記する場合を除いてこのREPLACINGそのものをまだ実行することに注意してください。追加行は、ただ情報を伝えるだけのために渡されます。

プリプロセッサが修正された行を見つけると、*resp-main*の値に32を付加して返します。このように値が設定された行の後には、*resp-main*の値が8である行が1行以上続きます。これらの行には修正後の行の内容が含まれています。これらの変更内容は情報を伝えるだけの目的で使用されるため、これらの行を受け取るほかのプロセッサは、スタック内の他のプリプロセッサにさらにこの情報を渡す必要があります。コンパイラは33から64の範囲の値が設定された行を受け取るとその値から32を引きます。また、8が設定された行は無視します。

プリプロセッサがこのように値が設定された行を修正する場合には、規定通りに行う必要があります。ただし、コンパイラに新しく挿入された行についてREPLACEまたはREPLACINGの結果内容をテストする必要がないことを知らせるためには、プリプロセッサはこの新しい行に1ではなく9を設定する必要があります。

2.4.3 CPの指令

CPプリプロセッサは、たくさんの指令を受け付けます。コマンド行を短くするために、これらにはフルネームの他に省略形の名前があります。通常の方法では、CPプリプロセッサはコマンド行を検査した後で指令用の環境変数CPDIRを見ます。

CPプリプロセッサでは次の指令を使用できます。

CONFIRM
LIMITED-SEARCH
PREPROCESS
SOL
SY
TRACE
WARNINGS

CONFIRM, C

指令が許可された場合に画面上に表示するかどうかを指定します。

構文:

省略値: NCONFIRM

LIMITED-SEARCH, LS

環境変数COPCPYで指定したディレクトリに範囲を限定してCOPYファイルを検索します。本指令を使用して環境変数COBCPYのディレクトリとCOPYEXT指令のファイル拡張子を慎重に選ぶと、ファイル検索の数を大幅に少なくすることができます。

構文:

省略値: NOLIMITED－SEARCH

注釈:

本指令は、COPYファイルの検索を環境変数COPCPYで指定したディレクトリに限定します。これによりコンパイル時間を短縮することができます。

本指令を設定すると、次のようになります。

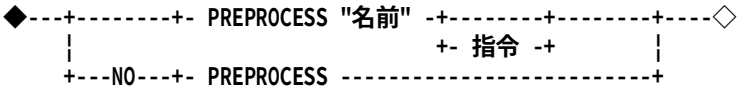
- COPYファイルは、環境変数COBCPYで原始ファイルがあるディレクトリを省略して指定されたディレクトリ内で検索されます。

- OSEXT指令で指定されたCOPYファイルの拡張子は、検索では使用されません（ただし、COPYEXTで指定された拡張子は使用されます）。

PREPROCESS, P

別のプリプロセッサをスタックします（つまり、このプリプロセッサに入力された原始ファイルが原始ファイルそのものではなく別のプリプロセッサから渡されるように指定します）

構文:

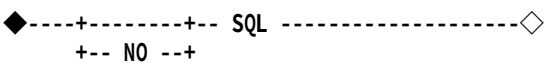


省略値: NOPREPROCESS

SQL, S

EXEC SQL INCLUDE文をCOPY文として扱うかどうかを指定します。

構文:



省略値: SQL

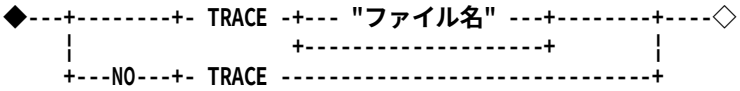
SY

本指令は、COBSQLで使用するために予約されています。本指令は、*Database Access*に記述されているようにしか使用できません。

TRACE, T

トレースファイルを作成するかどうかを指定します。また、オプションで使用するトレースファイルの名前を指定します。ファイル名を省略した場合は、*progname.cpt*の名前で作成されます。prognameには主な原始ファイルの基本名が入ります。

構文:



省略値: NOTRACE

WARNINGS, WARNING, W

指令の設定に関する警告を画面に表示するかどうかを指定します。

構文:

```
◆----+--------+-- WARNINGS --------------------◇
     +-- NO --+
```

省略値: WARNING

2.4.4 CPのエラーメッセージ

CPのエラーメッセージはすべて次の形式で表示されます。

```
*CP nnn-x
**     説明
```

変数の意味を以下に示します。

nnn **エラー番号**

x **重要度のレベル**

W **警告 - 処理は続行されます**

S **重要 - 初期化中の場合は処理は中断されますが、その他の場合は致命的なエラーではありません。**

U **回復不可能 - 処理は中断されます**

以下の番号をクリックすると、エラーの原因、対処法などエラーメッセージの説明を見ることができます。

初期エラー

101, 102, 103, 104, 105, 106, 107

メイン処理エラー

0, 200, 201, 202, 203

101 Illegal command line

1つ以上の指令が拒否されました。

対処:

CPプリプロセッサを明示的に呼び出している場合には、コマンド行を修正して再実行してください。CPプリプロセッサが他のプリプロセッサにより呼び出されている場合には、ベンダにお問い合わせください。

102 Compiler level set to 1; come features disabled

使用可能な機能をテストしています（内部使用のみ）。

対処:

必要ありません。

103 Open fail: filename

CPプリプロセッサは原始ファイルを見つけられませんでした。

対処:

正しい名前を入力してコマンドを再実行してください。

104 Open fail: filename

CPプリプロセッサは原始ファイルを作成できませんでした。

対処:

ファイルがすでに使用されていないか確認してください。また、ディスクの空き容量が十分かるか、ドライブがライトプロテクトされていないか確認してください。

105 Call to stacked preprocessor name failed

CP指令PREPROCESSにより指定されたプリプロセッサが見つかりませんでした。

対処:

プリプロセッサが存在するか、また呼出し可能かどうか確認してください。名前が正しくなければ、CPプリプロセッサが明示的に呼び出されている場合にはコマンド行を修正し、CPプリプロセッサが他のプリプロセッサにより呼び出されている場合には適切な製品マニュアルを参照してください。

106 Stacked preprocessor returned an error

スタックされたプリプロセッサが初期化に失敗しました。

対処:

CPプリプロセッサの対処はありません。スタックされたプリプロセッサが通知したエラーを直してください。

107 Unable to open a heap

内部エラーが発生したためヒープをオープンできませんでした。

対処:

技術サポートにお問い合わせください。

200 Internal stack full - contact technical support

内部処理エラーが発生しました。

対処:

技術サポートにお問い合わせください。

201 File error - contact technical support

内部処理エラーが発生しました。

対処:

技術サポートにお問い合わせください。

202 Copybook filename not found

COPY文が存在しないファイルを参照しようとしました。

対処:

原始プログラムを修正して再実行してください。

203 Nested REPACING is not allowed

COPY REPLACING文の処理中に別のCOPY REPLACING文を処理しようとしました。

対処:

原始プログラムを修正して再実行してください。

0 Undefined technical error - contact technical support

内部処理エラーが発生しました。

対処:

技術サポートにお問い合わせください。

2.5 統合型プリプロセッサの例

スタックされるように設計された2つの統合型プリプロセッサ例が、Micro Focus COBOLのサンプルプログラムに含まれています。また、プリプロセッサが処理する原始プログラムとCOPYファイルの例も提供されています。

これらの例をコピー・アンド・ペーストすると、これらのファイルを含むプロジェクトを作成してその動作を見ることができます。また、次の例をもとにしてユーザ独自のプリプロセッサを作成することもできます。

- PREPROC.CBL**プログラム**
- PREPROX.CBL**プログラム**

プリプロセッサ例は他のプリプロセッサの基本として使用でき、下記の事柄を具体的に説明しています。

- どのようにしてプリプロセッサは呼び出され、原始入力を開くか
- どのようにして原始コードは編集なしにコンパイラに渡されるか
- どのようにして原始コードは編集されそしてコンパイラに渡されるか
- どのようにして2個以上の動詞の入った行が拡張されるか
- どのようにしてCOPY文が処理されるか
- どのようにして警告とエラーは生成されるか
- どのようにしてプリプロセッサをスタックできるか

プリプロセッサ例の仕様

プリプロセッサ例は下記のようにいくつかの動詞を認識・処理します。

- MOVはMOVEに変換されます
- DISは2行に変換されます。最初の行がDISPLAY SPACE UPON CRT、2番目の行には語DISをDISPLAYと置き換えて元の行が入ります。
- PRINTはDISPLAYに変換されます。
- CPYはCOPYに変換され、従って、コンパイラにより拡張されます。COPYはプリプロセッサが拡張します。
- WARNにより警告メッセージがリストファイルに書き出されます。
- ERRORにより重大エラーメッセージが表示され、利用者にコンパイルを継続するかどうかたずねます。継続を選択した場合は、重大エラーカウント数が増分されます。

スタック可能プリプロセッサを具体的に説明するために、これらプリプロセッサの関数は2つの別々のプログラムに分けられます。これら2つのプリプロセッサは非常に良く似ており、多くの共通コードを持っています。

これらのプリプロセッサはCOBOL用字句解析系例を含んでおり、この解析系は複数レコードに渡っては続かない字句単位を扱います。

プログラム例

WORKING.CBLファイルとASMCOBOL.CBLファイルは、プリプロセッサによる翻訳を必要とするプリプロセスファイルです。これらのファイルは、それぞれPREPROCとPREPROXと共に使用します。

例とするプリプロセッサPREPROCとPREPROXは、これらのデモンストレーションを行う前にコンパイルしてください。

ベースレベルは、最も古いコンパイラのサポートレベルを意味します。

コンパイラがプリプロセッサを呼び出す場合、プリプロセッサにサポートレベルを知らせるためにはresponse-code-2を使用します。プリプロセッサがコンパイラに戻るときには、そのサポートレベル用のパラメータが使用されます。

機能を追加すると、サポートレベルも上がります。そのレベルにはそれまでのレベルのサポート内容が含まれます。従来のコンパイラがサポートされるようにするには、値8224を設定します。この値は、パラメータが初期化されていないことを示し、ベースレベルであることを意味します。同様に、サポートレベルを設定していない従来のプリプロセッサは、パラメータの値を変更しません。そのため、32767以下の値はすべてベースレベルとして扱われます。

ベースレベル以外の機能は、警告により強調表示されます。

第3章： H2CPY

3.1 概　説

H2cpyユーティリティはC言語のヘッダーファイルの内容をCOBOLの同等なCOPYファイルのCOBOL型定義とCALLプロトタイプ構文に翻訳します。COBOLシステムは、この構文を使用して、COBOLプログラム中で使用されているパラメータが元のC言語のヘッダーファイルの定義と整合するかどうかを検査します。H2cpyを使用すると、インタフェース定義がC言語のヘッダーファイルとして提供されているライブラリルーチンをCOBOLプログラムで利用することが容易になります。Windows 32bit用のライブラリを使用することはその一例です。

H2cpyは2つのCOBOLファイルを作成します。それらのファイルをCOPY文を使用して、COBOLプログラムに組み込むことができます。主となる（.cpy）ファイルには型定義と定数が収録されます。外部データ宣言があれば補助的な（.ext）ファイルに収録されます。このH2cpyはCOBOLプログラムでCインタフェースを使用するときに重宝なだけではなく、CプログラムをCOBOLで組み直すときにも役に立ちます。

H2cpyへの入力は任意の有効なANSIまたはISOの標準のCソースファイルです。有効でないCを入力すると、H2cpyは異常終了したり不正なCOBOLコードを生成したりすることがあります。

H2cpyはコマンド行から直接実行することができます。また、コンパイラ指令のPREPROCESS（headers）を指定して、間接的に呼び出すこともできます。Headersプリプロセッサを使用すると、COBOL原始ファイル中でCOPY文の中に拡張子が.hであるCヘッダーファイルを指定することができます。Headersプリプロセッサはそれをこのヘッダーファイルと認識して、H2cpyを呼び出して変換します。この方法は比較的小さなヘッダーファイルに適用すると非常に便利です。

この章で提示する例はプログラムの全体ではなく一部です。H2cpyが作成した完全なコードではありません。H2cpyによって生成されたコードの中には、『言語リファレンス』には含まれていないものがあります。構文の補足については、ディスクドキュメントを参照してください。

この章で記載されている例を使用すると、記述されたコードの一部に加えて余分なCOBOLデータ項目も生成されます。これらのデータ項目は、等価なCデータ型と対応する一般のCOBOL typedefであるので、出力されるCOBOLコピーファイルに含まれます。データ項目のサイズは実行環境に依存します。

32ビット環境

```
77    char           pic s9(2)    comp-5 is typedef.
77    uns-char       pic  9(2)    comp-5 is typedef.
77    short          pic s9(4)    comp-5 is typedef.
77    uns-short      pic  9(4)    comp-5 is typedef.
77    int            pic s9(9)    comp-5 is typedef.
77    uns-int        pic  9(9)    comp-5 is typedef.
77    long           pic s9(9)    comp-5 is typedef.
   77   uns-long       pic  9(9)   comp-5 is typedef.
77    d-l-float                   comp-2 is typedef.
77    d-float                     comp-2 is typedef.
```

```
77    float                       comp-1 is typedef.
77    proc-pointer      procedure-pointer is typedef.
77    data-pointer                pointer is typedef.
77    void              pic  9(2)    comp-5 is typedef.
```

16ビット環境では、以下のデータ項目が異なるサイズで出力されます。

```
77    int               pic s9(4)    comp-5 is typedef.
77    uns-int           pic  9(4)    comp-5 is typedef.
```

3.2 操　作

この節では、H2cpyの使用方法を説明します。まず最初に、コマンド行の構文から説明を開始します。

3.2.1 H2cpyの呼出し

H2CPYを呼び出すには、下記のコマンドを入力します。

h2cpy {[オプション**] [**ファイル名**]}･･･**

ここで、

オプション　下の「コマンド行オプション」の項に列挙するH2cpyオプションの1つです。

ファイル名　変換する対象のC言語のヘッダーファイルです。下の「入力ファイル名」の項を参照してください。

3.2.1.1 コマンド行オプション

オプションはハイフン（-）で始まり、大文字と小文字の区別があります。パラメータをとらない1文字のオプションは1つのハイフンの後ろにまとめて書いてもかまいません。たとえば、"-v -C -h"の3つのオプションを"-vCh"と書くことができます。オプションを指定する順序は任意です。

ハイフン（-）の代わりにスラッシュ（/）を使用することができます。ハイフンで始まるオプションは最初のファイル名を指定する前に使用してはなりません。

オプションを以下に列挙して詳しく説明します。配列はアルファベット順です。ただし、大文字のものを小文字のものの前に掲げます。オプションの文字とパラメータの間の空白は空けても空けなくてもかまいません。

オプション	摘　　要
-A	C言語をANSI流に厳密に解釈しないようにする。 現在のANSI/ISO C標準がそれ以前のC言語製品と異なる曖昧な領域がいくつかある。旧式のCが原因で発生していると思われる問題がある場合には、このオプションを使用してみるとよ

い。

オプション	摘　　要
-C	生成されたCOBOLの中にCコードを組み込む。 組み込まれたCコードの各行の先頭にはアスタリスク（*）と空白が付けられる。それらの行はCOBOLコンパイラによって無視される。組み込まれたCコードはCプリプロセッサを通ってきている。
-D 名前 [=値]	マクロ名を宣言する。値を指定することもできる。値は数値または文字列定数とすることができる。
-E	構造体と共用体をすべて展開する。 構造体および共用体用のCOBOLコードはそれらが最初に宣言されたときにのみ、全面的に示される。以降の参照は名前による。
-I パス	インクルード検索パスリストの先頭にパスを挿入する。このオプションは検索パスリストにエントリを追加するために使用することができる。検索パスリストに保持することの出来るエントリは14までである。それに達するまで、このオプションを使用することができる。 注： 検索パスリストは当初は空である。ただし、INCLUDE環境変数を使用した場合は別である。
-J	文字型の省略時の設定を符号なしに変更する。
-M	通常は認識される下記のMicrosoftのキーワードを無効にする。

```
Cdelc       pascal     _far        _near
Far         _asm       _fastcall   _pascal
Fortran     _based     _fortran    _saveregs
Huge        _cdecl     _hege       _segment
Interrupt   _emit      _interrupt  _segname
Near        _export    _loadds     _self
```

-P	Cの前処理だけを行い、COBOLコピーファイルを生成しない。 このオプションは-Cオプションに似ているが、さらにCOBOLコピーファイルの生成を行なわない。H2cpyによるCの前処理がCコンパイラと異なると疑われる場合に、このオプションは役に立つ。

オプション	摘　　要
-R 名前	COBOLの予約語を辞書に追加する。 現在分かっている予約語はすべて、すでにH2cpyに組み込まれている。このオプションを使用すると、新しい予約語を追加することができる。

-S　　#defineマクロ定義用のCOBOLを別のファイルに生成する。ファイル名は出力ファイル名に基づいて付けられるが、拡張子は.78とされる。

C言語の反復記号（...）に相当する標準的なCOBOLは現在のところ存在しない。したがって、H2cpyでは反復記号を無視する。このオプションを使用すると、それに相当する暫定的なCOBOLのコードを生成することができる。

-V　　可変数のCOBOLパラメータを生成する。

C言語の反復記号（...）に相当する標準的なCOBOLは現在のところ存在しない。したがって、H2cpyでは反復記号を無視する。このオプションを使用すると、それに相当する暫定的なCOBOLのコードを生成することができる。

-X　　文字列ポインタにCOBOLで相当するものをchar*だけからchar*とunsigned char*の両方に拡張する。

関数プロトタイプにおけるunsigned char*パラメータからは、省略時の設定では、BY REFERENCE uns-charが生成される。uns-char型は一般にヘッダーファイル以外に適する数字であり、文字列ポインタを定義するためにunsigned char*型を使用している）。そのようなファイルを変換するときには、-Xオプションを使用して、H2cpyがBY REFERENCE ANYを生成するようにし、文字列パラメータをPIC X（n）の項目として渡せるようにする。

-a型=n　　Cのデータ型用の桁寄せを設定する。ここで、［型］は次のどれかとする：int, char, short, long, pointer, float, dfloat, dlfloat。

上記のすべてのCのデータ型には桁寄せの省略値がある（-eオプションを参照）。このオプションを指定すると、それらの省略値を個々に変更することができる。

オプション	摘　　要

-c　　COBOL用の名前をすべて小文字で生成する。

H2cpyは最初に出てくるCOBOL用の名前を元のCでの名前とできるかぎり近いように付ける。その際、大文字と小文字の別は元のCでの名前と同じにする。

-e n　　ターゲットの環境を表わす。nは16または32であり、それぞれ最も一般的な16ビット環境と32ビット環境を示す。このオプションを指定しないと、省略値としてWindows 32bit用の32が採られる。

Cのデータ型のサイズと桁寄せはコードが実行される環境に依存する。たとえば、16ビットと32ビットと64ビットのどの環境で実行するかによって、Cのデータ型のサイズは異なる。

16ビット環境では、Cのデータ型のサイズと桁寄せは下記のようになる。

char 1:1, short 2:2, int 2:2, long 4:4, pointer 4:4,

float 4:4, double float 8:8, double long float 8:8

32ビット環境では、Cのデータ型のサイズと桁寄せは下記のようになる。

char 1:1, short 2:2, int 4:4, long 4:4, pointer 4:4,

float 4:4, double float 8:8, double long float 8:8

-f　　#define浮動小数点定数をCOBOLに渡す。

COBOLコンパイラは浮動定数のサイズと形式の妥当性を検査する。省略時の設定では、#defineに浮動小数点定数としてマクロが指定されていると、それに対応するCOBOLのコードは生成されない。このオプションを設定すると、それに相当するCOBOLのコードが生成されるようになる。しかし、COBOLコンパイラがその数値のサイズを受け入れないことがある。

浮動小数点定数が含まれている定数式をH2cpyが評価すると、その浮動小数点定数はゼロとして扱われ、警告が発せられる。

オプション　　摘　　要

-h　　16進の#define値を10進に変換する。

省略時の設定では、#defineに16進定数としてマクロが指定されていると、H2cpyは16進のCOBOLの定数を生成する。このオプションを設定すると、H2cpyはそれを10進で生成するようになる。

-0　名前　　生成されたCOBOL出力ファイルに名前を付ける。

省略時の設定では、生成されたCOBOLのコードが収められた出力ファイルの名前は最初のCヘッダーファイルの基本名に基づいて付けられる。その際、主コピーファイルには拡張子.cpyが付けられ、外部宣言ファイルには拡張子.extが付けられる。このオプションを指定すると、主コピーファイルに別の名前を使用することができる。生成された外部宣言ファイルが収められているファイルの基本名は主コピーファイルと必ず同じである。

-p　　CALLプロトタイプを生成しない。

省略時の設定では、H2cpyはCの関数プロトタイプをCOBOLのCALLプロトタイプに変換する。そして、COBOLコンパイラは行われるCALLをCALLプロトタイプに適合させるようにする。この適合性のチェックは必ずしも柔軟ではない。

H2cpyにCALLプロトタイプを生成させたくない場合に、このオプションを設定する。

数字定数（または78レベルのデータ項目）をパラメータとして渡す呼出しには注意を要する。コンパイラはそのデータ項目をCOMP-5に変換し、そのサイズをCOBOL CALLプロトタイプに示されている値に合わせる。CALLプロトタイプが存在しない場合は、サイズには省略値である2バイトが採られるが、それが正しくなくて実行時に呼出しが異常終了する可能性がある。そのような場合には、BY VALUE指定に対してSIZE句を使用して、正しいサイズを指定しなければならない。

-r　　COBOL予約語のリストを削除する。

生成されたCOBOLコードをコンパイルしない場合は、このオプションを指定して起動にかかる時間を短縮することができる。

-s 型=n	Cのデータ型のサイズを設定する。型は-aオプションの場合と同じ。

オプション	摘　　要
-u	COBOLのデータ名の中でアンダースコアを使用できるようにする。 標準のCOBOLコンパイラとは異なり、Micro Focus COBOLでは利用者語にアンダースコア文字を使用することを許している。
-v	画面上に情報メッセージを表示するモードを交互に切り替える。 メッセージ表示モードでは、プログラムの見出しを始めとする各種のメッセージが表示される。その中には、ネストされている#includeファイルの追跡も含まれる。メッセージ非表示モードでは、エラーメッセージだけが表示される。省略値はメッセージ表示モードである。

3.2.1.2　名前の翻訳

H2cpyでは通常、コピーファイルの中で使用するデータと手続きの名前を元のCヘッダーファイルと同じにする。しかし、CとCOBOLでは名前の付け方に関する標準に違いがある。そこで、Cのファイル中で下記の条件が発生する場合は、H2cpyは新しい名前を付ける。

- 2つの別々のデータ項目に同じ名前が使用されていて、しかも両者は別々の構造体ではない。
- 2つの名前の間に大文字か小文字かの違いしかない。
- 名前がアンダースコアで始まる（自動的にハイフンに変換）。
- データ名がCOBOLの予約語に当る。
- 2つの名前の先頭から30文字の間に違いがない。

上記の条件の中には、利用可能なオプションを設定することによって、変更できるものがあります。たとえば、-uオプションを使用すると、アンダースコアを使用できるようになります。

H2cpyによって付けられる新しい名前は古い名前の先頭に番号を付けた形を取ります。その番号は、元の名前が宣言されて以降、同じ名前が出てきた数です。

3.2.1.3　Headersプリプロセッサの使用

H2cpyをコマンド行から実行した方が便利な場合は、大きなCヘッダーファイルを変換するとき、生成されたコピーファイルを修正したいとき、いろいろなコマンド行フラグを使用したいときです。

簡単なヘッダーファイルの場合は、COBOLプログラムの中からCヘッダーファイルを直接参照した方が便利です。その場合、プログラムがコンパイルされるたびに、HeadersプリプロセッサによってH2cpyが自動的に呼び出されます。

そのような形でH2cpyを使用するには、（.h）ファイルを参照するCOPY文をCOBOLプログラムの中に含めるだけで済みます。COPY文を置く位置には制限はありません。さらに、必ずPREPROCESS（headers）指令を使用してください。その指定をする最も便利な方法は$SET文を使用することです。その例を下に示します。

```
$set proprocess(headers)
 working-storage section.
 01 data-item pic x(9).
     copy "cheader.h".
 01 cobol-b usage btype.
 procedure division
    copy"cheader2.h".
    move 6 to c-long     of cobol-b
    move 7 to c-int    of cobol-b
    move "Show"to c-char   of cobol-b
    display"C struct= " c-long " " c-int " "c-char level-78
```

上記のコード中のcheader.hの内容は下記のようになっています。

```
#define b B
int a ;
typedef struct btype {
   long c_jong;
   char c_char[5];
   int c_int;
   } b;
```

上記のコード中のcheader2.hの内容は下記のようになっています。

```
#ifdef b
#define level_78 "C literal"
#else
typedef struct btype {
   signed char c_uchar;
   float c_float[5];
   }b;
#endif
```

このプログラムを実行すると、下記のメッセージが表示されます。

```
C struct= +0000000006 +00007 Show C literal
```

3.2.1.4 指令の翻訳

H2cpyはいくつかのタイプの指令を翻訳します。その対象となる具体的な指令には、プリプロセッサ、マクロ、条件、#errorに関するものがあります。

プリプロセッサ指令

H2cpyは#include指令と#define指令の両方を取り扱います。

#include指令の翻訳

H2cpyは下記の形式の#include指令を取り扱います。

#include ＜名前＞

ここで、[名前]は拡張子つきの単純なファイル名または相対パス名を含むファイル名とすることができます。

#include ”名前”

ここで、[名前]は拡張子つきの単純なファイル名または相対パス名を含むファイル名または絶対パス名を含むファイル名とすることができます。

#include マクロ

ここで、[マクロ]は他の2つの形式のどちらかに展開されるものです。

H2cpyは指定されたファイルを探して、その中からCのソースコードを読み込みます。そのファイルの末尾にまで達すると、H2cpyは元のファイルの読込みを再開します。

#include指令を使用するときに、絶対パス名を指定する必要はありません。引用符（""）で囲んで絶対的ではないファイル名を指定すると、H2cpyはまず最初に現行ディレクトリのもとでそのファイルを探します。そこに求めるファイルが見つからないときは、H2cpyは続いてインクルード検索パスリスト中の各ディレクトリのもとを探します。

山かっこ（<>）で囲んでファイル名を指定すると、そのファイルはシステムヘッダーファイルであるとみなされます。そして、インクルード検索パスリスト中のディレクトリのもとでのみ、ファイルが探されます。現行ディレクトリはファイルを探す対象となりません。

インクルード検索パスリストには6つまでの絶対ディレクトリ名を登録することができます。H2cpyはその中のディレクトリのもとを順々に探していきます。そして、求めるファイルが見つかるか、リストの最後に達したときに、検索を終了します。結果としてファイルが見つからなかったときは、エラーメッセージが出されます。

インクルード検索パスリストの前にディレクトリを追加することができます。そのためには、「コマンド行オプション」の項で説明した-Iオプションを使用します。

インクルード検索パスリストは当初は空です。それに値を設定するには、INCLUDE環境変数を使用します。セミコロンで区切った14個までのディレクトリのリストをINCLUDE環境変数に設定することができます。

H2cpyはインクルード検索パス中のディレクトリ名の区切り文字は円記号（#）であるとみなします。ディレクトリ中にスラッシュ（/）があると、H2cpyはそれを円記号に置き換えてからファイルの検索を行います。

たとえば、INCLUDE環境変数を下記のように設定したとします。

include=d:#include;e:#sys#include

そして、下記の指令を使用するとします。

#include <abc#abc.h>

すると、H2cpyはまず最初に、d:#include#abc#abc.hを探します。それが見つからないと、H2cpyは次に、e:#sys#include#abc#abc.hを探します。

#define指令の翻訳

H2cpyは、パラメータを伴わないマクロだけではなく、パラメータを伴うマクロをも取り扱います。H2cpyはそれらのマクロを格納して、Cの原始コードの中にそのマクロ名が出てくると、その内容を展開します。マクロには任意の数のCのトークンを含めることができます。一般にマクロはCOBOLのコードに直接変換されません。

H2cpyは下記のマクロ定義をCOBOLの78レベルのデータ項目に翻訳します。

- 単一のCの文字列定数、文字、整数、浮動小数点定数
- 評価すると単一の整数定数となる数字定数式

前に同じCトークンを用いて両方の場合に定義されているマクロが出てくると、H2cpyはそれを無視します。定義が異なると、エラーメッセージが出されます。

以前に定義されているマクロ名の定義を取り消すために、#undef指令を使用することができます。

例：

Cのソースコードの一部

```
#define_H_STDLIB
#define XmNborderColor "borderColor"
#define CHAR_BIT 8
#define GMEM_MOVEABLE 0x2
#define GMEM_ZEROINIT 0x0040
#define GHND(GMEM_MOVEABLE ¦ GMEM_ZEROINIT)
#define VisualAllMask 0x1ff
#define INT_MAX(2147483647)
#define INT_MIN(-(INT_MAX + 1))
#define Button5Mask(k<<12)
#define HFILE int
#define HFILE_ERROR((HFILE)-1)
#define NULL((void *)0)
#define brdev(x) (x&0x7fffffff)
#define XT_CONVERT_FAIL(Atom)0x80000001
#define max(a,b)(((a)>(b)) ?(a): (b))
#define FLT_MAX 3.402823466385288e+38f
#define NBPW sizeof(int)
```

出力されるCOBOLのコード

```
78  XmNborderColor      value    "borderColor".
78  CHAR-BIT            value    8.
78  GMEM-MOVEABLE       value    h"02".
78  GMEM-ZEROINIT       value    h"0040"
78  GHND                value    66.
78  VisualAllMask       value    h"01ff".
78  INT-MAX             value    2147483647.
78  INT-MIN             value    -2147483648.
78  Button5Mask         value    4096.
78  FLT-MAX             value    3.402823466385288e+38.
```

生成されるCOBOLのコードは-fや-hなどのオプションの設定によって異なります。

マクロの展開と文字列定数の作成

H2cpyは、パラメータを伴うマクロを、文字列および結合演算子の#と##を使用して、展開します。たとえば、下記の#define指令があるとします。

```
#define DECLARE_HANDLE32(name) struct name##__{  int unused; };\
      typedef const struct name##__ _far* name
```

このマクロからはCOBOLの78レベルのデータ項目は生成されません。しかし、後でCのヘッダーが下記のようにこのマクロを呼び出したとします。

```
DECLARE_HANDLE32(HHOOK);
```

すると、このマクロはCの規則に従って展開され、それと同等のCOBOLのコードに翻訳されます。COBOLは、データポインタと手続きポインタを区別するとき以外は、ポインタを型定義することはありません。

```
01 HHOOK--        is typedef.
   02  unused                    usage int.
01 HHOOK          is typedef       usage data-pointer.
```

条件指令の翻訳

H2cpyは下記の条件指令を解釈します。それらの指令はネストさせることができます。

```
#if      #else      #endif
#elif    #ifdef     #ifndef
```

#if指令の後ろには、定数式が来なければなければなりません。式の中のマクロは展開され、定義されている演算子は評価され、未知の名前はゼロに置き換えられ、その結果の数字式が評価されて、数値が出されます。その結果がゼロであると、条件は偽と判定されます。そうでなければ、条件は真と判定されます。条件が偽であると、それに続く行は無視されます。そうでなければ、それらは翻訳されます。

#else指令と#endif指令は意味を切り替えて、先行する#ifと#ifdefと#ifndefの各指令の効力を停止させます。

#elif指令は#else指令の後ろに#if指令が続いているのと基本的に同じです。ただし、次の違いがあります。#elifの場合、後続の#ifによってネストのレベルは下がりません。また、その後ろに出てくる#endif指令は前に出てきている#if指令に対して適用されます。

#ifdefの後ろには名前を続けなければなりません。その名前が前にヘッダーファイル中または-Dオプションの結果として定義されていれば、条件は真と判定されます。そうでなければ、条件は偽と判定されます。条件が偽であると、それに続く行は無視されます。そうでなければ、それらは翻訳されます。

#ifndef指令は#ifdedf指令と同じように扱われます。ただし、その条件が逆になっています。

その他の指令の翻訳

#error指令の後ろに、その行上で、任意の注釈を続けることができます。-Cオプションを指定していると、H2cpyはその行全体を画面上に表示するとともに、出力ファイルにリストします。

先頭にシャープ記号（#）があるその他の行をH2cpyは無視します。

3.2.1.5　文の翻訳

H2cpyは定義と宣言に関する文を翻訳します。

H2cpyは上記以外のCの構成要素に関してはCOBOLコードを生成しません。

外部データ宣言

H2cpyは外部データとして定義されていないデータ宣言を無視します。Cのデータ宣言がexternと定義されていると、H2cpyはEXTERNAL指定を使用して、COBOLの宣言を生成します。そのようなデータ項目がCのモジュール内で宣言されていて、それとCOBOLのアプリケーションをリンクすることができる場合には、COBOLアプリケーションからそのデータ項目を参照できるようにします。

H2cpyは生成されたCOBOLコードを（.cpy）ファイルに出力します。ただし、外部データ宣言だけは別の（.ext）ファイルに出力します。

例：

Cの原始コードの一部

```
typedef struct WidegetClassRec *WidegetClass;
extern WindegetClass widgetClass;
```

生成されたCOBOLコード（.cpyファイルへの出力）

```
01 Widegetclass       is typedef        usage data-pointer
```

生成されたCOBOLコード（.extファイルへの出力）

```
01 1widgetclass       is external by “widgetClass”
                      usage data-pointer.
```

生成されたCOBOLコードにおいては、型定義の名前と外部データ項目の名前は別々でなければなりません。それが、外部データ項目の名前の前に1が追加されている理由です。Cの中で定義されている外部データ項目をリンカが解明するときは、記号名の"widgetClass"が使用されます。

型定義

文字が符号つきまたは符号なしと明示的に宣言されている場合、H2cpyはその文字を数字項目として扱います。文字が配列として定義されている場合、H2cpyはその文字をCOBOLの英数字項目として扱います。それ以外の場合も、H2cpyはその文字をCOBOLの英数字項目として扱います。2次元の配列からは、COBOLのOCCURS句が生成されます。

配列サイズが指定されていない配列は該当データ型へのポインタとして翻訳されます。

任意のCのデータ型へのポインタは型定義を通じてCOBOLのデータポインタにマップされます。同様に、Cの関数へのポインタはCOBOLの手続きポインタにマップされます。

例：

Cのソースコードの一部

```
typedef char t_slab[12], cnt, c_list[], c_table[12][5];
typedef unsigned char BYTE, *BYTE_ptr;
typedef t_slab *t_slab_ptr;
typedef void _near _cdecl _export _dmsEVENT(short ent_type);
typedef _dmsEVENT *fptr;
typedef void (*XtActionProc)(
#if NeedFunctionPrototypes
     Widget       /* widget */,
     Xevent*      /* event */,
     String*      /* params */,
     Cardinal*    /* num_params */
#endif
);
typedef XtActionProc* XtBoundActions;
```

生成されたCOBOLのコード

```
01  t-slab             is typedef     pic x(12)
01  cnt                is typedef     usage char.
01  c-list             is typedef     usage data-pointer.
01  c-talbe            is typedef.
    02  filler occures 5              pic x(12).
01  BYTE               is typedef     usage uns-char.
01  BYTE-ptr           is typedef     usage data-pointer.
01  t-slab-ptr         is typedef     usage data-pointer.
01  fptr               is typedef     usage proc-pointer.
01  XtActionProc       is typedef     usage proc-pointer.
01  XtBoundActions     is typedef     usage data-pointer.
```

同様のCOBOL型定義を生成する型定義の例をもう1つ示します。

Cのソースコードの一部

```
typedef signed int I, AI[12], *PI, I_Table[12][5];
```

生成されたCOBOLのコード

```
01  I            is typedef      usage int.
01  AI           is typedef.
    02  filler occure 12         usage int.
01  PI           is typedef      usage data-pointer.
01  I-Table      is typedef.
    02  filler occurs 5.
        03  filler occurs 12     usage int.
```

宣　言

構造体と共用体と列挙宣言には名前を付けても付けなくてもかまいません。それらは独立していてもかまいませんし、型定義の中または構造体または共用体の宣言の中に含まれていてもかまいません。

宣言に名前が付けられていないと、該当項目用のCOBOLコードが全面的に生成されます。

例：

Cの原始コードの一部

```
typedef struct fsid { long val[2]; } fsid_t;
typedef unsigned long ino_t;  /* inode number (filesystem)*/
typedef unsigned int uint_t;
#define FHSIZE       32
#define MAXFIDSZ    (FHSIZE - sizeof(fsid t) - sizeof(uint_t))
struct fileis {  /* this is for servers only! */
       uint_t  fid_len;
       ino_tf id_ino;
       uint_t  fid_gen;
       char fid_x[MAXFIDSZ - (sizeof(into_t) + 2) -
                                        sizeof(uint_t)];  };
```

生成されたCOBOLのコード

```
01  fsid             is typedef.
    02  val occurs 2                usage long.
01  fsid-t           is typedef    usage fsid.
01  ino-t            is typedef    usage uns-long.
01  uint-t           is typedef    usage uns-int.
78  FHSIZE           value 32.
01  fileid           is typedef.
    02  fid-len                     usage uns-int.
    02  fid-ino                     usage uns-long.
    02  fid-gen                     usage uns-int.
    02  fid-x                       pic x(10).
    02  filler                      pic x(2).
```

H2cpyはフィールドfid-ino（longで4バイトの境界に桁寄せ）が正しく桁寄せされるようにします。そのために、2バイトのfid-lenフィールドの後ろに、2バイトのFILLERが挿入されています。

宣言に名前が付けられていないと、該当項目用のCOBOLコードが全面的に生成されます。

例：

Cのソースコードの一部

```
typedef struct {
   int type;
   union {
          char b[20];
          short s[10];
          long l[5];
   } data;
} XclientMessageEvent;
```

生成されたCOBOLのコード

```
01  XClientMessageEvent    is typedef.
    02  ltype                          usage int.
    02  ldata.
        03  data.
            04  b                      pic x(20).
        03  filler redefines ldata.
            04  s occurs 10            usage short.
        03  filler redefines ldata.
```

```
    04  l occurs 5              usage long.
```

列挙型の名前つきの値に対しては、H2cpyは78レベルのデータ項目を生成します。構造体の中で列挙項目に名前が付けられていてもかまいません。また、構造体の中でビットフィールドが指定されていてもかまいません。

H2cpyはCOBOLコード中のデータ名の前に「l」をつけ加えます。「type」および「data」はCOBOLの予約語であり、これらの語がCOBOLデータ項目の名前として使用されると、コンパイラは構文チェックの時にエラーを出力します。

例：

Cの原始コードの一部

```
enum input_type { GPICK = 0x0001, GANK= 0x0002, GPFK = 0x0004,
                  FP_NOINBUF = 0x10000, FP_WTMAX = 0x20000 };
typedef struct qqel {           /* queue element*/
   enum input_type type;        /* enum input_type value */
   int time;                    /* timestamp */
   struct {
     unsigned bad_data :1;      /* extra data io failed */
     unsigned no_data  :1;      /* queue empty (for K_POLL) */
     unsigned reserved :6;      /* reserved  */
     char rsvd[3];              /* reserved  */
   } flags;
   int data_len;                /* length of appended data */
} q_qel;
```

生成されたCOBOLのコード

```
01  input-type    is tyedef          usage uns-int.
78  gpik                             value    1.
78  gank                             value    2.
78  gpfk                             value    4.
78  fp-noinbuf                       value    65536.
78  fp-wtmax                         value    131072.
01  qqel          is typedef.
    02  type                        usage input-type.
    02  time                        usage int.
    02  flags.
        03  bad-data.
            04  bad-data            usage uns-int.    *> 1 Bit
        03  filler redefines        bad-data.
            04  no-data             usage uns-int.    *> 1 Bit
        03  filler redefines        bad-data.
            04  reserved            usage uns-int.    *> 6 Bits
        03  rsvd                    pic x(3).
        03  filler                  pic x(1).
    02  data-len                    usage int.
01  q-qel         is typedef         usage qqel.
```

rsvdフィールドの長さは3バイトです。その前の8ビットには1バイトの記憶域が割り当てられたものと想定しています。実際、ANSI C標準では、ビットフィールドは整数項目内に取られるとしています。したがって、16ビットの領域が割り当てられます。そのために、H2cpyはflags構造体の末尾に1バイトを充てんしています。

関数プロトタイプの宣言

Cの関数プロトタイプには、戻り値の型が1つ指定されていなければならず、少なくとも1つのパラメータが指定されていなければなりません。パラメータが1つも指定されていない場合、その関数宣言は関数プロトタイプとはみなされません。したがって、それに対してはH2cpyはCOBOLのCALLプロトタイプを生成しません。戻り値の型が指定されていない場合は、int型であるとみなされます。

関数が値を何も返さない場合は、関数プロトタイプでは戻り値の型にvoidを使用します。H2cpyはそれを翻訳するときに、returning句を落します。関数がパラメータを何も取らないときは、関数プロトタイプにはvoid型のパラメータを1つ指定します。H2cpyはそれを翻訳するときに、using句を落すとともに、付随するBY

VALUE句またはBY REFERENCE句があればそれも落します。

例：

Cの原始コードの一部

```
extern XfontStruct *XloadQueryFont(
);
extern void XrmInitialoze(
);
extern char *XfetchBytes(
);
extern int (*XsetAfterFunction(
))();
extern XtAppContext XtCreateApplicationContext(
void
);
extern void XtInitializeWidgetClass(
WidgetClass
);
XmColorProc XmGetColorCalculation(void);
```

生成されたCOBOLのコード

```
entry "XtCreateApplicationContext"
    returning          data-pointer
    .
entry "XtIntializeWidgetClass" using
         by value      data-pointer
    .
entry "XmGetColorCalculation"
    returning          proc-pointer
    .
```

呼出し方式

何通りかの呼出し方式が使用される可能性があります。Cでは、呼出し方式はキーワードの_pascalおよび_cdeclによって区別されます。COBOLでは、手続き部より前の特殊名段落の中で呼出し方式を指定します。CALLプロトタイプ用の呼出し方式が前に使用したものから変わる場合、またはCALLプロトタイプが生成された後で型定義を生成する必要がある場合、新しいプログラムを宣言する必要があります。その場合、H2cpyはend-programレコードとprogram-idレコードを続けて生成します。

例：

Cの原始コードの一部

```
typedef HINSTANCE HMODULE;
DWORD _far _pascal GetVersion(void);
UINT _far _pascal GetFreeSystemResources(UINT);
#define GFSR_SYSTEMRESOURCES 0x0000
BOOL_far _pascal SetWinDebugInfo(const WINDEBUGINFO Far*
   Lpwdi);
void _far _cdecl DebugOutput(UINT flags, LPCSTR lpsz, …);
#define WDI_OPTIONS 0x0001
```

生成されたCOBOLのコード

```
01  HMODULE          is typedef    usage uns-int.
end program "c-typedefs".
program-id. "GetVersion" is external.
special-names. Call-convention 3 is pascal-convention.
procedure divition pascal-convention
   returning         uns-long
   .
entry "GetFreeSystemResources" using
        by value     uns-int
   returning         uns-int
   .
end prograsm "GetVersion".
program-id. "c-typedefs" is external.
special-names. call-convention 3 is pascal-convention.
78  GFSR-SYSTEMERESOURCES      value h"0000".
entry "SetWinDebugInfo" using
        by reference  windebuginfo
   returning          int
   .
end program "c-typedefs".
program-id. "DebugOutput" is extgernal.
special-names. call-convention 0 is c-convention.
procedure division c-convention using
        by value      uns-int
        by reference  any
        by value      any
   .
end program "DebugOutput".
program-id. "c-typedefs" is external.
special-names. call-convention 0 is c-convention.
78  WDI-OPTIONS                value h"0001".
```

H2cpyは省略時の外部プログラムであるc_typedefsの中でCOBOLコードの生成を開始します。このc_typedefsでは、省略時の呼出し方式が使用されます。次のGetVersionルーチンでは、別の呼出し方式であるPascalが使用されます。そこで、H2cpyは外部プログラム見出しを生成します。次に呼出しプロトタイプが来ます。そのために、H2cpyは手続き部のENTRY文を生成します。その次に、データ部中に78レベルのデータ項目を生成する必要があります。そこで、H2cpyは別のプログラム見出しを生成します。その次に生成されるのは手続き部の一部なので、プログラム見出しは必要はありません。DebugOutputルーチンでは呼出し方式のCが使用されます。したがって、別のプログラム見出しが必要となります。

呼出しパラメータ

Cでは、パラメータは概念的にスタックを経由して渡されます。パラメータは直接的に渡すことも、ポインタを通じて間接的に渡すこともできます。COBOLでは現在のところ4バイトを超える項目をスタックを通じて直接渡すことは許されていません。構造体または共用体をスタックを通じて渡すことは、Cの悪いプログラミング慣習と考えられています。そのような項目がポインタのサイズ（4バイト）よりも大きいと、非効率的でもあります。通常はCOBOLの制限が問題となることはありません。H2cpyはそのようなコードを検出すると、別のパラメータを追加して、スタックの完全性が保たれるようにします。

intやcharやpointerのような簡単なCのデータ型は容易にCOBOLに翻訳することができます。スタックを通じてパラメータを直接渡されるときは、H2cpyはBY VALUE句を生成します。Cではまた、ポインタを型定義して、パラメータを間接的に渡す手段として使用することができます。COBOLでも間接方式を取ることができますが、1レベルだけに限られます。型定義されたポインタが出てくると、H2cpyはBY REFERENCE句を生成します。

例：

Cの原始コードの一部

```
typedef unsigned char BYTE;
typedef char _far* LPSTR;
typedef int Hfile;
typedef Boolean  (*XtfilePredicate) ( /* String filename */ );
BYTE _far _pascal GetTempDrive (char);
long _far _pascal _llseek (HFILE, long, int) ;
long _far _pascal _hwrite (HFILE, const void _huge*, long );
LPSTR _far _pascal AnsiLower (LPSTR );
void _far _pascal Throw (const int _far*, int );
extern String XtFindFile(
     CONST String          /* path */,
     Substitution          /* substitutions */,
     Cardinal              /* num_substitutions */,
     XtFilePredicate       /* predicate */
);
```

生成されたCOBOLのコード

```
01  BYTE              is typedef    usage uns-char.
01  LPSTR             is typedef    usage data-pointer.
01  HFILE             is typedef    usage int.
01  XtFilePredicate   is typedef    usage proc-pointer.
entry "GetTempDrive" using
        by value      char
    returning         uns-char
    .
entry "_llseek" using
       by value      int
       by value      long
       by value      int
    returning        long
    .
entry "_hwrite" using
       by value      int
       by value      data-pointer
       by value      long
    returning        long
    .
entry "Ansilower" using
```

```
           by reference  any
       returning         data-pointer
   .
   entry "Throw" using
           by reference  int
           by value      int
       .
   entry "XtFindFile" using
           by reference  any
           by value      data-pointer
           by value      uns-int
           by value      proc-pointer
       returning             data-pointer
       .
```

_hwriteルーチンにおいては、voidを指すポインタからは値によって渡されるデータポインタが生成されていることに注意してください。それに対して、AnsiLowerルーチンでは、文字へのポインタであるLPSRT型からBY REFERENCE句が任意のパラメータが許される形で生成されています。通常はCOBOLの任意の英数字項目が適するので、charは限定的すぎます。したがって、H2cpyはanyを生成します。

Throwルーチンにおいては、値によって渡されるintと参照によって渡されるintとが使われています。XtFindFileルーチンにおいては、手続きポインタが値によって渡されています。

例：

Cの原始コードの一部

```
typedef int (_far _pascal* FARPROC)( );
typedef struct tagSEGINFO
{
    UINT offSegment;
    UINT cbSegment;
    UINT flags;
    UINT cbAlloc;
    HGLOBAL h;
    UINT alignShift;
    UINT reserved[2];
} SEGINFO;
typedef SEGINFO _ far* LPSEGINFO;
typedef struct tagPOINT
{
    int x;
    int y;
} POINT;
void _far _pascal GetCodeInfo(FARPROC lpProc, SEGINFO _far*
                              lpSegInfo);
BOOL _far _pascal PtInRect(const RECT _far*, POINT);
void _far _pascal ClientToScreen(HWND, POINT _far*);
```

生成されたCOBOLのコード

```
01  FARPROC      is typedef          usage proc-proc-pointer.
01  tagSEGINFO   is typedef.
    02  offsegment                  usage uns-int.
    02  cbsegment                   usage uns-int.
    02  flags                       usage uns-int.
    02  cballoc                     usage uns-int.
    02  h                           usage uns-int.
```

```
    02  alignshift                  usage uns-int.
    02  reserved occurs 2           usage uns-int.
01  SEGINFO            is typedef   usage tagseginfo.

01  LPSEGINFO          is typeder   usage data-pointer.
01  tagPOINT           is typedef.
    02  x                           usage int.
    02  y                           usage int.

01  POINT              is typeder   usage tagpoint.
entry "GetCodeInfo" using
        by value     proc-pointer
        by reference seginfo
    .
entry "PtInRect" using
        by reference rect
        by value     point
    returning        int
    .
entry "ClientToScreen" using
        by value     uns-int
        by reference point
```

データ型POINTのパラメータが値によっても参照によっても渡されています。

制　約

H2cpyはパラメータを伴うマクロを格納します。それは、後でそのマクロが展開されることに備えるためです。パラメータを伴うマクロの定義が物理的に2行ないし3行にわたる場合、H2cpyにはそのマクロの内容をすべて格納するのに十分なスペースがないことがありえます。そのような状態が発生した場合には、エラーメッセージが出されます。

H2cpyによるヘッダーファイルの前処理がCコンパイラによる前処理と同じではないという問題が発生したとします。その問題に対処するには、Cコンパイラを使用してヘッダーファイルの前処理だけを行い、その結果をH2cpyへの入力に使用します。

COBOLとは異なり、C言語のデータ型は複数の環境間で移植することは容易ではありません。そのために、同じライブラリを異なるマクロ間または異なるオペレーティングシステム間でインタフェースする必要があるときは、H2cpyを2回以上実行する必要がある場合があります。

3.2.1.6　H2cpyのメッセージ

H2cpyは3種類のメッセージを出します。具体的には、それらは情報メッセージと警告メッセージとエラーメッセージです。情報メッセージは何が起っているかをユーザーに知らせます。警告メッセージは、翻訳されたコードをコンパイルしてもエラーにはならないが、ユーザーが期待するとおりの結果が得られない可能性があることを示します。エラーメッセージは、何か間違いがあって、生成されたコードを正しくコンパイルできないことを示します。

注：

H2cpyはまた、内部エラーメッセージをいくつか発することがあります。それらのメッセージには内部メッセージであることを示すフラグが付けられています。通常は内部エラーメッセージが出されることはありません。もし、そのようなエラーメッセージが出された場合には、Cヘッダーファイルに含まれているのは有効なANSI C原始コードだけであることをチェックしてください。その内容が正しい場合には、貴社のテクニカルサポート担当者に連絡してください。

警告メッセージ

警告メッセージの前には必ずアスタリスク（*）が付けられています。警告メッセージは、何らかの理由で処理できない異常なC原始コードが見つかったので、H2cpyはその部分を無視したことを意味します。無視されたCの原始コードがCOBOLコピーファイルの生成または使用に直接影響することはありません。しかし、COBOLコピーファイルの生成または使用に間接的に影響する可能性があります。また、Cヘッダーファイルが正しくないことを示している可能性もあります。H2cpyから出される警告メッセージには下記のものがあります。

＊警告．[名前]に他の戻り値が既に定義されています

説明：　前に定義されている関数プロトタイプと、返されるデータ型が合致しない関数[名前]の宣言が出てきました。新しい宣言は無視されます。

＊警告．浮動小数点はゼロとして評価されます

説明：　数式の中に浮動小数点定数が出てきました。その数式を評価するにあたって、浮動小数点定数は値をゼロとして扱いました。

＊警告．表現バッファのオーバーフロー―無視されます

説明：　H2cpyが処理するには長くて複雑すぎる数式が出てきました。その数式は無視されます。Cの原始コードをチェックして、COBOLに翻訳する必要のあるCの定義の中でその数式を使用しないようにしてください。

＊警告．マクロが長すぎます、無視します：［名前］

説明：　数行にまたがりH2cpyが処理するには長すぎるマクロ［名前］の定義が出てきました。そのマクロ定義は無視されます。一般に、そのマクロ定義はヘッダーファイルの中で再び参照されることはありません。したがって、COBOLへの翻訳に影響が出ることはないでしょう。

＊警告．マクロにパラメータが多すぎます

説明：　非常に多くのパラメータを指定するマクロ定義が出てきました。そのマクロ定義は無視されます。一般に、そのマクロ定義はヘッダーファイルの中で再び参照されることはありません。したがって、COBOLへの翻訳に影響が出ることはないでしょう。

＊警告．マクロのパラメータが長すぎます―無視されます

説明：　非常に長いパラメータを指定するマクロ定義が出てきました。そのマクロ定義は無視されます。一般に、そのマクロ定義はヘッダーファイルの中で再び参照されることはありません。したがって、COBOLへの翻訳に影響が出ることはないでしょう。

＊警告.　**#define**の後に空白もかっこもありません、本行を無視します

説明：　Cヘッダーファイル中の#defineの形式がANSI Cの仕様から外れています。問題の行は無視されます。CヘッダーファイルをANSI Cに準拠するように修正してから、翻訳されたCOBOLコードを使用するようにしてください。

＊警告.　**#define**の再定義は無視します

説明：　#defineを使用してマクロを定義するにあたって、前と同じ形で定義されていないものが出てきました。そのようなCヘッダーファイルはANSI Cの仕様から外れています。問題の行は無視されます。CヘッダーファイルをANSI Cに準拠するように修正してから、翻訳されたCOBOLコードを使用するようにしてください。

＊警告.　**passed by value**で渡される5バイト以上の構造体/共用体にCOBOLから直接アクセスできません

説明：　Cでは4バイトよりも長いデータ項目を値によって渡すことが許されています。しかし、その実行時の効果は作成者の設計に依存します。したがって、そのデータ項目をCOBOLから直接アクセスすることはできません。そのようなコーディングの仕方は良くないと、Cのプログラマによって考えられています。そのようなCヘッダーファイルは修正すべきです。

エラーメッセージ　―　コマンド行エラー

下に示すエラーメッセージは、H2cpyに対するコマンド行が無効であり、翻訳は行われないことを意味します。エラーを修正してから、H2cpyを実行し直さなければなりません。

エラー．コマンド行が長すぎます

説明：　コマンド行にたくさん指定しすぎています。コマンド行上に[ファイル名]または-D[宣言名]オプションを指定する代りに、#include "ファイル名"および#define[宣言名]を使用して、小さなCヘッダーファイル中にヘッダーファイル名およびマクロ宣言を指定するようにしてください。

エラー．　**C**ヘッダファイルの指定がありません

説明：　入力ヘッダーファイルを指定し忘れています。

エラー．　入力ファイル名が多すぎます

説明：　コマンド行上にファイル名をたくさん指定しすぎています。Cヘッダーファイルを作成して、各ファイル名を指定した#includeを使用するようにしてください。

エラー．　マクロ［名前］は既に定義されています

説明：　2つの-Dオプションに同じ[名前]を指定しています。

エラー．　**-I**オプションに最大数を超えるディレクトリを指定しています、最大数=14

説明：　-Iオプションを指定した結果、インクルード検索リストがオーバフローしました。指定した-Iオプションの数を減らすか、INCLUDE環境変数中に指定されているディレクトリの数を減らすかしてください。

エラー．指定されたCの型を認識できません

説明：　-aオプションまたは-sオプションに指定されたパラメータが正しくありません。

エラー．**C**の型に割り当てた値が誤っています

説明：　-aオプションに指定されたパラメータが正しくありません。

エラー．**C**の型に誤ったサイズを指定しています

説明：　-sオプションに指定されたパラメータが正しくありません。

エラー．環境の指定に誤りがあります（取り得る値は16か32です）

説明：　-eオプションに指定されたパラメータが正しくありません。

エラー．オプション［フラグ］を認識できません

説明：　　フラグに指定された文字が正しくありません。

エラーメッセージ　－　無効な**C**ソースコード

下に示すエラーメッセージは、H2cpyに入力されたCヘッダーファイルにANSIの仕様から外れる無効なCソースコードが含まれていることを意味します。翻訳は行われません。有効なANSI

Cソースコードだけが含まれるようにヘッダーファイルを修正してから、H2cpyを使用してそれをCOBOLに翻訳し直さなければなりません。エラーの原因を正確に突き止めるには、該当するCの方言をチェックするように設定したC開発ツールを使用する必要があります。

エラー．型が必要です、使用した名前：［名前］
エラー．型が必要です
エラー．内部レベルでtypedefを検出しました、かっこをチェックしてください
予期しないトークン［トークン］
エラー．型が必要です
エラー．型が必要です、［名前］を検出しました
エラー．一意のtypedef名がありません
エラー．この名前は既に宣言されています
エラー．enum［名前］は宣言されていません
エラー．既に定義された名前：［名前］
エラー．［名前］で構造体/共用体を既に定義しています
エラー．［名前］他のパラメータが既に定義されています
エラー．宣言されたバッファのオーバーフロー
エラー．一意の宣言名がありません
エラー．一意のパラメータ名がありません
エラー．Cコードが無効です（トークンがありません）
エラー．［名前］の中に4バイトには多すぎる数字があります
エラー．##の後に正しいトークンの型が続いていません
エラー．#if行内の左右のかっこの数が合いません

第4章:　ランタイム構成

4.1　概　説

この章は2つの部分に分けられます。最初の部分では、ランタイム構成を調整する要素を使用して実行時のシステムの働きを調整する方法について説明します。その中で、構成ファイルとは何かということと、構成ファイルに対して行うことのできる2通りの調整手段について、詳しく説明します。2番目の部分では、使用できる環境変数とランタイムチューナーを列挙して説明します。

注：

ランタイムチューナーには移植性があると思わないでください。それらはWindows 32bit以外の環境ではサポートされていない可能性があります。

4.2　操　作

アプリケーションを起動すると、COBOLランタイム環境を初期化する一環として、ランタイム構成ファイルが読み込まれます。このランタイム構成ファイルはこのMicro Focus COBOLシステムのすべてのユーザーによって共有されます。

4.2.1　ランタイム構成ファイル

ランタイム構成ファイルはテキストファイルです。したがって、標準のテキストエディタを使用して編集することができます。これは$COBDIR#cobopt.cfgという名前のファイルです。このファイルを使用するかしないかは任意です（このファイルが存在しなくとも、エラーにはなりません）。

各ランタイム構成調整要素は新しい行に記します。空白の行は無視されます。注釈行は先頭に#を付けます（このファイルは実行時に処理されるので、注釈はあまり付けないように勧めます。そうすれば、その処理に要する時間が最小限に抑えられます）。1行の注釈行に含められる文字数は80です。その中には、改行文字および復帰文字も含まれます。エディタの中には、行末にそれらの文字を付加するものがあります。

構成ファイルの中では、下記の2通りの構成を行うことができます。

- 環境変数を設定する。dd_ファイル名のマップの指定はその一例です。その他に、Micro Focus COBOLにとって意味のある環境変数を設定します。
- ランタイムチューナーを設定する。ランタイムチューナーはCOBOLランタイムシステムのある種の働きを制御するものです。

4.2.2　環境変数の設定

行の先頭にキーワードの"setenv"を置き、その後ろに下記の要素を順に続けます。

1. 1つ以上のタブまたは空白
2. 環境変数の名前
3. 1つ以上のタブまたは空白
4. 環境変数に割り当てる値（単一の語または引用符で囲んだ複数の語）。ここで設定した環境変数はすでに設定されているものにとって代わります。

例：

```
setenv dd_MYFILE d:¥mydir¥myfile
setenv SOMESHORTNAMES "JOE FRED JACK MARY"
```

dd_ファイル名のマップの詳細については、「COBOLファイル処理」の章を参照してください。

4.2.3　ランタイムチューナーの設定

行の先頭にキーワードの"set"を置き、その後ろに下記の要素を順に続けます。

5. 1つ以上のタブまたは空白
2. ランタイムチューナーの名前
3. "="
4. ランタイムチューナーに設定する値。この値は該当ランタイムチューナーに許される型と値およびcobconfig_error_reportの値に基づいて、妥当性を検査されます。次の節に、ランタイムチューナーの名前とそれに設定することのできる値を詳しく説明します。

4.3　ランタイムチューナー

このCOBOLシステムで利用可能なランタイムチューナーを以下に列挙して説明します。

4.3.1　cobconfig_error_report

構成ファイルの処理中に検出されたエラーを報告するかどうかを指定します。

構文：　**set cobconfig_error_report= {TRUE|FALSE}**

値：　TRUE: エラーを報告します。
FALSE: エラーを報告しません。

省略値：　TRUE

説明：　TRUEが指定されると、エラーが構成ファイルで検出された場合に報告され、構成ファイルの処理は中断します。

4.3.2 command_line_linkage

プログラムを呼び出して、コマンド行をLinkage Sectionを介してアクセスされたパラメータとして、メインプログラムに渡すことができるようになります。

構文： **set command_line_linkage={TRUE|FALSE}**

値： TRUE: Linkage Sectionを介してコマンド行にアクセスできます。
FALSE: Linkage Sectionを介してコマンド行にアクセスできません。

省略値： FALSE。

4.3.3 environment_mapper

外部ファイルマッパ(Mfextmap)を使用可能にするかどうかを指定します。外部ファイルマッパを使うと環境変数への割り当てをファイル中で行えます。外部ファイルマッパが使用可能になると、システムはファイルmfextmap.datを探します。

構文： **set environment_mapper={TRUE|FALSE}**

値： TRUE: 外部ファイルマッパ(Mfextmap)が使用可能になります。
FALSE: 外部ファイルマッパ(Mfextmap)を使用不可能になります。

省略値： FALSE

4.3.4 isam_block_size

索引ファイルの作成時に使用するブロックサイズを選択します。

構文： **set isam_block_size={512|1024|2048|4096}**

値： 512 : 1ブロックを2分の1 Kbyteとしてファイルを作成します。
1024: 1ブロックを1 Kbyteとしてファイルを作成します。
2048: 1ブロックを2 Kbyteとしてファイルを作成します。
4096: 1ブロックを3 Kbyteとしてファイルを作成します。

省略値： 1024

4.3.5 isam_open_key_check

OPEN処理中にプログラムで指定されたキー指定が既存のファイルのそれと一致するかどうかのチェックを行うかを指定します。

構文： **set isam_open_key_check={TRUE|FALSE}**

値： TRUE:キー指定をチェックします。
FALSE:キー指定をチェックしません。

省略値： FALSE

4.3.6 lock_mode

使用するロック方法を選択します。

構文： **set lock_mode={0|1|2}**

値： 0: Micro Focus COBOL V3.1以前で提供されているロック機構を使用します。
1: DOSおよびOS/2のロック機能との互換性を保ちます。
2: 中間言語のロック機構を提供します。この方法ではレコードロックにより物理レコードがロックされます。

省略値： 1

4.3.7 printer_redirection

COBOLのWRITE文や報告書作成機能によるプリンタへの印刷の方法を制御します。日本語を印字するためにはこの指定をTRUEにする必要があります。

構文： **set printer_redirection= {TRUE|FALSE}**

値： TRUE:すべてのプリンタへの出力はプリントマネージャへリダイレクトされます。
FALSE:プリンタへのOSの通常の方式で行われます。

省略値： FALSE

4.3.8 same_proc_excl_detection

1つの実行単位において1つのファイルが複数回開かれようとしたことを検出します。

構文： **set same_proc_excl_detection= {TRUE|FALSE}**

値： TRUE:ファイルが開かれようとしたときに、そのファイルがすでに開かれているか否かがチェックされます。すでに開かれている場合は、さらにそのファイルが排他的に開かれているか否かがチェックされます。そのファイルが排他的に開かれていれば、ファイルを開く要求はエラーとされます。
FALSE:条件は適用されません。

省略値： FALSE

説明： 現在のところ、同じファイルが同じ実行単位によって2回開かれても、ファイルエラーは発生しません。

4.3.9 skip_on_lock

NODETECTLOCK指令を指定してコンパイルされたプログラム中でロックされているレコードのREAD WITHLOCK文が、それに続くREAD NEXT/PREVIOUS文に論理レコードNEXT/PREVIOUSまたは現在ロックされているレコードを再読み込みさせるかどうかを指定します。

構文： **set skip_on_lock= {TRUE|FALSE}**

値： TRUE:レコードが読み込まれます。
FALSE:レコードは読み込まれません

省略値： FALSE

4.4　環境変数

利用できるランタイム環境変数には下記のものがあります。

4.4.1　COBCONFIG

COBCONFIGはユーザー独自のランタイム構成ファイルを指します。ユーザー独自のニーズに適合するように、ランタイム構成調整要素に何らかの調整を加える場合に、この環境変数を使用します。

構文：　　**set COBCONFIG=パス名**

パラメータ： パス名　　ユーザーの構成ファイルの名前です。

第5章:　ランタイムスイッチの説明

5.1　概説 - ランタイムスイッチ

プログラムを実行するときにその動作を制御するスイッチを設定することができます。これらはランタイムスイッチ、またはランタイムシステムスイッチと呼ばれます。このスイッチの大部分は、COBOL機能の動作に影響を与えます。また、プログラムによって明示的に参照されるスイッチもあります。これらは、COBOLプログラムスイッチと呼ばれます。

設定可能なランタイムスイッチには、次の2種類があります。

「5.2　ランタイムスイッチ一般」
「5.3　OOプログラムのランタイムスイッチ」

5.2　ランタイムスイッチ一般

コマンド行または環境変数COBSWのどちらかにランタイムスイッチを入力してプログラムを実行することができます。ランタイムスイッチは、サブプログラムの実行中も設定されたままです。コマンド行でランタイムスイッチを設定すると、COBSWの設定よりも優先されます。

コマンド行におけるランタイムスイッチの構文は次のようになります。

```
({ {+|-|/}s } ...)
```

COBSWでランタイムスイッチを設定するには、次のようにオペレーティングシステムのコマンドを使用します。

```
set COBSW={ {+|-|/}s } ...
```

以下にこのコマンドの説明を示します。

{ | }	|で区切られた項目から1つを選択します。
{ }...	この項目を繰り返し設定できます。
s	ランタイムスイッチ。数字、文字（大文字または小文字）、または0から7までの数字を後に付けた文字を設定できます。文字だけと0が後ろに付いた文字とは同義になります。たとえば、B、b、B0、b0はすべて同じスイッチを指します。
+	スイッチをオンにします。
-	スイッチをオフにします。

たとえば、次のコマンドはランタイムスイッチTをオンにし、Oをオフにしたプログラムmyprogをロードします。

```
myprog (+T-O)
```

「5.4　各ランタイムスイッチの説明」
「5.4.1　凡例」

5.3　OOプログラムのランタイムスイッチ

OOプログラムだけで使用できるランタイムスイッチがあります。OOランタイムスイッチはすべて環境変数OOSWに設定します。スイッチをオフにする場合はマイナス記号（-）を、オンにする場合はプラス記号（+）をスイッチの前に記述します。複数のスイッチ設定をつなげて設定することもできます。

例えば、

set oosw=-v+d

このように設定すると、vスイッチはオフに、dスイッチはオンになります。スイッチ名の大文字小文字は区別されます。

「5.4　各ランタイムスイッチの説明」
「5.4.1　凡例」

5.4　各ランタイムスイッチの説明

各ランタイムスイッチに対して次の内容を説明します。

関数:

ランタイムスイッチが実行する関数

属性:

省略値:　スイッチの省略時の設定

タイプ:　スイッチが一般のランタイムシステムスイッチかOOプログラム（OO）用のランタイムシステムスイッチかを示します。

説明:

スイッチに関する追加情報

5.4.1　凡例

次の一般的なランタイムスイッチが使用できます

0,1,2,...8	プログラムスイッチ
A1	末尾の空白のDISPLAYスイッチ
B, B1	ロックされたレコードの飛越しスイッチ
C4	43行モードスイッチ
C5	50行モードスイッチ
D	ANSI COBOLデバッグスイッチ
E	エラースイッチ
F	数字項目検査スイッチ

K4　　エラーメッセージの表示維持スイッチ

L1　　ポップアップウィンドウの無効化スイッチ

L2　　レコード区切り文字設定スイッチ

N　　空スイッチ

S　　重複キー順整列スイッチ

S5　　ANSI.SYS互換操作卓I/Oスイッチ

T　　タブスイッチ

次のランタイムスイッチは、OOプログラムで使用できます。

d　　デバッグスイッチ

f　　トレースファイル名設定スイッチ

g1　　目的記憶域節前の監視ページ設定スイッチ

g2　　目的記憶域節後の監視ページ設定スイッチ

l　　クラスライブラリのロードスイッチ

t　　トレースファイル作成スイッチ

5.4.2　プログラムスイッチ 0,1,2,3,4,5,6,7 and 8

プログラムを入力するときに、関連するCOBOLスイッチを設定します。

属性:

省略値:　すべてオフ
タイプ:　一般

説明:

プログラムスイッチはプログラマがプログラムの特殊名段落に定義するCOBOLスイッチの0から8を有効にします（詳細については、『言語リファレンス』を参照してください）。これらのスイッチを指定する順序は任意です。ただし、各スイッチの前に＋またはーを付けなければなりません。

5.4.3　末尾の空白のDISPLAYスイッチ（A1）

DISPLAY ... UPON CONSOLE文で画面上に表示するデータの末尾の空白文字を削除します。

属性:

省略値:　オフ
タイプ:　一般

説明:

このスイッチは旧版のCOBOL製品との互換性のために用意されています。このスイッチがオフのとき、ANSI COBOLの要請に従って末尾の空白を削除しません。

5.4.4　ロックされたレコードの飛越しスイッチ（B，B1）

READ NEXT文を使用するときに、ロックされているレコードがあれば、それらを飛び越してレコードポインタを更新できるようにします。

属性:

省略値:　オフ
タイプ:　一般

説明:

Bスイッチは入力用または入出力用に開かれた索引順ファイルに対して設定します。B1スイッチは入力用に開かれた索引順ファイルに対して設定します。これらのスイッチのどちらかを設定すると、他のプロセスが該当ファイル内のレコードにロックを掛けている場合でも、そのファイルを順呼出しで読むことができます。

Bスイッチが設定されているときに、ロックが掛けられているレコードを読むと、「レコードがロックされている」というファイル状態が返されます。

B1スイッチが設定されているときに、ロックが掛けられているレコードを読むと、ファイル状態00が返されます。

注:

このスイッチは、CALLFH指令を設定してプログラムをコンパイルした場合は、関連するレコード順ファイルに影響を与えます。

5.4.5　43行モードスイッチ（C4）

EGAまたはVGAモニター、またはCOBOLテキストウィンドウにおいて、画面を強制的に43行モードにします。

属性:

省略値:　オフ
タイプ:　一般

説明:

オペレーティングシステムに戻ったときに、元の画面モードに復帰します。
このスイッチを設定すると、コンポーネントは43行モードで起動されます。

5.4.6　50行モードスイッチ（C5）

EGAまたはVGAモニター、またはCOBOLテキストウィンドウにおいて、画面を強制的に50行モードにします。。

属性:

省略値:　オフ
タイプ:　一般

説明:

オペレーティングシステムに戻ったときに、元の画面モードに復帰します。
このスイッチを設定すると、コンポーネントは50行モードで起動されます。

5.4.7 ANSI COBOLデバッグスイッチ（D）

標準のANSI COBOLデバッグモジュールを呼び出します。

属性:

省略値: オフ
タイプ: 一般

説明:

このスイッチはコマンド行上で指定しなければなりません。

例:

myprog （+D）

このように設定すると、プログラムmyprogがロードされるときに、標準のANSI COBOLデバッグモジュールも併せて呼び出されます。

5.4.8 デバッグスイッチ（d）

終了したオブジェクトからのオブジェクトハンドルを再利用しないようにすることでデバッグを可能にします。

属性:

省略値: オフ
タイプ: 00

説明:

通常は、オブジェクトが終了するごとにそのオブジェクトハンドルがランタイムシステムで再利用可能になり、新しいオブジェクトを作成するときに割り当てることができます。+dを設定すると、ランタイムシステムはオブジェクトハンドルの再割当ができなくなります。

このスイッチをアプリケーションを作成するために使用しないでください。それは、ランタイムシステムが新しいハンドルを割り当てるためのメモリを使い果たしてしまう可能性があるからです。

5.4.9 エラースイッチ（E）

中間コードに「重大」レベルのコンパイルエラーがあっても実行されるようにします。

属性:

省略値: オフ
タイプ: 一般

説明:

このスイッチがオフに設定されているときに、「重大」レベルのコンパイルエラーが含まれる中間コードのプログラムを実行しようとすると、ランタイムシステムエラーが発生します。

5.4.10　数字項目検査スイッチ（F）

中間コードを処理するときに、数字データ項目の妥当性をチェックします。

属性:

省略値:　オン
タイプ:　一般

説明:

有効でない数字データ項目が検出されると、ランタイムエラー163 "数字項目に違法な文字がある"が発生します。このスイッチをオフに設定すると、このチェックは抑制されます。

このスイッチはCHECKNUMコンパイラ指令で作成された生成コードと中間コードに対してのみ有効です。

ランタイムシステムおよび生成コードは多種類の数字操作を最適化しようとします。これらの操作は、数字作用対象に有効な数字データが含まれていない場合に不正な結果を出すことがあります。省略時は、数字操作が不正な数字データにより実行されるとランタイムシステムがランタイムシステムエラー163を出力します。

中間コードおよび元のコードによる不正なデータの処理方法は異なります。プログラムの開発中は数字フィールド使用可能にすることをお勧めします。

ランタイムエラー163が発生した場合は、不正なデータを使用しないようにコードを修正することをお勧めします。+Fスイッチを設定してプログラムをアニメートすると、ランタイムシステムがエラー163を検出するまでコードに不正な数字データを入れておくことができます。

5.4.11　トレースファイル名設定スイッチ（f）

メッセージトレースファイルの名前を付けることを可能にします。

属性:

省略値:　オフ
タイプ:　OO

説明:

このスイッチの形式は、次のようになります。

f*trace.log*

*ftrace.log*はメッセージトレースファイルの名前です。省略時のトレースファイルの名前は、trace.logです。

5.4.12　目的記憶域節前の監視ページ設定スイッチ（g1）

目的記憶域節の前に監視ページを設定します。

属性:

省略値: オフ
タイプ: OO

説明:

+g1+g2を設定することはできません。

5.4.13 目的記憶域節後の監視ページ設定スイッチ（g2）

目的記憶域節の後に監視ページを設定します。

属性:

省略値: オフ
タイプ: OO

説明:

+g1+g2を設定することはできません。

5.4.14 エラーメッセージの表示維持スイッチ（K4）

ランタイムシステムエラーメッセージをキーが押されるまで画面上に表示させたままにします。

属性:

省略値: オフ
タイプ: 一般

説明:

このスイッチは、ランタイムシステムエラーメッセージによりバッチランのどのプログラムが中断したのかを調べる場合、またはにメニューを表示するシステムで役に立ちます。それは、ランタイムシステムエラーメッセージが即書き換えられることがあるからです。

5.4.15 クラスライブラリのロードスイッチ（l）

ランタイムシステムが事前にロードするクラスライブラリを設定します。

属性:

省略値: オン
タイプ: OO

説明:

このスイッチの形式は、次のようになります。

l*class*

*class*は、事前にロードするクラスの名前です。

このスイッチをオフにすると、クラスライブラリのロードができません。クラスライブラリをデバッグする場合には、オフにするべきです。

ユーザ独自のクラスライブラリが.dllファイルにある場合には、Micro Focusライブラリの代りにそのライブラリをロードすることができます。次のように設定します。

+l*filename*

*filename*は、.dllファイルの名前です。 また、+lスイッチにより複数のクラスライブラリをロードすることもできます。この場合は、次のように名前をセミコロン（;）で区切ります。

+l*class1;class2*

名前の間にスペースを入れないでください。

5.4.16　ポップアップウィンドウの無効化スイッチ（L1）

例外エラーが発生した場合に表示される対話型ポップアップウィンドウを使用不可能にします。

属性:

省略値:　オフ
タイプ:　一般

説明:

対話型ポップアップウィンドウの代りに、一般的なWindowsのエラーメッセージが表示され、制御はユーザの介在なしにオペレーティングシステムに戻されます。

5.4.17　レコード区切り文字設定スイッチ（L2）

行順ファイルを読み込むときにレコード区切り文字として使用する文字を決めます。

属性:

省略値:　オフ
タイプ:　一般

説明:

このスイッチをオフにすると（-L2を設定）、x"0A"がレコード区切り文字とされます。

このスイッチをオンにすると（+L2を設定）、x"0D0A"がレコード区切り文字とされます。これによりWRITE...AFTER ADVANCING文を使用して書かれたファイルを正確に読むことができるので、Micro Focus COBOLの初期バージョンとの互換性が保たれます。

このスイッチは、読み込み操作だけに効果があり、書込み操作には効果がありません。

5.4.18 空スイッチ（N）

プログラム中のすべての行順ファイルに空文字を挿入できるようにします。

属性:

省略値: オン
タイプ: 一般

説明:

このスイッチはファイルの形式がこのCOBOLシステムのバージョンと互換性がない場合に特に役に立ちます。

このスイッチをオンに設定すると、プログラムが行順ファイルにレコードを書き出すときに、レコード内に制御文字（つまり、ASCIIコードでx"1F"以下のすべての文字）が含まれていると、各制御文字の前に空文字（x"00"）が付け加えられます。同様に、行順ファイルからレコードを読み込むときには、制御文字からそれらの空文字が除去されます。

このスイッチの設定は行順ファイル中にデータを物理的に格納することにのみ影響します。書き出したときと同じ設定で読み込まれたレコードの内容は元通りになります。他方、書き出したときとは違う設定でレコードを読み込むと、結果はどうなるか分からないので、注意してください。

このスイッチをオフに設定すると、制御文字を他の文字と同様に読み書きすることができます。

このスイッチがオンであるときの処理は下記のようになります。

- 行順ファイルのデータレコード中に任意の文字を含めることができます。
- 各レコードは書き出される前に検査されます。値がx"1F"以下の文字はすべて、前に空文字（x"00"）が付加されて2バイトの組として書き出されます。ディスクに出力する際に、後行の空白は削除されて、レコードの末尾を示すためにレコード終了文字（x"0D0A"）が付加されます。
- ファイルはレコードとして読み込まれます。レコードの中には、最初のバイトが空文字である2バイトの組を含むものもあります。レコードはレコード区切り文字によって区切られています。レコード中の、最初のバイトが空文字である2バイトの組からは空文字が除去されます。2バイトの組を構成しないタブ（x"09"）は拡張されます。

注：

制御文字はオペレーティングシステムによって異なります。それらの16進の値は常にx"1F"未満です。DOS、Windows、およびOS/2では、復帰（x"0D"）、改行（x"0A"）、タブ（x"09"）、空文字（x"00"）、ファイル終了（x"1A"）はすべて制御文字として認識されます。

このスイッチがオフであるときの処理は下記のようになります。

- 行順ファイル中のデータレコードにはASCIIテキスト文字（16進値がx"1F"よりも大きい文字）しか含めることはできません。
- ディスクに出力する際に、後行の空白は削除されて、レコードの末尾を示すためにレコード区切り文字（x"0D0A"）が付加されます。プリンタ制御文字もディスクに書き出すことができます。
- ファイルはレコードとして読み込まれます。レコードはレコード区切り文字によって区切られています。データレコード中にx"0D"またはx"0A"が含まれていると、結果はどうなるか分かりません。タブ（x"09"）は空白に拡張されます。それ以外は、データレコードは元のままです。

例:

a:myprog （-N）

この命令はドライブa:からプログラムmyprogをロードします。その際、行順ファイル中の16進値がx"1F"よりも小さい文字をすべてデータとして扱うようにします。ただし、タブとファイル制御文字は例外です。x"1F"よりも小さい文字の前には空文字は付加されません。

5.4.19 重複キー順整列スイッチ（S）

SORT文を実行中に、重複するレコードの順序を保つようにします。このスイッチはSORT文だけではなくMERGE文に対しても効力を発揮します。それと対象的に、WITH

DUPLICATES IN ORDER指定はSORT文に対してのみ有効な構文です。

属性:

省略値: オフ
タイプ: 一般

説明:

SORT文を実行したときに重複するすべてのレコードの順序を保つようにする方法は、このスイッチを設定することしかありません。原始コード中にWITH DUPLICATES IN ORDER指定を明示的に指定することはできますが、それは実行時には効力がありません。

SORTは一時作業ファイルをシステムによって定義されるTMP環境変数によって指定されるディレクトリのもとに置きます。TMPが設定されていないと、代りに現行ディレクトリが使用されます。

注:

このスイッチを設定すると、アプリケーションの処理速度が遅くなることがあります。

5.4.20 ANSI.SYS互換操作卓I/Oスイッチ（S5）

ANSI互換コンソールI/Oを可能にします。

属性:

省略値: オフ
タイプ: 一般

説明:

この機能はランタイムシステムおよびランタイム環境によって実現されます。このシステムがオンであると、ACCEPT FROM/DISPLAY UPON CONSOLE文はすべて、画面への出力およびキーボードからの入力が、オペレーティングシステムのコマンド行で、">"および"<"を使用してリダイレクトできます。

5.4.21　タブスイッチ（T）

すべての行順ファイルにタブ文字を挿入できるようにします。

属性:

省略値:　オフ
タイプ:　一般

説明:

このスイッチはファイルの形式がこのCOBOLシステムのバージョンと互換性がない場合に特に役に立ちます。

ディスクに出力する際に、タブストップの前にある複数の空白が1つのタブ文字として書き出されます。このスイッチは後続する空白には影響を与えません。それらは行順レコードの一部を構成しないからです。したがって、それらは除去されます。タブ位置は8文字ごとに置かれており（たとえば、9、17、25等々）、変更することはできません。

入力時には、タブ文字は必ず次の位置まで空白に拡張されます。それはこのスイッチが設定されているか否かにかかわらず行われます。

このスイッチがオフに設定されているときは、行順ファイル中の空白はすべて、空白文字としてディスクに書き出されます。

5.4.22　トレースファイル作成スイッチ（t）

アプリケーションにより送られたメッセージをその都度出力するトレースファイルを作成します。

属性:

省略値:　オフ
タイプ:　00

説明:

省略時のトレースファイル名は、trace.logです。fスイッチを使用すればこのファイル名を別の名前に指定することができます。

トレースファイルは、アプリケーションによって送られたメッセージを表示します。これはデバッグ時に役立ちます。このスイッチを製品化されたアプリケーションには設定しないでください。それは、トレースにより処理が遅くなり、アプリケーションの実行中にログファイルのサイズが大きくなるためです。

第6章:　コンパイラ用指令

この章はコンパイラを制御するのに使用できる指令を列挙・説明しています。最初の項は特定の目的のための指令を見つけるために構成されており、指令の名前をカテゴリー別に列挙し、各指令の目的について簡単な説明を付けています。2番目の項では、アルファベット順に各指令の完全な説明をしています。

コンパイラ指令の概要については、「第2章　コンパイラ」を参照してください。

6.1 コンパイラ指令のカテゴリー

コンパイラ指令のカテゴリーは以下の通りです。

- 言語機能の実現
- ランタイム動作の選択
- マルチスレッド
- インターネットアプリケーションプログラミング
- コンパイラ制御
- デバッグ用コンパイル
- 利用者データファイルの形式
- 目的コード、サイズおよび最適化
- 報告書
- 予約指令

6.1.1 言語機能の実現

これらの指令は、コンパイラがどの言語機能を受け入れるかを変更します。

- 付加機能
- 方言
- 明示予約語制御
- 速度に影響する機能
- マルチスレッド

6.1.1.1 付加機能

CONSTANT	**定数を定義**
FCDREG	**ファイル管理記述のレジスタ**
PREPROCESS	**プリプロセッサからの原始プログラム**

REWRITE-LS	LINE SEQにたいするREWRITE
SEQCHK	行番号のチェック
SOURCEFORMAT	自由または固定COBOL方式を選択
SQL	SQLを可能とする

6.1.1.2 方言

ACTUAL-PARAMS	パラメータ付きクラスに仮パラメータではなく実パラメータを指定する
ANS85	ANSI85
COMS85	ANSI85通信モジュール
DBCHECK	2バイト文字を調べる
DBCS	2バイト文字サポート
DG	Data General
DIALECT	使用する方言に従ってチェックタイムおよびランタイム動作を設定する
FLAG	方言用フラグ
FLAGCD	矛盾する指令へのフラグ
FLAGSTD	ANSI '85レベルフラグ
IBM-MS	IBM COBOL VI、Microsoft COBOL V1.0
ISO2000	ISO2000
MF	Micro Focus方言レベル
MF-OO	Micro Focus オブジェクト指向
MS	Microsoft V1またはV2
NCHAR	Micro Focus日本語およびPIC Nサポート
OOCTRL	OOプログラム言語要素
PC1	IBM COBL VI、Microsoft COBOL V1.0
RM	Ryan-McFarland
XOPEN	X/Open

6.1.1.3 明示予約語制御

ADDRSV	予約語の付加
ADDSYN	同義語の付加
MAKESYN	同義とする
OVERRIDE	意味の変更
REMOVE	予約を取り消す

6.1.1.4 速度に影響する機能

ALTER	ALTERを可能とする
FASTCALLS	呼び出されるプログラムの動作を制御する
FASTLINK	パラメータを制約する
FIXOPT	ある制御領域の位置を制御する
NESTCALL	プログラムの入れ子を使用可能とする
QUAL	修飾を可能とする
QUALPROC	修飾された手続き名を使用可能とする
SEG	区分化を可能とする
TRICKL	PERFORMを制約する

6.1.1.5 マルチスレッド

REENTRANT	プログラムを入力するためにマルチスレッドを可能とする
SERIAL	プログラムを入力できるスレッドの数を制限する

6.1.2 ランタイム動作の選択

これらの指令は言語機能の定義を変更し、従って、プログラムの論理を変更します。「言語機能の実現」の項の「方言」の項に列挙された指令のいくつかもまた動作に影響します。

- 一般
- 他のベンダのCOBOL語との互換性
- 古いMicro Focus製品との互換性
- プログラム速度またはサイズに影響する動作

6.1.2.1 一般

ACCEPTREFRESH	ACCEPT文
ARITHMETIC	中間演算結果を制御
ASSIGN	EXTERNALまたはDYNAMIC
ASSIGN-PRINTER	プリンタ出力
BWZSTAR	PIC * 付きのBWZ
CHARSET	ASCIIまたはEBCDIC
CHECKDIV	ゼロによる除算の制御
CHECKNUM	数字フィールドのチェック
COBFSTATCONV	EXTFH状態コード
CONVERTRET	RETURNING項目タイプ
CURRENCY-SIGN	通貨記号
CURRENT-DATE	DDMMYYまたはMMDDYY
DEFAULTBYTE	データ部の初期化
DEFAULTCALLS	CALL変換
DETECTLOCK	レコードロックを検出
EARLY-RELEASE	Micro Focusの初期の利用者構文を使用できるようにする
FOLD-CALL-NAME	CALL名の大・小文字変換
FOLD-COPY-NAME	COPYファイル名の大・小文字変換
INDD	ACCEPTをREADに変換する
INITCALL	モジュールの実行
INTDATE	組み込み関数dateで使用する整数フォーマットの日付に開始日を指定する
IXNUMKEY	索引キーでの真の数字ソーティングを可能にする
LOCKTYPE	ロックされたレコードの読込み
LOGICOPT	CBL_logicの最適化
NATIVE	文字の大小順序
NLS	国別言語サポート
OUTDD	DISPLAYをWRITEに変換する
PRINT-EXT	印刷用拡張子
PROTECT-LINKAGE	保護連結
REPORT-LINE	報告書ライン
SPZERO	数字データ項目でスペース=ゼロ

STICKY-LINKAGE	パラメータの連結を保持
TERMPAGE	報告書ページを充てんする
TRACE	READY TRACEをオンにする
ZEROLENGTHFALSE	ゼロ長試験
ZWB	空白ならばゼロの数字比較

6.1.2.2 他のベンダのCOBOL語との互換性

他のCOBOLを扱うのに以下の指令が必要になることがあります。

ALPHASTART	ALPHABETでの番号付け
APOST	引用符='
CASE	プログラム名の大文字・小文字
COMP-6	COMP-6項目フォーマット
DBSPACE	DBCSスペース
FDCLEAR	書込み後にレコードバッファをクリア
IBMCOMP	語記憶モード
NTDATE	組み込み関数dateで使用する整数フォーマットの日付の開始日
ODOSLIDE	可変長テーブル
OPTIONAL-FILE	全てのファイルがオプション
PERFORM-TYPE	PERFORMからの戻り
QUOTE	引用符="
RETRYLOCK	ロックされたレコードに再試行する
SIGN	含まれた符号
SSRANGE	添字、索引および参照修正項目のランタイムチェック
STICKY-PERFORM	PERFORMの動作
SYMBSTART	SYMBOLICでの番号付け
TRUNC	2進データ項目の切り捨て
SWITCH-TYPE	ANSI互換のスイッチ動作

6.1.2.3 古いMicro Focus製品との互換性

古いMicro Focus COBOL製品を扱うのに以下の指令が必要になることがあります。

AUTOLOCK	省略時ロック
COMP	計算用部分集合
COMP-5	COMP-5動作
DE-EDIT	数字編集形式動作
FILESHARE	省略時ロックを自動にする
IOCONV	READ-INTO/WRITE-FROM
MF	Micro Focus方言レベル
OLDBLANKLINE	BLANK LINEの効果
OLDINDEX	指標=添字
OLDNEXTSENTENCE	NEXT SENTENCEの効果
OLDREADINTO	READ INTOの効果
WRITELOCK	省略時ロック

6.1.2.4 プログラム速度またはサイズに影響する動作

ALIGN	データけたよせを決定
BOUND	添字の範囲チェック
BOUNDOPT	表示用途添字を最適化
FIXOPT	制御領域の場所の制御

LINKCHECK　　Linkage Section項目のチェック
PARAMCOUNTCHECK　　パラメータの省略
PERFORMOPT　　空の段落のPERFORMを最適化

6.1.3 マルチスレッド

REENTRANT　　プログラムを再入可能とする
APOST　　プログラムをシリアルとする

6.1.4 インターネットアプリケーションプログラミング

この指令は、インターネットアプリケーションにおいて、COBOLプログラムとwebサーバ間で使用するプロトコルを設定します。

WEBSERVER

6.1.5 コンパイラ制御

- コンパイル／リンクファイル
- 指令の使用
- エラーおよびフラグメッセージ
- リスト
- 画面

6.1.5.1 コンパイル／リンクファイル

CANCELLBR　　COPY .lbrファイルのクローズ
CONVSPACE　　原始コードスペース
COPYLBR　　COPYライブラリ（ .lbr）ファイル
GNT　　目的コード用ファイル
INT　　中間コード用ファイル
INTLEVEL　　移植性レベル
KEEP-INT　　エラーがあっても.intファイルを生成
KEYCHECK　　キーの数のチェック
OBJ　　目的コード用ファイル
OSEXT　　原始ファイル名拡張子
PREPROCESS　　プリプロセスの原始プログラム
RDFPATH　　リポジトリーファイルの場所の指定
REPOSITORY　　リポジトリーファイルの作成

6.1.5.2 指令の使用

COBOLDIR　　cobol.dirを使用/無視

CONFIRM　　指令の表示
DIALECT　　使用する方言に従ってチェックタイムおよびランタイム動作を設定する
DIRECTIVES　　指令のファイル
DIRECTIVES-IN-COMMENTS　注釈行中の指令を有効化
SETTING　　指令の印刷
SHOW-DIR　　指令ファイルの印刷
USE　　指令のファイル

6.1.5.3 エラーおよびフラグメッセージ

BRIEF　　メッセージテキストなし
CHANGE-MESSAGE　　メッセージの重大度を変更
EDITOR　　エディタ用エラーファイルの作成
ERRLIST　　メッセージのみを印刷
ERRQ　　エラー発生時休止
FLAG　　方言外のフラグ
FLAGAS　　エラー、警告または情報としてフラグを示す
FLAGCD　　矛盾する指令
FLAGQ　　フラグを立てるとき休止
FLAGSINEDIT　　エラーファイル中のフラグ
FLAGSTD　　ANSI85レベルを越えるフラグ
HIDE-MESSAGE　　隠すメッセージを設定
INFORETURN　　情報メッセージのリターンコード値
MAX-ERROR　　コンパイラエラーの最大数を指定
MOVE-LEN-CHECK　　原始プログラムと英数字のMOVE操作の対象の長さをチェックする
QUERY　　COPYファイルが見つからないとき休止
STDERR　　メッセージをSTDERRに書き込む
WARNING　　出力するメッセージのレベル

6.1.5.4 リスト

ASM　　(.GRP) ファイルの隠れ生成コード
ASMLIST　　アセンブリーリスト用のファイル
COPYEXT　　COPYファイルの拡張子
COPYLIST　　COPYファイルのリスト
DATAMAP　　データ項目のリスト
DATE　　リスト用日付
ERRLIST　　原始コードおよびエラーメッセージのリスト
FORM　　ページ長
LINE-COUNT　　リストの終わりにある情報の詳細を制御
LIST　　原始リスト用ファイル
LISTPATH　　リストファイルのパスを指定
LISTWIDTH　　ページ幅
MFCOMMENT　　副形式注記
PREPLIST　　プリプロセッサのリスト情報作成
PRINT　　原始リスト用ファイル
RAWLIST　　日付等の情報を省いたリスト
REF　　段落のアドレスのリスト
REFNO　　リスト内のコンパイラ番号
RESEQ　　行番号の作成
SEQCHK　　行番号のチェック
SETTING　　指令の印刷
SHOW-DIR　　指令ファイルの印刷
SOURCEASM　　アセンブラリスト内の原始コード
TIME　　リストに時間を載せる
VERBOSE　　コンパイラメッセージ

XREF	相互参照表の作成
ZEROSEQ	行番号内のゼロ列

6.1.5.5 画面

BELL	停止時の警報
CONFIRM	指令の表示
ECHO	エラーの表示
ECHOALL	完全なリストの表示
SUPFF	ページ頭書きなし

6.1.6 デバッグ用コンパイル

これらの指令は、プログラムについての追加情報の入ったファイルを、デバッグに使用されるソフトウェアへの入力用に、コンパイラに作成させます。

ANIM	Animator用
COBIDY	Animator情報ファイル用パス
FLAGSINEDIT	エラーファイル中のフラグ

6.1.7 利用者データファイルの形式

これらの指令は利用者データファイルのデータ格納形式を変更します。

CALLFH	外部ファイル・ハンドラ
DATACOMPRESS	データ圧縮
FILETYPE	データファイル形式
IDXFORMAT	索引ファイル構造
KEYCOMPRESS	キー圧縮
RECMODE	固定長または可変長
SEQUENTIAL	SEQ編成の傍系

6.1.8 目的コード、サイズおよび最適化

これらの指令は、プログラムの論理を変更せずに、生成された目的コードを変更します。

- サイズと速度
- 目的ファイル形式
- プログラム間連絡
- ファイル処理
- 外部ハンドラ

6.1.8.1 サイズと速度

FIXOPT	制御領域の場所の制御
HOSTSIGNS	不正な記号ニブル
LNKALIGN	連結寄せとする
OPT	最適化レベル
SEG	区分化
SCHEDULER	Intelプロセッサ用のCOBOL最適化
TARGET	プロセッサ固有の命令

6.1.8.2 目的ファイル形式

OMF	OBJまたはGNT

6.1.8.3 プログラム間連絡

LITLINK	定数を共有とする
LITVAL-SIZE	BY VALUEサイズ
RTNCODE-SIZE	RETURN-CODEサイズ

6.1.8.4 ファイル処理

WRITETHROUGH	バッファリングなしの書出し

6.1.8.5 外部ハンドラ

CALLFH	外部ファイル・ハンドラ
CALLSORT	外部ソートハンドラ
CALLMCS	外部MCSハンドラ

6.1.9 報告書

この指令は報告書制御モジュールの動作に影響します。

RWHARDPAGE

6.1.10 予約指令

これらの指令は古い製品との互換のために予約、または、保留されているもので効果はありません。使用しないでください。

CSI
ENSUITE

```
FASTSORT
LOCALCOUNT
WB
WB2
WB3
```

6.2 コンパイラ指令（アルファベット順リスト）

ACCEPTREFRESH	ACCEPT文
ACTUAL-PARAMS	パラメータ付きクラスに仮パラメータではなく実パラメータを指定する
ADDRSV	予約語の付加
ADDSYN	同義語の付加
ALIGN	データけたよせを決定
ALPHASTART	ALPHABETでの番号付け
ALTER	ALTERを可能とする
ANIM	Animator用
ANS85	ANSI85
APOST	引用符='
ARITHMETIC	中間演算結果を制御
ASM	(.GRP）ファイルの隠れ生成コード
ASMLIST	アセンブリーリスト用のファイル
ASSIGN	EXTERNALまたはDYNAMIC
ASSIGN-PRINTER	プリンタ出力
AUTOLOCK	省略時ロック
BELL	停止時の警報
BOUND	添字の範囲チェック
BOUNDOPT	表示用途添字を最適化
BRIEF	メッセージテキストなし
BWZSTAR	PIC*付きのBWZ
CALLFH	外部ファイル・ハンドラ
CALLMCS	外部MCSハンドラ
CALLSORT	外部ソートハンドラ
CANCELLBR	COPY.lbrファイルのクローズ
CASE	プログラム名の大文字・小文字
CHANGE-MESSAGE	メッセージの重大度を変更
CHARSET	ASCIIまたはEBCDIC
CHECKDIV	ゼロによる除算の制御
CHECKNUM	数字フィールドのチェック
COBFSTATCONV	EXTFH状態コード
COBIDY	Animator情報ファイル用パス
COBOLDIR	cobol.dirを使用/無視
COMP	計算用部分集合
COMP-5	COMP-5動作
COMP-6	COMP-6項目フォーマット

COMS85	ANSI85通信モジュール
CONFIRM	指令の表示
CONSTANT	定数を定義
CONVERTRET	RETURNING項目タイプ
CONVSPACE	原始コードスペース
COPYEXT	COPYファイルの拡張子
COPYLBR	COPYライブラリ（.lbr）ファイル
COPYLIST	COPYファイルのリスト
CSI	
CURRENCY-SIGN	通貨記号
CURRENT-DATE	DDMMYYまたはMMDDYY
DATACOMPRESS	データ圧縮
DATAMAP	データ項目のリスト
DATE	リスト用日付
DBCHECK	2バイト文字を調べる
DBCS	2バイト文字サポート
DBSPACE	DBCSスペース
DE-EDIT	数字編集形式動作
DEFAULTBYTE	データ部の初期化
DEFAULTCALLS	CALL変換
DETECTLOCK	レコードロックを検出
DG	DataGeneral
DIALECT	使用する方言に従ってチェックタイムおよびランタイム動作を設定する
DIRECTIVES	指令のファイル
DIRECTIVES-IN-COMMENTS	注釈行中の指令を有効化
EARLY-RELEASE	MicroFocusの初期の利用者構文を使用できるようにする
ECHO	エラーの表示
ECHOALL	完全なリストの表示
EDITOR	エディタ用エラーファイルの作成
ENSUITE	
ERRLIST	メッセージを印刷
ERRQ	エラー発生時休止
FASTCALLS	呼び出されるプログラムの動作を制御する
FASTLINK	パラメータを制約する
FASTSORT	
FCDREG	ファイル管理記述のレジスタ
FDCLEAR	書込み後にレコードバッファをクリア
FILESHARE	省略時ロックを自動にする
FILETYPE	データファイル形式
FIXOPT	制御領域の場所の制御
FLAG	方言外のフラグ
FLAGAS	エラー、警告または情報としてフラグを示す
FLAGCD	矛盾する指令へのフラグ

FLAGQ	フラグを立てるとき休止
FLAGSINEDIT	エラーファイル中のフラグ
FLAGSTD	ANSI85レベルを越えるフラグ
FOLD-CALL-NAME	CALL名の大・小文字変換
FOLD-COPY-NAME	COPYファイル名の大・小文字変換
FORM	ページ長
GNT	目的コード用ファイル
HIDE-MESSAGE	隠すメッセージを設定
HOSTSIGNS	不正な記号ニブル
IBM-MS	IBMCOBOLVI、MicrosoftCOBOLV1.0
IBMCOMP	語記憶モード
IDXFORMAT	索引ファイル構造
INDD	ACCEPTをREADに変換する
INFORETURN	情報メッセージのリターンコード値
INITCALL	モジュールの実行
INT	中間コード用ファイル
INTDATE	組み込み関数dateで使用する整数フォーマットの日付に開始日を指定する
INTLEVEL	移植性レベル
IOCONV	READ-INTO/WRITE-FROM
ISO2000	ISO2000
IXNUMKEY	索引キーでの真の数字ソーティングを可能にする
KEEP-INT	エラーがあっても.intファイルを生成
KEYCHECK	キーの数のチェック
KEYCOMPRESS	キー圧縮
LINE-COUNT	リストの終わりにある情報の詳細を制御
LINKCHECK	LinkageSection項目のチェック
LIST	原始リスト用ファイル
LISTPATH	リストファイルのパスを指定
LISTWIDTH	ページ幅
LITLINK	定数を共有とする
LITVAL-SIZE	BYVALUEサイズ
LNKALIGN	連結寄せとする
LOCALCOUNT	
LOCKTYPE	ロックされたレコードの読込み
LOGICOPT	CBL_logicの最適化
MAKESYN	同義とする
MAX-ERROR	コンパイラエラーの最大数を指定
MF	MicroFocus方言レベル
MF-OO	MicroFocusオブジェクト指向
MFCOMMENT	副形式注記
MOVE-LEN-CHECK	原始プログラムと英数字のMOVE操作の対象の長さをチェックする
MS	MicrosoftV1またはV2
NATIVE	文字の大小順序

NCHAR	MicroFocus日本語およびPICNサポート
NESTCALL	プログラムの入れ子を使用可能とする
NLS	国別言語サポート
OBJ	目的コード用ファイル
ODOSLIDE	可変長テーブル
OLDBLANKLINE	BLANKLINEの効果
OLDINDEX	指標=添字
OLDNEXTSENTENCE	NEXTSENTENCEの効果
OLDREADINTO	READINTOの効果
OMF	OBJまたはGNT
OOCTRL	OOプログラム言語要素
OPT	最適化レベル
OPTIONAL-FILE	全てのファイルがオプション
OSEXT	原始ファイル名拡張子
OUTDD	DISPLAYをWRITEに変換する
OVERRIDE	意味の変更
PARAMCOUNTCHECK	パラメータの省略
PC1	IBMCOBLVI、MicrosoftCOBOLV1.0
PERFORM-TYPE	PERFORMからの戻り
PERFORMOPT	空の段落のPERFORMを最適化
PREPLIST	プリプロセッサのリスト情報作成
PREPROCESS	プリプロセスの原始プログラム
PRINT	原始リスト用ファイル
PRINT-EXT	印刷用拡張子
PROTECT-LINKAGE	保護連結
QUAL	修飾を可能とする
QUALPROC	修飾された手続き名を使用可能とする
QUERY	COPYファイルが見つからないとき休止
QUOTE	引用符="
RAWLIST	日付等の情報を省いたリスト
RDFPATH	リポジトリーファイルの場所の指定
RECMODE	固定長または可変長
REENTRANT	プログラムを入力するためにマルチスレッドを可能とする
REF	段落のアドレスのリスト
REFNO	リスト内のコンパイラ番号
REMOVE	予約を取り消す
REPORT-LINE	報告書ライン
REPOSITORY	リポジトリーファイルの作成
RESEQ	行番号の作成
RETRYLOCK	ロックされたレコードに再試行する
REWRITE-LS	LINESEQにたいするREWRITE
RM	Ryan-McFarland
RTNCODE-SIZE	RETURN-CODEサイズ
RWHARDPAGE	

SCHEDULER	Intelプロセッサ用のCOBOL最適化
SEG	区分化を可能とする
SEQCHK	行番号のチェック
SEQUENTIAL	SEQ編成の傍系
SERIAL	プログラムを入力できるスレッドの数を制限する
SETTING	指令の印刷
SHOW-DIR	指令ファイルの印刷
SIGN	含まれた符号
SOURCEASM	アセンブラリスト内の原始コード
SOURCEFORMAT	自由または固定COBOL方式を選択
SPZERO	数字データ項目でスペース=ゼロ
SQL	SQLを可能とする
SSRANGE	添字、索引および参照修正項目のランタイムチェック
STDERR	メッセージをSTDERRに書き込む
STICKY-LINKAGE	パラメータの連結を保持
STICKY-PERFORM	PERFORMの動作
SUPFF	ページ頭書きなし
SWITCH-TYPE	ANSI互換のスイッチ動作
SYMBSTART	SYMBOLICでの番号付け
TARGET	プロセッサ固有の命令
TERMPAGE	報告書ページを充てんする
TIME	リストに時間を載せる
TRACE	READYTRACEをオンにする
TRICKL	PERFORMを制約する
TRUNC	2進データ項目の切り捨て
USE	指令のファイル
VERBOSE	コンパイラメッセージ
WARNING	出力するメッセージのレベル
WB	予約済み
WB2	予約済み
WB3	予約済み
WEBSERVER	ISAPIおよびNSAPIの性能優位性を取得
WRITELOCK	省略時ロック
WRITETHROUGH	バッファリングなしの書出し
XOPEN	X/Open
XREF	相互参照表の作成
ZEROLENGTHFALSE	ゼロ長試験
ZEROSEQ	行番号内のゼロ列

6.3 解説の読み方

コンパイラ指令全てについてアルファベット順に説明します。各説明の内容は以下の通りです。

DIRECTIVE-NAME

DIRECTIVE-NAMEの簡単な説明

構文:

```
◆---+-------+-+---------- DIRECTIVE-NAME ----"paramater(s)"-+----◇
    +-- / --+ +-+--------+-- DIRECTIVE-NAME -----------------+
              +-- NO --+
```

ダイヤグラムを使用したコンパイラ指令の構文を示します。ダイヤグラムは、指令とそのパラメータが、それらが書き出されるべき順序を示す線で結ばれて示されています。ダイヤグラムは左から右に読みます。各ダイヤグラムは◆で始まり、◇で終わります。時に、その線は選択肢を示すために分岐し、また合流します。各線の長さには意味はありません。

DOS*、*OS/2*および*Windows

特に指定のない限り指令のパラメータは引用符（" "）で囲まれて示されます。括弧を代わりに使用することも可能です。引用符を使用した場合、パラメータにスペースを入れることができます。一方、括弧で囲まれたパラメータにスペースを入れることはできません。パラメータをコンマの後に指定して、ファイル名が削除されている場合、その指令の前に斜線（/）を入れる必要があります。そうしないと、その指令は間違ってファイル名と取られます。

コンパイラ指令設定の詳細については、「第2章　コンパイラ」の章を参照してください。

UNIX

指令のパラメータは引用符（" "）で囲まれて示されます。特に指定のない限り括弧または等号（=）を代わりに使用することも可能です。引用符を使用した場合、パラメータにスペースを入れることができます。一方、括弧で囲まれたパラメータにスペースを入れることはできません。引用符または等号を使用する場合、逆斜線文字（¥）を用いてそれらをエスケープする必要があります。ダイヤグラムにおける総称指令用斜線（/）は、DOS、OS/2およびWindowsにしか適用できません。斜線はUNIX環境では無視でき、UNIX向け指令のダイヤグラムには使用されません。

パラメータ:

指令について有効なパラメータが指定されている場合、それらを列挙・説明します。

属性:

省略値:　その指令の省略時の設定を示します。DIALECT指令にある値を設定する場合、他の指令は省略値をオーバーライドする値またはその指令に設定しようとしている値に設定されます。DIALECTによって設定される指令は、Dialectに示されています。ここをクリックすると、DIALECTのページを表示してDIALECT固有の設定を確認することができます。さらに詳細については、「DIALECT指令」の項を参照してください。

段階:　　構文チェック、生成、またはその両方。その指令がコンパイラのどの段階を制御するかを示します。

$SET:　「初期」、「任意」または「不可」。その指令を原始プログラム内の$SET文に入れられるかどうか示します。「初期」は初期$SET文にのみ入れられることを意味します。

依存性:

この指令の設定がそれ以外の指令の設定を変更する場合、またはそれ以外の指令がこの指令の設定を変更する場合に依存性が示されます。

いくつかの異なるケースがあり、次のキーワードで識別できます。

直ちに:　指定された変更がこの指令の処理の一部として直ちに実行されます。これにより、最初の指令の次に2番目の指令を指定することで値をリセットすることができます。ただし、これは何をしようとしているかを分かっているときだけ行ってください。

最後に: 指定された変更がすべての指令の処理が終わったときに実行されます。これにより、新しい値を書きかえるようなことは起こりません。

説明:

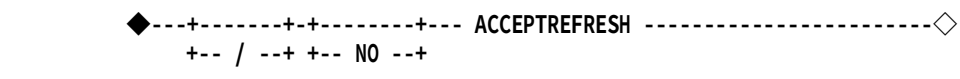
その指令に関する追加情報などを示します。

6.4 各コンパイラ指令の解説

ACCEPTREFRESH

画面節データに関連するデータ領域を、ACCEPT文の前の対応する作業場所節項目に基づいて更新するかどうか指定します。

構文:



パラメータ:

なし

属性:

省略値:

NOACCEPTREFRESH

段階:

構文チェック

$SET:

初期

依存性:

MS、IBM-MS、またはPC1により直ちにACCEPTREFRESHが設定されます。

説明:

画面節データ項目を参照するACCEPTの前にACCEPTREFRESHを指定すると、受け取られる項目に従属的な画面節項目に関連する全てのデータ領域が次のように修正されます。

- 画面節項目にUSING指定が入っていると、暗黙MOVEが、対応する作業場所節項目から生成されます。
- 画面節項目にTO指定が入っていると、データ領域はスペースに初期化されます。

NOACCEPTREFRESHが指定されると、画面節データ領域は最後のACCEPTまたはDISPLAYの後にそのまま残されます。

ACTUAL-PARAMS

パラメータ付きクラスに仮パラメータではなく実パラメータを指定します。

構文:

```
  ◆---+-------+-+-------------- ACTUAL-PARAMS --- "parameter" --------+---◇
      +-- / --+ +-+----NO------ ACTUAL-PARAMS ----------------------+
```

パラメータ:

スペースで区切られた最大32個までのパラメータ

属性:

省略値:

NOACTUAL-PARAMS

段階:

構文チェック

$SET:

初期

説明:

少なくとも1つのパラメータを指定する必要があります。また、指定できるパラメータの最大数は32個です。指定するパラメータは、パラメータ付きクラスの仮パラメータではなく実パラメータです。

ADDRSV

1つまたは複数の特定の予約語を予約語リストに付加し、プログラム内で予約語としてそれらが認識されるようにします。方言指令が有効となっているかどうかに関わらず、指定された予約語が付加されます。

構文:

```
                                +---------------+
                                -               |
  ◆---+-------+---- ADDRSV ------ "予約語" ------+-----------------◇
      +-- / --+
```

パラメータ:

予約語 COBOLのある方言での予約語で、必ずしも本コンパイル用に指定された方言用ではありません。

属性:

省略値:

予約語を更に付加することはない

段階:

構文チェック

$SET:

初期

説明:

方言指令が有効となっているかどうかに関わらず、指定された予約語が付加されます。

本指令はSETTING指令で作成されたリストには出てきません。

ADDSYN

現存する予約語と同義の利用者予約語を定義します。

構文:

```
  ◆---+-------+---- ADDRSN --- "予約語" = "利用者語" --------------◇
      +-- / --+
```

パラメータ:

予約語 現存する予約語

利用者語 現存する予約語と同じでない任意のCOBOL語

属性:

省略値:

予約語の同義語は作成されません

段階:

構文チェック

$SET:

初期

説明:

等号（=）の両端にスペースを入れる必要があります。

本指令はSETTING指令で作成されたリストには出てきません。

ALIGN

01レベルまたは77レベルのデータ項目を割り付けるメモリ境界を指定します。

構文:

```
  ◆---+-------+--- ALIGN --- "整数" -------------------------------◇
      +-- / --+
```

パラメータ:

整数 1つの01レベルから次までの距離がこの整数の倍数となります。

1から255までの任意の値をとることができます。

属性:

省略値:

ALIGN"8" （Dialect）

段階:

構文チェック

$SET:

初期

依存性:

IBMCOMPまたはNORMにより直ちにALIGN"8"が設定されます。

RMまたはRM"ANSI"により直ちにALIGN"2"が設定されます。

説明:

01レベルのデータ項目は、与えられた値の倍数のバイト境界に割り付けられます。

これによりより効果的な実行結果が生成できますが、メモリの使用量が増えることを意味しています。

「整数」を4の倍数でない数字に設定すると、生成されたコードの効率に悪影響をおよぼします。

ALPHASTART

ALPHABET句をコンパイルするとき、コンパイラが文字の大小順序における位置を計数し始める数字を設定します。

構文:

```
◆---+-------+--- ALPHASTART --- "整数" --------------------------◇
    +-- / --+
```

パラメータ:

整数 使用される数字

属性:

省略値:

ALPHASTART "1" （Dialect）

段階:

構文チェック

$SET:

初期

説明:

ANSIに準拠するためには、ALPHASTART"1" を使用します。

例:

例えば、ALPHASTART"1" であると、

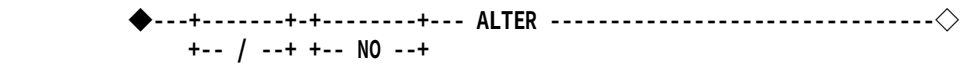
ALPHABET MYALPHA IS 66, 67

このCOBOL文は "A" と "B" から成るアルファベットを宣言します。その理由は、1から計数すると、ASCIIの文字の大小順序ではこれらの文字は第66番目と67番目の文字だからです。ALPHASTART"0" では、MYALPHAは "B" と "C" で構成されます。

ALTER

プログラムのALTER文を使用できるようにします。

構文:



パラメータ:

なし

属性:

省略値:

ALTER （Dialect）

段階:

構文チェックおよび生成

$SET:

初期

説明:

プログラムにALTER文が入っていない場合は、NOALTERを指定すると、コンパイラはやや効率の向上したコードを生成できます。

ANIM

コンパイラに追加情報を生成させ、Animatorを使用できるようにします。

構文:

パラメータ:

なし

属性:

省略値:

NOANIM

段階:

構文チェックおよび生成

$SET:

初期

依存性:

CSIにより最後にANIMが設定されます。

NOINTにより最後にNOANIMが設定されます。

説明:

拡張子（ .idy）の付いたファイルもまた作成されます。このファイルには、プログラムをアニメートする上で必要な追加情報が入っています。（ .idy）ファイルの場所は、COBIDY指令により制御されます。

構文チェックの段階で次の指令のいずれかを指定することにより中間コードファイルを作成できます: NOOBJ、NOGNT、またはINT()。中間コードは、拡張子（.int）の付いたファイルに保持されます。

生成の段階では、（.obj）ファイルが作成されます。（.obj）ファイルをリンクして（.exe）ファイルまたは（.dll）ファイルを作成すると、原始コードをデバッグすることができます。

GNT()指令を指定した場合には、（.obj）ファイルの代りに（.gnt）ファイルが生成段階で作成されます。

ANS85

ANSI85 COBOLで予約されている語が予約語として扱われることを指定し、ある機能の動作を変更してANSI85と互換が取れるようにします。

構文:

```
◆---+-------+-+------------- ANS85 --- "SYNTAX" ------------+---◇
    +-- / --+ +-+--------+-- ANS85 ------------------------+
              +-- NO --+
```

パラメータ:

SYNTAX　その指令が影響するのは構文のみとさせ、動作には影響させないようにします。

属性:

省略値:

ANS85 （Dialect）

段階:

構文チェック

$SET:

初期

説明:

本指令は次の動作を変更します。

- 入出力状態値はANSI85で定義されたものです。
- ALPHABETIC字類試験では、小文字を英字とみなします。
- PERFORM動作での制御変数は、ANSI85で指定された順に初期化されます。
- 可変長集団項目へのMOVE（転記）は、DEPENDING ON項目がその集団内にあれば、受取り側項目の最大長を使用します。

APOST

コンパイラに表意定数QUOTEを一重引用符（ ’）として解釈させます。

構文:

```
◆------ APOST ------------------------------------------------◇
```

パラメータ:

なし

属性:

省略値:

APOST

段階:

構文チェック

$SET:

任意

説明:

本指令の反対はQUOTE指令で、これは二重引用符文字を使用させます。

ARITHMETIC

算術式がどのように評価されるべきか指定します。

構文:

```
◆---+-------+--- ARITHMETIC -- "算術演算種類" --------------------◇
    +-- / --+
```

パラメータ:

算術演算種類 採用する動作

属性:

省略値:

ARITHMETIC"MF" （Dialect）

段階:

構文チェック

$SET:

不可

説明:

「算術演算種類」の値として可能なものは次の通りです。

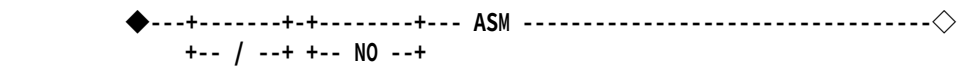
MF 中間結果に対し切り捨ては行わない

ARITHMETIC"MF" を指定すると、切り捨ては行われないので、算術式はできるだけ正確に計算されます。

ASM

ジェネレータの報告書ファイル（ .GRP）に隠されていた生成コードをコンパイラに挿入させます。

構文:



パラメータ:

あて先 完全なファイル指定または装置名

属性:

省略値:

NOASM

段階:

生成

$SET:

初期

説明:

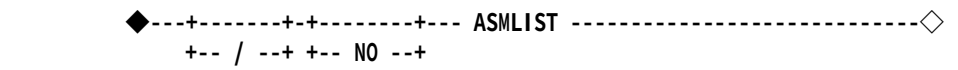
報告書ファイルには、必ずアセンブリリストが含まれています。NOASMを指定すると、リストファイルは作成されません。ファイル名を付けないでASMを指定すると、アセンブリリストが画面に表示されます。既存のファイルを指定した場合には、既存のファイルは上書きされます。

ASM()は、アセンブリリストを原始ファイル名.grpファイルに入れます。この原始ファイル名は、コンパイルするプログラムの基本名です。ASMLISTを指定すると、コンパイラはアセンブラコードだけでなく生成済みコードの16進ダンプを含むリストファイルを作成します。アセンブラではこのファイルが必要な場合があります。

ASMLIST

ジェネレータ報告書ファイルにある16進ダンプを制御します。

構文:



パラメータ:

あて先 完全なファイル指定または装置名

属性:

省略値:

NOASMLIST

段階:

生成

$SET:

初期

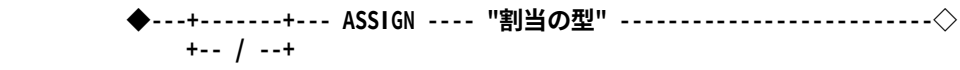
説明:

リストファイルは、ASM指令が制御します。

ASSIGN

SELECT文にEXTERNALもDYNAMICもどちらもない場合どのようにファイル名を割り当てるか指定します。

構文:



パラメータ:

割当の型 EXTERNAL（外部）またはDYNAMIC（動的）。方法を定義。

属性:

省略値:

ASSIGN "DYNAMIC" （Dialect）

段階:

構文チェック

$SET:

任意

説明:

任意の索引ファイルにおいて、またはCALLFH指令を使用してコンパイルされたプログラムの場合はすべてのファイルにおいて、ファイル名のマッピングにより構文定義またはASSIGN指令が変更されます。

ASSIGN-PRINTER

ASSIGN TO PRINTER句がファイル名を指定していないとき、その句からの出力をどのように割り当てるかを指定します。

構文:

パラメータ:

ファイル名 ASSIGN TO PRINTER句と関連するファイル

属性:

省略値:

NOASSIGN-PRINTER

段階:

構文チェック

$SET:

任意

説明:

ファイル名をASSIGN TO PRINTER句の一部として指定すると、本指令は効果がありません。

NOASSIGN-PRINTERを指定すると、出力は装置LST:に送られます。

ASSIGN-PRINTER "ファイル名" を指定すると、指定された「ファイル名」のファイルに出力が転送されます。装置、パス名、ファイル名、および拡張子を含ませた完全なファイル名を指定することが可能です。

ASSIGN-PRINTER（ ）を指定すると、次のCOBOL文を含んだ場合と同じ動作になります。

select ファイル名1 assign to printer ファイル名1

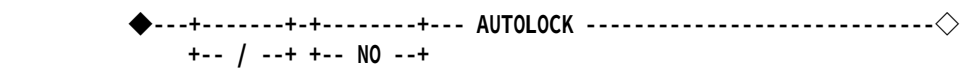
つまり、COBOLプログラムで使用されるファイル名は、また、出力のファイル名としても使用されるということです。内部ファイル名がオペレーティングシステムが処理できる長さを超える場合、オペレーティングシステムが許容する最大長にファイル名が切り捨てにより縮められます。

省略時には、ファイル名は拡張子を含みませんが、PRINT-EXT指令を用いて拡張子を指定できます。

AUTOLOCK

マルチユーザー環境でI-OまたはEXTENDで開いたファイルへの、省略時ロックをEXCLUSIVE（排他）ではなくAUTOMATIC（自動）にします。

構文:



パラメータ:

なし

属性:

省略値:

NOAUTOLOCK

段階:

構文チェック

$SET:

初期

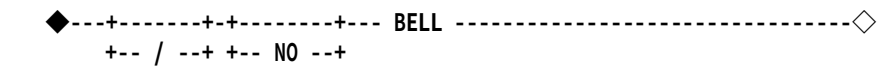
本指令はその状態がWRITELOCKと同じであると、SETTINGリストには表れません。この場合、これら2つの指令の状態はFILESHARE指令により示されます。

本指令は初期のファイル共有製品との互換を取るために提供されています。新しいプログラムを書くときは、本指令ではなくロック構文を使用する必要があります。

BELL

全てのNOTE（注記）点（例えば、エラーが発生したか、コンパイルが完了して、コンパイルが停止したとき）で警報を鳴らさせます。

構文:



パラメータ:

なし

属性:

省略値:

NOBELL

段階:

構文チェックおよび生成

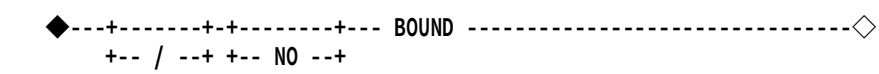
$SET:

初期

BOUND

各表アクセスごとに添字または指標値をチェックして、それがOCCURS句で定義された回数内にあることを確認するように指定します。

構文:



パラメータ:

なし

属性:

省略値:

BOUND （Dialect）

段階:

構文チェックおよび生成

$SET:

初期

依存性:

BOUNDが最後にNOBOUNDOPTを設定します。

説明:

多次元の表では、合成添字のみチェックされます。個々の添字または索引が制限を超えたが参照は章の中に残ったままの場合には、エラーは発生しません。

BOUNDOPT

USAGE DISPLAY添字用に生成されたコードを最適化します。

構文:

パラメータ:

なし

属性:

省略値:

BOUNDOPT

段階:

生成

$SET:

初期

依存性:

BOUNDが最後にNOBOUNDOPTを設定します。

説明:

BOUNDOPTを使用する場合、表のサイズを超える符号なしUSAGE DISPLAY添字の桁は無視されます。

本指令は、NOBOUNDが指定されている場合のみ使用でき、OPT指令で作成された生成コードに影響します。また、プログラムが表の終わりを越えて参照する場合、NOBOUNDOPTを指定する必要があります。

例:

例えば、50個の記述項がある表の場合、PIC9(3）DISPLAY添字はPIC9(2）として扱われ、最上位桁は無視されます。

BRIEF

コンパイラにエラー番号のみを生成させ、メッセージテキストは生成させません。

構文:

パラメータ:

なし

属性:

省略値:

NOBRIEF

段階:

構文チェック

$SET:

任意

BWZSTAR

BLANK WHEN ZERO句を"*"PICTURE記号により定義されたフィールドのデータ部で使用できるかどうかを決定します。

構文:

```
◆---+-------+-+--------+--- BWZSTAR -----------------------------◇
     +-- / --+ +-- NO --+
```

パラメータ:

なし

属性:

省略値:

NOBWZSTAR

段階:

構文チェック

$SET:

任意

説明:

本指令は、このCOBOLシステムの後のリビジョンで変更または削除される可能性があります。

BWZSTARを指定すると、BLANK WHEN ZERO句を"*"PICTURE記号により定義されたフィールドで使用できます。また、BLANK WHEN ZERO句はフィールドが編集操作の対象である場合に有効となり結果はゼロです。

NOBWZSTARを指定すると、PIC*フィールドに関連するBLANK WHEN ZERO句は使用できません。

CALLFH

コンパイラが、呼出し可能ファイルハンドラインタフェースを使用して、すべてのファイル入出力操作を直接呼出すようにします。

構文:

```
◆---+-------------- CALLFH --- "handler-name" ---+---------------◇
     +-+--------+--- CALLFH ---------------------+
       +-- NO --+
```

パラメータ:

ハンドラ名 ファイルハンドラとして呼び出すプログラムの基本名。

属性:

省略値:

CALLFH"EXTFH" （Dialect）

段階:

構文チェック

$SET:

初期

説明:

次のいずれかのハンドラ名を指定することができます。

EXTFH - このCOBOLシステムで提供しているファイルハンドラ

FHREDIR - Fileshare Version2用

ユーザー独自のファイルハンドラ

NOCALLFHを指定すると、ランタイムシステムがファイル入出力を処理します。CALLFHを指定すると、ファイル入出力文はすべてこの指令により指定されたファイルハンドラの呼出しに変換されます。

ハンドラ名を指定しないと、モジュールExtfhとみなされます。

本指令は、変換用モジュールによりすべてのファイル処理操作を直接呼び出し、COBOL以外のファイルハンドラを利用するためにも使用されます。

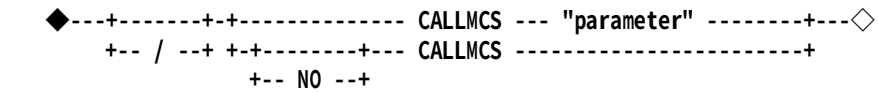
本指令を使用して、標準ファイルハンドラの代りにユーザ独自のファイルハンドラをプログラムに呼び出させることもできます。

CALLMCS

本指令は、システムが内部で使用するために予約されており、その設定を変更できません。

すべてのメッセージ制御システム操作に対して、指定されたハンドラ経由のルートを決めます。

構文:



パラメータ:

MCS名 メッセージ制御システム（MCS）操作を処理するために呼び出すプログラムの基本名。

属性:

省略値:

NOCALLMCS

段階:

構文チェック

$SET:

初期

説明:

MCS名を指定しない場合、EXTMCSとみなされます。

CALLSORT

全てのSORTおよびMERGE動作を処理するために呼び出されるプログラムを定義します。

構文:

```
◆---+-------+-+-------------- CALLSORT --- "整列名" -------+------◇
    +-- / --+ +-+--------+--- CALLSORT --------------------+
              +-- NO --+
```

パラメータ:

整列名 整列併合を処理するために呼び出されるプログラムのルート名。

属性:

省略値:

CALLSORT"EXTSM" （Dialect）

段階:

構文チェック

$SET:

初期

CANCELLBR

COPY文でライブラリとして使用される（ .lbr）ファイルを、その複写動作が終了したら、コンパイラに閉じさせます。

構文:

```
◆---+-------+----------+--- CANCELLBR ----------------------------◇
    +-- / --+ +-- NO --+
```

パラメータ:

なし

属性:

省略値:

CANCELLBR （Dialect）

段階:

構文チェック

$SET:

任意

NOCANCELLBRはそのような（ .lbr）ファイルをコンパイルの終わりまで開いたままとしておきます。ライブラリを指定しない後続のCOPY文は、現行ディレクトリの前の全ての開いている（ .lbr）ライブラリを探索（最後に開かれたものを最初に探索）してCOPYファイルを捜します。

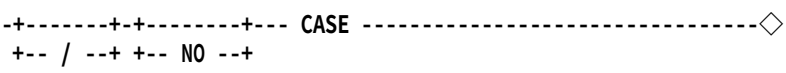
COPYLBRがオンの場合にのみ本指令は関連があります。

NOCANCELLBRは任意の$SETで使用できます。$SETの後のCOPY文で参照されている（ .LBR）ライブラリのみが影響を受けます。前のCOPY文から開いたままとなっている（ .LBR）ライブラリが閉じられるのは、CANCELLBR指令に続くCOPY文上で再度指定される場合のみです。

CASE

外部記号（PROGRAM-IDや呼ばれるプログラムの名前）が大文字に変換されるのを防ぎます。

構文:



パラメータ:

なし

属性:

省略値:

NOCASE

段階:

生成

$SET:

任意

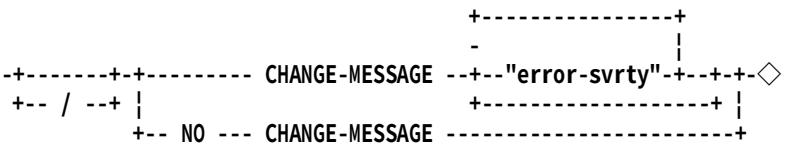
説明:

名前が大文字・小文字に依存している 'C' プログラムを呼び出している場合に本指令は有用です。

CHANGE-MESSAGE

エラー／メッセージの重大度を変更します。メッセージを表示しないようにすることも、元の重大度に戻すこともできます。

構文:



パラメータ:

エラー 変更するメッセージの番号またはすべてのメッセージを意味する "ALL"

重大度 新しい重大度

S 重大

E エラー

W 警告

I 通知

N メッセージを表示しない

R 通常の重大度に戻す

PS 重大。プリプロセッサ生成コードが原因。

PE エラー。プリプロセッサ生成コードが原因。

PW 警告。プリプロセッサ生成コードが原因。

PI 通知。プリプロセッサ生成コードが原因。

属性:
省略値: NOCHANGE-MESSAGE
段階: 構文チェック
$SET: 任意
依存性:
CHANGE-MESSAGEは、FLAGASのような指令の一般設定を変更します。
説明:

CHANGE-MESSAGEにはパラメータを必要なだけ指定することができます。複数回指定したCHANGE-MESSAGEの効果は累積されます。

CHANGE-MESSAGE指令はHIDE-MESSAGE指令に代わるものです。しかし、HIDE-MESSAGEをCHANGE-MESSAGEと一緒に使用すると、その効果も累積されます。

重大エラーの重大度を下げることはできません。ただし、他のCHANGE-MESSAGE指令またはFLAGAS指令ないしFLAGCD指令によって（重大度の低いものから）重大に変更されていたものは別です。

プリプロセッサにより生成されたメッセージ重要度の省略値を変更するには、Pxformを使用します。通常、重大ではないメッセージはすべてプリプロセッサにより挿入された行に生成されるときに抑制されます。

例:

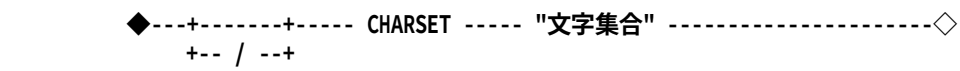
CHANGE-MESSAGE"ALL R"

CHANGE-MESSAGE"10 S 135 E 100 N"

CHARSET

環境の文字集合を定義します。

構文:



パラメータ:

文字集合 "ASCII" または "EBCDIC"

属性:

省略値:

CHARSET"ASCII" （Dialect）

段階:

構文チェック

$SET:

初期

依存性:

CHARSET"ASCII"は、直ちにDEFAULTBYTE"32"、SIGN"ASCII"およびNATIVE"ASCII"を設定します。

CHARSET"EBCDIC" は、直ちにDEFAULTBYTE"0"を設定し、最後にSIGN"EBCDIC"およびNATIVE"EBCDIC"を設定します。

説明:

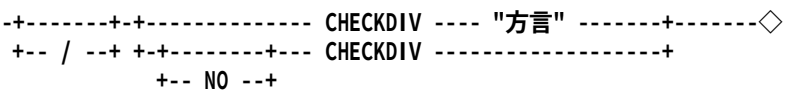
全ての定数と文字の大小順序が、指定された文字集合で扱われます。

CHARSET"EBCDIC" を設定すると、任意のパラメータにおいて英数字データを受け取ったり返したりするCOBOLシステムライブラリルーチンは動作しません。英数字データは、必ず"ASCII"でなければなりません。

CHECKDIV

ON SIZE ERROR指定のない文でゼロによる除算を試みた場合の、プログラムの動作を制御します。

構文:



パラメータ:

方言 ANSIのみ

属性:

省略値:

CHECKDIV"ANSI"（Dialect）

段階:

構文チェック

$SET:

初期

説明:

CHECKDIVまたはCHECKDIV"ANSI"を指定すると、ゼロによる除算を試みた場合でもプログラムは続行し、結果は不定です。NOCHECKDIVを設定すると、動作は不定となります。NOCHECKDIVを設定すると、除算についての最適なコードが生成されます。

ON SIZE ERROR指定を使用する算術文には本指令は効果ありません。

CHECKNUM

数字フィールドに対して操作を実行する前に、その数字フィールドに有効データがあるかどうかをチェックします。

構文:

```
  ◆---+-------+-+--------+--- CHECKNUM ----------------------------◇
       +-- / --+ +-- NO --+
```

パラメータ:

なし

属性:

省略値:

NOCHECKNUM

段階:

構文チェック

$SET:

初期

説明:

本指令を指定すると、余分なコードが生成されます。これは、数字フィールドに対して操作を実行する前にその数字フィールドに有効データがあるかどうかを確かめるためです。

CHECKNUMを設定すると、数字フィールドに数字データ以外のデータが含まれている場合に、ランタイムエラーメッセージ163（数字フィールドでは不正な文字（致命的なエラー））が表示されます。このエラーは、-F RTS スイッチを使用することにより無効にできます。

CHECKNUMを指定すると、余分なコードが生成されるため、プログラムの効率が低下します。効率を上げるには、CHECKNUMは指定しない方がいいでしょう。

COBFSTATCONV

ファイル上でI-Oエラーが検出されると、COBFSTATCONV環境変数で指定した利用者のモジュールを使用して、ファイル状態コードを変換します。

構文:

```
  ◆---+-------+-+--------+--- COBFSTATCONV ------------------------◇
       +-- / --+ +-- NO --+
```

パラメータ:

なし

属性:

省略値:

NOCOBFSTATCONV

段階:

構文チェック

$SET:

初期

COBIDY

Animator情報ファイル（ .idyファイル）が書き出されるパスを指定します。

構文:

```
◆---+-------+-+-------------- COBIDY ---- "パス名" -------+-------◇
    +-- / --+ +-+--------+--- COBIDY --------------------+
              +-- NO --+
```

パラメータ:

パス名 パス指定

属性:

省略値:

NOCOBIDY

段階:

構文チェック

$SET:

不可

説明:

パス名を指定しないと、COBIDY環境変数内のパス名が使用されます。環境変数に複数のパス名が入っている場合は、最初のパス名が使用されます。NOCOBIDYを設定すると、（ .idy）ファイルが目的プログラムと同じパスに書き出されます。

COBOLDIR

cobol.dirファイル中の指令を構文チェックプログラムが処理すべきか無視すべきかを指定します。

構文:

```
◆---+-------+-+--------+--- COBOLDIR ----------------------------◇
    +-- / --+ +-- NO --+
```

パラメータ:

なし

属性:

省略値:

COBOLDIR

段階:

構文チェック

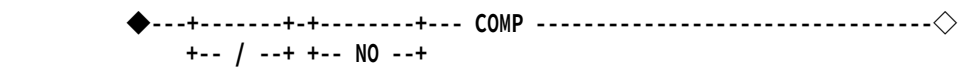
$SET:

不可

COMP

COMPデータ項目を含んだいくつかの文について、COMP項目をCOMP-Xとして扱うことによって、コンパイラに非常にコンパクトで効率的なコードを生成させます。

構文:



パラメータ:

なし

属性:

省略値:

NOCOMP

段階:

構文チェック

$SET:

初期

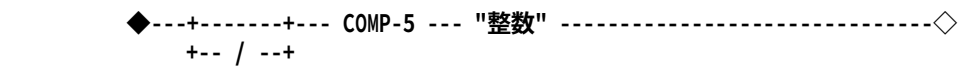
説明:

コンパイラにより生成された本COMPコードは、数字オーバフローの場合にANSI規格とは異なった動作をします。プログラムが数字オーバフローにならないことが分かっている場合、あるいは、オーバフロー時、定義された（しかし、規格外）動作を使用したい場合にのみ本指令を使用する必要があります。

COMP-5

符号なしCOMP-5データ項目に値を入れるとき符号を取り去るかどうか指定します。

構文:



パラメータ:

整数 1または2とする必要があります

属性:

省略値:

COMP-5"2"

段階:

構文チェック

$SET:

初期

説明:

「整数」の値として可能なものは次の通りです。

１ 符号は取り去られます。この動作は本コンパイラの初期のバージョンで使用されています。

２ 符号は取り去られません。負数は2の補数形で格納されます。従って、それら負数のバイト順がマシン依存である場合を除いて、符号なしCOMP-5項目はCOMP-Xのような動作をします。この結果、符号なしCOMP-5項目に対する演算が非常に効率的になります。

COMP-6

COMP-6データをバイナリまたはパック10進数のどちらの形式で保存するかを指定します。

構文:

```
◆---+-------+--- COMP-6 --- "整数" ------------------------------◇
     +-- / --+
```

パラメータ:

整数 1または2とする必要があります

属性:

省略値:

COMP-6"2"

段階:

構文チェック

$SET:

初期

依存性:

ADDRSV"COMP-6"を設定する必要があります。

説明:

「整数」の値として可能なものは次の通りです。

１ バイナリ形式。

２ パック10進数形式。項目が符号付きの場合、形式はCOMP-3と同じです。項目が符号なしの場合、符号付きフィールドは存在しません。

COMP-6指令を指定しても、予約語COMP-6は選択された方言に属する場合、またはADDRSV"COMP-6"指令が指定されている場合のみ認識されます。

例:

COMP-6"2"を指定した場合、
PIC 99 COMP-6 VALUE 87 は、x"87"として1バイトで保存されます。
PIC S99 COMP-6 VALUE 87 は、x"087"Cとして2バイトで保存されます。

COMS85

ANS85 COBOL規格の通信モジュールで採用されている構文をプログラムに入れることが可能になります。

構文:

```
  ◆---+-------+-+--------+--- COMS85 ------------------------------◇
      +-- / --+ +-- NO --+
```

パラメータ:

　なし

属性:

省略値:

　NOCOMS85 （Dialect）

段階:

　構文チェック

$SET:

　初期

CONFIRM

コンパイラに全ての後続指令を画面表示させます。

構文:

```
  ◆---+-------+-+--------+--- CONFIRM ------------------------------◇
      +-- / --+ +-- NO --+
```

パラメータ:

　なし

属性:

省略値:

　NOCONFIRM

段階:

　構文チェックおよび生成

$SET:

　不可

依存性:

　VERBOSEが直ちにCONFIRMを設定します。

CONSTANT

プログラムで使用する定数を宣言します。

構文:

```
  ◆---+-------+--- CONSTANT --定数名---+----- (数字定数) -----+-----◇
      +-- / --+                        +-----"英数字定数"-----+
```

パラメータ:

定数名 データ名。定数の名前

数字定数 数字定数。定数の値（ゼロまたは正の整数）

英数字定数 英数字定数。定数の値

属性:

省略値:

設定なし

段階:

構文チェック

$SET:

任意

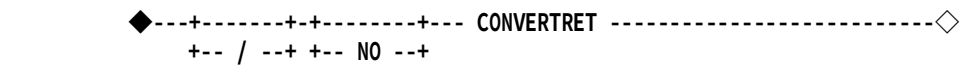
説明:

あたかも「定数名」がプログラムに78レベルで定義されたかのような効果があります。定数の値は括弧または引用符で囲まれた値です。括弧が使用されている場合、定数は項類が数字となり、引用符が使用されている場合、定数は項類が英数字となります。本機能を使用していくつかの表意定数を宣言するには、CONSTANT指令を繰返し使用します。

CONVERTRET

CALL ...RETURNINGとEXIT PROGRAM ...RETURNING指定で指定されたCOMP項目をCOMP-5に変換させます。

構文:



パラメータ:

なし

属性:

省略値:

NOCONVERTRET

段階:

構文チェック

$SET:

初期

CONVSPACE

COBOL原始ファイルに埋め込まれた2バイトの間隔文字を入力上1バイトのスペース列に変換するかどうか決定します。

構文:

パラメータ:

なし

属性:

省略値:

CONVSPACE

段階:

構文チェック

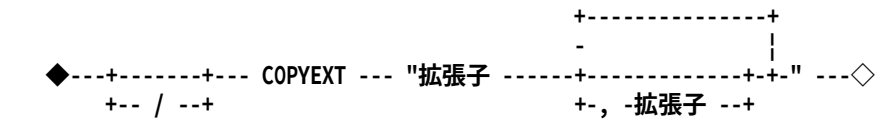
$SET:

初期

COPYEXT

COPY文内のファイル名が拡張子なしで指定されている場合、コンパイラが捜すCOPYファイルのファイル名拡張子を指定します。

構文:



パラメータ:

拡張子 ファイル名拡張子

属性:

省略値:

COPYEXT"cbl, cpy"

段階:

構文チェック

$SET:

初期

説明:

8個までの拡張子を指定でき、各拡張子の長さは最大10文字とします。空拡張子は拡張子がないことを示すために使用されます。例えば、

COPYEXT "SRC,,TXT"

COPY文に指定されたファイル名に拡張子またはファイル名の後にピリオドがない場合、可能性のある拡張子のリストがファイルが見つかるかリストがなくなる（エラーが通知されます）まで次々に表示されます。リストはCOPYEXTおよびOSEXT指令の値の組み合わせによるものです。そのリストの最初の拡張子は、OSEXT指令の値またはCOPYEXT指令の最初の値のどちらかです。これは、この2つの指令のどちらが最後に指定されたかによります。

リストの残りの値は、COPYEXT指令（終りまでの値が2つ）の値の残りです。例えば、

COPYEXT "CPY, CBL, TXT "OSEXT "SRC"

これは以下のものと同じ効果があります。

COPYEXT "SRC, CBL, TXT"

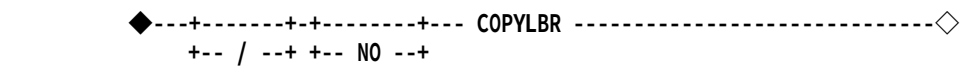
拡張子を指定していないたくさんのCOPY文がある場合、COPYEXTを使用してプログラムのコンパイル速度を上げることができます。

例えば、COPYファイル全てに拡張子（ .CPY）がついている場合、COPYEXT"CPY, CBL" を指定すると、不必要なファイルアクセスが避けられます。

COPYLBR

COPY文で指定されたライブラリを（ .lbr）ファイルとしてコンパイラに扱わせます。

構文:



パラメータ:

なし

属性:

省略値:

NOCOPYLBR （Dialect）

段階:

構文チェック

$SET:

不可

説明:

NOCOPYLBRを設定すると、コンパイラはそのライブラリがパス名であるとみなします。

COPYLIST

コンパイラにCOPY文で指定されているファイルの内容をリスト表示させます。

構文:

パラメータ:

整数 0か50～99とします。区分番号。

属性:

省略値:

COPYLIST

段階:

構文チェック

$SET:

任意

説明:

整数はCOBOL区分の番号です。整数は0または50～99の範囲とする必要があります。整数が指定されないと、全てのCOPYファイルの内容がリスト表示されます。整数が指定されると、最初の3個の部（つまり、見出し部、環境部、およびデータ部）、ルート区分、および与えられた区分内の全てのCOPYファイルの内容がリスト表示されます。整数0は最初の3個の部および全てのルート区分を指します。

NOCOPYLISTは任意のCOPYファイルの内容のリスト表示を防ぎます。整数がNOCOPYLISTで指定されると、その区分内のCOPYファイルだけがリスト表示されます。例えば、

COPYLIST"53" 最初の3個の部、ルート区分、および区分53内の全てのCOPYファイルをリスト表示

NOCOPYLIST"53" 区分53内のCOPYファイルのみをリスト表示

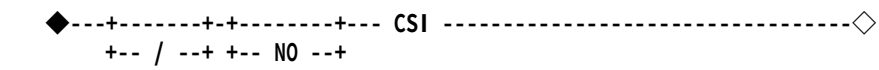
本指令の状態がどのようであっても、ページ頭書きが出力されているとき開いている任意のCOPYファイルの名前が、その頭書き内に与えられます。

CSI

コンパイラに追加情報を生成させ、利用者がCSIを使用できるようにします。

本指令はMicro Focusから提供されるオプション製品での使用に限定されています。該当するオプション製品を使用していない場合は、その設定を変更しないでください。

構文:



パラメータ:

なし

属性:

省略値:

CSI

段階:

構文チェック

$SET:

初期

依存性:

CSIを指定するとANIMが設定されます。

CURRENCY-SIGN

PICTURE句で認識される通貨記号を指定します。

構文:

```
◆---+-------+---- CURRENCY-SIGN ---- "整数" ----------------------◇
    +-- / --+
```

パラメータ:

整数 文字のASCIIコード。10進値で入力。

属性:

省略値:

CURRENCY-SIGN"36"（つまり、ドル記号 "$" ）で、日本語版以外の本COBOLシステムに適用されます。

CURRENCY-SIGN"92"（つまり、円記号 "¥" ）を、本COBOLシステムの日本語版用の省略値として、COBOL.DIRファイルに入れることができます。インストール中にCOBOL.DIRファイルのオプションを選択すると、この設定がCOBOL.DIR中に入れられます。

段階:

構文チェック

$SET:

初期

説明:

有効なPICTURE句記号を指定することはできません。これらのリストについては、「言語リファレンス」を参照してください。

CURRENT-DATE

CURRENT-DATE特殊レジスタに格納されるデータの形式を指定します。

構文:

```
◆---+-------+---- CURRENT-DATE ---- "日付形式" ------------------◇
    +-- / --+
```

パラメータ:

日付形式 DDMMYYまたはMMDDYY

属性:

省略値:

CURRENT-DATE"MMDDYY"

段階:

構文チェック

$SET:

任意

説明:

「日付形式」パラメータはCURRENT-DATEを必要な形式で格納させます。本パラメータは大文字・小文字のどちらでも指定できます。

DATACOMPRESS

順ファイルおよび索引ファイルに対して行うデータ圧縮の種類を指定します。

構文:

```
◆---+-------+-+--------- DATACOMPRESS --- "整数" ----+------------◇
     +-- / --+ +-- NO --- DATACOMPRESS ---------------+
```

パラメータ:

整数 下記のどれかでなければなりません。

０ NODATACOMPRESSに相当

１ 実行長の符号化を指定

３ 2バイト文字セット用の実行長の符号化を指定

属性:

省略値:

NODATACOMPRESS

段階:

構文チェック

$SET:

任意

説明:

このシステムでサポートしているデータ圧縮用に指定できる値は1と3だけです。128～255の範囲の値は利用者定義の圧縮ルーチンを示します。

データ圧縮を指定する必要があるのはファイルを作成するときだけです。その後、ファイルが開かれるときデータ圧縮が検出されます。

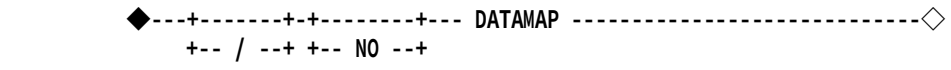
個別ファイルにデータ圧縮をかけるには、$SET文を利用者の原始コードで使用します。これにより、そのファイルのSELECT文が入った原始コード部分についてのみ本指令が有効となります。

データ圧縮は、呼出し可能ファイル・ハンドラのみがサポートしています。全索引ファイルが呼出し可能ファイル・ハンドラによって処理されます。順ファイルにデータ圧縮を使用する場合はファイルを参照するプログラムがCALLFH"EXTFH"指令を用いてコンパイルされていなければなりません。

DATAMAP

コンパイラにデータ項目に関する情報を出力させます。

構文:



パラメータ:

なし

属性:

省略値:

NODATAMAP

段階:

構文チェック

$SET:

初期

依存性:

なし

説明:

本指令は、コンパイラの確認段階でデータ項目（ユーザ指定のデータ項目およびRETURN-CODE特殊レジスタ用のデータ領域のようなコンパイラが自動的に生成するデータ項目）に関する情報を出力させます。

情報は、リストファイルの最後に出力されます。データ項目に関する情報は、コンパイラによってユーザ独自のプログラムにおける順番と同じ順番で出力されます。先頭行はデータ項目が属しているプログラムのPROGRAM-IDです。

PROGRAM-IDにより、入れ子になっているプログラムまたはOOクラスプログラム（クラス、オブジェクトメソッドなど）のあらゆる領域に関連するデータ項目を判別できます。

データ項目ごとに次の情報がリスト表示されます。

行番号
データ名
データ項目のアドレス。プログラムのデータ領域の先頭からのオフセットです。
バイト単位のデータ項目サイズ
データ項目の属性。次のフォーマットの文字列で構成されます。

```
ss xxxx cccccccccc R E G
```

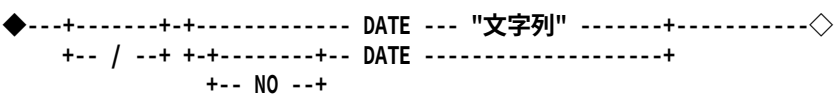
次の表は、上記のフォーマットで指定できる値のリストです。

ss	データ項目の位置	
	IO	入出力節
	FD	ファイル記述
	WS	作業場所節
	TL	スレッド局所節
	OS	オブジェクト記憶節
	LC	局所記憶節
	LS	連絡節
	CS	通信節
	RD	報告書節
	SS	画面節
	PG	特殊レジスタのようなコンパイラチェック段階で作成された変数
	PD	search文で使用されるような手続き部で生成された一時変数
xxxxx	G	集団項目
	GO	集団発生項目
	GSO	集団副発生項目
	E	基本項目
	EO	基本発生項目
	ESO	基本副発生項目
	ESOO	基本副再発生項目
ccccccccc	データ型の説明。例えば、Display、Comp、Comp-3、Numeric E、およびAlphNum J（英数字位置揃え用）など。	
R	データ項目が他のデータ項目の再定義である。	
E	データ項目が外部データ項目である。	
G	データ項目がグローバルデータ項目である。	

DATE

DATE-COMPILED段落とリストの各ページの先頭に日付をいれます。

構文:



パラメータ:

文字列 英数字定数

属性:

省略値:

DATE

段階:

構文チェック

$SET:

不可

説明:

日付と時間はオペレーティングシステムにより保持されています。DATEを指定すると、日付と時間は自動的にリストのDATE-COMPILED段落に挿入されます。しかし、利用者自身で日付をパラメータとして入力することもできます。NODATEを設定すると、DATE-COMPILED段落は変更しないで残されます。

DATEを設定すると、システム日付または利用者が入力する文字列がリストの各ページの先頭に現れます。NODATEを設定すると、スペースが代わりに使用されます。

DBCHECK

どの2バイト文字集合（DBCS）定数も有効な16ビットDBCS文字のみを含んでいることをコンパイラに確認させます。

構文:

パラメータ:

なし

属性:

省略値:

DBCHECK （Dialect）

段階:

構文チェック

$SET:

初期

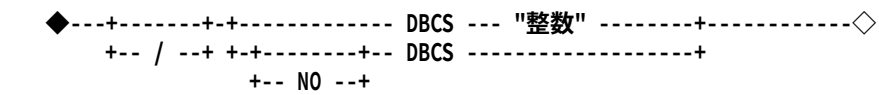
説明:

DBCSの妥当性検査をサポートする環境でDBCHECKを指定すると、有効な16ビットDBCS文字だけで構成されていない任意の定数は構文エラー1048（DBCS定数が無効データを含んでいる）となります。

DBCS

日本語、中国語および韓国語のような表意言語で使用する2バイト文字集合（DBCS）の文字を、コンパイラに受け取らせます。

構文:



パラメータ:

整数 1、2または3とする必要があります。

属性:

省略値:

DBCS"3" （Dialect）

段階:

構文チェック

$SET:

初期

依存性:

MF"7"が直ちにDBCS"2"を設定します。

MF"7"が直ちにDBCS"3"を設定します。

説明:

「整数」の値として可能なものは次の通りです。

１．本コンパイラの初期バージョンでの動作を提供します。

２．DBSPACEが設定されます。

IBM COBOL/370の場合のようにDBCS支援を含みます。「区切り文字N」で指定されるPIC N、PIC GおよびDBCS定数の使用をこれは含みます。

DBSPACE

表意定数SPACEがDBCS表意定数として使用されたとき、その表意定数をシステムの提供する2バイト間隔文字としてコンパイラに解釈させます。

構文:

```
◆---+-------+-+--------+--- DBSPACE -----------------------------◇
     +-- / --+ +-- NO --+
```

パラメータ:

なし

属性:

省略値:

DBSPACE （Dialect）

段階:

構文チェック

$SET:

任意

依存性:

MF"7"が直ちにDBSPACEを設定します。

説明:

DBSPACEが設定されると、コンパイラはシステムの提供する2バイト間隔文字を使用します。NODBSPACEは、2バイト間隔文字が2個のASCII間隔文字（x"2020"）であった本コンパイラの前のバージョンとの互換性を提供します。

DE-EDIT

数字編集項目から他の数字編集項目、または、数字項目への逆編集転記の動作を指定します。

構文:

```
◆---+-------+---- DE-EDIT ---- "整数" --------------------------◇
    +-- / --+
```

パラメータ:

整数 1または2とする必要があります。必要な互換性を示しています。

属性:

省略値:

DE-EDIT"2" （Dialect）

段階:

構文チェック

$SET:

任意

説明:

「整数」として可能な値は次の通りです。

1．送出し側項目のPICTURE句を無視します。この動作は本コンパイラの初期のバージョンのものと同じです。

2．送出し側項目のPICTURE句に従って逆編集します。これによりANSI85に準拠するようになります。

例:

```
01 a pic 909v99  value "30456".
01 b pic 9(5).
...
move a to b
```

DE-EDIT"1" を設定すると、Bには30456が入ります。DE-EDIT"2" を設定すると、Bには00034が入ります。3の後の0は落ちています。その理由は、その0はAの形式文字列内の挿入文字0に対応するからです。また、Bには小数位がないので .56も落ちています。

DEFAULTBYTE

データ部の各未定義バイトを与えられた文字に初期化します。

構文:

```
  ◆---+-------+---- DEFAULTBYTE ---- "整数" ----------------------◇
      +-- / --+
```

パラメータ:

整数 文字のASCIIコード、10進

属性:

省略値:

DEFAULTBYTE"32" （Dialect）

段階:

構文チェック

$SET:

初期

依存性:

CHARSET"ASCII"が直ちにDEFAULTBYTE"32"を設定します。

CHARSET"EBCDIC"、MS,IBM-MS、またはPC1が直ちにDEFAULTBYTE"0"を設定します。

説明:

整数は、10進数です。例えば、EBCDICスペースを指定する場合にはDEFAULTBYTE"64"を使用し、ASCIIスペースを指定する場合にはDEFAULTBYTE"32"を使用します。

DEFAULTCALLS

省略時呼出し方式を指定します。

構文:

```
  ◆---+-------+-+------------ DEFAULTCALLS --- "整数" ---+---------◇
      +-- / --+ +-+--------+-- DIRECTIVE-NAME ------------+
                +-- NO --+
```

パラメータ:

整数 省略時呼出し方式

属性:

省略値:

NODEFAULTCALLS

段階:

構文チェック

$SET:

任意

説明:

補助引数のない DEFAULTCALLS は、PROCEDURE DIVISION USING 文に指定されている呼出し方式を省略時呼出し方式として使用することを指定します。

DEFAULTCALLS"整数" は、「整数」で示される呼出し方式を省略時呼出し方式として使用することを指定します。

NODEFAULTCALLS は DEFAULTCALLS"0"と等価です。

個々の CALL 文はこれらの省略値に優先します。

DETECT-LOCK

DETECTLOCK

レコードが別のプログラムでロックされると、それをREAD文に検出させます。

構文:

```
  ◆---+-------+-+--------+-+--- DETECT-LOCK ----+-------------------◇
       +-- / --+ +-- NO --+ +--- DETECT-LOCK ---+
```

パラメータ:

なし

属性:

省略値:

DETECT-LOCK

段階:

構文チェック

$SET:

初期

説明:

コンパイラは、ハイフンのない名前、すなわち DETECTLOCK も認めます。

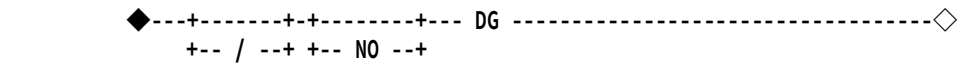
DETECT-LOCK を設定してあると、READ 文が別のプログラムでロックされたレコードを読み込むと、READ 文は 9/068 という入出力状態を返します。NODETECTLOCK を設定してあると、READ 文は 0/000 を返します。どちらの場合でも、READ 文はそのレコードの読込みには成功します。

DETECT-LOCK が指定されると、READ ... IGNORELOCK 構文を使用することによって、個々の READ 文はロックを無視します。

DG

ある機能の動作を変更してData General対話形COBOL rev 1.30との互換をとります。

構文:



パラメータ:

なし

属性:

省略値:

NODG （Dialect）

段階:

構文チェック

$SET:

初期

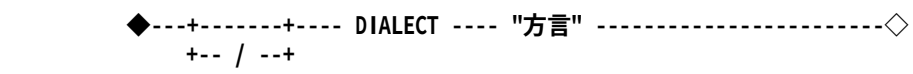
説明:

Data General 対話形 COBOL rev 1.30 構文支援の詳細については、「言語リファレンス」を参照してください。

DIALECT

チェックタイムおよびランタイム動作を指定された方言に合わせます。

構文:



パラメータ:

方言 MF、ANS85、またはISO2000。使用する方言を指定します。

属性:

省略値:

DIALECT"MF"

段階:

チェック

$SET:

初期

依存性:

省略時には、方言の設定によってあらゆるコンパイラ指令が指定されます。

DIALECTにより設定されるコンパイラ指令

コンパイラ指令DIALECT"方言"により、あらゆる指令が設定されます。次の表に、ANS85方言およびISO2000方言を指定した場合に設定される指令を示します。DIALECT"MF"を指定すると、NetExpressの省略時の指令がリセットされます。

ANS85	ISO2000
ALIGN"8"	ALIGN"8"

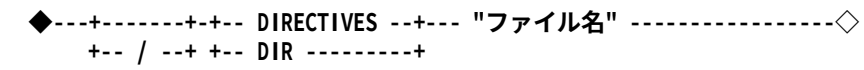
ALPHASTART"1"	ALPHASTART"1"
ALTER	ALTER
NOAMODE	NOAMODE
ANS85	NOANS85
ARITHMETIC"MF"	ARITHMETIC"MF"
CALLFH	CALLFH
CALLSORT"EXTSM"	CALLSORT"EXTSM"
CANCELLBR	CANCELLBR
CHARSET"ASCII"	CHARSET"ASCII"
CHECKDIV"ANSI"	CHECKDIV"ANSI"
COMS85	NOCOMS85
COPYEXT"cbl,CPY,"	COPYEXT"cbl,CPY,"
COPYLBR	COPYLBR
CURRENT-DATE"MMDDYY"	CURRENT-DATE"MMDDYY"
NODBCHECK	NODBCHECK
NODBCS	NODBCS
NODBCSOSSI	NODBCSOSSI
NODBSPACE	NODBSPACE
DE-EDIT"2"	DE-EDIT"2"
DEFAULTBYTE"32"	DEFAULTBYTE"32"
NODG	NODG
DYNAM	DYNAM
ECHO	ECHO
NOEXTINDEX	NOEXTINDEX
NOFDCLEAR	NOFDCLEAR
NOFLAG	FLAG"ISO2000"
NOFLAGAS	NOFLAGAS
FLAGCD"w"	FLAGCD"w"
NOFLAGMIG	NOFLAGMIG
FLAGSTD"H C2 D2 S2 R O"	NOFLAGSTD
FOLDCALLNAME"UPPER"	FOLDCALLNAME"UPPER"
FOLDCOPYNAME"UPPER"	FOLDCOPYNAME"UPPER"
NOHOSTNUMCOMPARE	NOHOSTNUMCOMPARE
NOHOSTNUMMOVE	NOHOSTNUMMOVE
NOIBM-MS	NOIBM-MS
INFORETURN"U"	INFORETURN"U"
	INTLEVEL"4"
IOCONV	IOCONV
NOISO2000	ISO2000
NOLIBRARIAN	NOLIBRARIAN
NOMAPNAME	NOMAPNAME
NOMF	NOMF
NOMFCOMMENT	NOMFCOMMENT
NOMS	NOMS

NESTCALL	NESTCALL
NOODOSLIDE	NOODOSLIDE
NOOLDCOPY	NOOLDCOPY
NOOLDINDEX	NOOLDINDEX
NOOLDNEXTSENTENCE	NOOLDNEXTSENTENCE
NOOLDREADINTO	NOOLDREADINTO
NOOLDSTRSUB	NOOLDSTRSUB
NOOPTIONAL-FILE	NOOPTIONAL-FILE
NOPC1	NOPC1
PERFORM-TYPE"MF"	PERFORM-TYPE"MF"
NOPROGID-COMMENT	NOPROGID-COMMENT
NOPROTECT-LINKAGE	NOPROTECT-LINKAGE
QUAL	QUAL
QUALPROC	QUALPROC
QUOTE	QUOTE
NORDW	NORDW
RECMODE"F"	RECMODE"F"
REMAINDER"1"	REMAINDER"1"
NORESEQ	NORESEQ
NORM	NORM
NOSEQCHK	NOSEQCHK
SOURCEFORMAT"FIXED"	SOURCEFORMAT"FIXED"
NOSPZERO	NOSPZERO
NOSTICKY-LINKAGE	NOSTICKY-LINKAGE
NOSTICKY-PERFORM	NOSTICKY-PERFORM
SYMBSTART"1"	SYMBSTART"1"
NOTRACE	NOTRACE
TRUNC"ANSI"	TRUNC"ANSI"
NOTRUNCCOPY	NOTRUNCCOPY
WARNING"3"	WARNING"3"
NOXOPEN	NOXOPEN
ZEROLENGTHFALSE	ZEROLENGTHFALSE
NOZEROSEQ	NOZEROSEQ
NOZWB	NOZWB

DIRECTIVES

コンパイラにファイルから指令を読み込ませます。

構文:



パラメータ:

ファイル名 完全なファイル指定

属性:

省略値:

なし

段階:

構文チェック

$SET:

任意

説明:

指令ファイルは、指令を含むテキストファイルです。ファイル内の各指令はスペースでお互いに分離しておく必要があり、指令を 2 行に渡って分割することはできません。

アンパサンドを行の 1 カラム目に置き、その後にスペースを入れることによって注釈をファイルに記述することができます。ただし、スペースを省略すると注釈は指令と認識され、そのファイルの構文チェックはエラーになります。

ファイルの終わりに達するか、あるいは、別の DIRECTIVES 指令が見つかるまで、これらの指令はファイルから読み込まれます。1 行の最大長は 128 文字です。

指令ファイル内に DIRECTIVES"ファイル名"を指定することによって、指令が 2 個以上のファイルをプログラムに指定できます。指令ファイル内に DIRECTIVES 指令を指定すると、コンパイラはその新しい指令ファイルに切り替わり、その中の全ての指令を読み込み、元の指令ファイルに戻り、そして、DIRECTIVES 指令の後に指定された指令の読込みを継続します。指令ファイルを任意の深さに入れ子にできます。

現行ディレクトリと COBOL システムディレクトリを探索して指令ファイルを捜します。拡張子が指定されてないと、探索を行う前に拡張子（ .dir）が付加されます。ファイルが見つからない場合は、拡張子を取り除いて探索を繰り返します。

DIRECTIVES-IN-COMMENTS

注釈行の中に記されている$SET文をコンパイラに処理させるようにします。

構文:

パラメータ:

なし

属性:

省略値:

NODIRECTIVES-IN-COMMENTS

段階:

構文チェック

$SET:

任意

説明:

この指令を設定すると、Micro Focus のコンパイラでコンパイルするときは適用し、メインフレームのコンパイラを始めとする他のコンパイラでは（注釈行として）無視する指令を、原始コード中に含めることができます。

$SET は、行のどこに置いてもかまいません。ただし、それが行の中で最初に出てくるものとします。この指令を設定すると、注釈行（カラム 7 にアスタリスク）の中の$SET 文でも処理されます。

EARLY-RELEASE

Micro Focusの初期の利用者構文を使用できるようにします。

構文:

```
◆---+-------+-+--------+--- EARLY-RELEASE -----------------------◇
     +-- / --+ +-- NO --+
```

パラメータ:

なし

属性:

省略値:

NOEARLY-RELEASE

段落:

構文チェック

$SET:

初期

依存性:

NOEARLY-RELEASE および MF が設定され、MF の省略値は MF"10"に設定されます。

EARLY-RELEASE および MF が設定され、MF の省略値は MF"12"に設定されます。

説明:

Micro Focus COBOL のあるバージョンで使用できる early-release 指令を使用するには、本指令および MF 指令を指定する必要があります。詳細は、MF 指令の項を参照してください。

ECHO

コンパイラにエラー行とエラーメッセージを画面に表示させます。

構文:

```
◆---+-------+-+--------+--- ECHO --------------------------------◇
     +-- / --+ +-- NO --+
```

パラメータ:

なし

属性:

省略値:

ECHO （Dialect）

段階:

構文チェック

$SET:

任意

依存性:

ECHOALL が直ちに NOECHO を設定します。

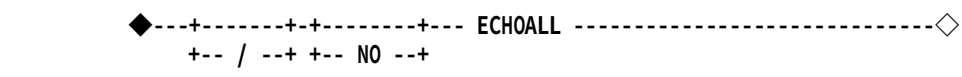
説明:

各エラーごとに、エラー番号および（BRIEF が設定されていなければ）説明文と一緒に原始コード行が表示されます。

ECHOALL

LISTまたはPRINT指令で指定されたプリンタまたはその他の装置と同様に画面にも完全なリストを送出します。

構文:



パラメータ:

なし

属性:

省略値:

NOECHOALL

段階:

構文チェック

$SET:

不可

EDITOR

コンパイラに、指定されたエディタと互換のある形式でファイルに、エラーメッセージを送出させます。

構文:

パラメータ:

エディタ識別子 MF、MF2、または MS とする必要があります

属性:

省略値:

NOEDITOR

段階:

構文チェックおよび生成

$SET:

不可

依存性:

EDITOR"MS"が直ちに NOENSUITE を設定します。

EDITOR"MS"が直ちに ENSUITE"1"を設定します。

説明:

「エディタ識別子」の値として可能なものは次の通りです。

MF Micro Focus エディタ

MF2 Micro Focus アニメータ V2

MS Microsoft プログラマーズワークベンチ

EDITOR 指令に加えて NOECHO または NOQUERY 指令を使用されることを推奨します。

ENSUITE

本指令はシステムが内部で使用するために予約されており、その設定を変更できません。

ERRLIST

リストファイルにおけるエラーメッセージの形式を指定します。

構文:

```
◆---+-------+-+--------+--- ERRLIST -----------------------------◇
    +-- / --+ +-- NO --+
```

パラメータ なし

オプション 必要なオプションを指定する定数

属性:

省略値:

ERRLIST"EMBED"

段階:

構文チェック

$SET:

不可

依存性:

ERRQ および FLAGQ が ERRLIST(EMBED)を設定します。

説明:

「オプション」の値として可能なものは次の通りです。

TERSE エラーを含む行だけがリストファイルに出力されます。原始プログラムの行とエラーの両方が出力されます。

EMBED エラー発生時に、組み込みエラーメッセージのある原始プログラムリストがリストファイルに出力されます。

END リストファイルにおいて、原始プログラムリストの後にエラーメッセージのリストが出力されます。バッチコンパイルでは、画面リストが選択されている場合のみエラーメッセージが画面に出力されます。

VERBOSE リストファイルにおいて、エラー発生時に、組み込みエラーメッセージのある原始プログラムリストの後にエラーメッセージの各リストが出力されます。

メインフレーム互換のリストは、ERRLIST"END"を使用すると取り出せます。特にリストファイルを生成するバッチコンパイルにおいては、この指令を使用すると、次のような効果があります。

- プログラムにエラーがある場合、画面には次のメッセージが表示されるだけです。

  ```
  Compilation completed with errors
  ```

- リストファイルでは、最後のブロックにエラーが出力されます。エラーごとにその行およびカラムがエラー番号と説明とともに表示されます。説明がLISTWIDTH指令が指定した形式の行に適さない場合には、再書式化され数行に分割されることもあります。

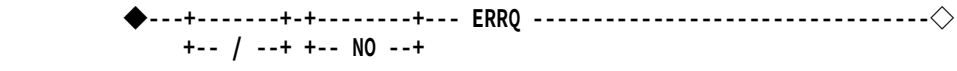
致命的なエラーが発生すると、形式はERRLIST(EMBED)に戻り、コンパイラの構文チェック段階は直ちに終了します。ERRLIST(END)がまだ有効な場合は、リストが最後に達した場合を除いてエラーは表示されません。

ERRQまたはFLAGQ指令のいずれかが設定されると、その指令が正常に動作しているかどうかを確認するために、コンパイラはERRLIST(END)をERRLIST（EMBED)に設定します。

ERRQ

コンパイラがエラーメッセージを出すたびに、コンパイルを停止するかどうかコンパイラが利用者に尋ねるようにします。

構文:



パラメータ:

なし

属性:

省略値:

NOERRQ

段階:

構文チェック

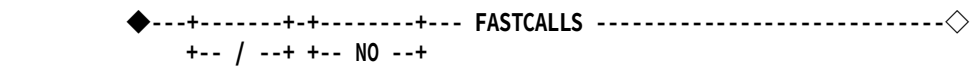
$SET:

任意

FASTCALLS

呼び出されたプログラムの動作を制御します。FASTCALLSを指定すると、プログラムは、それがメインプログラムであるかどうかを決定させるような情報を提供しません。そのため、ランタイムシステムはプログラムはメインプログラムではないものと仮定し、EXIT PROGRAMが必ず呼出しプログラムを終了させます。

構文:



パラメータ:

なし

属性:

省略値:

NOFASTCALLS

段階:

生成

$SET:

初期

説明:

FASTCALLS を指定してコンパイルされたプログラムは、そこからさらに呼び出されるプログラムに、それがメインプログラムであるかどうかを判別できるような情報を提供しません。したがって、FASTCALLS を指定してコンパイルされたプログラムが呼び出すプログラムも FASTCALLS を指定してコンパイルする必要があります。

また、FASTCALLS 指令を指定すると、プログラムにさらに制限を加える NOPARAMCOUNTCHECK 指令が有効になります。これによりプログラムはそれがいくつのパラメータを指定して呼び出されたのか分からなくなるため、STICKY-LINKAGE"2"または PROTECT-LINKAGE 動作を正しくサポートできません。これらのいずれかがユーザアプリケーションで重要な場合には、FASTCALLS 指令を使用しないでください。詳細は、PARAMCOUNTCHECK 指令の項を参照してください。

FASTLINK

コンパイラに、手続き部の文および各ENTRY文のUSING句のパラメータがある制限に準拠していることを知らせます。これにより、コードの処理が速くなります。

構文:

パラメータ:

なし

属性:

省略値:

NOFASTLINK

段階:

生成

$SET:

初期

説明:

パラメータは、どのエントリポイントにおいても同じ順番でなければなりません。ただし、この順番は連絡節中の01レベルの項目として表示される順番と同じである必要はありません。

プログラムは、NOFIXOPTおよびNOTYPECHECKコンパイラ指令を指定してコンパイルしなければなりません。

FASTSORT

この指令は、システムが内部で使用するために予約されています。設定を変更してはいけません。

構文:

パラメータ:

なし

属性:

省略値:

FASTSORT

段階:

構文チェック

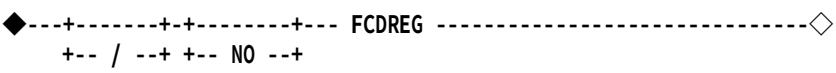
$SET:

初期

FCDREG

ファイルのファイル管理記述（FCD）とキー定義ブロックへのアクセスのための特殊レジスタを、コンパイラに定義させます。

構文:



パラメータ:

なし

属性:

省略値:

NOFCDREG

段階:

構文チェック

$SET:

初期

説明:

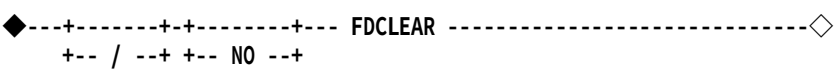
FCDREG は特殊レジスタ FH--FCD をプログラム内の各ファイル定義（FD）ごとに作成させます。本レジスタはファイルについてのファイル管理記述（FCD）を指します。従って、プログラムは FCD 内の情報を読み込んだり、修正したりできます。

各索引ファイルごとに、追加の特殊レジスタ FH--KEYDEF が作成されます。本レジスタはファイルについてのキー定義ブロックを指します。

FDCLEAR

書出し操作を行うたびに、ファイルのレコードバッファをクリアすることを指定します。バッファをクリアした後の値を何にするかは、DEFAULTBYTE指令に指定します。

構文:



パラメータ:

なし

属性:

省略値:

NOFDCLEAR （Dialect）

段階:

構文チェック

$SET:

任意

説明:

FDCLEAR 指令は SELECT 句を処理している間だけ有効です。選択したファイルにこの機能を適用するには、各 SELECT 句の前後に$SET 文を置きます。その他に、どの SELECT 文よりも前に 1 つだけこの指令を置く方法があります。その場合は、プログラム中のすべてのファイルにこの指令が適用されます。

I-O-CONTROL 段落中に SAME RECORD AREA 句が指定されているファイルに関しては、FDCLEAR 指令は無視されます。

例:

下に示す例に置いては、FDCLEAR は file-2 に対してのみ適用されます。

```
 file-control.
     select file-1 ...
$set fdclear
     select file-2 ...
"set nofdclear
     select file-3 ...
```

FILESHARE

マルチユーザー環境でのファイルについて、省略時ロックをEXCLUSIVE（排他的）ではなくAUTOMATIC（自動）とし、プログラムが複数レコードをロックするとき、WRITEまたはREWRITE文に関するレコードを自動的にロックします。

本指令は、Fileshareのバージョン2とは関係ありません。したがって、Fileshareのバージョン2を使用するときに、本指令を使用しないでください。

構文:

```
  ◆---+-------+-+--------+--- FILESHARE ---------------------------◇
       +-- / --+ +-- NO --+
```

パラメータ:

なし

属性:

省略値:

NOFILESHARE

段階:

構文チェック

$SET:

初期

説明:

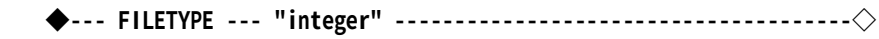
本指令は以前のファイル共有製品との互換を取るためのものです。新しいプログラムを書くとき、本指令ではなくロック構文を使用する必要があります。

FILESHARE は WRITELOCK と AUTOLOCK を合わせたものに相当します。NOFILESHARE は NOAUTOLOCK と NOWRITELOCK を合わせたものに相当します。

FILETYPE

ファイルを作成するとき使用するファイル形式を指定します。

構文:



パラメータ:

整数 使用する型を示す整数

属性:

省略値:

FILETYPE"0"

段階:

構文チェック

$SET:

任意

依存性:

FILETYPE"integer"は直ちに IDXFORMAT"integer"を設定します。

説明:

「整数」の値として可能なものは次の通りです。

０ システム特有省略値（本 COBOL システムの場合、3 と同じ）

１ C-ISAM 形式

２ Micro Focus Level II 形式

３ Micro Focus COBOL 形式

４ 本システムにより使用される形式を最適化したもので、重複キー処理が速くなります

５～255 将来の予備

本指令は、呼出し可能ファイルハンドラによって処理されたファイルに対してのみ有効です。索引ファイル以外のファイルを処理している場合には、CALLFH 指令を使用します。

FIXOPT

生成コードが目的（.obj）コードのデータ部にアクセスする方法を変更します。

構文:

パラメータ:

なし

属性:

省略値:

NOFIXOPT

段階:

生成

$SET:

初期

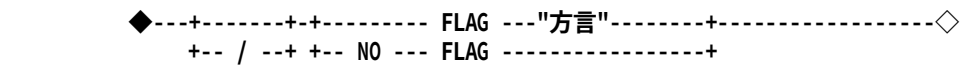
説明:

目的コードの作成時にコード生成エラー 78（"Too many code relocations (limit=limit, actual=actual-relocations)"）を受け取った場合には、FIXOPT を指定して再コンパイルする必要があります。

FLAG

コンパイラがCOBOLの指定された方言ではない構文を見つけたとき、コンパイラに言語レベルの検証フラグを生成させます。

構文:



パラメータ:

方言 方言を識別する定数

属性:

省略値:

NOFLAG （Dialect）

段階:

構文チェック

$SET:

任意

説明:

「方言」の値として可能なものは次の通りです。

MF Micro Focus

ANS74 ANSI COBOL 標準 X3.23, 1974

ANS85 ANSI COBOL 標準 X3.23, 1985

ISO2000 推奨 ISO2000 COBOL 標準

DG、RM、または MS を方言として使用することはできません。

ANSI '85 に完全に準拠したプログラムを作成するには、

ANS85 FLAG"ans85"

これを使用し、また、フラグメッセージを生成する原因となるものを修正します。

FLAGAS

フラグメッセージをエラーメッセージ、警告メッセージまたは情報メッセージとして、コンパイラに出力させます。

構文:

```
  ◆---+-------+-+--------- FLAGAS ---"重大度"--------+-------------◇
      +-- / --+ +-- NO --- FLAGAS ------------------+
```

パラメータ:

重大度 フラグメッセージに割り付ける重大度を示す定数

属性:

省略値:

NOFLAGAS （Dialect）

段階:

構文チェック

$SET:

任意

説明:

「重大度」の値として可能なものは次の通りです。

S 重大エラー

E エラー

W 警告

I 情報

FLAGCD

FLAG指令と一緒に使用されるとき、COBOLの指定された方言との互換性のない任意の指令設定に対してFLAGCDはフラグを立てます。

構文:

```
  ◆---+-------+-+--------- FLAGCD ---"重大度"--------+-------------◇
      +-- / --+ +-- NO --- FLAGCD ------------------+
```

パラメータ:

重大度 矛盾する指令に起因するフラグメッセージに割り付ける重大度を示す定数

属性:

省略値:

NOFLAGCD （Dialect）

段階:

構文チェック

$SET:

任意

説明:

「重大度」の値として可能なものは次の通りです。

S 重大エラー

E エラー

W 警告

I 情報

FLAGCD がパラメータなしで指定されると、FLAGAS 指令が指定されていない限り、メッセージはフラグメッセージとして出力されます。FLAGAS が指定されており、FLAGCD がパラメータなしで指定されると、メッセージには FLAGAS 指令により与えられる重大度が割り当てられます。

FLAGQ

コンパイラがフラグメッセージを出すたびに、コンパイルを停止するかどうかコンパイラが利用者に尋ねるようにします。

構文:

パラメータ:

なし

属性:

省略値:

NOFLAGQ

段階:

構文チェック

$SET:

任意

説明:

本指令を IDE 内で使用しないでください。

FLAGSINEDIT

フラグメッセージをエラーファイルに入れるかどうか指定します。

構文:

パラメータ:

なし

属性:

省略値:

FLAGSINEDIT

段階:

構文チェックおよび生成

$SET:

不可

説明:

NOEDITOR が指定されていると本指令は効果がありません。EDITOR と FLAGSINEDIT が指定されると、生成されるエラーファイルはコンパイラにより生成される全てのフラグメッセージを格納します。

FLAGSTD

コンパイラがANSI85の指定したレベルの一部ではない構文を見つけたとき、コンパイラに言語レベルの検証フラグを生成させます。

構文:

```
◆---+-------+-+--------- FLAGSTD ---- "文字列" --------+----------◇
     +-- / --+ +-- NO --- FLAGSTD ---------------------+
```

パラメータ:

文字列 下記参照

属性:

省略値:

NOFLAGSTD （Dialect）

段階:

構文チェック

$SET:

初期

説明:

文字列には、ANSI 標準の言語レベルまたは選択モジュールをそれぞれ定義したパラメータをリスト形式で指定します。ある機能が、指定されたレベル以下のレベル、または指定された選択モジュールの 1 つにある場合には、フラグは設定されません。しかし、ある機能が標準よりも高いレベル、または指定されていない選択モジュールにおける拡張である場合には、フラグが設定されます。

さらに、ANSI 標準によって廃言語要素とみなされた要素にフラグを設定することもできます。

「文字列」には次のようなパラメータの組み合わせを指定できます。

M　ANSI85 に定義された最小 COBOL 部分集合を越える全ての項目に対してフラグを立てます。

I　ANSI85 に定義された中間 COBOL 部分集合を越える全ての項目に対してフラグを立てます。

H　ANSI85 に定義された高 COBOL 部分集合を越える全ての項目に対してフラグを立てます。

C1　通信選択モジュールレベル 1 のフラグ設定を除外します。

C2　通信選択モジュールレベル 2 のフラグ設定を除外します。

D1　デバッグ選択モジュールレベル 1 のフラグ設定を除外します。

D2　デバッグ選択モジュールレベル 2 のフラグ設定を除外します。

S1　区分化選択モジュールレベル 1 のフラグ設定を除外します。

S2　区分化選択モジュールレベル 2 のフラグ設定を除外します。

R　報告書作成選択モジュールのフラグ設定を除外します。

O　全ての廃言語要素のフラグ設定を含みます。

パラメータは任意の順序とできますが、少なくとも 1 個のスペースで分離する必要があります。次のようなフラグの組み合わせを指定できます。

- 次のどれか1つを選択する必要があります: M、I、H
- 次のものを必要なだけ選択できます: CIまたはC2

D1 または D2

S1 または S2

R

- 次のフラグを選択することも可能です: 0

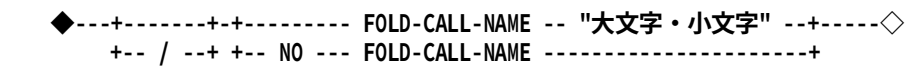
FLAG と FLAGSTD は同等の機能性を提供します、従って、どちらか 1 つだけを使用してください。

ANS85 指令はオンとする必要があります。

FOLD-CALL-NAME

CALL、CANCEL、ENTRY、およびCHAIN文に関連する識別子／定数、およびPROGRAM-ID段落のプログラム名を大文字または小文字に変換します。

構文:



パラメータ:

大文字・小文字 "UPPER"または"LOWER"

属性:

省略値:

NOFOLD-CALL-NAME （Dialect）

段階:

構文チェック

$SET:

初期

説明:

コンパイラは、ハイフンのない指令名、すなわち FOLDCALLNAME も認めます。

CALL"識別子"文において呼び出される名前は、本指令で指定した大文字または小文字で与えられます。ただし、識別子の実際の内容に変わりはありません。

本指令は、大文字小文字を区別しない環境から区別する環境に COBOL コードを転送する場合に役に立ちます。

また、本指令は、コンパイラを ANSI85 標準のプログラム間連絡モジュールに準拠させます。ANSI85 標準では、CALL 文のプログラム名に大文字小文字の区別がありません。

本指令を指定すると、通常の COBOL 規定では、引用符で囲まなければプログラム名は大文字に変換されます。これは、入れ子になっているプログラムの基本プログラムに適用されます。副プログラム名については、引用符で囲まれていてもすべて大文字に変換されます。（プログラム名を引用符で囲むというのは、Micro Focus の拡張機能です。）

プログラム名は、完全にリンクされた実行可能ファイルにおいては、エントリポイントとしてもアクセス可能です。

FOLD-COPY-NAME

COPYファイルの名前を大文字または小文字に変換すべきかどうか決定します。

構文:

```
◆---+-------+-+--------- FOLD-COPY-NAME -- "大文字・小文字" --+-----◇
     +-- / --+ +-- NO --- FOLD-COPY-NAME ---------------------+
```

パラメータ:

大文字・小文字 "UPPER"または"LOWER"

属性:

省略値:

NOFOLD-COPY-NAME （Dialect）

段階:

構文チェック

$SET:

任意

説明:

コンパイラは、ハイフンのない指令名、すなわち FOLDCOPYNAME も認めます。

本指令は、テキストおよびライブラリ名がユーザ定義語または定数として指定されているかどうかに関わらず、どちらにも作用します。

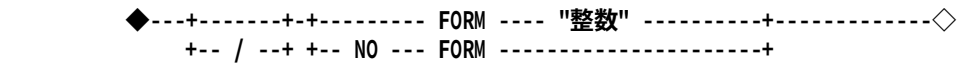
例:

FOLDCOPYNAME"UPPER"を指定すると、COPY ファイルの名前を大文字に変換します（大文字に折りたたむとも言われます）。

FORM

リストの各ページの行数を指定します。

構文:



パラメータ:

整数 4 以上を指定する必要があります

属性:

省略値:

FORM"60"

段階:

構文チェックおよび生成

$SET:

任意

依存性:

NOLIST または MASM が最後に NOFORM を設定します。

LIST は直ちに FORM"60"を設定します。

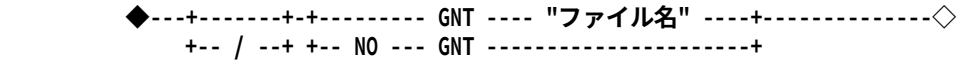
説明:

FORM"整数"を設定すると、リストファイルの先頭で書式送り文字がいつも生成されます。NOFORM を指定すると、書式送り文字およびページ頭書きのいずれも、リストのどこにも生成されません。

GNT

目的コードファイルまたは生成コードファイルの名前を指定します。

構文:



パラメータ:

ファイル名 完全なファイル指定

属性:

省略値:

GNT"原始ファイル名 .GNT"（OMF"GNT"を設定した場合）

GNT"原始ファイル名 .OBJ"（OMF"OBJ"を設定した場合）

段階:

生成

$SET:

任意

説明:

本指令は OMF 指令よりも優先させて使用してください。

本指令は、指定された名前で指定された形式のファイルを作成します。たとえば、GNT"myprog.gnt"を指定した場合には、生成コードを含むファイル myprog.gnt が生成されます。

NOGNT を設定すると、目的コードファイルは生成されません。

HIDE-MESSAGE

この指令はCHANGE-MESSAGE指令に置き換えられました。互換性を保つためだけに、この指令は残されています。将来のリリースにおいては、この指令は削除される予定です。

構文チェックエラーの番号を隠すために登録します。従って、そのエラーが見つかっても無視されます。

構文:

パラメータ:

整数 隠そうとする構文チェックエラーメッセージの番号

属性:

省略値:

NOHIDE-MESSAGE

段階:

構文チェック

$SET:

任意

説明:

コンパイラはハイフン抜きの指令名も受け付けます。つまり、HIDEMESSAGE も使用可能です。

HIDE-MESSAGE"整数"を指定すると、最大 20 個の構文チェックエラーメッセージ番号を収容するリストに整数を追加します。本機能を用いていくつかのエラーメッセージ番号を隠すには、HIDE-MESSAGE 指令を繰返し使用する必要があります。

プログラムが構文チェックを受けているとき、レベルが E、W、I、フラグのメッセージは、いかなるリストにも示されません。コンパイルの最後のエラー一覧にも含まれません。

レベルが S のメッセージは、CHANGE-MESSAGE、FLAGAS、または FLAGCD 指令によってレベル S が与えられた場合のみ隠すことができます。

NOHIDE-MESSAGE は番号リストを消去しますので、メッセージは隠されません。

"NOHIDE-MESSAGE"整数"を指定すると、指定した番号のメッセージだけがリストから削除されます。

HOSTSIGNS

不正な符号ニブルを持つCOMP-3項目の算術により期待する結果が得られるようにします。

構文:

パラメータ:

整数 1 または 2

属性:

省略値:

NOHOSTSIGNS

段階:

生成

$SET:

初期

説明:

HOSTSIGNS を指定すると、COMP-3 算術式により最適化されていないルートで処理が行われるため、プログラムの効率が低下します。

効率を上げるためには、HOSTSIGNS を指定しないことです。

IBM-MS

IBM COBOL 1.00で予約されている語を予約語とするよう指定し、その製品と互換のある、ある機能の動作を変更します。

構文:

パラメータ:

なし

属性:

省略値:

NOIBM-MS （Dialect）

段階:

構文チェック

$SET:

初期

依存性:

IBM-MS は、DEFAULTBYTE"0"および ACCEPTREFRESH を直ちに設定します。

説明:

本指令は PC1 および MS"1"指令と同義です。

IBMCOMP

語記憶モードをオンにします。

構文:

パラメータ:

なし

属性:

省略値:

NOIBMCOMP （Dialect）

段階:

構文チェック

$SET:

初期

依存性:

IBMCOMP は、ALIGN"8"を直ちに設定します。

説明:

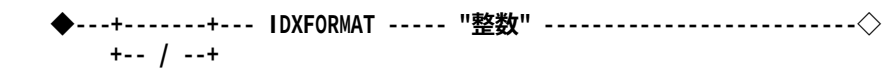
語記憶モードでは、USAGE COMP または USAGE COMP-5 の各データ項目は、2 バイトまたは 4 バイトの倍数を占有します。IBMCOMP を指定し、プログラムにおいて USAGE COMP または USAGE COMP-5 の任意の項目で SYNCHRONIZED 句を使用する場合、ALIGN"1"を指定しないでください。

任意の Micro Focus システムルーチンを直接呼び出す場合には、IBMCOMP の使用には十分注意してください。それは、そのルーチンに指定したパラメータが正しく整列されなかったり、サイズが不正になってしまうことがあるからです。この場合、ルーチンは誤った結果を返します。このようなパラメータには、CBL_ライブラリルーチン、Adis の呼出しまたは呼出し可能ファイルハンドラの方言呼出し、Extfh 用のパラメータなどがあります。

IDXFORMAT

索引ファイルを作成するときに使用する形式を指定します。

構文:



パラメータ:

整数 使用する型を示す整数

属性:

省略値:

IDXFORMAT"0"

段階:

構文チェック

$SET:

任意

依存性:

FILETYPE"整数"が直ちに IDXFORMAT"整数"を設定します。

説明:

「整数」の値として可能なものは次の通りです。

0 定システム特有省略値（本 COBOL システムでは、3 と同じ）

1 将来の予備

2 Micro Focus Level II V2.5 形式

3 本 COBOL システムにより使用される形式

4 本システムにより使用される形式を最適化したもので、重複キー処理が速くなります

5 Btrieve 形式ファイル（ANSI 準拠エミュレーションあり）

6 Btrieve 形式ファイル（ANSI 準拠エミュレーションなし）

7 将来の予備

8 大索引付き

9～255 将来の予備

既存のファイルは、どのような形式でも正しく処理されますので、この指令を指定する必要はありません。この指令は、新しいファイルを作成するときの形式を制御します。

3 を指定すると、必ずこのシステムで使用する形式でファイルが作成されます。ところが、0 を指定し別のシステムのファイルハンドラをプログラムに適用していると、そのシステムの省略時の形式でファイルが作成されます。

4 を指定すると、IDXFORMAT"3"の場合よりも、ファイルを大きくします。

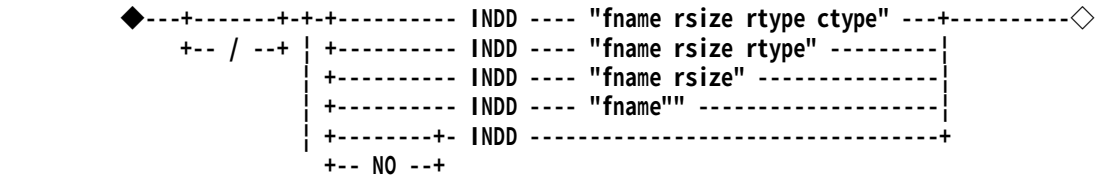
Micro Focus Level II V2.5 形式のファイルは DOS のもとで稼働する Micro Focus 製品と互換性があります。そのような Micro Focus 製品の具体例としては、Level II COBOL V2.5、Professional COBOL V1.2、V5 COBOL Workbench V1.3 以上が挙げられます。

IDXFORMAT"2"を使用するときは、ANS85 指令を指定して ANSI '85 の動作を有効化してはなりません。しかし、ANS85"SYNTAX"を指定して ANSI'85 の構文を使えるようにしてもかまいません。

INDD

ACCEPT文を指定したファイルから読めるようにします。

構文:



パラメータ:

fname 指定した ACCEPT 文を読み込むファイルの名前。このパラメータを指定しない場合は、ファイル名は SYSIN と設定されます。

rsize ファイル中のデータレコードのサイズ。このパラメータを指定しない場合は、サイズは 80 と設定されます。

rtype 行順ファイルの L またはレコード順ファイルの R。このパラメータを指定しない場合は、L が設定されます。

ctype ASCII ファイルの A または EBCDIC ファイルの E。このパラメータを指定しない場合は、A が設定されます。E は CHARSET"EBCDIC"指令を使用した場合のみ効果があります。

属性:

省略値:

NOINDD

段階:

構文チェック

$SET:

初期

説明:

INDD を指定すると、FROM オプションが指定されていないか FROM SYSIN（または SYSIN に関連する mnemoric-name）を指定する形式 1 の ACCEPT 文はすべて READ 文に変換されます。そして指定した外部ファイル名のファイルから読み込まれます。

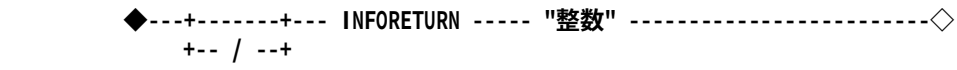
ファイル名は、他のファイルと同じように外部ファイル名を使用して物理ファイル名にマップすることができます。つまり、環境変数または外部ファイルマッパを使用します。

本指令の省略時の設定は、SYSIN 指令が使用されるときのこのタイプの ACCEPT 文用に使用される設定と同じです。

INFORETURN

コンパイラが情報メッセージのみを生成するとき、コンパイラにより返されるリターンコード値を指定します。

構文:



パラメータ:

整数 0～4 とする必要があります

属性:

省略値:

INFORETURN"0" （Dialect）

段階:

構文チェック

$SET:

任意

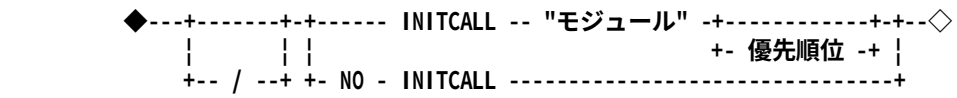
説明:

コンパイラが終了すると、コンパイルの成功・失敗を決めるためにオペレーティングシステムコマンドで試験できる値を、コンパイラは返します。コンパイラが情報メッセージのみを出す場合は、本指令を用いて返される値を設定できます。

INITCALL

プログラムの最初の文が実行される前に直ちに呼び出されるモジュールを指定します。

構文:



パラメータ:

モジュール 呼び出されるモジュール

優先順位 モジュールの実行に割り当てる優先順位

属性:

省略値:

NOINITCALL

段階:

構文チェック

$SET:

初期

説明:

「優先順位」の値として可能なものは次の通りです。

H 高優先順位（優先順位が指定されていない場合の省略値）
L 低優先順位

INITCALL を指定すると、コンパイラに対して指定モジュールに呼出しを挿入させます。実行時、任意の手続きコードが実行される前に指定モジュールは呼び出されます。呼ばれるモジュールにパラメータを渡すことはできません。

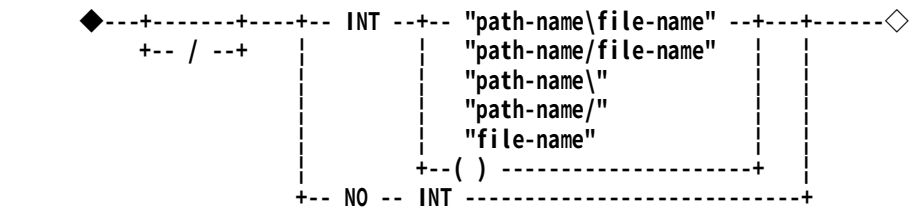
本機能を用いていくつかのモジュールを呼び出すには、INITCALL 指令を繰返し使用する必要があります。NOINITCALL は呼ばれるモジュールのリストを消去します。

高優先順位の呼出しは、全ての低優先順位の呼出しおよびコンパイラによるその他の呼出しの前に置かれます。低優先順位呼出しはコンパイラによるその他の呼出しの後に置かれます。同じ優先順位の呼出しは指定された順に実行されます。

INT

中間コードファイルの名前を指定します。

構文:



パラメータ:

パス名 パスまたはパスを指定する環境変数

ファイル名 完全なファイル指定

属性:

省略値:

INT "原始ファイル名 .int"

段階:

構文チェック

$SET:

不可

依存性:

NOINT は NOANIM を直ちに設定します。

説明:

省略時に INT()指令が実行された場合、その動作により一時中間コードファイルが構文チェック段階で作成されます。このファイルは、(.obj）ファイルまたは（.gnt）ファイルの作成が正常終了すると削除されます。

コンパイラによる（.obj）ファイルまたは（.gnt）ファイルの作成後も（.int）ファイルを保持したい場合には、INT()指令が省略時に設定されることに頼らず、明示的に INT()を設定する必要があります。

NOINT は中間コードファイルが生成されるのを防ぎます。

現存するファイルを指定すると、そのファイルは上書きされます。

「ファイル名」が指定されないと、コンパイラは原始ファイル名に拡張子（ .int）を付けて使用します。原始ファイル名は、原始プログラムファイル名の基本名です。ファイル名ではなくパス名を指定した場合には、コンパイラはパス名を使用してそれに「原始ファイル名.int」を付けます。

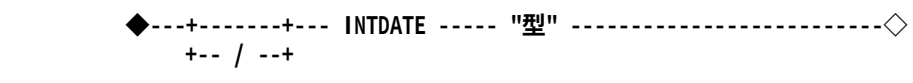
INT（）を指定すると、中間コードが標準ファイル「原始ファイル名 .INT」に格納されます。このパラメータの場合、引用符ではなく括弧を使用する必要があることに留意してください。したがって、INT""では結果が得られません。

INT 指令の使用には注意してください。間違った使用はコンパイル処理を中止させることがあります。

INTDATE

組み込み関数により整数形式の日付で開始日を選択します。

構文:



パラメータ:

型 開始日の型。

属性:

省略値:

ANSI

段階:

構文チェック

$SET:

任意

説明:

本指令は、組み込み関数で使用される整数形式の日付で開始日を選択します（つまり、DATE-OF-INTEGER、DAY-OF-INTEGER、INTEGER-OF-DATE、INTEGER-OF-DAY）。

型には、次の値を指定できます。

ANSI コンパイラに ANSI COBOL 標準の開始日（Day1=Jan1, 1601）を使用するように指示します。

LILIAN コンパイラに Lilian の開始日（Day1=Oct15, 1582）を使用するように指示します。

date 組み込み関数についての詳細は、「言語リファレンス」を参照してください。

INTLEVEL

コンパイラにより中間コードが作成されているとき、本指令は作成されるコードの他の環境にある異なるバージョンのMicro Focus製品への移植性のレベルを制御します。

構文:

```
◆---+-------+-------- INTLEVEL ---- "整数" --------+-------------◇
    +-- / --+-- NO -- INTLEVEL --------------------+
```

パラメータ:

整数 移植性のレベルを指定します。2 または 4 を指定します。

属性:

省略値:

INTLEVEL"2"

段階:

構文チェック

$SET:

不可

説明:

中間コード移植性についての詳細説明は、関連する Micro Focus オプション製品と一緒に提供されています。

NOINTLEVEL を指定すると、本環境でのみの実行に適した中間コードが作成されます。

INTLEVEL"整数"を指定すると、あるバージョンの Micro Focus 製品により他の環境で実行できる中間コードが作成されます。環境間の移植を可能にするため、コンパイルに使用される整数の値は、その上で中間コードを実行しようとする各環境の Micro Focus 製品によりサポートされている必要があります。

INTLEVEL"整数"はプログラムで使用できる構文を制限することがあります。

推奨 ISO2000 標準で定義されている構文を使用しているプログラムをコンパイルする場合は、INTLEVEL"4"を使用します。INTLEVEL"4"は、DIALECT"ISO2000"を指定した場合には省略時に設定されます。これらの指令を設定した場合には、ISO2000 標準で提案された OO 構文は NetExpress で使用された構文とは異なるので、NetExpress を使用して作成された OO プログラムは失敗します。

IOCONV

INTO/FROM句で指定したデータ項目への集団転記およびINTO/FROM句で指定したデータ項目からの集団転記をREAD...INTOおよびWRITE...FROMで行います。

これはANSIの動作です。

構文:

```
◆---+-------+--------+--- IOCONV --------------------------------◇
    +-- / --+-- NO --+
```

パラメータ:

なし

属性:

省略値:

IOCONV （Dialect）

段階:

構文チェック

$SET:

任意

説明:

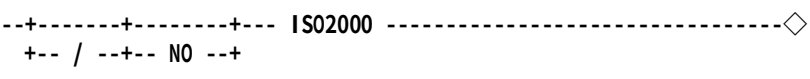
NOIOCONV を指定すると、READ ... INTO および WRITE ... FROM の操作に単純な集団転記が適用されます。これは下位互換性のために用意されています。

IOCONV を指定すると、必要に応じて、基本レコード記述の変換が行われます。これは通常の ANSI の動作であり、省略値となっています。

ISO2000

推奨ISO2000 COBOL標準で予約されている語を予約語として扱うように指定し、その標準と互換性のある機能の動作を変更します。

構文:



パラメータ:

なし

属性:

省略値:

NOISO2000 （Dialect）

段階:

構文チェック

$SET:

初期

説明:

推奨 ISO2000 標準と完全に互換性を持たせるためには、DIALECT"ISO2000"指令を設定してください。これらの指令を設定した場合には、ISO2000 標準で提案されている OO 構文は NetExpress で使用された構文とは異なるので、NetExpress を使用して作成された OO プログラムは失敗します。

IXNUMKEY

索引キーで真の数字ソーティングを可能にします。

構文:

```
◆---+-------+--------+--- IXNUMKEY --------------------------------◇
    +-- / --+-- NO --+
```

パラメータ:

なし

属性:

省略値:

NOIXNUMKEY

段階:

構文チェック

$SET:

任意

説明:

索引キーは、通常は数字データを含んでいる場合でも英数字として扱われます。そのため、READ NEXT 操作の結果は期待通りではありません。本指令は、真の数字の順番が必要なときに使用します。

一般的には、符号付き数字として定義されたキーだけに効果がありますが、符号なしデータの中にも効果があるもの（たとえば COMP-5 など）があります。

たとえば、pic s9(5)と定義したキーがあり、そのキーの値に次のように 1、-2、3、-4 の 4 つのレコードが与えられているものとします。

```
00001+
00002-
00003+
00004-
```

これらのキーが英数字として扱われる場合には、上記の順番で取り出されます。しかし、数字として扱われる場合（IXNUMKEYが指定されている場合）には、数字の順番で格納されます。そのため、READ NEXTを使用して取り出すことができます。

```
00004-
00002-
0000+1
0000+3
KEEP-INT
```

コンパイルが不成功に終わった（重大または回復不能なエラーが発生した）場合でも、コンパイラに中間コードを生成させるか否かを指定します

構文：

```
◆---+-------+--------+--- KEEP-INT ------------------------------◇
    +-- / --+-- NO --+
```

パラメータ： なし

属性:

省略値： KEEP-INT

段階： 構文チェック

$SET： 任意

説明:

コンパイラはハイフン抜きの指令名（KEEKPINT）でも受け付けます。

プログラムが区分化されている場合、コンパイルが不成功に終わったどれかの区分に NOKEEP-INT が指定されていると、どの区分の分も中間コードは生成されません。

KEYCHECK

ファイルオープン時にキーの数をチェックするかどうかを指定します。

構文：

```
◆---+-------+--------+--- KEYCHECK --------------------------------◇
     +-- / --+-- NO --+
```

パラメータ： なし

属性:

省略値： KEYCHECK

段階： 構文チェック

$SET： 任意

説明:

NOKEYCHECK を指定すると、ファイルハンドラはファイル情報領域に保持されているキーの数に対する SELECT 文で指定されたキーの数はチェックしません。

KEYCOMPRESS

索引ファイルに対して行われるキー圧縮の種類を指定します。

構文:

```
◆---+-------+--------- KEYCOMPRESS --- "整数" ------+-------------◇
     +-- / --+-- NO --- KEYCOMPRESS -----------------+
```

パラメータ:

整数 1～7 とする必要があります

属性:

省略値:

NOKEYCOMPRESS

段階:

構文チェック

$SET:

任意

説明:

「整数」の値として可能なものは次の通りです。

1 重複キーの繰返しを抑制します
2 前のキーと同じ先行文字列を抑制します
４ 後続スペース列を抑制します

これらの値を加算することでこれらの番号の任意の組み合わせを指定できます。

KEYCOMPRESS"0"と NOKEYCOMPRESS は同義です。

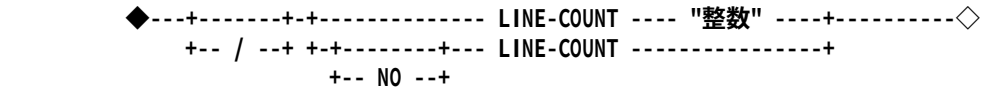
ファイルを作成するときにのみキー圧縮を指定する必要があります。その後、ファイルを開いたときにキー圧縮が検出されます。

個々のファイルについてキー圧縮を行うには、原始プログラムにおいて$SET 文を使用し、関連する KEY 句をファイルの SELECT 文に含んでいる原始プログラムの部分についてのみ本指令が有効となるようにします。

LINE-COUNT

原始プログラムリストの最後に表示される情報の詳細を制御します。

構文:



パラメータ:

整数 1 または 2

属性:

省略値:

NOLINE-COUNT

段階:

構文チェック

$SET:

初期

依存性:

LINE-COUNT"2"は直ちに CSI を設定します。

説明:

「整数」には以下の値を指定でき、次のように詳細情報を表示します。

1 空白行および注釈行以外のチェックされた原始プログラム行の数が原始プログラムリストの最後に表示されます。

2 チェックされた原始プログラム行の数だけでなく、たくさんの詳細な追加情報が原始プログラムリストの最後に表示されます。表示されるものには、呼出しの数、節の数、およびプログラム内の条件の数などがあります。

「整数」を指定せずに LINE-COUNT を指定すると、LINE-COUNT"2"を指定したのと同じ結果になります。

LINKCHECK

連絡節項目を参照するごとに連絡節項目が存在するかどうかをチェックするように指定します。

構文:

パラメータ： なし

属性:

省略値:

NOLINKCHECK

段階:

生成

$SET:

初期

説明:

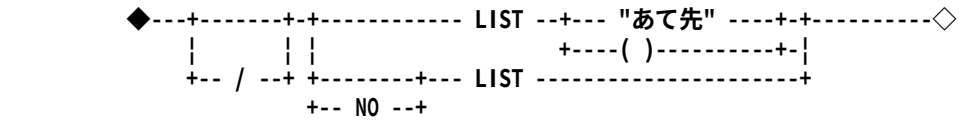
LINCKCHECK 指令を使用する場合は、PARAMCOUNTCHECK 指令も指定する必要があります。

存在しない連絡節項目を参照した場合には、ランタイムエラー 203（CALL パラメータが指定されていない）が生成されます。

LIST

原始プログラムリストファイルのあて先を指定します。

構文:



パラメータ:

あて先 完全なファイル指定または装置名

属性:

省略値:

NOLIST

段階:

構文チェックおよび生成

$SET:

任意（LIST および NOLIST 用）

なし（LIST"ファイル名"および LIST()用）

依存:

NOLIST が NOFORM、NOREF、NOSETTING、NOSOURCEASM、および NOXREF を最後に設定します。

LIST が FORM"60"を直ちに設定します。

ASM が LIST を直ちに設定します。

説明:

現存するファイルを指定すると、そのファイルは上書きされます。

NOLIST を指定すると、原始プログラムリストは生成されません。あて先なしで LIST を指定すると、原始プログラムリストは画面に送られます。あて先または（）を指定すると、$SET 文で本指令を使用することはできません。

装置名は任意の適当な装置とすることができます。例えば、画面用の CON:などです。

パラメータのない NOLIST および LIST は、プログラムの選択された部分をリストするためにプログラム内の $SET 文で使用できます。リストのあて先をこの方法で変更することはできません。

LIST（）を指定すると、原始プログラムリストは「原始ファイル名 .lst」ファイルに入ります。ここで、「原始ファイル名」はコンパイルされるプログラムのルート名です。

例:

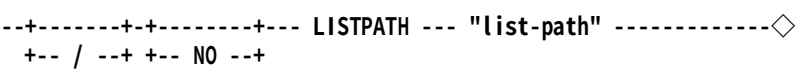
実行するコンパイルごとに原始プログラムのリストをファイルに書き出す場合は、cobol.dir ファイルに LIST（）指令を記述します。これは省略値の NOLIST に優先します。

既に COBOL .DIR に LIST 指令が入っており、各リストを画面に送るようにしている場合は、コマンド行で LIST（）を用いてこれに優先させることができます。

LISTPATH

リストファイルを書き出す先のパスを指定します。リストファイルの名前は「原始ファイル名.lst」です。

構文:



パラメータ:

リストパス リストファイルを書き出す先のパス、またはそれを指定している環境変数です。

属性:

省略値:

NOLISTPATH

段階:

構文チェック

$SET:

初 期

LISTWIDTH

LW

リストの幅を設定します。

構文:

```
  ◆---+-------+--+-- LISTWIDTH --+-- "width" ----------------------◇
      +-- / --+  +-- LW ---------+
```

パラメータ:

幅 幅（単位は文字)。72～132 とする必要があります。

属性:

省略値:

LISTWIDTH"80"

段階:

構文チェック

$SET:

任意

依存性:

REF 指令が設定されていて幅が 90 よりも小さい場合、直ちに LISTWIDTH"90"を設定します。

幅が 90 よりも小さい場合、最後に NOREF を設定します。

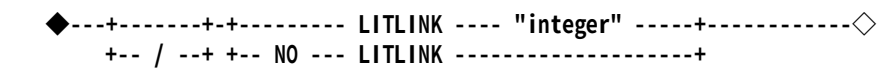
説明:

LISTWIDTH"132"を指定すると、リスト表示された各行ごとに付加情報が表示されます。この情報には、現 COPY ファイル名の最初の 8 文字（主ファイルの場合はスペース）とそのファイルの先頭からの相対行番号が含まれます。

LITLINK

「CALL 定数」文中の定数を共有記号としてコンパイラに定義させます。そうすると、定数は実行時ではなくリンク時に解明されます。（このように生成された呼出しは、LITLINK式として参照されます。）

構文:



パラメータ:

整数 1 または 2 でなければなりません。

属性:

省略値:

NOLITLINK

段階:

生成

$SET:

初期

依存性:

OMF"GNT"の場合、LITLINK が OMF"OBJ"を最後に設定します。

説明:

この指令は.obj ファイルだけに影響を与えます。

LITLINK"2"を指定すると、「CALL 定数」文中の名前の先頭に下線が 1 つ付いている場合（"_名前"）、入口名"_名前"への LITLINK 式の呼出しが生成されます。名前の先頭に下線が 2 つ付いている場合（"__名前"）、入口名"名前"への LITLINK 式の呼出しが生成されます（つまり、入口名は共有記号として定義されます）。

LITLINK"1"またはパラメータなしの LITLINK を指定すると、すべての「CALL 定数」文によって、LITLINK 式の呼出しが生成されます。このことは、定数の先頭に下線が付いているか否かに左右されません。

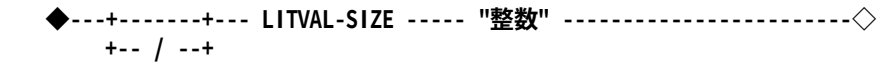
NOLITLINK を指定すると、「CALL 定数」文によって、LITLINK 式の呼出しは生成されません。このことは、定数の先頭に下線が付いているか否かに左右されません。

LITLINK"2"は 16 ビットの COBOL システム用に組まれたコードとの下位互換性を保つことだけを意図したものです。新しいプログラムには呼出し方式の 8 を使用してください。

LITVAL-SIZE

数字定数をBY VALUEで渡すときSIZE句が省略される場合、渡すバイト数を指定します。

構文:



パラメータ:

整数 最大 4 までの数。渡すバイト数。

属性:

省略値:

LITVAL-SIZE"4"

段階:

構文チェック

$SET:

任意

LNKALIGN

USING文で指定された任意の連絡レコードが、01または77レベル項目でありALIGN指令に従って割り付けられていると指定します。

構文:

```
  ◆---+--------+--- LNKALIGN ---------------------------------------◇
      +-- NO --+
```

パラメータ:

なし

属性:

省略値:

NOLNKALIGN

段階:

生成

$SET:

不可

説明:

LINKALIGN 指令を使用すると、連絡項目へのアクセスに必要な時間を減らすことができますが、使用にあたっては注意をしてください。プログラムをコンパイルするとき、正しい ALIGN 指令設定を使用する必要があります。項目が割り付けられていることを確認するための検査は行われません。

本指令の使用結果はマシン依存であることに留意してください。その使用に疑問を感じられたときは決して使用しないでください。割付けされていない項目を渡すと、予期しない結果となることがあります。

LOCALCOUNT

この指令は、システムが内部的に使用するために予約されています。設定されている指定のリストにこれが載せられることがあるので、完全性を示すためにこの指令をここに含めます。しかし、この指令はユーザーのアプリケーションのために使用するものではないので、その設定を変更してはなりません。

構文:

```
  ◆---+-------+--- LOCALCOUNT ----- "integer" ----------------------◇
      +-- / --+
```

パラメータ:

整数 0 から 65536 の範囲になければなりません。

属性:

省略値:

LOCALCOUNT"0"

段階:

構文チェック

$SET:

初期

LOCKTYPE

レコードロックの種類を指定します。

構文:

```
◆---+-------+--- LOCKTYPE ----- "整数" --------------------------◇
     +-- / --+
```

パラメータ:

整数 0 または 1。ロックの種類。

属性:

省略値:

LOCKTYPE"0"

段階:

構文チェック

$SET:

初期

説明:

「整数」の値として可能なものは次の通りです。

０ プログラムはロックされたレコードを読み込むことができますが、他の方法でアクセスすることはできません。これは Micro Focus の標準的手法です。

１ プログラムはロックされたレコードにはアクセスできません。これは COBOL 以外の言語の場合です。

CALLFH 指令が使用されている場合にのみ本指令は効果があります。

LOGICOPT

COBOL論理呼出しがRETURN-CODEの値に影響しないようにこれらの呼出しを最適化します。

構文:

```
◆---+--------+--- LOGICOPT ------------------------------------◇
     +-- NO --+
```

パラメータ:

なし

属性:

省略値:

NOLOGICOPT

段階:

生成

$SET:

不可

説明:

LOGICOPT を指定すると、CBL_AND のような COBOL 論理関数の呼出しが最適化され、RETURN_CODE が影響を受けないようになります。

MAKESYN

1個の予約語を別の予約語と同義にします。

構文:

```
  ◆---+-------+--- MAKESYN ----- "予約語１" ＝ "予約語２" ----------◇
      +-- / --+
```

パラメータ:

予約語 1 予約語

予約語 2 その意味を変更する予約語

属性:

省略値:

同義の予約語を作成しない。

段階:

構文チェック

$SET:

初期

説明:

等号（=）の両側にスペースを入れる必要があります。

本指令は、SETTING 指令で作成されたリストには現れません。

MAX-ERROR

発生したエラーの件数が指定されている値に達したら、コンパイラを異常終了させるようにします。

構文:

```
  ◆---+-------+-+------- MAX-ERROR --- "err-cnt" -+---------+---+---◇
      +-- / --+ ¦                                 +- svrty -+   ¦
               +- NO -- MAX-ERROR ---------------------------+
```

パラメータ:

エラー件数 エラーの最大件数を指定します。

重大度 数を数える重大度の最低レベルです。重大度がこの値以上のメッセージだけが数えられます。下記のどれかを指定します。

Ｆ フラグ
Ｉ 通知
Ｗ 警告
Ｅ エラー
Ｓ 重大

指定を省略すると、F とみなされます。

属性:

省略値:

NOMAX-ERROR

段階:

構文チェック

$SET:

任意

説明:

コンパイラはハイフン抜きの指令名（MAXERROR）でも受け付けます。

指定した重大度以上のメッセージが最大件数に達すると、コンパイラは停止します。実際に発生したメッセージだけが数えられます。また、メッセージの重大度が変更された場合には、変更後の区分に基づいて数えられます。

例:

下記のように指定したとします。

MAX-ERROR"100 E"

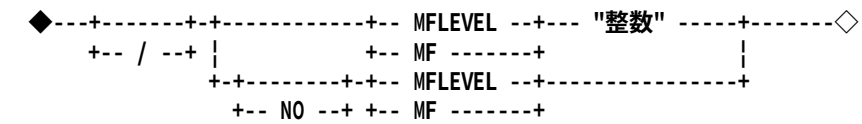
すると、重大度が E と S のメッセージが 100 件発生すると、コンパイルは終了します。

MFLEVEL

MF

Micro Focus固有の予約語を選択的に使用可能とし、特定のバージョンと互換のある、ある機能の動作を変更することにより、将来のMicro Focus製品との互換を容易にします。

構文:



パラメータ:

整数 互換のある Micro Focus COBOL のレベル

属性:

省略値:

MF"11" （Dialect）

段階:

構文チェック

$SET:

初期

依存性:

MF"7"は DBCS"2"を直ちに設定します。

「整数」の値が 7 よりも大きい場合、DBCS"3"および DBSPACE を直ちに設定します。

NOEARLY-RELEASE および MF が設定されている場合の MF の省略値は MF"11"です。

EARLY-RELEASE および MF が設定されている場合の MF の省略値は MF"12"です。

説明:

「整数」の値として可能なものは次の通りです。

1 Professional COBOL v1.0, v1.1, v1.2

Level II COBOL v2.5, v2.6

Level II COBOL/ET v1.1

2 VS COBOL Workbench v1.2

VS COBOL v.1.2

3 VS COBOL Workbench v1.3

VS COBOL Workbench v2.0

Professional COBOL v2.0

VS COBOL v1.5

4 COBOL/2 v1.1

Professional COBOL/2

COBOL/2 Workbench v2.2

Microsoft COBOL v3.0

5 COBOL/2 v1.2

COBOL/2 Workbench v2.3

6 COBOL/2 v2.4

COBOL/2 Workbench v2.4

Microsoft COBOL v4.0

７ COBOL/2 v2.5

COBOL/2 Workbench v2.5

Microsoft COBOL v4.5

８ COBOL v3.0

COBOL Workbench v3.0

Microsoft COBOL v5.0

９ COBOL V3.1

COBOL Workbench V3.1

10 COBOL V3.1 で、初期のバージョンの構文を有効としているもの

COBOL Workbench V3.1 で、初期のバージョンの構文を有効としているもの

COBOL V3.2, v3.3

COBOL Workbench V3.2, v3.3

Object COBOL v3.2, v3.3

11 Visual Object COBOL v1.0

COBOL V3.4, v4.0 で、初期のバージョンの構文を有効としているもの

COBOL Workbench V3.4, v4.0 で、初期のバージョンの構文を有効としているもの

12 Visual Object COBOL v1.0 で、初期のバージョンの構文を有効としているもの

本製品の以前のバージョンにおける BLANK LINE 指定の動作は、OLDBLANKLINE 指令を用いて実現できます。本製品の以前のバージョンにおける NEXT SENTENCE 指定の動作は、OLDNEXTSENTENCE 指令を用いて実現できます。

MF"11"には、V3.2 および v3.3 の MF-OO 指令で使用されているすべての予約語が含まれています。

MF"12"には、MF"11"および初期のバージョンの構文で使用されているすべての予約語が含まれています。パラメータを付けずに MF を指定すると、EARLY-RELEASE 指令が設定されている場合には、MF"12"に設定されます。そうでない場合には、MF"11"に設定されます。EARLY-RELEASE 指令が設定されていないかぎり、MF"12"を設定することはできません。

MFCOMMENT

1カラム目がアスタリスク(*)の行を注釈行と同様に扱いますが、原始プログラムリストには出力しません。

構文:

パラメータ:

なし

属性:

省略値:

MFCOMMENT （Dialect）

段階:

構文チェック

$SET:

任意/P >

依存性:

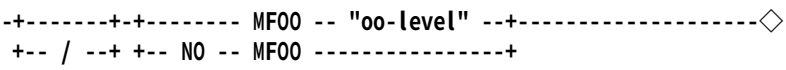
SOURCEFORMAT"FREE"が最後に MFCOMMENT を設定します。

説明:

MFCOMMENT を指定すると、コンパイラは 1 カラム目がアスタリスクの行を無視し、そのような行は原始プログラムリストには現れません（ただし、デバッガでは表示されます)。NOMFCOMMENT を指定すると、アスタリスクは連続番号の一部を構成し、特に意味を持ちません。

MF-OO

構文:



パラメータ:

なし

属性:

省略値:

NOMF-OO

段階:

構文チェック

$SET:

初期

依存性:

MF-OO は、MF を直ちに設定します。

説明:

MF-OO 指令は、ハイフン抜き（MFOO）でも指定できます。

MOVE-LEN-CHECK

構文:

パラメータ:

なし

属性:

省略値:

NOMOVE-LEN-CHECK

段階:

構文チェック

$SET:

任意

説明:

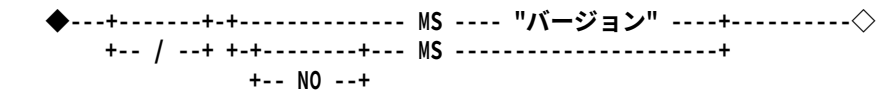
本指令を指定すると、コンパイラは原始プログラムと英数字 MOVE 操作の対象の長さを比較し、違っていれば警告メッセージを出します。(166)

この指令により生成された警告メッセージはどれも WARNINGS"2"または WARNINGS"3"を選択した場合のみ見えます。

MS

Micro Focus固有の予約語を選択的に使用可能とし、特定のバージョンと互換のある、ある機能の動作を変更することにより、Microsoft製品との互換を容易にします。

構文:



パラメータ:

バージョン 1 または 2 とする必要があります

属性:

省略値:

NOMS （Dialect）

段階:

構文チェック

$SET:

初期

依存性:

MS が DEFAULTBYTE"0"および ACCEPTREFRESH を設定します。

説明:

「バージョン」の値として可能なものは次の通りです。

1 Microsoft COBOL バージョン 1

2 Microsoft COBOL バージョン 2

パラメータなしの MS を指定することは、MS"2"を指定することと同じです。MS"1"は IBM-MS および PC1 と同じです。

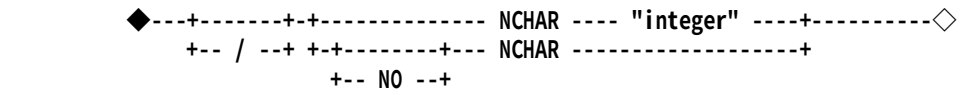
本指令は、Microsoft COBOL バージョン 3.0 よりも前のバージョンと互換を取るためのものです。Microsoft COBOL バージョン 3.0 以降のバージョンとの互換を取るには NOMS MF"整数"を使用します。

MF 指令で使用する整数の値は、MF 指令のところで記述されています。

NATIVE

比較用に省略時の文字の大小順序を指定します。

構文:



文字の大小順序 "ASCII"または"EBCDIC"

属性:

省略値:

NATIVE "ASCII"

段階:

構文チェック

$SET:

初期

依存性:

CHARSET"ASCII"が直ちに NATIVE"ASCII"を設定します。

CHARSET"EBCDIC"が最後に NATIVE"EBCDIC"を設定します。

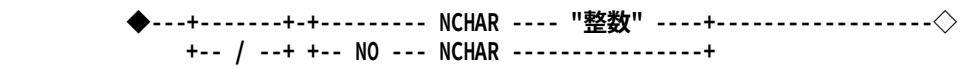
説明:

索引ファイルにおけるキーの順序は ASCII の照合順序に従っています。

NCHAR

Micro Focus2バイト言語拡張（PIC、日本語データ名および日本語手続き名）を提供します。

構文:



パラメータ:

整数 1 または 2 とする必要があります。必要とするサポートのレベルを示します。

属性:

省略値:

NONCHAR

段階:

構文チェック

$SET:

初期

説明:

「整数」の値として可能なものは次の通りです。

1 以前のバージョンの日本語 Micro Focus 製品との互換
2 拡張 PIC N サポート。本設定は DBSPACE 指令を使用可能にします。

パラメータなしの NCHAR を指定することは NCHAR "2"を指定することと同じです。

NCHAR は JAPANESE 指令と同義で、JAPANESE 指令に取って代わります。

NCHAR および DBCS 指令は、互いに排他的です。

NCHAR"2"によって PIC X を PIC N に変換する場合、スペース（X"20"）は 1 つの 2 バイトスペースに拡張されます。したがって、X"2020"は 2 つの 2 バイトスペースに拡張されます。NCHAR"1"を指定すると、X"2020"は 1 つの 2 バイト文字に拡張されます。

NESTCALL

プログラムの入れ子が利用者のプログラムに現れるのを可能にします。

構文:

```
 ◆---+-------+-+--------+--- NESTCALL ----------------------------◇
      +-- / --+ +-- NO --+
```

パラメータ:

なし

属性:

省略値:

NONESTCALL （Dialect）

段階:

構文チェック

$SET:

初期

NLS

国別言語サポートを可能にします。

構文:

```
 ◆---+-------+-+--------+--- NLS --------------------------------◇
      +-- / --+ +-- NO --+
```

パラメータ:

なし

属性:

省略値:

NONLS

段階:

構文チェック

$SET:

不可

説明:

NLS は、実行時にプログラムをユーザの国に適した文字セット、通貨記号、および編集用記号に自動的に適合させます。

OBJ

目的コードファイルおよび生成コードファイルの名前と形式を指定します。

構文:

```
◆---+-------+-+--------- OBJ ---- "ファイル名" ----+-------------◇
     +-- / --+ +-- NO --- OBJ ---------------------+
```

パラメータ:

ファイル名 完全なファイル指定

属性:

省略値:

OBJ "原始ファイル名 .OBJ"（OMF "OBJ"指定）

OBJ "原始ファイル名 .GNT"（OMF "GNT"指定）

段階:

生成

$SET:

任意

説明:

本指令は、OMF 指令よりも優先させて使用してください。

本指令は、指定された名前で指定された形式のファイルを作成します。たとえば、OBJ"myprog.gnt"を指定した場合には、生成コードを含むファイル myprog.obj が生成されます。

NOOBJ を指定すると、目的コードファイルは生成されません。本指令が指定されないと、目的コードファイルの名前は原始ファイル名に拡張子（ .OBJ）を付けたものと同じになります。

ODOSLIDE

同じレコード内の可変長表に続くデータ項目を、その表の長さの変化に応じて転記します。

構文:

パラメータ:

なし

属性:

省略値:

NOODOSLIDE （Dialect）

段階:

構文チェック

$SET:

初期

説明:

本指令は同じレコード内の可変長表の後にくるデータ項目に影響します。つまり、OCCURS DEPENDING 句のある項目の後で、しかし、その項目に従属するものではありません。

ODOSLIDE を指定すると、表の現在サイズにかかわらず、常にこれらの項目は表の直後に続きます。これは表サイズの変更に合わせてこれら項目のアドレスが変化することを意味しています。NOODOSLIDE を指定すると、これらの項目は固定アドレスを持ち、最大長の表に割り付けられた領域の終わりから始まります。

OLDBLANKLINE

画面節のBLANK LINE句の動作を変更します。

構文:

パラメータ:

なし

属性:

省略値:

NOOLDBLANKLINE

段階:

構文チェック

$SET:

任意

説明:

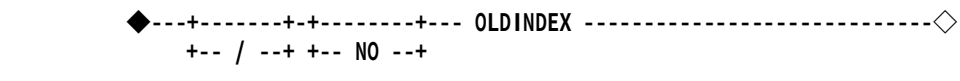
OLDBLANKLINE が指定されると、BLANK LINE 句は ERASE EOL と完全に同じ動作をします。つまり、カーソルの右側の文字は全て削除されます。

NOOLDBLANKLINE が指定されると、BLANK LINE 句は行全体を削除します。

OLDINDEX

指標が添字としてコンパイルされるようにします。

構文:



パラメータ:

なし

属性:

省略値:

NOOLDINDEX （Dialect）

段階:

構文チェック

$SET:

初期

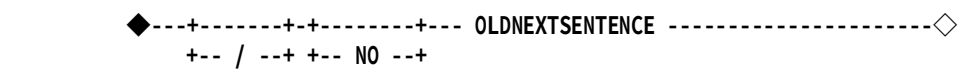
説明:

本指令は初期の製品との互換を図るためのものです。

OLDNEXTSENTENCE

NEXT SENTENCE文の動作を変更します。

構文:



パラメータ:

なし

属性:

省略値:

NOOLDNEXTSENTENCE （Dialect）

段階:

構文チェック

$SET:

任意

説明:

OLDNEXTSENTENCE を指定すると、NEXT SENTENCE 文は CONTINUE 文のように動作します。

OLDREADINTO

READ...INTO文の動作を変更します。

構文:

```
◆---+-------+-+--------+--- OLDREADINTO -------------------------◇
    +-- / --+ +-- NO --+
```

パラメータ:

なし

属性:

省略値:

NOOLDREADINTO （Dialect）

段階:

構文チェック

$SET:

任意

説明:

OLDREADINTO が指定されると、READ が成功しない場合でも、ファイルのレコード領域から INTO 指定で指定されたデータ項目への IMPLICIT 転記が実行されます。

NOOLDREADINTO が指定されると、READ が成功した場合、MOVE のみが発生します。

OMF

どの形式の目的コードまたは生成コードを生成するか指定します。

構文:

```
◆---+-------+--- OMF ----- "コード" ----------------------------◇
    +-- / --+
```

パラメータ:

コード OBJ または GNT

属性:

省略値:

OMF "OBJ"

段階:

生成

$SET:

不可

「コード」の値として可能なものは次の通りです。

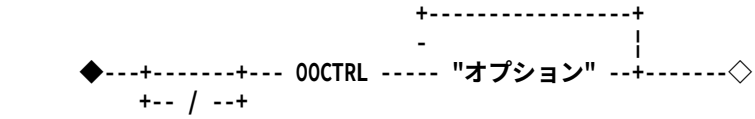
OBJ （ .OBJ）形式を生成します。

GNT （ .GNT）コードを生成します。

OOCTRL

目的COBOLクラスをコンパイルするときに、言語オプションを変更できるようにします。

構文:



パラメータ:

+ オプションを有効にする

- オプションを無効にする

オプション OO プログラムの構文をどのように扱うかを決めます。

属性:

省略値:

OOCTRL"-W+N"

段階:

構文チェック

$SET:

任意

説明:

「オプション」の値として可能なものは次の通りです。

G インスタンスでクラスデータをグローバルにするかどうかを指定します。このオプションを使用することはお勧めしません。このオプションは、初期バージョンとの互換を取るために提供されています。

N 構文 FACTORY、END FACTORY、CLASS-OBJECT、および END CLASS-OBJECT を使用可能にするかどうかを指定します。

P パラメータ型情報をランタイムシステムで使用可能にするかどうかを指定します。このオプションは、OLE および SOM に送られたメッセージで必要となります。

Q 次の内容を指定します。

- メソッドインタフェース定義の動詞記号でデータ名の後ろの任意の場所を無効にするかどうか。
- メソッドインタフェース定義の動詞記号で任意の動詞の使用を無効にするかどうか。

W 次の領域を使用するかどうかを指定します。

- インスタンスおよびクラスオブジェクトにおける目的記憶域用の作業用記憶域
- メソッドにおける局部記憶域用の作業用記憶域

また、-N を指定すると、クラスで目的記憶域用に作業用記憶域が使用されます。

作業用記憶域が局部記憶域または目的記憶域として使用される場合、局部記憶域または目的記憶域節をそれぞれ指定しないでください。

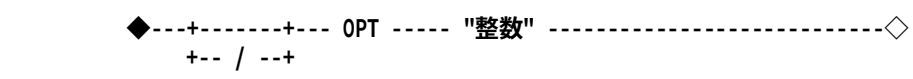
これにより、開発中の ISO COBOL 標準との互換性が保たれます。

以前のコードとの互換を取るためには-N を指定してください。しかし、旧形式で新しいクラスを作成することはお勧めしません。＋W を指定して ANSI97 標準で提案されているように作業用記憶域を使用してください。

OPT

（.obj）ファイルで作成されたコードの最適化レベルを指定します。

構文:



パラメータ:

整数 1 または 2

属性:

省略値:

OPT"2"

段階:

生成

$SET:

任意

説明:

「整数」の値として可能なものは次の通りです。

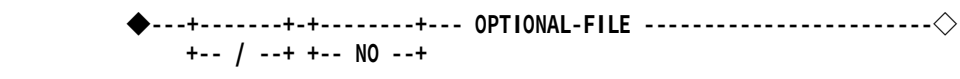
1 標準目的コード

2 最適化されたオブジェクトコード。最適化によりプログラムのロジックが変更されるようなことはありません。しかし、最適化により、混乱（関連するコードを持たない文がある、変数を直ちに更新できない、など）がないかどうかをデバッグすることはあります。

OPTIONAL-FILE

I-OまたはEXTEND用に開いている全てのファイルを不定ファイルとしてコンパイラに取り扱わせます。

構文:



パラメータ:

なし

属性:

省略値:

OPTIONAL-FILE （Dialect）

段階:

構文チェック

$SET:

初期

説明:

ANSI85 標準 COBOL の下では、ファイルの SELECT 文に OPTIONAL 指定がある場合にのみ、ファイルは不定として扱われます。ANSI85 標準と互換を取るには、NOOPTIONAL-FILE 指令を指定する必要があります。

OSEXT

コンパイラに対して原始ファイルの名前に省略時に何という拡張子を付けるかを指示します。これは、COPYファイルが使用する拡張子だけに影響します。

構文:

```
◆---+-------+--- OSEXT ----- "拡張子" ---------------------------◇
    +-- / --+
```

パラメータ:

拡張子 拡張子

属性:

省略値:

OSEXT "cbl"

段階:

構文チェック

$SET:

任意

説明:

コマンドまたはプロンプトで使用される原始ファイル名に拡張子または後続ピリオドが付いていない場合に、本指令で指定した拡張子が付加されます。COPY 文に指定された原始ファイル名に拡張子または後続ピリオドがない場合、本指令は可能性のある拡張子のリストを順番に検索します。この検索は、ファイルが見つかるまで、またはリストがなくなるまで（この場合エラーが表示されます）行われます。リストは、COPYEXT および OSEXT 指令の値の組み合わせによるものです。そのリストの最初の拡張子は、OSEXT 指令の値または COPYEXT 指令の最初の値のどちらかです。これは、この 2 つの指令のどちらが最後に指定されたかによります。

リストの残りの値は、COPYEXT 指令（終りまでの値が 2 つ）の値の残りです。例えば、

COPYEXT "CPY, CBL, TXT "OSEXT "SRC"

これは以下のものと同じ効果があります。

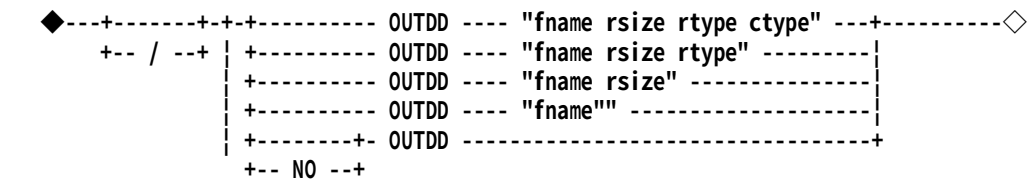
COPYEXT "SRC, CBL, TXT"

拡張子を指定しない場合は（OSEXT""）、ファイル名に拡張子が付加されません。

OUTDD

DISPLAY文とEXHIBIT文、およびTRACEからの出力を指定した出力ファイルに書き込みます。

構文:



パラメータ:

fname 指定した DISPLAY 文、EXHIBIT 文、および TRACE の出力を書き込むファイルの名前。このパラメータを指定しないと、ファイル名は SYSOUT になります。

rsize ファイル中のデータレコードのサイズ。このパラメータを指定しないと、レコードサイズは 132 になります。

rtype 行順ファイルの場合は L、レコード順ファイルの場合は R。このパラメータを指定しないと、L が設定されます。

ctype ASCII コードの場合は A、EBCDIC コードの場合は E。このパラメータを指定しないと、A が設定されます。E は、CHARSET(EBCDIC)指令が使用されている場合のみ効果があります。

属性:

省略値:

NOOUTDD

段階:

構文チェック

$SET:

初期

依存性:

OUTDD は直ちに NOSYSIN を設定します。

SYSIN は直ちに NOOUTDD を設定します。

説明:

OUTDD を指定すると、UPON オプションが指定されていないか UPON SYSOUT が指定されている形式 1 の DISPLAY 文すべて、EXHIBIT 文すべて、および TRACE からの出力は WRITE 文に書き換えられます。これらは、指定した外部ファイルに書き込まれます。

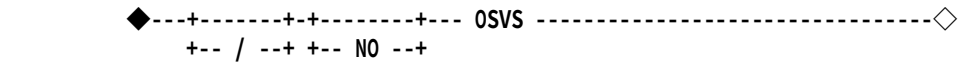
ファイル名は、外部ファイルと同じように、つまり、環境変数または外部ファイルマッパを使用して論理ファイル名にマップされます。

本指令の省略時の設定は、SYSIN 指令を使用した場合にこのタイプの DISPLAY 文、EXHIBIT 文、および TRACE の出力で使用された設定と同じになります。

OSVS

IBM OS/VS COBOLで予約された語を予約語として扱うように指定します。

構文:



パラメータ:

なし

属性:

省略値:

NOOSVS

段階:

構文チェック

$SET:

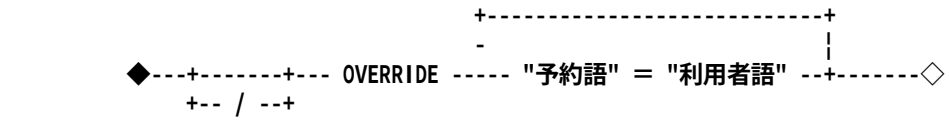
初期

OSVS を指定すると NODOSVS が設定されます。条件文の条件内で部分参照を使用する場合は、Micro Focus COBOL システムの下で原始プログラムをコンパイルするとき、NOOSVS を設定する必要があります。

OVERRIDE

ある予約語を新しい予約語に置き換えます。

構文:



パラメータ:

予約語 現存する予約語

利用者語 任意の COBOL 語で、現存する予約語と同じでないもの。

属性:

省略値:

予約語の変更はありません。

段階:

構文チェック

$SET:

初期

説明:

本指令は現存する予約語を指定された利用者語と同等にします。これにより、プログラムにおいて、「利用者語」は予約語として扱われ、「予約語」は利用者語として扱われます。

等号の両側にスペースを入れる必要があります。パラメータを繰り返す場合は、それぞれをスペースで分離する必要があります。

本指令は、SETTING 指定で作成されたリストには現れません。

PARAMCOUNTCHECK

関連する入口点のUSING句で指定されたパラメータよりも少ないパラメータでプログラムを呼び出します。

構文:

```
◆---+-------+-+--------+--- PARAMCOUNTCHECK ---------------------------------◇
    +-- / --+ +-- NO --+
```

パラメータ:

なし

属性:

省略値:

PARAMCOUNTCHECK

段階:

生成

$SET:

初期

説明:

本指令は、次のいずれかにあてはまる場合に使用する必要があります。

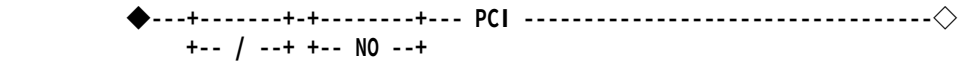
- プログラムがパラメータ変数付きで呼び出されている場合。
- プログラムに連絡節が含まれていて、プログラムがコマンド行からメインプログラムとして実行され、また他のプログラムからサブプログラムとしても呼び出される場合。
- プログラムがSTICKY-LINKAGE"2"を指定してコンパイルされている場合。

また、プログラムは、処理効率を上げるためには NOPARAMCOUNTCHECK を指定してコンパイルします。プログラムが COBOL 以外の言語のプログラムから呼び出された場合は、本指令の設定に関わらず NOPARAMCOUNTCHECK でコンパイルされます。

PC1

IBM COBOL 1.00で予約されている語を予約語として見なすよう指定し、ある機能の動作を変更してその製品との互換を図ります。

構文:



パラメータ:

なし

属性:

省略値:

NOPC1

段階:

構文チェック

$SET:

初期

依存性:

PC1 は直ちに DEFAULTBYTE"0"および ACCEPTREFRESH を設定します。

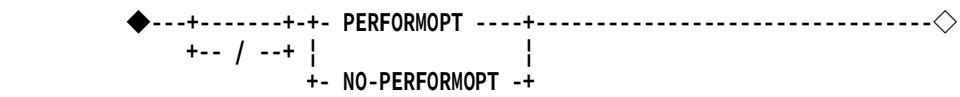
説明:

本指令は IBM-MS および MS "1"指令と同義です。

PERFORMOPT

最適化のために、空の段落に対するPERFORMを、生成される固有コードに含めないようにするように指定します。

構文:



パラメータ:

なし

属性:

省略値:

PERFORMOPT

段階:

生成

$SET:

初期

説明:

効率を最大にするために、コンパイラは通常は空の段落を参照する PERFORM 文用のコードを生成しません。たとえばタイミングを取るために、PERFORM 文を残しておく必要がある場合には、NOPERFORMOPT を指定します。

PREPLIST

コンパイル中、生成されるリストファイルに、プリプロセッサがコンパイラに渡した全データファイルと同様に、元の原始プログラムとプリプロセッサが作成した、変更された原始プログラムの両方が示されるようにします。

構文:

パラメータ:

なし

属性:

省略値:

NOPREPLIST

段階:

構文チェック

$SET:

不可

説明:

本指令は、プリプロセッサの作成者にデバッグを援助するために提供されています。

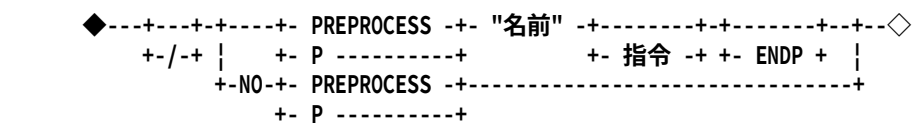
本指令は、リストファイルが生成されている場合には、それに含まれているものにだけ影響します。本指令はリストファイルを生成するかどうか、またはファイルの名前は決めません。

PREPROCESS

P

コンパイラに原始ファイルの代わりにプリプロセッサから原始プログラムを取らせます。

構文:



パラメータ:

名前 使用するプリプロセッサ

指令 プリプロセッサへ直接渡される指令

注:

- プリプロセッサに渡す指令を終了するには、ENDP COBOL指令を使用する必要があります。ENDPの後に置かれた指令がCOBOLコンパイラに渡されます。そのため、ENDP指令がなければ、コンパイラ指令はCOBOLコンパイラではなくプリプロセッサに渡され続けます。
- COBSQLプリプロセッサに指令を送る場合は、END-Cを使用してCOBSQLに渡す指令とプリプロセッサに渡す指令を区別する必要があります。たとえば、以下のコマンド行ではEND-Cの前に置かれた指令がCOBSQLに渡されます。END-CとENDPの間に置かれた指令は、COBSQL経由でプリプロセッサに渡されます。詳細については、「NetExpress Database Access book」の「COBSQL」を参照してください。

```
Preprocess(Cobsql) csqltype=oracle end-c comp5=yes endp;
```

属性:

省略値:

NOPREPROCESS

段階:

構文チェック

$SET:

原始プログラムの第 1 行のみ（PREPROCESS 指定の場合）

不可（NOPREPROCESS 指定の場合）

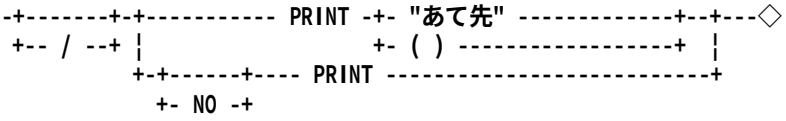
説明:

本指令はコンパイラに統合型プリプロセッサが使用されることを知らせます。本指令の使用の詳細については、「第 2 章 統合型プリプロセッサ」を参照してください。

PRINT

原始リストファイルのあて先を指定します。

構文:



パラメータ:

あて先 完全なファイル指定、または、装置名

属性:

省略値:

NOPRINT

段階:

構文チェック

$SET:

任意

説明:

PRINT は LIST と同義です。

既存のファイルを指定した場合は、上書きされます。

NOPRINT を指定すると、原始リストは作成されません。PRINT を「あて先」なしで指定すると、原始リストが画面に表示されます。「あて先」または()を指定した本指令を$SET 文に使用することはできません。

「あて先」に適切な装置の名前を指定することもできます。画面の場合は、CON を使用します。

プログラム中の選択された部分をリスト表示するには、プログラムの$SET 文に NOPRINT または PRINT をパラメータなしで使用できます。この方法では、リストのあて先は変更できません。

PRINT()は、原始リストを原始ファイル名.lst ファイルに入れます。原始ファイル名は、コンパイルするプログラムのファイル名の基本名です。

例:

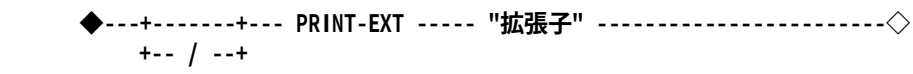
原始プログラムをコンパイルするごとにファイルにリスト表示するには、PRINT()指令を cobol.dir ファイルに指定します。これにより省略時の NOPRINT が書き換えられます。

また、すでに cobol.dir ファイルに PRINT 指令が指定されている場合は、リストが作成されるたびに画面に表示されます。cobol.dir ファイルの PRINT 指令は、コマンド行で PRINT()を使用すると書き換えられます。

PRINT-EXT

ASSIGN TO PRINTER句に関連するファイル名に付加される拡張子を指定します。

構文:



パラメータ:

拡張子 付加される拡張子

属性:

省略値:

拡張子は付加されません。

段階:

構文チェック

$SET:

任意

説明:

ASSIGN-PRINTER（）がファイル名なしで指定されない限り、本指令は無視されます。

PROTECT-LINKAGE

呼出しプログラムと呼び出されたプログラムのパラメータ長がそれぞれ異なってもいいように標準COBOLの意味を拡張します。

通常は、制約条件を無視すると未定義の結果が出されます。そしてときには保護違反やメモリアクセスエラーなどの重大なエラーを意味することもあります。

構文:

```
◆---+-------+-+--------+--- PROTECT-LINKAGE ----------------------◇
     +-- / --+ +-- NO --+
```

パラメータ:

なし

属性:

省略値:

NOPROTECT-LINKAGE （Dialect）

段階:

構文チェック

$SET:

初期

説明:

標準 ANSI COBOL の総則では、CALL 文に渡す各パラメータについて次のように明言されています。「呼び出されたプログラムのデータ項目の記述は、呼出しプログラムの関連するデータ項目の記述と同じ数でなければならない。」

この制約は、本 COBOL システムを使用して PROTECT-LINKAGE 指令を指定せずにプログラムをコンパイルした場合に有効になります。

この制約は、PROTECT-LINKAGE 指令が設定されている場合は無効です。つまり、呼び出されたプログラムは、項目を文に渡すときだけは誤った組み合わせのパラメータを使用できますが項目を受け取るときにはそのようなパラメータは使用できせん。

呼び出されたプログラムと呼出しプログラムのパラメータが一致していない場合、そのパラメータにおける文字位置も一致していません。そのような不一致文字の内容は、呼び出されたプログラムの送信項目として使用されるパラメータには定義されていません。

本指令を使用すると、アプリケーションの性能に影響が出ます。

例:

プログラムの呼出し

```
   ...
   03 x1  pic x.
   03 x2  pic x(100).
 procedure division.
    ...
   call subprog using x1
    ...
```

サブプログラム

```
 working-storage section.
 01  y1       pic x(1000).
 linkage section.
 01  z1       pic x(1000).
 procedure division using z1.
 move z1 to y1
```

このプログラムは正常に動作します。つまり、x1の内容を転送します。また、x1に続くデータを呼出しプログラムに転送します。そのデータサイズは、1000バイトまで、または割り当てられたメモリの限界までです。1000バイトよりも小さいデータは転送され、y1の残りの部分には空白が詰められます。

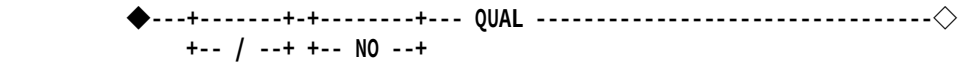
```
 move y1 to z1.
```

この操作は保護されていないため、呼出しプログラムのx1を超えたデータを壊すか、割り当てられたメモリを超えて書き込もうとして失敗します。これは、保護違反の原因になります

QUAL

プログラムに修飾されたデータ名と手続き名を入れるのが可能になります。

構文:



パラメータ:

なし

属性:

省略値:

QUAL （Dialect）

段階:

構文チェック

$SET:

任意

説明:

原始コードに修飾されたデータ名または手続き名が入っていない場合、NOQUAL を指定できます。これによりコンパイル速度が向上します。

QUALPROC

プログラムに修飾された手続き名を入れるのが可能になります。

構文:

パラメータ:

なし

属性:

省略値:

QUALPROC （Dialect）

段階:

構文チェック

$SET:

任意

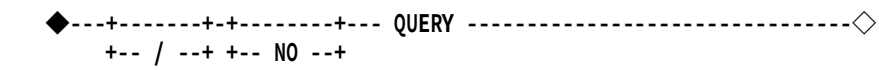
説明:

修飾された手続き名が原始プログラム内にない場合は、NOQUALPROC を指定できます。これによりコンパイル速度が向上します。修飾されたデータ名はあるが、修飾された手続き名がない場合、QUAL と NOQUALPROC を指定する必要があります。

QUERY

COPYファイルを見つけることができないとき、取るべき処理をコンパイラに尋ねさせます。

構文:



パラメータ:

なし

属性:

省略値:

QUERY

段階:

構文チェック

$SET:

任意

説明:

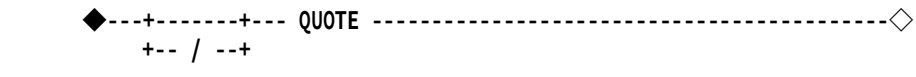
QUERY を指定すると、コンパイラが COPY ファイルを見つけることができない場合、コンパイラは利用者にコンパイルを終了するかどうか尋ね、探索をもう一度試み、エラーメッセージを作成してコンパイルを継続するか、あるいは、利用者により指定された代替パスに沿って探索をもう一度試みます。

NOQUERY を指定すると、コンパイラは単にエラーメッセージを作成してコンパイルを続けます。

QUOTE

コンパイラに表意定数QUOTEを二重引用符（"）として解釈させます。

構文:



パラメータ:

なし

属性:

省略値:

QUOTE （Dialect）

段階:

構文チェック

$SET:

任意

RAWLIST

ページ見出しや、日付、時間、コンパイラのリリースなどの変動する情報がリストに出力されないようにします。

構文:

パラメータ:

なし

属性:

省略値:

NORAWLIST

段階:

構文チェック

$SET:

任意

説明:

この指令は、リストが生成されるかどうか、またはリストの名前などには影響しません。

RDFPATH

リポジトリーファイルの場所を指定します。

構文:

```
◆---+-------+-+--------- RDFPATH --+--- "パス" --------------◇
     +-- / --+ +-- NO -- RDFPATH ---+
```

パラメータ:

パス リポジトリーファイルを保管するパス名

属性:

省略値:

NORDFPATH

段階:

構文チェック

$SET:

初期

説明:

この指令は、リストが生成されるかどうか、またはリストの名前などには影響しません。

RECMODE

ファイルの省略時の形式を指定します。

構文:

```
◆---+-------+--- RECMODE ----- "形式" ---------------------------◇
     +-- / --+
```

パラメータ:

形式 F、V または OSVS

属性:

省略値:

RECMODE"F" （Dialect）

段階:

構文チェック

$SET:

初期

説明:

「形式」の値として可能なものは次の通りです。

F 固定長レコード形式

V 可変長レコード形式

個々のファイルの場合、RECORD IS VARYING 指定（可変形式を指定）または RECORDING MODE 句が FD に入っていると、本指令はオーバライドされます。

REENTRANT

マルチスレッドプログラムの再入を制御します。プログラムをマルチスレッドアプリケーションに再入できるようにするには、本指令を指定してコンパイルする必要があります。REENTRANT指令が指定されると、大量のプログラム領域が動的に割り当てられ、そのため安全にプログラムを複数コピーして実行させることができます。

構文:

```
◆---+-------+-+--------- REENTRANT --+--- "n" ---------------◇
    +-- / --+ +-- NO -- REENTRANT ---+
```

パラメータ:

n 1 または 2

属性:

省略値:

NOREENTRANCT

段階:

構文チェック

$SET:

初期

説明:

「n」の値として可能なものは次の通りです。

1 コンパイラが生成した一時作業領域が各スレッドに割り当てられます。環境部およびデータ部で割り当てられたユーザデータ領域および FD ファイル領域は、すべてのスレッドが共有できます。CBL_同期呼出しが使用するプログラムのデータを順番に並べるのはプログラマの役割です。

2 環境部およびデータ部で割り当てられたユーザデータ領域および FD ファイル領域が各スレッドに割り当てられます。外部データおよび外部ファイルは、必ずすべてのスレッドが共有できます。

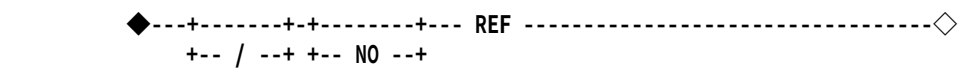
再入したいプログラムがある場合はいつでも REENTRANT"1"を使用します。既存のプログラムを初めて再入可能プログラムに変換するときだけ REENTRANT"2"を使用します。

REF

コンパイラに対し、各データ項目または手続き部文の中間コードアドレスを、原始プログラムリストへ含むようにさせます。

このアドレスは、各手続き部文のアドレスをリストする目的コードに含まれます。

構文:



パラメータ:

なし

属性:

省略値:

NOREF

段階:

構文チェック

$SET:

任意

依存性:

REF は LISTWIDTH がすでに 90 を超えた値である場合以外は LISTWIDTH"90"を直ちに設定します。

NOASMLIST、NOLIST または 90 よりも小さい値の LISTWIDTH により最後に NOREF が設定されます。

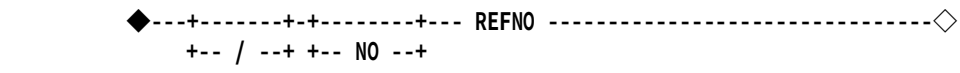
説明:

原始コードリストと目的コードリストを一緒に使用して、原始コードの各行について生成されたコードを識別できます。本指令はまたランタイムエラーメッセージで報告された場所を決定する上で有用です。

REFNO

コンパイルの開始時と各リストの最後でコンパイラにその内部参照番号を表示させます。

構文:



パラメータ:

なし

属性:

省略値:

NOREFNO

段階:

構文チェック

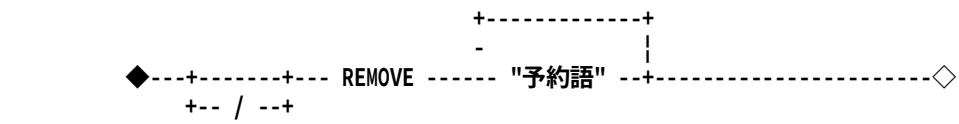
$SET:

不可

REMOVE

予約語リストから語を削除し、利用者語として使用できるようにします。

構文:



パラメータ:

予約語 予約語

属性:

省略値:

予約語は削除されません

段階:

構文チェック

$SET:

初期

説明:

本指令は SETTING 指令で作成されたリストには現れません。

REPORT-LINE

報告書行の最大長を指定します。

構文:

パラメータ:

整数 最大長（文字数）

属性:

省略値:

REPORT-LINE"256"

段階:

構文チェック

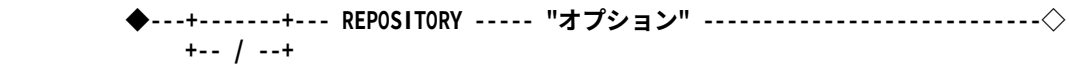
$SET:

初期

REPOSITORY

リポジトリーファイルを作成します。

構文:



パラメータ:

オプション 下記の「説明」を参照してください。

属性:

省略値:

REPOSITORY"OFF"

段階:

構文チェック

$SET:

初期

説明:

「オプション」の値として可能なものは次の通りです。

OFF 何も操作を行いません。

UPDATE コンパイルするクラス、プログラム、関数のインタフェース用にリポジトリーファイルを作成します。

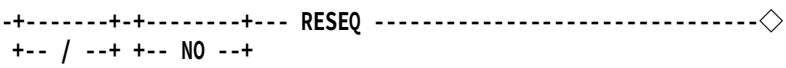
CHECKING CALL 文および INVOKE 文のパラメータの適合性をチェックするためにリポジトリーファイルを使用します。このチェックは、NetExpress の本バージョンでは強制的ではありません。

リポジトリーファイルは、RDFPATH 指令で指定されたディレクトリに作成されます。

RESEQ

コンパイラに行番号を作成させます。

構文:



パラメータ:

なし

属性:

省略値:

RESEQ （Dialect）

段階:

構文チェック

$SET:

初期

依存性:

XREF により直ちに RESEQ が設定されます。

SEQCHK により直ちに NORESEQ が設定されます。

SOURCEFORMAT"FREE"により最後に NORESEQ が設定されます。

説明:

作成されるのは COBOL 行順番号で、1 で始まり 1 づつ増加します。

RETRYLOCK

レコードがロックされていることをREAD文が検出したとき、そのレコードが使用可能になるまで読込み動作を繰返し試みるよう指定します。

構文:

```
  ◆---+-------+-+--------+--- RETRYLOCK ---------------------------◇
       +-- / --+ +-- NO --+
```

パラメータ:

なし

属性:

省略値:

NORETRYLOCK

段階:

構文チェック

$SET:

任意

依存性:

NORM により直ちに NORETRYLOCK が設定されます。

説明:

ファイルに FILE STATUS 項目があってそのファイル用の宣言がない場合にのみ、本指令はファイルに影響します。

REWRITE-LS

REWRITE文を行順ファイルに入れることが可能となります。

構文:

```
  ◆---+-------+-+--------+--- REWRITE-LS --------------------------◇
       +-- / --+ +-- NO --+
```

パラメータ:

なし

属性:

省略値:

REWRITE-LS

段階:

構文チェック

$SET:

不可

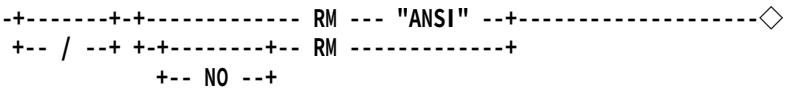
説明:

利用者が書き出すレコードは、それと置き換えられるレコードと同じサイズであることを確認する必要があります。

RM

Ryan-McFarland COBOL V2.0で予約されている語を予約語とするよう指定し、ある機能の動作を変更してその製品との互換を図ります。

構文:



パラメータ:

ANSI 下記の「説明」を参照 （Dialect）

属性:

省略値:

NORM

段階:

構文チェック

$SET:

初期

依存性:

RM は、COMP-6"1"、SEQUENTIAL"LINE"、NOTRUNC、OLDINDEX、NOOPTIONAL-FILE、RETRYLOCK、および ALIGN"2"を直ちに設定します。

RM"ANSI"は、COMP-6"1"、SEQUENTIAL"RECORD"、NOTRUNC、OLDINDEX、NOOPTIONAL-FILE、RETRYLOCK、および ALIGN"2"を直ちに設定されます。

NORM は、COMP-6"2"、SEQUENTIAL"RECORD"、TRUNC"ANSI"、NOOLDINDEX、OPTIONAL-FILE、NORETRYLOCK、および ALIGN"2"を直ちに設定します。

説明:

パラメータに ANSI と指定すると、ANSI スイッチを設定してプログラムをコンパイルしたのと同じように、これらの機能が働きます。『言語リファレンス』を参照してください。

RTNCODE-SIZE

特殊レジスタRETURN-CODEのサイズとそのメモリにおける桁合わせを指定します。

構文:

```
◆--- RTNCODE-SIZE --- "integer" ---------------------------------◇
```

パラメータ:

整数 2、4、または 8

属性:

省略値:

RTNCODE-SIZE"4"

段階:

構文チェック

$SET:

初期

依存性:

XOPEN により直ちに RTNCODE-SIZE"4"が設定されます。

説明:

「整数」の値として可能なものは次の通りです。

２ PIC S9（4）COMP:サイズ 2 バイト、2 バイト境界に合わせられます。

４ PIC S9（9）COMP:サイズ 4 バイト、4 バイト境界に合わせられます。

８ PIC S9（18）COMP:サイズ 8 バイト、8 バイト境界に合わせられます。

4 バイト RETURN-CODE のあるプログラムが 2 バイト RETURN-CODE のあるプログラムを呼び出すと、引取り時、呼ばれるプログラムからの値は、4 バイト RETURN-CODE の下位 2 バイトに入っています。上位 2 バイトの内容は不定です。

8 バイト RETURN-CODE のあるプログラムが 4 バイト RETURN-CODE のあるプログラムを呼び出すと、引取り時、呼ばれるプログラムからの値の下位 4 バイトは、4 バイト RETURN-CODE に入っています。上位 4 バイトの内容は不定です。

あるプログラムがもう一つのプログラムを呼び出す場合、それらのプログラムの RETURN-CODE レジスタのサイズが一致している必要はありません。呼出しプログラムの RETURN-CODE レジスタが呼ばれるプログラムの RETURN-CODE レジスタよりも大きい場合、呼出しプログラムのその余ったバイトの内容は不定です。たとえば、2 バイト RETURN-CODE のあるプログラムが 4 バイト RETURN-CODE のあるプログラムを呼び出すと、引取り時、呼ばれるプログラムからの値の下位 2 バイトは、2 バイト RETURN-CODE に入っています。上位 2 バイトの内容は不定です。

呼出しプログラムの RETURN-CODE レジスタが呼ばれるプログラムの RETURN-CODE レジスタよりも小さい場合、その呼出しプログラムのレジスタに入りきらないバイトの内容は失われます。たとえば、4 バイト RETURN-CODE のあるプログラムが 8 バイト RETURN-CODE のあるプログラムを呼び出すと、引取り時、呼ばれるプログラムからの値の下位 4 バイトは、4 バイト RETURN-CODE に入っています。上位 4 バイトの内容は失われます。

RWHARDPAGE

最後の項目がページに出力された後で、通常の複数空白行の代りに書式送りを実行するモジュールを報告書に制御させます。

構文:

パラメータ:

なし

属性:

省略値:

NORWHARDPAGE

段階:

構文チェック

$SET:

初期

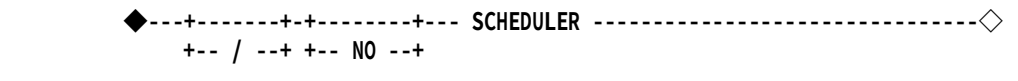
説明:

本指令を指定した CODE IS 句を使用する場合、2 文字コードはページの終りを拡張しません。

SCHEDULER

Intel COBOL Optimizerの起動を制御します。

構文:



パラメータ:

なし

属性:

省略値:

NOSCHEDULER

段階:

生成

$SET:

初期

依存性:

ANIM により直ちに NOSCHEDULER が設定されます。

説明:

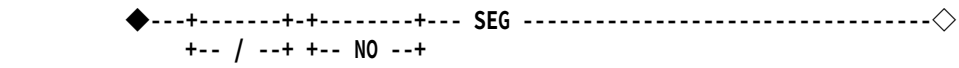
本指令は、Micro Focus コンパイラが作成した生成コードをさらに最適化する Intel COBOL Optimizer（Copyright Intel 1994-1996）の起動を制御します。アプリケーションの性能を向上させたい場合には、本指令を使用してください。

本指令を使用すると、NetExpress デバッガを使用したデバッグを行えないコードが作成されます。その結果、コード生成中に ANIM 指令を選択すると、SCHEDULER がオフになります。

SEG

COBOL区分化をオンにします。

構文:



パラメータ:

なし

属性:

省略値:

SEG

段階:

構文チェック

$SET:

初期

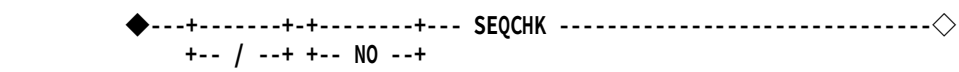
説明:

NOSEG を指定すると、コンパイラはコード内の全節番号をあたかもゼロであるかのように扱います。これは、コンパイラは区分化を無視し、オーバレイなしで 1 つのプログラムを作成することを意味しています。

SEQCHK

コンパイラにカラム1～6の一連番号をチェックさせ、順序をはずれた原始コード行を識別させます。

構文:



パラメータ:

なし

属性:

省略値:

NOSEQCHK （Dialect）

段階:

構文チェック

$SET:

任意

依存性:

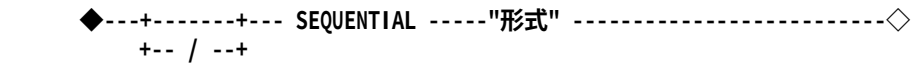
SEQCHK は NORESEQ を直ちに設定します。

SOURCEFORMAT"FREE"により最後に NOSEQCHK が設定されます。

SEQUENTIAL

ORGANIZATION SEQUENTIALと定義された（暗黙的または明示的）ファイルについて省略時のファイル形式を指定します。

構文:



パラメータ:

形式 ADVANCING、ANSI、LINE または RECORD

属性:

省略値:

SEQUENTIAL"RECORD" （Dialect）

段階:

構文チェック

$SET:

初期

説明:

「形式」の値として可能なものは次の通りです。

ADVANCING 行送り付きレコード順

ANSI ANSI 準拠のレコード順ファイル出力

LINE 行順

RECORD レコード順（標準順ファイル）

SERIAL

プログラムをシリアルプログラムとして指定します。本指令を指定すると、ランタイムシステムはプログラムをあるスレッドでアクティブにして他のスレッドがそのアクティブプログラムに入るのを防ぐかどうかが決定されます。これを決定するコードは処理に時間がかかり、どのプログラムの入口でも呼び出されるということに注意してください。プログラムは、マルチスレッド環境で実行される場合はこの指令でコンパイルしなければなりません。

パラメータ:

なし

属性:

省略値:

NOSERIAL

段階:

構文チェック

$SET:

初期

SETTING

SETTINGS

コンパイラに指令の設定のリストを原始プログラムリストへ含ませます。

構文:

パラメータ:

形 式 リストの形式。構文チェックの段階の間に作成されるリストに対してのみ影響します。

属性:

省略値:

NOSETTING

段階:

構文チェック

$SET:

任意

依存性:

NOLIST により最後に NOSETTING が設定されます。

説明:

「形式」には下記のどれかを指定することができます。

LINE 行に指令を列記します。次の指令との間を空白で区切ります。
COL 1 行あたりに指令を 1 つだけ記します。
COL3 1 行あたりに指令を 3 つ、カラム状に記します。
「形式」を指定しないと、LINE と想定されます。

他の COBOL システムとの互換をとるためにのみ提供されているいくつかの指令（COMMIT など）は示されていません。コード生成を制御する指令、特定の変更を予約語リストに加える指令、あるいは、他の指令を指定する指令もまた示されていません。

指令リストは原始プログラムを処理する前に作成され、従って、プログラムにおける$SET 文内の任意の指令は示されません。$SET 文に指令が指定されている場合は、リストはその文の指令すべてが処理されたときに一度だけ作成されます。どちらの場合も、リストにはその時点の各指令の状態が含まれています。

SETTING 指令はまた SETTINGS とすることもできます。

SHOW-DIR

コンパイラに原始プログラムリスト内の指令ファイルの内容を示させます。

構文:

パラメータ:

なし

属性:

省略値:

NOSHOW-DIR

段階:

構文チェック

$SET:

任意

説明:

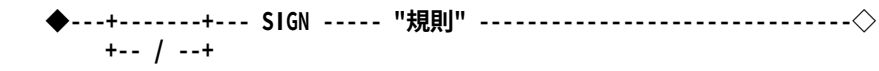
指令ファイルは cobol .dir、または、DIRECTIVES または USE 指令に存在するファイルを意味します。cobol .dir の内容が原始プログラムリストに現れるようにするには、SHOW-DIR は cobol .dir の先頭に指定されている必要があります。それは、このファイルが他の指令に先だって処理されるからです。

指令ファイルの所々のみをリストに出力するため、SHOW-DIR と NOSHOW-DIR を選択的に使用できます。

SIGN

符号を含んだ数字DISPLAY項目について、符号をASCII規則に従って解釈するか、あるいは、EBCDICに従うかを指定します。

構文:



パラメータ:

規則 ASCII または EBCDIC

属性:

省略値:

SIGN"ASCII"

段階:

構文チェック

$SET:

初期

依存性:

CHARSET"ASCII"によって直ちに SIGN"ASCII"が設定されます。

CHARSET"EBCDIC"によって直ちに SIGN"EBCDIC"が設定されます。

SOURCEASM

原始コード文をアセンブラリストへ含むようにコンパイラを動作させます。

構文:

```
◆---+-------+-+--------+--- SOURCEASM ---------------------------◇
     +-- / --+ +-- NO --+
```

パラメータ:

なし

属性:

省略値:

NOSOURCEASM

段階:

生成

$SET:

不可 <!--◆◆◆ -->

依存性:

NOLIST によって最後に NOSOURCEASM が設定されます。

SOURCEASM は最後に ANIM を設定します。

SOURCEFORMAT

COBOL原始プログラムを固定方式とするか自由方式とするかを選択します。

構文:

```
◆---+-------+--- SOURCEFORMAT -----"原始プログラム形式" ----------◇
     +-- / --+
```

パラメータ:

原始プログラム形式 FIXED または FREE

属性:

省略値:

SOURCEFORMAT"FIXED" （Dialect）

段階:

構文チェック

$SET:

任意

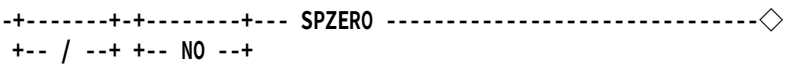
依存性:

SOURCEFORMAT"FREE"は最後に NOMFCOMMENT、NORESEQ および NOSEQCHK を設定します。

SPZERO

USAGE DISPLAYの数字データ項目内の間隔文字を、固有コードのゼロとして扱うようにします。

構文:



パラメータ:

なし

属性:

省略値:

NOSPZERO （Dialect）

段階:

生成

$SET:

任意

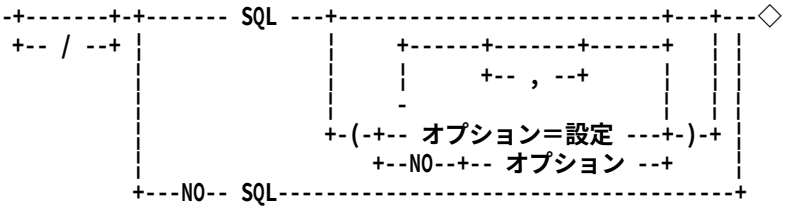
説明:

NOSPZERO を指定すると、数字データ項目内の間隔文字は予期しない結果をもたらします。

SQL

プログラムにODBC用の埋め込みSQLを使用できるようにします。

構文:



パラメータ:

オプション SQL オプション（下記参照）から 1 つ選択

設定 オプション用の設定

使用可能なオプションを以下にリストします。詳細は、各オプションをクリックしてください。

ANSI92ENTRY	ANSI SQL 92サポートのENTRYレベルを可能にします。
AUTOCOMMIT	各SQL文を個別のトランザクションとして処理します。
CHECK	コンパイル時にSQL文を原始データに渡します。
CTRACE	リスト出力に診断結果を入れます。
DB	ODBCデータの原始名を指定します。
DB2	宣言部以外で定義されたデータ項目をホスト変数として許可されるように指定します。
DBMAN	データベースマネージャを指定します。
ESQLVERSION	OpenESQLの適合レベルを設定します。
INIT	プログラムにSQLを初期化させます。
NIST	完全なNIST準拠を可能にします。
PASS	ユーザIDとパスワードを指定します。
TARGETDB	データベース固有の最適化を使用します。

SQLオプション - ANSI92ENTRY

ANSI SQL 92サポートのENTRYレベルを可能にします。

構文:

パラメータ:

なし

属性:

省略値:

NOANSI92ENTRY

説明:

本オプションを設定すると、プリプロセッサは標準 SQL ANSI92 エントリレベルに準拠します。

SQLオプション - AUTOCOMMIT

各SQL文を個別のトランザクションとして処理します。

構文:

パラメータ:

なし

属性:

省略値:

NOAUTOCOMMIT

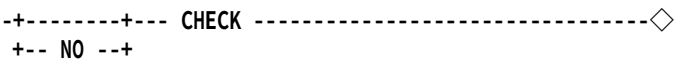
説明:

本オプションを設定すると、各 SQL 文は個別のトランザクションとしてしょりされ、実行されると直ちにコミットされます。本オプションを設定しない場合は、トランザクションを明示的にコミット（またはロールバック）する必要があります。

SQLオプション - CHECK

コンパイル時にSQL文を原始データに渡します。

構文:



パラメータ:

なし

属性:

省略値:

NOCHECK

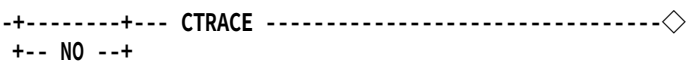
説明:

本オプションを設定すると、コンパイル時に各 SQL 文が原始データに渡されます。これにより、たとえば表およびカラム名が正しいかどうかをチェックできます。

SQLオプション - CTRACE

診断を入れてリスト出力します。

構文:



パラメータ:

なし

属性:

省略値:

NOCTRACE

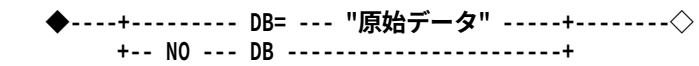
説明:

本オプションは、リストファイルが生成されている場合には、それに含まれているものにだけ影響します。本オプションはリストファイルを生成するかどうか、またはファイルの名前は決めません。

SQLオプション - DB

ODBCデータの原始名を指定します。

構文:



パラメータ:

原始データ ODBC の原始データを指定する英数字の文字列

属性:

省略値:

NODB

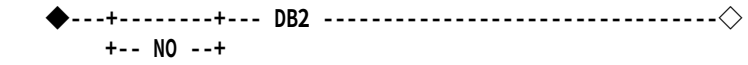
説明:

本オプションは、INIT オプションによりデータベース接続が自動的に行われるように指定されている場合に使用します。

SQLオプション - DB2

宣言部以外で定義されたデータ項目をホスト変数として許可されるように指定します。

構文:



パラメータ:

なし

属性:

省略値:

DB2

説明:

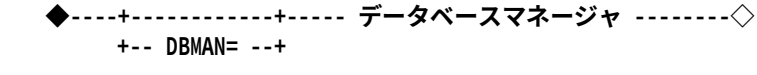
NODB2 を指定すると、宣言部以外で定義されて SQL 文で使用されているデータ項目は SQLDA と想定されます。

DB2 を指定すると、データ項目がホスト変数または SQLDA かどうかを調べるために検査されます。

SQLオプション - DBMAN

データベースマネージャを指定します。

構文:



パラメータ:

データベースマネージャ ODBC 原始データへのアクセス用にプリコンパイルするために ODBC を指定します。

属性:

省略値:

DBMAN=IBM

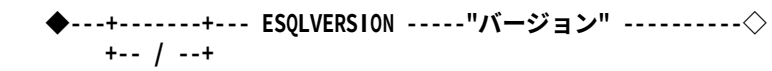
説明:

ODBC 原始データへのアクセス用にプリコンパイルするには ODBC を使用します。

SQLオプション - ESQLVERSION

OpenESQLの適合レベルを設定します。

構文:



パラメータ:

バージョン OpenESQL の適合レベルを指定します。

1.0 - MS SQL Server 1.0 版 ESQL Toolkit に適合

2.0 - MS SQL Server 2.0 版 ESQL Toolkit に適合

3.0 - 新しい ODBC ベースの機能性

属性:

省略値:

ESQLVERSION=3.0

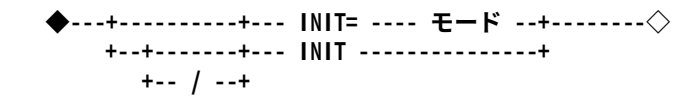
説明:

「バージョン」は OpenESQL が準拠する MS SQL Server のバージョンを指定するものです。本オプションは、バージョン間で変更のあったリターンコード値などに影響します。

SQLオプション - INIT

プログラムにSQLを初期化させ、データベースにログオンします。また、データベースをプログラムの異常終了から守るためにプロセスを登録します。

構文:



パラメータ:

モード このパラメータに指定できる値は、PROT のみ

属性:

省略値:

INIT=PROT

説明:

INIT（パラメータなしの場合）は、DB または PASS オプションにより提供された情報を使用してプログラムにデータベースへの接続を初期化させます。この処理の一環として、STOP RUN が発生したときにデータベースのクローズが常に正常に行われるように手続きを登録します。本オプションを指定しなければ、プログラムが処理を完了する前に終了してしまった場合などにデータベースが不整合な状態のままになってしまうことがあります。それは、たとえばプログラムの完了前にアニメーションのセッションが中断された場合などです。

PROT オプションは、STOP RUN 時にデータベースを守る必要があるが初期化を必要としない SQL プログラム用に提供されています。PROT オプションは、プログラムが終了したときにアクティブなトランザクションをロールバックします。

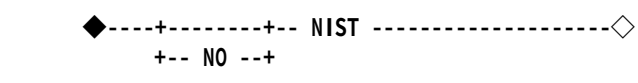
本指令を指定していなくても、プログラムの終わりに達する前に終了するとき、障害状態のデータベースを抜けることができます。例えば、プログラムが完了する前にアニメートセッションを終了するような場合です。この保護機能は初期化しなくても SQLPROT 指令を使用することで適用されます。

NOINIT は他の SQL プログラムに呼び出される SQL プログラムについて指定する必要があります。NOINIT はまた実行単位の最初の SQL プログラムについても指定できますが、任意の EXEC SQL 文を実行する前に、そのプログラムは CONNECT 文を実行する必要があります。

SQLオプション - NIST

完全なNIST準拠を可能にします。

構文:



パラメータ:

なし

属性:

省略値:

NONIST

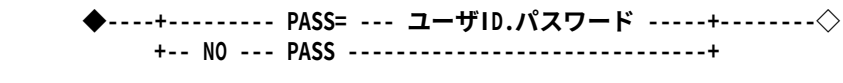
説明:

本オプションを設定すると、プリプロセッサは完全な NIST 準拠を提供します。

SQLオプション - PASS

データベースのユーザIDおよびパスワードを指定します。SQL(INIT)を指定している場合に本オプションを使用する必要があります。

構文:



パラメータ:

ユーザ ID.パスワード 英数字の文字列。次のどちらかの形式で指定します。

ユーザ ID[.パスワード]

ユーザ ID[/パスワード]

属性:

省略値:

NOPASS

説明:

本オプションは、INIT オプションによりデータベースが自動的に接続されるようになっている場合に指定する必要があります。パスワードは接続をするときに要求されます。

これ以外の場合はオプションです。

SQLオプション - TARGETDB

データベース固有の最適化を使用します。

構文:

```
◆---+--------+--- TARGETDB -----"データベース" ----------◇
    +-- NO --+
```

パラメータ:

データベース 指定可能な値は、以下の 2 つです。

MSSQLSERVER

ORACLE7

属性:

省略値:

NOTARGETDB

説明:

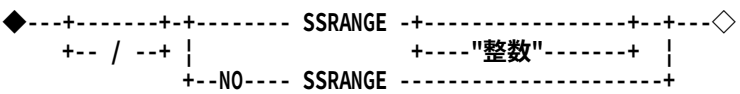
本オプションを設定すると、特定のデータベースの性能が最適化されます。

SSRANGE

実行時に次の指定を行います。

- 添字または索引が表の外部の領域を参照していないかどうかをチェックします。
- 参照修正項目の値が負またはゼロでないかどうかをチェックします。

構文:



パラメータ:

整数 必要な SSRANGE サポートのレベル

属性:

省略値:

NOSSRANGE

段階:

構文チェック

$SET:

初期

依存性:

直ちに BOUND を設定します。

説明:

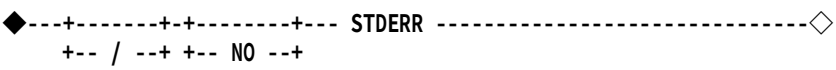
「整数」の値として可能なものは次の通りです。

1 長さに負またはゼロの値を指定して参照修正式を使用しようとした場合に、エラー 153 を発生させます。

STDERR

エラーメッセージをコンソール（STDOUT）ではなくSTDERRに表示するようにします。

構文:



パラメータ:

なし

属性:

省略値:

NOSTDERR

段階:

構文チェック

$SET:

任意

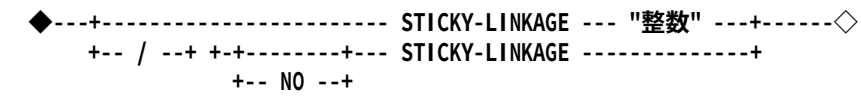
説明:

本指令はコマンド行からコンパイルを実行する場合のみ使用できます。

STICKY-LINKAGE

プログラムに対するパラメータを後続のプログラム呼出し中連結されたままとします。

構文:



パラメータ:

整数 1 または 2

属性:

省略値:

NOSTICKY-LINKAGE （Dialect）

段階:

構文チェック

$SET:

初期

説明:

COBOL 連絡節の目的は、連絡節の中に宣言されている 01 レベルまたは 77 レベルのデータ項目（連絡項目）と動的に指定されるデータ項目とを結び付けて、両者が同じ記憶域を共有するようにすることです。この連絡を確立する方法は 3 通りあります。具体的には、プログラム名を参照する CALL 文、入口名を参照する CALL 文、SET 文です。ANSI の標準に設定されているのは最初の方法です。残りの 2 つの方法はこの COBOL システムにおける COBOL 言語の拡張機能です。

ANSI COBOL においては、連絡節を使用して、CALL 文から渡されるパラメータを呼び出されるプログラムでアクセスできるようにします。CALL 文の USING 指定に含まれるパラメータの数は PROCEDURE DIVISION 見出しの USING 指定に含まれるパラメータの数と等しくなければなりません。連絡節の中の 01 レベルまたは 77 レベルのデータ項目で PROCEDURE DIVISION 見出しの USING 指定に現われないものを参照してはなりません。

ENTRY 文は、プログラムへの代替入口点を提供するための、COBOL 言語の拡張機能です。ENTRY 文も PROCEDURE DIVISION 見出しと同様の USING 指定を取ります。ENTRY 文の場合も、CALL 文の場合と同じように、プログラムが呼び出される入口名に対応する USING 指定の中に現われない連絡データ項目を参照してはなりません。

SET 文を使用すると、記憶場所を識別する POINTER の値によって連絡項目を指させることができます。そして、指された連絡項目を参照して、その記憶場所に保持されているデータをアクセスすることができます。

以下に、連絡項目がいつリンクされ、リンクを解除されるかについて、概念的に詳しく説明します。

COBOL システムは実行時に連絡項目が参照されたときに、それがデータ項目とリンクされたことを確認します。参照された連絡項目が記憶場所とリンクされないと、COBOL システムはランタイムエラーを発生させます。このチェックを促進するために、プログラムが呼び出しされるたびにその最初の文が実行される直前に、すべての連絡項目のリンクがまず解除され、呼出し元のプログラムから渡されたパラメータに対応する連絡項目がリンクされます。このことは、呼び出されたプログラムの中の連絡項目で SET 文によってデータ項目にリンクされていたものが、その後でそのプログラムが呼び出されたときには、リンクを解除されることを意味します。したがって、SET 文は各呼出しごとに実行しなければなりません。

他の COBOL 製品の中には、ANSI の規則の適用を甘くしているものが数多くあります。そこで、STICKY-LINKAGE 指令にパラメータを 2 つ用意して、ANSI の規則の適用を厳格にするか甘くするかを指定できるようにしています。

STICKY-LINKAGE"1"を指定すると、呼ばれるプログラムの USING 指定の中に現われる連絡項目だけがリンクを解除されます。そうでない連絡項目は、それまでにリンクがなされていれば、その状態が維持されます。その USING 指定は、プログラム名を使用して呼び出された場合は、PROCEDURE DIVISION 見出しの USING 指定を指します。そうでない場合は、プログラムの呼出しに使用された入口点に対応する USING 指定が、その USING 指定にあたります。

この方式を取る場合でも、呼出し元のプログラム中のパラメータの指定が足りなくて、CALL 文の USING 指定にはないが呼ばれるプログラムの USING 指定中に現われる連絡項目が参照されると、ランタイムエラーが発生します。

STICKY-LINKAGE"2"を指定すると、プログラムが呼ばれたときに、連絡項目のリンクは通常は解除されません。ただし、そのプログラムが起動後の初期状態にある場合、またはそのプログラムに対して CANCEL 文が実行された場合を除きます。すべての連絡項目は、呼出し元のプログラムからパラメータを渡されるか SET 文でリンクを変更されないかぎり、それ以前の呼出しのときにリンクされた状態を維持します。

例：

下記のプログラムを STICKY-LINKAGE 指令を指定しないでコンパイルして実行すると、2 番目の CALL 文によって"ent"に制御が渡され、その中の DISPLAY 文によって 2 つの連絡項目が参照された時点で、「CALL パラメータがたりない」というランタイムエラー 203 が発生します。DISPLAY 文によって参照された 2 つの連絡項目が入口点"ent"の USING 指定に宣言されていないからです。

このプログラムに STICKY-LINKAGE"1"指令を指定してコンパイルし直してから実行すると、2 番目の CALL 文は無事に実行されますが､3 番目の CALL 文で再びエラーが発生します｡制御を渡された"sub"の中の DISPLAY 文によって参照される連絡項目は PROCEDURE DIVISION 見出しの USING 指定中に現われますが、CALL 文から対応するパラメータを渡されていないからです。

このプログラムに STICKY-LINKAGE"2"指令を指定してコンパイルしてから実行し直すと、処理は最後まで正常に行われます。

```
program－id. main.
working-storage section.
```

```
01 param pic x value "a"
procedure division.
    call "sub"using param
    call "ent"
    call "sub"
    srop run.
end program main.
program-id. sub.
linkage section.
01 link-1 pic x.
01 link-2 pic x.
procedure division using link-1.
    set address of link-2 to address of link-1
    display link-1 link-2
    exit program.
entry "ent"
  display link-1 link-2
  exit program.
```

STICKY-PERFORM

プログラムが再入力されるとき、PERFORM文の動作を指定します。

構文:

```
  ◆---+-------+-+--------+--- STICKY-PERFORM ----------------------◇
      +-- / --+ +-- NO --+
```

パラメータ:

なし

属性:

省略値:

NOSTICKY-PERFORM （Dialect）

段階:

構文チェック

$SET:

初期

説明:

本指令は、PERFORM-TYPE"RM"を指定してプログラムをコンパイルした場合のみ効果があります。

コンパイル時に PERFORM-TYPE"RM"を指定すると、PERFORM 文はプログラム中の PERFORM 文の最後の部分に関連する記憶域またはバケットを確保することにより実行されます。

PERFORM 文が実行されると、返されたアドレスはバケットに格納されます。そしてそのバケットは PERFORM 文の最後の最後で制御をどこに戻すかを決めるために読み込まれ、それからクリアされます。

STICKY-PERFORM を指定しないと、EXIT PROGRAM 文がこのバケットをクリアします。そのため、プログラムを再入力してもそれまでに実行された PERFORM 文が返したアドレスはまったく残っていません。
STICKY-PERFORM を指定すると、バケットが EXIT PROGRAM 文によってクリアされるのを防ぎます。これは、プログラムを再入力したときにそれまでに実行された PERFORM 文が返したアドレスがすべて残っていることを意味します。

STOP RUN または CANCEL 文は、本指令の設定に関わらずバケットをクリアします。

SUPFF

コンパイルリストが画面に送られる場合、コンパイルリスト上の書式送り文字を抑制します。

構文:

パラメータ:

なし

属性:

省略値:

SUPFF

段階:

構文チェック

$SET:

任意

SWITCH-TYPE

プログラム可能なスイッチの動作を標準ISO2000に準拠させます。

構文:

パラメータ:

整数 1 または 2

属性:

省略値:

SWITCH-TYPE

段階:

構文チェック

$SET:

任意

説明:

「整数」には次の値を設定します。

1 NetExpress 動作
2 ISO2000 動作

標準 ISO2000 では、スイッチは実行中は持続して他から見えます。たとえば、プログラム A がプログラム B を呼び出す場合、A のスイッチに関する変更は B が呼び出された時点で表面化し、B で行われた変更は制御が A に戻った時点で表面化します。ただし、A が終了して B が呼び出された場合、A によって行われた変更内容は失われます。スイッチが格納される領域は、ある実行単位のプログラムすべてが共有できるものであるため、スイッチの変更内容はすべてのプログラムから見えます。

NetExpress および Micro Focus COBOL の前バージョンまでは、呼び出されるサブプログラム（この例では B のサブプログラム）はすべて他のプログラムとは独立したそれ自身の COBOL スイッチのセットを持っています。プログラム A がプログラム B を呼び出す場合、プログラム B のスイッチにはプログラム A のスタート時点のスイッチの値が設定されます。つまり、A がスイッチに対して行う変更は無視されます。制御がプログラム A に戻ると、A のスイッチはプログラムの制御が変わる前と同じようになります。つまり、B がスイッチに対して行う変更は無視されます。プログラム B のスイッチは（中断されない限り）保持されます。そのため、再度プログラム B が呼び出されると、そのスイッチはプログラム A を呼び出したときと同じように設定されます。この場合は、スイッチが格納される領域は各プログラムが個別に使用できます。

SYMBSTART

SYMBOLIC CHARACTERS句をコンパイルするとき、コンパイラが文字の大小順序における順序位置を数え始める番号を設定します。

構文:

```
◆---+-------+--- SYMBSTART ----- "整数" -------------------------◇
    +-- / --+
```

パラメータ:

整数 使用される番号

属性:

省略値:

SYMBSTART"1" （Dialect）

段階:

構文チェック

$SET:

初期

説明:

ANSI に準拠するには SYMBSTART"1"を使用し、前の製品との互換を図るには SYMBSTART"0"を使用します。

例:

SYMBSTART"1"を指定すると、

symbolic characters bee is 67

上記の COBOL 文は"B"を表す記号文字 BEE を宣言します。その理由は、1 からカウントを始めると、"B"は ASCII 文字の大小順序における 67 番目の文字だからです。SYMBSTART"0"を指定すると、BEE は"C"を表します。

TARGET

コンパイラにあるマイクロプロセッサだけで使用可能なある命令を生成できるかどうかを知らせます。この命令を使用すると、処理速度が速くなり、コードのサイズもわずかに小さくなります。

構文:

```
◆---+-------+--- TARGET ----- "プロセッサID" ------------------------◇
    +-- / --+
```

パラメータ:

プロセッサ ID プロセッサの種類を識別します

属性:

省略値:

TARGET"386"

段階:

生成

$SET:

初期

説明:

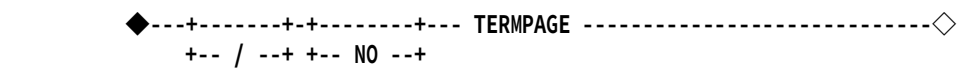
「プロセッサ ID」に指定可能な値は以下の通りです。

386	コードは80386、80486、およびPentiumプロセッサ上で実行されます。
486	80486およびPentiumプロセッサのみで使用できる命令を含みます。
PENTIUM	486用ですが、コードサイズが大きくなる代りにコードの処理速度を向上するように最適化を行う。この最適化はコードを486上で実行しますが、その利点はコードがPentium上で実行されたときほどは多くありません。

TERMPAGE

報告書ファイルの最後のページが完全なページ長さになるまで空白行を充てんするかどうか決めます。

構文:



パラメータ:

なし

属性:

省略値:

TERMPAGE

段階:

構文チェック

$SET:

初期

説明:

TERMPAGE を指定すると、レポートの最後のページに、ページがいっぱいになるまで、空白行が追加されます。

PAGE 指定がその報告書記述（RD）項に指定されている任意の報告書ファイルに本指令は影響します。

TIME

リストの各ページの先頭に時刻を入れます。

構文:

パラメータ:

なし

属性:

省略値:

TIME

段階:

構文チェック

$SET:

不可

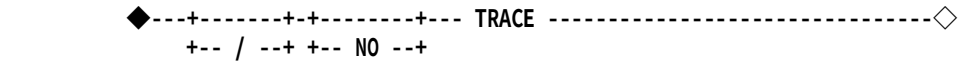
説明:

NODATE が指定されていると、本指令は効果がありません。

TRACE

READY TRACEとRESET TRACEをオンにします。

構文:



パラメータ:

なし

属性:

省略値:

NOTRACE （Dialect）

段階:

構文チェック

$SET:

初期

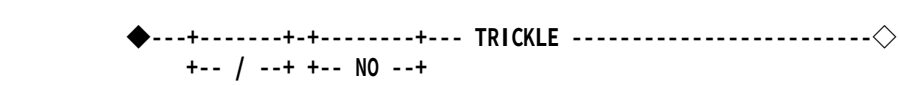
説明:

TRACE を設定すると、実行時 READY TRACE 文が追跡機能をオンにし、これにより、各段落または節の見出しの名前が、実行されるときに表示されます。RESET TRACE はこの機能をオフにします。NOTRACE を設定すると、READY TRACE と RESET TRACE 文は効果がなくなります。

TRICKLE

コンパイラに論理的だが構造化されていない実行範囲がプログラムに含まれていて、そのためにある最適化を行えないことを知らせます。

構文:



パラメータ:

なし

属性:

省略値:

TRICKLE

段階:

生成

$SET:

初期

説明:

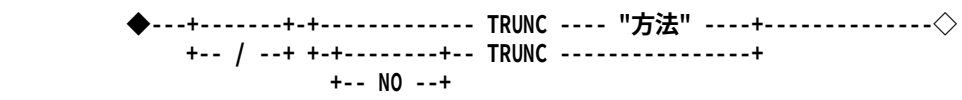
NOTRICKLE はコンパイラにプログラムがある実行範囲から別の実行範囲への動作を制御できないことを知らせます。そのようなプログラムは制御フローを構造化しています。つまり、段落および節が含まれていません。段落および節は、通常は実行されることによって、また制御がその前のコードからその段落または節に移ることによって記述されるものです。また、そのようなプログラムでは PERFORM 文の最後および別の PERFORM 文の範囲に何も含まれていません。

NOTRICKLE を指定すると、コンパイラは PERFORM 文用により処理効率の高いコードを生成することができます。

TRUNC

USAGE COMP, USAGE BINARY, USAGE COMP-4のどれかが指定されているデータ項目に収める値を、その項目のPICTURE句によって指定されるサイズまたは項目の最大サイズに合わせて、切り捨てるか否かを指定します。

構文:



パラメータ:

方法 下記の「説明」を参照

属性:

省略値:

NOTRUNC （Dialect）

段階:

構文チェック

$SET:

初期

依存性:

NORM によって TRUNC"ANSI"が直ちに設定されます。

RM または RM"ANSI"によって NOTRUNC が設定されます。

説明:

「方法」の値として可能なものは次の通りです。

TRUNC　　COMP, BINARY, COMP-4 のデータ項目に値を収めるときは必ず、PICTURE 句によって指定される桁数に合わせて 10 進数で切捨てを行います。

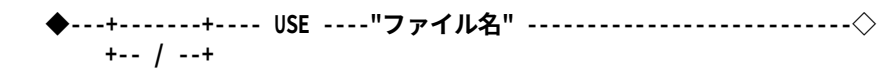
NOTRUNC　COMP, BINARY, COMP-4 のデータ項目に値を収めるときは必ず、割り当てられている記憶容量に合わせて 2 進数で切捨てを行います。

TRUNC"ANSI"　　COMP, BINARY, COMP-4 のデータ項目に対して算術演算以外の操作結果を収めるときは、PICTURE 句によって指定される桁数に合わせて 10 進数で切捨てを行います。ON SIZE ERROR 指定が指定されていない算術文を実行したときに桁あふれ条件が発生した場合、結果はどうなるか分かりません。

USE

コンパイラにファイルから指令を読み込ませます。

構文:



パラメータ:

ファイル名 完全なファイル指定

属性:

省略値:

設定されません

段階:

構文チェック

$SET 任意

説明:

USE は DIRECTIVES と同義です。DIRECTIVES に適用する全ての規則が USE にも適用されます。

VERBOSE

コンパイラから画面へメッセージを送ります。

構文:

パラメータ:

なし

属性:

省略値:

VERBOSE

段階:

構文チェック

$SET:

不可

説明:

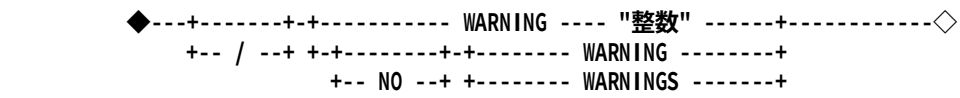
VERBOSE を指定すると、受け取った指令およびコードとデータ領域のサイズに関するメッセージが画面上に表示されます。

WARNING

WARNINGS

報告するエラーの最低重大度レベルを指定します。

構文:



パラメータ:

整数 1、2 または 3

属性:

省略値:

WARNING"1" （Dialect）

段階:

構文チェック

$SET:

任意

説明:

「整数」の値として可能なものは次の通りです。

1 レベル U、S または E のエラーのみ

2 レベル U、S、E または W のエラーのみ

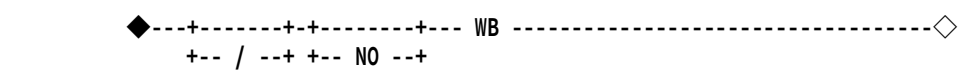
3 全レベル。つまり、レベル U、S、E、W および I

NOWARNING を指定すると、レベル U または S のエラーのみが報告されます。

WB

この指令は、システムが内部的に使用するために予約されています。設定されている指定のリストにこれが載せられることがあるので、完全性を示すためにこの指令をここに含めます。しかし、この指令はユーザーのアプリケーションのために使用するものではないので、その設定を変更してはなりません。

構文:



パラメータ:

なし

属性:

省略値:

NOWB

段階:

構文チェック

$SET:

不可

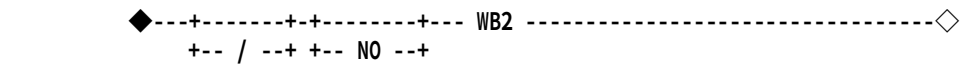
依存性:

WB は直ちに ANIM を設定します。

WB2

この指令は、システムが内部的に使用するために予約されています。設定されている指定のリストにこれが載せられることがあるので、完全性を示すためにこの指令をここに含めます。しかし、この指令はユーザーのアプリケーションのために使用するものではないので、その設定を変更してはなりません。

構文:



パラメータ:

なし

属性:

省略値:

NOWB2

段階:

構文チェック

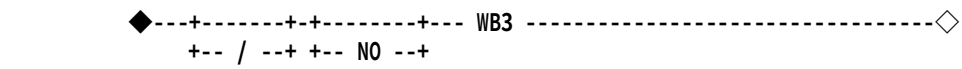
$SET:

不可

WB3

この指令は、システムが内部的に使用するために予約されています。設定されている指定のリストにこれが載せられることがあるので、完全性を示すためにこの指令をここに含めます。しかし、この指令はユーザーのアプリケーションのために使用するものではないので、その設定を変更してはなりません。

構文:



パラメータ:

なし

属性:

省略値:

NOWB3

段階:

構文チェック

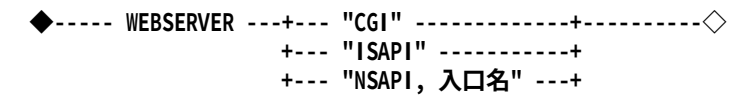
$SET:

不可

WEBSERVER

COBOLプログラムとWebサーバ間で使用するプロトコルを設定します。

構文:



パラメータ:

入口名 NSAPI プロトコルを使用する場合にサーバ構成ファイルで使用する入口点の名前

属性:

省略値:

WEBSERVER"CGI"

段階:

構文チェック

$SET:

初期

説明:

ISAPI および NSAPI を使用する場合、COBOL プログラムは複数回同時に実行されるので、マルチスレッドの問題点を知っておく必要があります。この問題点を処理するには、CASE および REENTRANT(1,2)コンパイラ指令を使用してプログラムを目的コードにコンパイルします。

詳細は、「インターネットアプリケーション」の CGI、ISAPI,NSAPI プログラムに関する章を参照してください。

WRITELOCK

WRITE-LOCK

マルチユーザー環境においてプログラムが共有データファイル内の複数レコードをロックするとき、WRITEおよびREWRITE文にレコードのロックを獲得させます。

構文:

パラメータ:

なし

属性:

省略値:

NOWRITELOCK

段階:

構文チェック

$SET:

初期

説明:

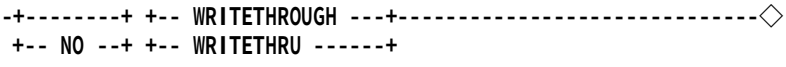
本指令は以前のファイル共有製品との互換を図るために含まれています。新しいプログラムを書くときは、本指令ではなくロック構文を使用してください。

WRITETHROUGH

WRITETHRU

ディスク書出しのバッファリングをしないことを指定します。

構文:



パラメータ:

なし

属性:

省略値:

NOWRITETHROUGH

段階:

構文チェック

$SET:

任意

依存性:

CALLFH を設定する必要があります。

説明:

WRITETHROUGH を指定すると、システムはディスク書き出しをバッファリングしません。

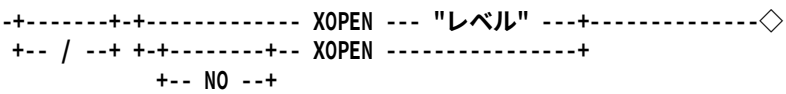
各ファイルに対して本指令を指定するには、＄SET 文を原始プログラムで使用して指令がそのファイルの SELECT 文を含む原始プログラムの部分だけに有効となるようにしてください。

WRITETHROUGH を使用すると、各書込み操作がディスクファイルに対して直接行われ、コンピュータがクラッシュした場合でもデータが失われる可能性が低くなるため、データファイルの完全性が向上します。ただし、これはキャッシュ法とブロック法を使用しないため、性能は低下します。

XOPEN

COBOLのX/Open定義の下で予約されている語が予約語として扱われることを指定します。

構文:



パラメータ:

レベル 3 または 4 とする必要があります。互換を取ろうとしている X/Open 定義 COBOL のレベル。

属性:

省略値:

NOXOPEN （Dialect）

段階:

構文チェック

$SET:

初期

依存性:

XOPEN は RTNCODE-SIZE"4"を直ちに設定します。

説明:

「レベル」の値として可能なものは次の通りです。

３ X/Open 移植性ガイド 1988（XPG-3）と互換があります

４ X/Open CAE 仕様（XPG-4）と互換があります

パラメータなしで XOPEN を指定することは、XOPEN"4"を指定することと同じです。XPG-4 が幾種類かの環境についてのオプションを指定することに注意してください。国別文字サポートは DBCS"3"指令を用いて使用可能となります。

XREF

コンパイラに相互参照表を作成させます。

構文:

```
◆---+-------+-+--------+--- XREF --------------------------------◇
     +-- / --+ +-- NO --+
```

パラメータ:

なし

属性:

省略値:

NOXREF

段階:

構文チェック

$SET:

初期

依存性:

XREF は RESEQ を直ちに設定します。

NOLIST により最後に NOXREF が設定されます。

説明:

LIST 指令が指定されていないと、本指令は効果ありません。

相互参照表を作成するためには、コンパイラディスク上に余分の作業領域を必要とします。必要な記憶域はデータ項目数と手続き名数およびデータ項目・手続き名が参照される回数に依存します。

XREF 指令が指定されると、生成される（ .lst）ファイルの終わりに追加情報が付加されます。

- データ項目の名前
- データ項目の種類
- n#（つまり、データ項目が定義された行番号）
- n*（つまり、データ項目が更新された行番号）
- n?（つまり、データ項目が試験された行番号）
- （X n）（つまり、データ項目が相互参照表に現れた回数）
- 手続き名
- 手続きの種類

例;

ある簡単なプログラム用の（ .LST）ファイルから抽出したものを以下に示します。

```
    1  WORKING-STORAGE SECTION.
    2  01 A PIC 9 (2) .
    3
    4  PROCEDURE DIVISION.
    5  main section.
    6  MOVE 1 TO A
    7  IF A = 1 DISPLAY "HELLO" END-IF
    8  stop run.
. . .
* A               numeric DISPLAY
*    2#   6*   7?
 (X      3)
*       1 data-names
* MAIN            Section
*    5#
 (X      1)
*       1 procedure-names
* End of cross reference listing
```

このリストファイルからの相互参照情報は、1個のデータ項目Aがあり、その種類は数字表示で、行2に定義されており、行6で更新され、行7で試験されることを示しています。そのデータ項目は相互参照表に3度現れています。（X3）は、行の最後で相互参照リストにデータ項目が現れる回数を参照します。手続き名Mainも、1度だけ参照される節として、その相互参照表に現れています。

ZEROLENGTHFALSE

ゼロ長項目を伴う字類試験を行う方法を変更します。

構文:

パラメータ:

なし

属性:

省略値:

NOZEROLENGTHFALSE （Dialect）

段階:

構文チェック

$SET:

任意

説明:

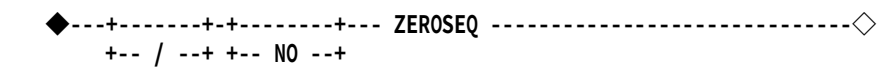
ZEROLENGTHFALSE を設定してある場合、長さがゼロの集団項目間の比較および長さがゼロの項目と表意定数との間の比較を行うと、すべて FALSE が返されます。ZEROLENGTHFALSE が設定されていない場合は、TRUE が返されます。

ANSI に準拠するためには、ZEROLENGTHFALSE を設定してください。

ZEROSEQ

カラム1～6の一連番号に先行ゼロ列が現れるようにします。

構文:



パラメータ:

なし

属性:

省略値:

NOZEROSEQ

段階:

構文チェック

$SET:

任意

説明:

NOZEROSEQ はこれらの先行ゼロ列を抑制します。

第7章:　システムとプログラミング上の制限

7.1　概説 - COBOLシステムの制限

コンパイラ、ランタイムシステム、ハードウェア、およびオペレーティングシステムにおけるさまざまな制限以外にも、注意して使用しなければならないものが多数あります。

コンパイラおよびコンパイラが作成する中間コードは、環境に依存しません。また、コンパイラに関わる制限も環境に依存せず、したがってどの環境でも制限を超えることはありません。ランタイムシステム、オペレーティングシステム、およびハードウェアの制限は、コンパイラの制限よりもかなり低くなっています。

この節では、次の項目に関わる制限または制約を提示します。

「7.2　ALTER文とACCEPT文」
「7.3　ENTRY文」
「7.4　ファイル名の長さ」
「7.5　浮動小数点数」
「7.6　IF文, CALL文, PERFORM文の入れ子」
「7.7　クラス中のメソッドの数」
「7.8　数値のサイズ」
「7.9　データ項目の大きさと数」
「7.10　利用者語、定数、PICTURE文字列の長さ」
「7.11　USE文の数」
「7.12　USING句のパラメタの数」
「7.13　GIVING句のファイルの数」
「7.14　プログラム、部、節の大きさ」
「7.15　プログラムソースファイル」
「7.16　表の次元」
「7.17　コマンド行の長さ」
「7.18　ハードウェアまたはオペレーティングシステム」

7.2　ALTER文とACCEPT文

次の文を使用する場合には注意が必要です。

- ALTER：　セグメント化しているプログラムにおいては、現在のセグメントの外のコードを変更するべきではありません。現在のセグメントの外のコードを変更しても、コンパイラはそれをエラーとして検出しません。変更先のセグメントに入ったときに、コンパイラはコードを変更します。

- ACCEPT *識別子* FROM LINE NUMBER： この構文は、一般的に多くの端末回線をサポートしているようなオペレーティングシステムのみで使用できます。Windowsでは、常にゼロが返されます。

7.3 ENTRY文

ENTRY文を使用して定義された入口点は、主入口点（プログラム名によって識別されます）が呼び出された後でのみ、アクセスすることができます。ただし、プログラムが静的にリンクされている場合とMFENTMAP機能が使用されている場合は別です。

ネストされているプログラムの中にENTRY文を含めることはできません。

入口点は、プログラムへの迂回ルートです。つまり、入口点は個別のプログラムとしては扱われません。

プログラムの入口点名はすべて30文字で切り捨てられます。LITLINK呼出し名は、リンカーにより31文字で切り捨てられます。

7.4 ファイル名の長さ

ファイルは、COBOLプログラムではユーザが定義したファイル名で識別されます。COBOLシステム以外では、外部のオペレーティングシステムの形式に合ったファイル名で識別されます。それには次の4つのコンポーネントがあります。

- ドライブ
- パス
- 基本名
- 拡張子

ファイル名の長さは、261文字までと制限されています。基本名と拡張子の長さは、合わせて256文字です。

COBOLでは、他に制限がありません。ただし、オペレーティングシステムがファイル名を正しく処理するためには、プログラムの実行時にどのオペレーティングシステムの制限にも従わなければなりません。

7.5 浮動小数点数

COBOLシステムは、IEEE浮動小数点をサポートしています。これにより、Microsoft Visual C++ V4,0などMicrosoftが提供しているさまざまな言語との互換性が完全に保たれます。

浮動小数点データ型のそれぞれ使用可能な値の範囲は、下記の通りです。

COMP-1	8.43E-37から 3.37E38まで

-8.43E-37から-3.37E38まで

COMP-2　　4.19E-307から 1.67E308まで
-4.19E-307から-1.67E308まで

上記の範囲は内部でサポートされる値ですが、本COBOLシステムでは2桁の指数しかサポートしません。その値の範囲は、実際には次の通りとなります。

COMP-2　　4.19E-99から 1.67E99まで
-4.19E-99から-1.67E99まで

浮動小数点の正確な値の範囲は、次のようになります。

COMP-1(4バイト)　　6～7桁の有効数字
COMP-2(8バイト)　　15～16桁の有効数字

定数では、浮動小数点の値は次の範囲でなければなりません。

0.54E-78から0.72E+76まで
-0.54E-78から-0.72E+76まで

7.6　IF文，CALL文，PERFORM文の入れ子

中間コードにおいては、標準のPERFORM文のネストの深さの最大値は100です。生成コードの場合は、その値はオペレーティングシステムによって割り当てられるメモリの量によって変わります。

ネスト文の種類	深さ
PERFORM文	
-中間コード	100
-生成コード	オペレーティングシステムが割り当てるメモリの量に依存
IF文（コンパイル済みのコードのみ）	255（ランタイムシステムにより制限されます）
CALL文	特に制限なし

7.7　クラス中のメソッドの数

クラス中のメソッドの数は、999個までと制限されています。

7.8　数値のサイズ

ANSI標準によると、数値の有効桁数は10進の18桁までです。すべての有効数字はその範囲内に収まらなければなりません。

COBOLシステムでは、乗算または除算の結果が36桁よりも大きくなると、ON SIZE ERRORが発生します。同様に、加算または減算の結果が37桁よりも大きくなった場合にも、ON SIZE ERRORが発生します。

7.9 データ項目の大きさと数

データ項目	最大サイズまたは最大数
英数字	256MB
英数字編集/数字編集	32の部分から構成される。全長が512バイトまでとなるように、そのそれぞれに、1つのタイプの編集文字を16個まで含めることができる。
数値	18桁
COMP/COMP-5形式	8バイト
EXTERNALデータ項目	特に制限なし
ACCEPT FROM CONSOLE	制限は（ある場合は）、オペレーティングシステムによって異なる。
ANSI DISPLAY	本COBOLシステム上では無制限

7.10 利用者語、定数、PICTURE文字列の長さ

言語要素の制限は、次の通りです。

言語要素	最大サイズ
英数字定数	2048バイト
PICTURE文字列	30文字
PICTUREレプリケーション（PIC X(n)のnなど）	256,000,000
プログラム名	30文字
入口点名	30文字
プログラマ定義の語	30文字
外部ファイル名（原始プログラム、オブジェクトファイル、COPYファイル、データファイルなど）	ファイル名は261文字、基本名と拡張子は合わせて256文字
動的ロード可能プログラム名	ファイル名は261文字、基本名と拡張子は合わせて256文字、先頭から30文字はユニークであること

7.11 USE文の数

1つのプログラムにおけるUSE文の数は、100までとする。

7.12 USING句のパラメタの数

次のパラメータの最大数は、各種類のUSING句で認められています。

USING句を使用した文	最大パラメータ数
CALL文	255
PROCEDURE DIVISIONヘッダ	255
ENTRY文	255
CALLプロトタイプ	40

7.13 GIVING句のファイルの数

SORT ファイル名 GIVING句またはMERGE ファイル名 GIVING句におけるファイルの最大数は、255です。

7.14 プログラム、部、節の大きさ

次の表は、プログラムの部および節の最大サイズを示しています。

オブジェクト	最大サイズ
全長	無制限
データ部	256MB
手続き部	816MB
局所記憶節	プラットフォームに依存します。サイズは良い性能を維持できる最小のサイズに保たれます。
COGOLセグメント	16MB

7.15 プログラムソースファイル

次の制限は、プログラムソースファイルに適用されます。

ソースファイル項目	項目数または項目のサイズ
ソース行	999,999
データおよび手続き名	64,000
ホスト変数	8,000またはデータベースシステムの制限
定数値	無制限
COPY仮原文の語	65,000
COPY文のBY句	65,000
実固定長形式行	80バイト
有効固定長形式行	72バイト
自由形式行	160バイト

7.16　表の次元

表要素	制限
表サイズ	無制限
OCCURS句のネスト	無制限
画面節	特に制限なし
添字の数	16
画面節中のOCCURS句	これはフィールドの数と反復のレベルのどちらか一方または両方によって決まる。具体的な制限値を予め示すことはできない。この制限を超えると、プログラムをコンパイルしたときにエラーが発生する。

7.17　コマンド行の長さ

COBOLでは、コマンド行の最大長はオペレーティングシステムによって決められます。

コマンド行の長さを128文字までと制限しているMicro Focusツールも中にはあります。ただし、ユーザアプリケーションは、オペレーティングシステムの制限に従うべきです。

7.18　ハードウェアまたはオペレーティングシステム

オペレーティングシステムまたはハードウェアの構成およびパラメータ設定は、COBOLアプリケーションを直接抑制することになります。オペレーティングシステムおよびハードウェアの制限を確認してください。次の項目を考慮する必要があります。

項目	制限
レコードロック	オペレーティングシステムに依存
ファイルロック	オペレーティングシステムに依存
CALL文のネスト	オペレーティングシステムが割り当てたスタック領域により制限されます。
DD_環境変数の設定値の長さ	オペレーティングシステムの1パスあたりの最大サイズまで
データおよび手続きの合計サイズ	使用可能な空きメモリ容量により制限されます。
同時にロードできるサブプログラム	使用可能な空きメモリ容量により制限されます。
ファイルサイズおよびレコードサイズ	オペレーティングシステムの環境により制限されます。
同時にオープンできるファイル	オペレーティングシステムの環境により制限されます。
ファイル名のサイズおよび形式	オペレーティングシステムに依存

索引キーのサイズおよび数	標準以外のファイルハンドラが使用されている場合は制限されます。
整列キーのサイズおよび数	標準以外のSORTサポートルーチンが使用されている場合は制限されます。

第8章　ライブラリルーチン（名前による呼出し）

本製品と一緒に提供されるランタイムシステムは、COBOL CALL文を用いて使用可能なルーチンのライブラリを持っています。これらのルーチンは一般にCOBOL構文を用いてアクセスできない機能を提供します。名前を呼び出すことによりアクセスされるこれらのルーチンは、名前による呼出しルーチンと呼ばれています。

8.1　概説

本製品にランタイムシステムと一緒に提供される名前による呼出しライブラリルーチンは、オペレーティングシステム機能のようなCOBOL構文を用いてアクセスできない機能を提供します。

このような機能の多くは、オペレーティングシステムの一部として提供されている関数を呼び出すことにより、直接アクセスできます。しかし、このような呼出しは、プログラムを特定のオペレーティングシステムに結び付けるという不都合があります。例えば、機能にアクセスするためにOS/2 API関数を使用する場合、API呼出しがファミリーAPIの一部であり、プログラムがファミリーアプリケーションに結合されているとすると、DOSからしかアクセスできません。しかし、そのようなプログラムはWindowsまたはUNIXの下では実行できません。

8.2 名前による呼出しルーチンの移植性

本COBOLシステムの古いバージョンでは、全てのシステムライブラリルーチンは、1個の16進数を呼び出すことでアクセスされていました。これらのルーチンは数字による呼出しルーチンとして知られており、名前を呼び出してアクセスされるルーチン（名前による呼出しルーチン）によって置き換えられています。ディスクマニュアルに含まれているリストは、各名前による呼出しルーチンごとに、置き換えた数字による呼出しルーチンがあれば、その数字を示しています。名前による呼出しルーチンが追加機能を持っている場合もあります。

可能な限り数字による呼出しルーチンではなく名前による呼出しルーチンを使用してください。同様に、接頭語"CBL_"の付いたCOBOLランタイムシステムルーチンが存在する場合、これらルーチンの使用をオペレーティングシステム関数の呼出しに優先させてください。その理由は、これらの呼出しの使用により、DOSとOS/2間の移植性、また、UNIX用Micro Focus COBOLのような同じ呼出しをサポートしている他のCOBOLシステムへの移植性が提供されるからです。

名前による呼出しルーチンに付けられた接頭語は、次のようにそのルーチンの使用可能レベルを示します。

CBL_	先に述べたように本COBOLシステムでサポートされている環境上で使用可能です。
PC_	DOS、OS/2およびWindows環境上でのみ使用可能で、通常これらの環境に特定されます。

CBL_ルーチンは全てのランタイムシステム間で完全な移植性があるようになっており、1つまたは2つのランタイムシステムにおいてのみ現在使用可能ものがいくつかあることに留意しておいてください。

CBL_ルーチンを用いてプログラムの移植性を保証するには、プログラム内でルーチンの名前がかならず大文字で符号化されている必要があります。Micro Focus COBOLシステムが、与えられた全ての環境において使用可能となるような環境においてのみ、移植性仕様は使用可能です。

注:

8.3 カテゴリー別ルーチン

この項は、特定の目的に使用するルーチンを見つけやすくするためのものです。ルーチンの名前を項類別編成で列挙しており、各ルーチンにその目的についての簡単な説明を付けています。「ルーチンの説明」の項は、アルファベット順にルーチンの詳細説明を行っています。

Micro Focus COBOLと一緒に提供される名前による呼出しルーチンのカテゴリーは次の通りです。

- アプリケーションサブシステム
- バイトストリームファイル
- 文字セット変換ルーチン
- デバッグルーチン
- 表示属性ルーチン
- 閉止手続き
- エラー手続き
- ファイル名
- ファイル
- キーボード
- 論理演算子
- メモリ割付け/割付け解除
- マウス
- マルチスレッド
- NLSメッセージファイル処理
- オペレーティングシステム情報
- 移植
- プリンタ
- プログラム取り消し
- プログラム情報
- 実行単位処理
- 画面
- 状態保守
- テキスト変換
- 仮想ヒープ
- Windows

各カテゴリーのルーチンを以下に列挙します。各カテゴリーのルーチン全てに共通の詳細説明も付けてあります。特別なルーチンは全てのMicro Focus環境で使用できないことがあることに留意しておいてください。詳細については、「名前による呼出しルーチンの移植性」の項を参照してください。

8.3.1 アプリケーションサブシステム

本ルーチンは、多数のプログラムを論理的にグループ分けします。これにより、これらのプログラム、つまりサブシステムを単独の文で中断することができます。

CBL_SUBSYSTEM	宣言/割付け解除サブシステム

8.3.2 バイトストリームファイル

CBL_CLOSE_FILE	ファイルのクローズ
CBL_CREATE_FILE	バイトストリームファイルの作成
CBL_FLUSH_FILE	バイトストリームファイルバッファのディスクへのフラッシュ
CBL_FREE_RECORD_LOCK	ファイルに対するレコードロックの解除
CBL_GET_RECORD_LOCK	ファイルに対するレコードロックの取得
CBL_OPEN_FILE	バイトストリームファイルのオープン
CBL_READ_FILE	バイトストリームファイルの読込み
CBL_TEST_RECORD_LOCK	ファイルに対するレコードロックのテスト
CBL_WRITE_FILE	バイトストリームファイルの書出し

Micro Focusバイトストリームファイルルーチンを用いると、COBOLレコード定義に限定されずにデータファイルの読込み・書出しができます。

これらのルーチン全てについて、ルーチンの実行が成功すると、RETURN_CODEレジスタがゼロに設定されます。ルーチンの実行が失敗すると、RETURN_CODEレジスタには、失敗を示すファイル状態値が入ります。このファイル状態は常に標準ANSI74ファイル状態値です。ANSI74ファイル状態がエラーについて定義されてない場合は、標準Micro Focusエラー状態が返されます（9/nnn、ここで***nnn***はRTSエラー番号です）。

これを動作させるには、RETURN-CODEを使用し、RETURNING句を記述しないでください。バイトストリームルーチンの呼出し後、RETURN-CODEがゼロ以外の場合、そのRETURN-CODEをファイル状態として処理するPIC COMP-Xデータ項目に移動してください。

例:

```
01 file-stat     pic xx comp-x.
01   redefines file-stat.
   03 fs-byte-1  pic x.
   03 fs-byte-2  pic x comp-x.
  . . .
   call "CBL_xxx_FILE" using <parameters>
   if return-code not = 0
      move return-code to file-stat
         . . .
```

この時点で、fs-byte-1には"9"が入り、fs-byte-2にはランタイムシステムエラーが入ります。

8.3.3 文字セット変換ルーチン

本ルーチンは、OEM文字セットおよびANSI文字セットの間で文字の変換を行います。

PC_WIN_CHAR_TO_OEM	ANSI文字のバッファをOEM文字セットに変換
PC_WIN_OEM_TO_CHAR	OEM文字のバッファのANSI文字セットに変換

8.3.4 デバッグルーチン

CBL_DEBUGBREAK	デバッガの起動

8.3.5 表示属性ルーチン

本ルーチンにより一般属性が使用できます。画面属性は、環境およびターミナルタイプに関わらず移植可能です。

CBL_SCR_ALLOCATE_COLOR	RGB値用カラーの割当て
CBL_SCR_ALLOCATE_VC_COLOR	仮想カラーマップのカラーの割当て
CBL_SCR_CREATE_VC	仮想カラーマップの作成
CBL_SCR_DESTROY_VC	仮想カラーマップの削除
CBL_SCR_GET_ATTR_INFO	属性情報の取得
CBL_SCR_GET_ATTRIBUTES	属性値の取得
CBL_SCR_NAME_TO_RGB	カラー名をRGB値に変換
CBL_SCR_QUERY_COLORMAP	カラーマップエントリの問い合わせ
CBL_SCR_RESTORE_ATTRIBUTES	属性表のリカバリ
CBL_SCR_SAVE_ATTRIBUTES	属性表の保存
CBL_SCR_SET_ATTRIBUTES	属性値の設定

一般属性を使用しているプログラムから表示される画面は、Windows環境のIBM PCでもUNIX環境のシリアルターミナルでも、同じか非常に似ています。

一般属性は、シングルビットの一般属性テーブルの索引です。このテーブルには文字属性に関する情報（太字、下線など）、使用される各索引の前景／背景色、などが含まれています。この2つのカラー項目はカラーマップの索引です。つまり、使用中の画面固有の赤、緑、青（RGB）のテーブルです。

カラーマップは、読み込み専用、または読み書き可能にすることができます。読み込み専用のカラーマップでは、ある色から別の色へと動的にRGB値を変更することができず、システムが提供しているカラーマップの色のみ使用できます。読み書き可能なカラーマップでも、要求された色どおりにハードウェアが作成できるとは限りません。

テーブルパラメータを使用するルーチンには、テーブルの最初の要素を渡すか、動的に割り当てられたメモリにテーブルを作ってそのアドレスを渡します。

ルーチンは、次のようにRETURN-CODEを設定します。

0　成功
1　指定された色がカラーデータベースにありません。
2　カラーマップ索引が範囲外です。

3　属性索引が範囲外です。

4　完全に一致しません。

5　この環境では仮想カラーマップを使用できません。

6　仮想カラーマップを割り当てられません。

7　使用中の仮想カラーマップがありません。

ユーザ属性ルーチン

ユーザ属性は、各プログラムにおいて最初はオフになっています。一度オンにすると、画面にテキストを表示する方法をいくつか使用することができます。たとえば、DISPLAY...UPON CRT、画面入出力サブプログラムおよびACCEPTやDISPLAYなどの拡張構文はすべてこの使用可能になった属性を使用します。ANSI形式のDISPLAY文やACCEPT文（UPON CONSOLE指令を使用している）は、他のDISPLAYまたはACCEPT文が使用されている場合のみユーザ属性を使用します。

ユーザ属性は、オンにされると画面上のテキストを表示するときに使用され、ディスプレイ上の文字位置用に設定された画面属性と入れ替わります。

X"A7"ルーチンを使用すると、コードは他の環境に移植できなくなります。一般属性ルーチンを使用すれば、汎用的なカラーサポートを提供することができます。

WRITE to CONを使用する場合、出力は常に省略値のウィンドウに直接送られます。

ユーザ属性が使用できない場合は、その文字位置用に現在設定されている画面属性は変更されません。

DISPLAY...UPON CRT-UNDERを使用すると、ユーザ属性が設定されていれば強制的に強調表示します。DISPLAY SPACE UPON CRTは、画面をクリアして各画面位置用のユーザ属性を設定します。

8.3.6 閉止手続き

本ルーチンにより、アプリケーションが正常または異常終了したときにランタイムシステムによって実行される利用者独自のルーチンを登録することができます。

CBL_EXIT_PROC	閉止手続きの登録
CBL_GET_EXIT_INFO	閉止手続き呼び出し時の状況通知

8.3.7 エラー手続き

本ルーチンにより、アプリケーションが異常終了したときにランタイムシステムによって実行される利用者独自のルーチンを登録することができます。

CBL_ERROR_PROC	利用者エラー手続きの通知および削除

UNIX

本ルーチンはUNIX環境においてのみ使用可能で、DOS、OS/2またはWindows上では効果がありません。

8.3.8 ファイル名

これらのルーチンを用いると、ファイル名をその構成要素文字列に分割でき、また、文字列を結合してファイル名を形成することもできます。これらのルーチンを一緒に用いて、ファイル名の構成要素（拡張子など）を置き換えることができます。これらのルーチンは空で終わるファイル名およびスペースで終わるファイル名の両方を取り扱います。

ファイル名は装置、基本名および拡張子に分割されます。 例えば、

d : ¥dir1¥dir2¥file .dat

上記のようなファイル名では、装置はD : ¥dir1¥dir2、基本名はfile、そして拡張子はdatとなります。

ルーチンは長さが65,535文字までの文字列を扱えますが、利用者の環境またはランタイムシステムによりファイル名の最大長に制限が加わります。

基本名.extの最大長	256文字
ファイル名全体の最大長	261文字

CBL_JOIN_FILENAME	ファイル名の部分・部分を結合
CBL_SPLIT_FILENAME	ファイル名を部分・部分に分割

ファイル名ルーチンの例

```
*****************************************************************
*                                                               *
*             (C) Micro Focus Ltd. 1991                         *
*                                                               *
*                  SPLTJOIN.CBL                                 *
*                                                               *
*    This program demonstrates the use of the routines that     *
*    enable you to separate a filename into its component       *
*    strings (CBL_SPLIT_FILENAME), and to join strings          *
*    together to form a filename (CBL_JOIN_FILENAME).           *
*                                                               *
*****************************************************************

 working-storage section.

 78 environ         value "dos".

 01 split-buffer           pic x(65).
 01 split-params.
    03  param-length       pic xx comp-x   value 24.
    03  splitjoin-flg1     pic x  comp-x   value 0.
    03  splitjoin-flg2     pic x  comp-x.
    03  path-strt         pic xx comp-x.
    03  path-len          pic xx comp-x.
    03  basename-strt      pic xx comp-x.
    03  basename-len       pic xx comp-x.
    03  extension-strt     pic xx comp-x.
    03  extension-len      pic xx comp-x.
    03  total-length       pic xx comp-x.
    03  split-buf-len      pic xx comp-x   value 65.
    03  join-buf-len       pic xx comp-x   value 65.
```

```
    03  first-path-len     pic xx comp-x.
 01 join-buffer            pic x(65).
 01 path-buffer            pic x(65).
 01 basename-buffer         pic x(65).
 01 extension-buffer        pic x(3) value "cbl".

 procedure division.

* Set up lengths
    move 65 to split-buf-len
               join-buf-len

* Set flag for space-terminated, fold to upper
    move 1 to splitjoin-flg1

$if environ = "unix"
    move "/dir/file.ext" to split-buffer
$else
    move "a:¥dir¥file.ext" to split-buffer
$end
    move 1 to splitjoin-flg1
    call "CBL_SPLIT_FILENAME" using split-params
                                  split-buffer

* This sets up most of the parameters you need for a join

* The join below replaces the original extension in split-buffer
* with the extension in extension-buffer, and puts the result in
* join-buffer.
    move 1 to extension-strt
    move 3 to extension-len
    call "CBL_JOIN_FILENAME" using split-params
                                join-buffer
                                split-buffer
                                split-buffer
                                extension-buffer

$if environ = "unix"
    if join-buffer = "/DIR/FILE.CBL" then
$else
    if join-buffer = "A:¥DIR¥FILE.CBL" then
$end
       display "first test passed"
    else
       display "first test failed"
    end-if

* It is harder to set up a join without doing a split first,
* but this is what you would need to do.
    move 1 to path-strt
              basename-strt
              extension-strt

    move length of path-buffer to path-len
    move length of basename-buffer to basename-len
    move length of extension-buffer to extension-len
    move length of join-buffer to join-buf-len

    move 0 to splitjoin-flg1
    move 24 to param-length
```

```
$if environ = "unix"
    move "/path" to pat-buffer
$else
    move "c:¥path" to pat-buffer
$end
    move "basename" to bas-buffer
    move "ext" to extension-buffer

    call "CBL_JOIN_FILENAME" using split-params
                                join-buffer
                                path-buffer
                                basename-buffer
                                extension-buffer

$if environ = "unix"
    if join-buffer = "/path/basename.ext" then
$else
    if join-buffer = "c:¥path¥basename.ext" then
$end
        display "second test passed"
    else
        display "second test failed"
    end-if
  stop run.
```

8.3.9 ファイル

CBL_CHANGE_DIR	現行ディレクトリを変更
CBL_CHECK_FILE_EXIST	ファイルが存在するかどうかチェック
CBL_COPY_FILE	ファイルのコピー
CBL_CREATE_DIR	ディレクトリを作成
CBL_DELETE_DIR	ディレクトリの削除
CBL_DELETE_FILE	ファイルの削除
CBL_LOCATE_FILE	ファイルの位置探査/パスの展開
CBL_FILENAME_CONVERT	スペースおよび空で終わるファイル名形式間の変換
CBL_GET_CURRENT_DIR	現行ディレクトリをリターン
CBL_FILENAME_MAX_LENGTH	オペレーティングシステムが扱えるファイル名の最大長をリターン
CBL_RENAME_FILE	ファイルの再命名
PC_FIND_DRIVES	有効ドライブの検索
PC_READ_DRIVE	現行ドライブの読込み
PC_SET_DRIVE	現行ドライブの設定

8.3.10 キーボード

CBL_GET_KBD_STATUS	キーボードでの文字の試験
CBL_READ_KBD_CHAR	キーボードからの文字の読込み（表示なし）

CBL_キーボードルーチンをx"AF"キーボードルーチンと共に使用することはできません。

キーボードルーチンの例

次のコードは、CBL_READ_KBD_CHARと一緒にキー呼出し関数を使用する方法の一例です。

```
 procedure division.
     ...
* Set up function key list
     call x"B0" using B0-function
                      B0-parameter-block
     ...
* Read a character from the keyboard
     call "CBL_READ_KBD_CHAR" using char
     if char = x"0D"

* A key defined in the function key table was pressed
         evaluate B0-return-byte
           when 1
            ...
           when 2
            ...
         end-evaluate
     else
         if char = x"00"
* A function key not defined in the table was pressed
            call "CBL_READ_KBD_CHAR" using char
* char contains the key's scan code
            ...
         else
* char contains a character
            ...
         end-if
     end-if.
```

8.3.11 論理演算子

CBL_AND	論理積（AND）
CBL_EQ	論理等価（EQ）
CBL_IMP	論理暗示（IMP）
CBL_NOT	論理否定（NOT）
CBL_OR	論理和（OR）
CBL_XOR	排他的論理和（XOR）

これらのルーチンは論理演算を実行します。CBL_NOTを除く全ての演算は2つの作用対象を有します。

長さが定数として指定され、RETURNING句がない場合、元のコードは内コードを生成するために最適化されます。

作用対象が2個のルーチンでは、2つの作用対象、つまり源と目標を入れ換えても、CBL_IMP以外は結果は変わりません。しかし、結果は常に第2作用対象、つまり目標作用対象に格納されます。

長さがどちらかのデータ項目より長いと、そのデータ項目に続くバイトが指定された長さまで使用されます。

パラメータ長さは次の構文で置き換え可能です。

length of 源

または、

length of 目標

データ項目の全バイトを使用するとします。

論理ANDおよびOR演算は、VALUE句を使用して実行することができます。

RETURN-CODEはこれらのルーチンによって影響を受けません。

8.3.12 メモリ割付け/割付け解除

CBL_ALLOC_MEM	動的メモリ割付け
CBL_ALLOC_DYN_MEM	スレッド格納またはスレッド固有のヒープメモリの割付け
CBL_ALLOC_SHMEM	スレッド格納またはスレッド固有のヒープメモリの割付け
CBL_ALLOC_THREAD_MEM	スレッド格納またはスレッド固有のヒープメモリの割付け
CBL_FREE_DYN_MEM	CBL_ALLOC_DYN_MEMによる割付けメモリの解放
CBL_FREE_MEM	CBL_ALLOC_MEMによる割付けメモリの解放
CBL_FREE_SHMEM	CBL_ALLOC_SHMEMによる割付けメモリの解放
CBL_FREE_THREAD_MEM	CBL_ALLOC_THREAD_MEMによる割付けメモリの解放

8.3.13 マウス

CBL_GET_MOUSE_MASK	マウス事象マスクの獲得
CBL_GET_MOUSE_POSITION	マウス画面座標の獲得
CBL_GET_MOUSE_STATUS	待ち行列内事象数の獲得
CBL_HIDE_MOUSE	マウスポインタを隠す
CBL_INIT_MOUSE	マウス支援を初期化
CBL_READ_MOUSE_EVENT	マウス事象待ち行列の読込み
CBL_SET_MOUSE_MASK	マウス事象マスクの設定
CBL_SHOW_MOUSE	マウスポインタの描画
CBL_TERM_MOUSE	マウス支援の終了

CBL_マウスルーチンをx"AF"マウスルーチンと共に使用することはできません。

ルーチンの使用

ユーザーが選択項目のリストから選択したり、あるいは、画面上で対象を移動させる必要のあるアプリケーションには、マウスが有用です。

これらのルーチンを使用するには、ご使用のシステムにマウスが付属されていることを確かめる必要があります。

マウスポインタのある画面の領域に操作する任意のANSI ACCEPTまたはDISPLAY文を実行している間は、マウスを隠ぺいする必要があります。

ルーチンの記述内で参照されている属性は画面属性であり、利用者属性ではありません。画面の左上の角が行0、カラム0になります。

マウス事象

マウスを移動したり、マウスボタンを押したりあるいは離したりすると、マウスハードウェアは割込みを発生します。マウスドライバに制御が渡り、設定してあるマスクに従ってその割込みを待ち行列にセーブするか、あるいは、無視します。アプリケーションがその事象を読み込む前に後続の割込みが発生した場合、これは事象が失われるのを防ぎます。マウスルーチンを用いて、事象待ち行列を読込み、その待ち行列に事象がいくつあるか知ることができます。

事象が生成されると、その記述が事象データと呼ばれるデータ構造体に格納されます。マスクが許せば（下記参照）、これは待ち行列に追加されます。「事象データ」のレイアウトは次の通りです。

事象種類　　PIC X（2）COMP-X.
事象時間　　PIC X（4）COMP-X.
事象行　　PIC X（2）COMP-X.
事象カラム　　PIC X（2）COMP-X.

ここで、

事象種類　　発生した動作（つまり、状態の変化）:
ビット7-4設定=予約済み
ビット3設定=ボタン3が押された
ビット2設定=ボタン2が押された
ビット1設定=ボタン1が押された
ビット0設定=マウスが移動した
ボタンを離す動作は、そのボタンに対応するビットが1から0に変化することにより示されます。例えば、マウスが移動して、同時にボタン1が押されると、「事象種類」には3が入ります。
事象時間　　事象が発生したときと任意の固定開始時間の間の時間差
事象行　　事象が発生したときのマウスの行位置
事象カラム　　事象が発生したときのマウスのカラム位置

事象マスク

事象マスクは、プログラマが提供し、どのような種類の事象を待ち行列に入れるべきか、また、無視すべきかを、システムに告げます。事象マスクは「事象種類」と同じ構造をしています。事象が待ち行列に登録されるのは、その事象が対応するマスクビットがオンであるときに、あるいは、マスクビットがオンである別の状態がオンであるときに発生する場合に限られます。事象データが待ち行列に登録されると、各状態のビットが正しく設定されます。つまり、マスクがそれらの事象をマスクしません。

例えば、たとえ「マウスの移動」用マスクビットがオフとなっていても、あるいは、オペレータがマウスボタンを押したままで、そのボタン用のマスクビットがオフであったとしても、マウスを移動しているオペレータは事象を発生させます。

8.3.14 マルチスレッド

名前による呼出しライブラリルーチンは、アプリケーションでマルチスレッドを実行したり制御したりできるようにプログラムインタフェースを提供します。このインタフェースは、次の3つのカテゴリーに分けられます。

- スレッドの制御
- スレッドの同期化
- スレッド固有データの処理

スレッドの制御

CBL_THREAD_CREATE	名前付き入口点からスレッドを作成
CBL_THREAD_CREATE_P	手続きポインタからスレッドを作成
CBL_THREAD_DETACH	未解放スレッドを解放
CBL_THREAD_EXIT	現行スレッドを終了
CBL_THREAD_IDDATA_ALLOC	IDデータ領域をスレッドに割り付け
CBL_THREAD_IDDATA_GET	IDデータ領域へのポインタを取得
CBL_THREAD_KILL	スレッドを中断
CBL_THREAD_LIST_END	スレッドリスト処理を終了
CBL_THREAD_LIST_NEXT	スレッドリストの次のスレッドを取得
CBL_THREAD_LIST_START	スレッドリスト処理を開始
CBL_THREAD_LOCK	スレッド処理ルーチンをロック
CBL_THREAD_PROG_LOCK	スレッドをロック
CBL_THREAD_PROG_UNLOCK	スレッドのロックを解除
CBL_THREAD_RESUME	サスペンドされたスレッドの再開
CBL_THREAD_SUSPEND	スレッドをサスペンド
CBL_THREAD_SELF	スレッドの識別子を格納
CBL_THREAD_SLEEP	スレッドによるCPU制御を停止
CBL_THREAD_UNLOCK	スレッド処理ルーチンのロックを解除
CBL_THREAD_WAIT	未解放スレッドの終了を待機
CBL_THREAD_YIELD	スレッドのタイムスライスを放棄

スレッド制御ルーチンのRETURN-CODE値

次の値は、スレッド制御ルーチンのRETURN-CODE値です。

0　　エラーはありませんでした。

1000　　メモリ割り付けエラーです。

1001　　スレッド識別子が不正です。このスレッドでは操作は無効です。指定されたスレッド識別子は有効でもすでに終了していたか、解放された可能性があります。

1002　　スレッド識別子が解放され、終了しました。このスレッドでは操作は無効です。

1003　　スレッドが解放されました。スレッドがすでに解放されているため、CBL_THREAD_DETACHはスレッドを解放できません。また、CBL_THREAD_WAITはスレッドを待つことができません。

1004　　スレッドが多すぎます。システムにおけるスレッドの数が多すぎるため、CBL_THREAD_CREATEはこれ以上スレッドを作成することができません。

1005　　スタックサイズが不正です。オペレーティングシステムがこのスタックサイズの使用を許可していないため、CBL_THREAD_CREATEはこのスレッドを作成することができません。

1006　　不正な操作です。CBL_THREAD_CREATEは、システムが初期化されていない場合はスレッドを作成することができません。CBL_THREAD_CREATEで作成されていないスレッドに対して、不正なスレッド制御ルーチンを呼び出そうとしました。CBL_THREAD_PROG_LOCKは、COBOLプログラムから直接的または間接的に呼び出さなければなりません。

1007　　システムエラーです。予期しないシステムエラーにより、本関数が失敗しました。

1008　　スレッドは本ランタイムシステムでサポートされていません。スレッドアプリケーションが、スレッドをサポートしていないランタイムシステムで使用しようとしました。

1009　　パラメータが不正です。有効範囲外、またはこのルーチン用ではないパラメータが検出されました。

1011　　プログラムまたは入口点の名前が見つかりません。CBL_THREAD_CREATEに指定されたプログラムまたは入口点の名前が見つかりませんでした。または、その読み込み中にエラーが発生しました。

スレッドの同期化

事象:

CBL_EVENT_CLEAR　　指定された事象を消去
CBL_EVENT_CLOSE　　与えられた事象ハンドルをクローズ
CBL_EVENT_OPEN_INTRA　　プロセス内事象を作成
CBL_EVENT_POST　　指定した事象を通知
CBL_EVENT_WAIT　　事象の通知を待機

モニター:

CBL_MONITOR_BROWSE　　スレッドの走査機能を取得
CBL_MONITOR_BROWSE_TO_READ　　走査機能を読み込み機能に変換
CBL_MONITOR_BROWSE_TO_WRITE　　走査機能を書き込み機能に変換
CBL_MONITOR_CLOSE　　与えられたモニターハンドルをクローズ
CBL_MONITOR_OPEN_INTRA　　モニターを作成
CBL_MONITOR_READ　　読み込み機能の取得
CBL_MONITOR_RELEASE　　モニターのロックを解除
CBL_MONITOR_UNBROWSE　　走査機能を解除

CBL_MONITOR_UNREAD　　読み込み機能を解除

CBL_MONITOR_UNWRITE　　書き込み機能を解除

CBL_MONITOR_WRITE　　書き込み機能の取得

CBL_MONITOR_WRITE_TO_BROWSE　　モニター書き込み機能を走査機能に変換

MUTEX:

CBL_MUTEX_ACQUIRE　　MUTEXを取得

CBL_MUTEX_CLOSE　　指定されたMUTEXをクローズ

CBL_MUTEX_OPEN_INTRA　　プロセス間MUTEXを作成

CBL_MUTEX_RELEASE　　指定されたMUTEXを解除

セマフォ:

CBL_SEMAPHORE_ACQUIRE　　セマフォのリソースの1つを取得

CBL_SEMAPHORE_CLOSE　　セマフォをクローズ

CBL_SEMAPHORE_OPEN_INTRA　　プロセス間セマフォを作成

CBL_SEMAPHORE_RELEASE　　セマフォのリソースの1つを解放

MUTEX、セマフォ、モニター、および事象は、スレッドを同期化する場合に使用できます。同期化オブジェクトには名前がついています。その名前は、システムからは見えますが、作成プロセスでは見えません。

次に、MUTEX、セマフォ、モニター、および事象の一般注意事項を説明します。

モニター - 一般注意事項

モニターは、読み込み、走査、および書込み操作に対してロックをかけられる同期化オブジェクトです。このタイプの同期化は、一般的に、異なるスレッドが読み書きを行うデータ構造体を保護するために使用されます。

モニターは、プロセス間同期化オブジェクトとしてのみ使用できます。

ライブラリルーチンを使用して作成または開かれたモニターは、ランタイムシステムが終了すると自動的に閉じられます（たとえば、STOP RUNによって）。

書き込みロックは、他の読み込み、走査、および書き込みロックを許可しません。一般的に、書き込みロックはデータ構造体に書込み操作を行う場合に使用されます。

走査ロックは、他の書き込みおよび走査ロックを許可しません。しかし、読み込みロックは同時に行えます。一般的に、走査ロックは、データ構造体を読み込むときに、その読み込みの結果によってデータ構造体に書き込みを行う目的で使用されます。このために、他の書き込みを許さずに走査ロックを書き込みロックに変換する関数が提供されています。

読み込みロックは、書き込みロックのみ許可しません。一般的に、読み込みロックは、データ構造体に書き込むのではなくただそれを読み込むだけの目的で使用されます。

ある特定のモニターに対して読み込みロックをかけたスレッドは、同じモニターに対してさらに読み込みロックをかけることができます。ただし、読み込みロックがかけられると、外部からの走査または書き込みロックが存在してそこに読み込みロックが入れ子になっていない限り、スレッドが走査または書き込みロックを要求するようなことはできません。この規則を破ろうとしても、ランタイムシステムエラーになるだけです。モニターに対して優先順位の変更が要求された場合は、入れ子になった読み込みロックが書き込み完了の待機をブロックする可能性があります。しかし、最初の読み込みロックがまだアクティブなので、書き込みのアクセスは許可されません。これを、シングル・スレッド・デッドロックといいます。入れ子になった読み込みロックを取得したい場合は、読み込みの優先権を使用するか、最高レベルの書き込みまたは走査ロックを取得してください。

ある特定のモニターに対して走査ロックをかけたスレッドは、同じモニターに対してさらに走査ロックをかけることができます。また、書き込みロックもかけることができます。さらに、最初の走査ロックが書き込みロックに変換される前、および書き込みロックが要求される前に読み込みロックが解除された場合に限って、読み込みロックもかけることができます。

ある特定のモニターに対して書き込みロックをかけたスレッドは、同じモニターに対してさらに書き込み、走査、または読み込みロックをかけることができます。

セマフォ - 一般注意事項

セマフォは、同一プロセス内でスレッドの同期化を行うために使用できます。

セマフォは、それに関連するカウントを持つ同期化オブジェクトです。そして一般的には、ある限られたリソースを持っています。スレッドは、取得呼出しを使用してこのリソースの一つを取得します。リソースが使用可能でない場合は、呼出しはブロックするか「wait」オプションの指定によってはエラーで戻ります。リソースが使用可能な場合（カウントがゼロ以外）、カウントは1つ減らされて呼出しがエラーなしで戻ります。

スレッドは、解放呼出しを使用してリソースを解放します。これによりカウントが1つ増えます。リソースが使用可能でないために（カウントはゼロ）他のスレッドがブロックしている場合、そのスレッドの一つが解放されます。その他のスレッドは、ブロックしたままです。これは、最初のスレッドの解放処理中に、カウントがゼロに戻されるからです。

MUTEX - 一般注意事項

MUTEXは、同一プロセス内でスレッドの同期化を行うために使用できます。

MUTEXは、オペレーティングシステムを直接呼び出さないと実行されません。したがって、モニターよりも機能性のレベルが低いといえます。

MUTEXは、ゼロまたは1つのスレッドによって一度所有される同期化オブジェクトです。あるスレッドがMUTEXを取得し、それを他のスレッドが要求すると、2番目のスレッドは「wait」オプションに従ってMUTEXを取得できずに戻るか、最初のスレッドがそのMUTEXを解放するまでブロックします。いくつかのスレッドがMUTEXを待機してブロックしている場合、その中の1つのスレッドだけがブロックを解除されます。その他のスレッドは待機を続けます。

事象 - 一般注意事項

事象は、同一プロセス内でスレッドの同期化のために使用できます。

事象は、オペレーティングシステムを直接呼び出さないと実行されません。したがって、モニターよりも機能性のレベルが低いといえます。

事象は、異なるスレッドによって通知または消去される同期化オブジェクトです。スレッドは事象が通知されるようになるのを待ちます。それを待機するスレッドがいくつかある場合、スレッドはすべて待機しつづけることが許されます。

同期ルーチンのRETURN-CODE値

次の値は、スレッド同期ルーチンのRETURN-CODE値です。

0　　エラーはありませんでした。

1000　　メモリ割り付けエラーです。

1001　　ハンドルが不正です。同期化オブジェクトに対して無効なハンドルです。ハンドルでは操作は無効です。指定されたハンドルは有効でもすでに閉じていたか、メモリが再利用した可能性があります。

1002　　ハンドルが閉じられました。このハンドルでは操作は無効です。ハンドルは有効でしたが閉じられていました。

1007　　システムエラーです。最後の操作により低レベルのオブジェクト処理でシステムエラーが発生しました。

1008　　スレッドは本ランタイムシステムでサポートされていません。スレッドアプリケーションが、スレッドをサポートしていないランタイムシステムで使用しようとしました。

1009　　マルチスレッドライブラリルーチンに渡されたパラメータが不正です。有効範囲外、またはこのルーチン用ではないパラメータが検出されました。

1010　　使用可能なリソースがありません。カウントがゼロでユーザが待ちなしを要求したため、セマフォまたはMUTEXを取得できません。事象は通知されず、ユーザはCBL_EVENT_WAIT呼出しを待機しません。

スレッド固有データの処理

スレッドローカルデータを処理する最も簡単な方法は、THREAD-LOCAL-STORAGE SECTIONをプログラムで使用することです。ここでは、THREAD-LOCAL-STORAGEについての知識があることを前提に説明します。

ライブラリルーチンは、スレッド固有のデータを処理する方法を制御できるように提供されています。このルーチンにより、プログラムに次のような処理を行わせることが可能です。

- ダイナミックヒープからスレッドローカルデータの割り付け

- 初期化されたスレッド格納ハンドルからスレッドローカルデータへのアクセス

CBL_TSTORE_CLOSE　　スレッド格納領域の割り付けを解除

CBL_TSTORE_CREATE　　スレッド格納領域を作成

CBL_TSTORE_GET　　スレッド格納領域へのポインタを取得

ダイナミックヒープからのスレッドローカルデータの割り付け

CBL_ALLOC_MEMまたはCBL_ALLOC_THREAD_MEMによりダイナミックヒープからスレッドローカルデータを割り付けることができます。この割り付けの利点は、サイズの異なる複数のメモリブロックをスレッドまたはプログラムに割り付けそれと関連付けることがでるということです。

CBL_ALLOC_MEMは、入力パラメータの設定によって、割り付けられたヒープデータをスレッドおよび呼出しプログラムと関連付けることができます。CBL_ALLOC_THREAD_MEMは、常に、割り付けられたヒープデータをスレッドおよびオプションで呼出しプログラムと関連付けます。

これらのルーチンによって割り付けられたメモリは、スレッドが終了すると自動的に解放されます

これらのルーチンによって返されるスレッドローカル格納領域へのポインタはTHREAD-LOCAL-STORAGEが定義したポインタに格納されるため、その値をスレッドローカル格納領域を割り付けたプログラムの後に続く呼出しで取り出すことができる、ということに注意してください。返されたポインタをWORKING-STORAGEが定義したポインタに保存してその値を使用すると、スレッドローカルヒープ領域はスレッド間で共有され、したがってこれは悪いプログラミング習慣です。

初期化されたスレッド格納ハンドルからのスレッドローカルデータへのアクセス

これらのルーチンは、プログラムを初期化されたスレッド格納ハンドルからスレッドローカルデータにアクセスできるようにするために提供されています。このハンドルは、CBL_TSTORE_CREATEルーチンによって返されます。メモリはCBL_TSTORE?GETによりある特定のハンドルを使用して返されます。メモリのサイズは固定で、したがってそのスレッド内で実行されるすべてのCBL_TSTORE_GETを通して同じサイズです。これらのルーチンは本質的にまったく別のスレッドローカル格納領域を各初期化されたハンドルに割り付けます。

スレッド固有データ処理ルーチンのRETURN-CODE値

次の値は、スレッド同期化ルーチンのRETURN-CODE値です。

0　　エラーはありませんでした。

1000　　メモリ割り付けエラーです。

1001　　スレッド格納ハンドルが不正です。このスレッド格納ハンドルでは操作は無効です。指定されたハンドルは有効でもすでに閉じていたか、メモリが再利用した可能性があります。

1002　　スレッド格納ハンドルが不正です。ハンドルは閉じられていたので、無効です。

1009　　スレッド固有データ処理ライブラリルーチンに渡されたパラメータが不正です。有効範囲外、またはこのルーチン用ではないパラメータが検出されました。

1010　　リソースがビジーです。同期化オブジェクトがスレッド終了時にまだロックされたいました。また、スレッド作成時に、ユーザがこのような状況が発生した場合はエラーを通知するように要求していました。

8.3.15 NLSメッセージファイル処理

CBL_NLS_CLOSE_MSG_FILE　　メッセージファイルをクローズ
CBL_NLS_COMPARE　　2つの文字列を比較
CBL_NLS_INFO　　国別言語情報を取得または設定
CBL_NLS_OPEN_MSG_FILE　　メッセージファイルをオープン
CBL_NLS_READ_MSG　　メッセージファイルからメッセージの読み込み

8.3.16 オペレーティングシステム情報

CBL_GET_OS_INFO　　オペレーティングシステム環境情報を獲得

8.3.17 移植

本ルーチンは、既存のアプリケーションを他の環境に移植できるようにするために提供されています。代替機構が提供されているような新しいアプリケーションでは、本ルーチンを使用しないでください。

CBL_SCR_SET_PC_ATTRIBUTES　　IBM-PC属性パレットをセットアップ

8.3.18 プリンタ

プリンタ処理ルーチンにより、標準COBOL構文よりもより広範囲にわたって印刷を制御できます。

PC_PRINT_FILE　　ファイルを印刷
PC_PRINTER_CLOSE　　プリンタチャネルをクローズ
PC_PRINTER_CONTROL　　プリンタコマンドをプリンタに送信
PC_PRINTER_FREE_BMP　　メモリからビットマップを解放
PC_PRINTER_GET_COLOR　　プリンタカラーを設定
PC_PRINTER_GET_FONT　　プリンタフォントを設定
PC_PRINTER_INFO　　プリンタ情報の取得
PC_PRINTER_LOAD_BMP　　メモリにビットマップをロード
PC_PRINTER_OPEN　　プリンタチャネルをオープン
PC_PRINTER_SET_COLOR　　プリンタカラーを設定
PC_PRINTER_SET_DEFAULT　　プロセス共通のデフォルトプリンタを設定
PC_PRINTER_SET_FONT　　プリンタフォントを設定
PC_PRINTER_WRITE　　プリンタへのテキスト書き込み
PC_PRINTER_WRITE_BMP　　プリンタへのビットマップ書き込み

プリンタ処理ルーチンのRETURN-CODE

次の値は、プリンタ処理ルーチンのRETURN-CODE値です。

0	成功
1	プリンタデバイスを開けません。
2	不正なプリンタ制御コードが指定されました。
3	指定されたハンドルに関連するプリンタデバイスがありません。
4	印刷中にメモリ不足になりました。
5	ファイルのオープンに失敗しました。
6	ファイルのスプーリング中にディスクフルになりました。
7	印刷ジョブが中断されました。プリントスプーラに送られたジョブはありません。
8	プリンタ情報構造体が不正に構成されました。
9	デフォルトプリンタが見つかりません。
10	エラーがダイアログを表示しようとしています。
11	書き込みエラーです。
12	このプリンタで使用できるフォントが見つかりません。
13	要求されたフォントは存在しません。
14	ユーザがプリントジョブを中断しました。
15	予約済み
16	予約済み
17	予約済み
18	ビットマップのロードに失敗しました。
19	不正なビットマップIDです。
20	ビットマップの解放に失敗しました。
21	ビットマップの印刷に失敗しました。
22	不正なパラメータです。
23	内部エラーです。

8.3.19 プログラム取り消し

プログラム取り消しルーチンは、COBOLプログラムの取り消しを制御できるようにします。

CBL_CANCEL_PROC	プログラム取り消しルーチンを制御

8.3.20 プログラム情報

プログラム情報ルーチンは、プログラムに関する情報を取得できるようにします。

CBL_GET_PROGRAM_INFO	指定されたプログラムまたは現行呼出しスタック内のプログラムの情報をリターン

8.3.21 実行単位処理

複数の実行単位ルーチンにより、システムの同時性サポートを利用することができます。

これらのルーチンは、.intファイルおよび.gntファイルでのみ動作します。.exeファイルまたは.dllファイルでは動作しません。

CBL_ABORT_RUN_UNIT	既存のスレッド状態に関わらず現行の実行単位を中断
CBL_EXEC_RUN_UNIT	実行単位を作成
CBL_GET_SHMEM_PTR	名前付き値の読み込み
CBL_PUT_SHMEM_PTR	名前付き値の作成/更新
CBL_YIELD_RUN_UNIT	現行実行単位を譲渡

8.3.22 画面

CBL_CLEAR_SCR	画面を消去
CBL_GET_CSR_POS	カーソル位置の獲得
CBL_GET_SCR_GRAPHICS	図形文字の獲得
CBL_GET_SCR_LINE_DRAW	線画文字の獲得
CBL_GET_SCR_SIZE	画面サイズの獲得
CBL_READ_SCR_ATTRS	属性列の読込み
CBL_READ_SCR_CHARS	文字列の読込み
CBL_READ_SCR_CHATTRS	文字列および属性列の読込み
CBL_SET_CSR_POS	カーソル位置の設定
CBL_SWAP_SCR_CHATTRS	文字および属性の交換
CBL_WRITE_SCR_ATTRS	属性列の書出し
CBL_WRITE_SCR_CHARS	文字列の書出し
CBL_WRITE_SCR_CHARS_ATTR	属性付き文字列の書出し
CBL_WRITE_SCR_CHATTRS	文字列および属性列の書出し
CBL_WRITE_SCR_N_ATTR	属性書出しの繰返し
CBL_WRITE_SCR_N_CHAR	文字書出しの繰返し
CBL_WRITE_SCR_N_CHATTR	文字および属性書出しの繰返し
CBL_WRITE_SCR_TTY	文字をTTY様式で書出し

これらルーチンのいくつかは「画面位置」パラメータを指定します。本COBOLシステムでは、画面の左上が行0、カラム0となります。例えば、行5、カラム8で始まる属性を変更したい場合、行番号=4とカラム番号=7とを指定します。

CBL_GET_SCRを用いれば、原始プログラムを変更しなくても、サポートされた環境で使用可能な図形文字の表現を可能な限り最善のものにするため、線画文字システムを使用できます。

8.3.23 状態保守

MF_CLIENT_STATE_ALLOCATE	クライアント識別子を割り当て
MF_CLIENT_STATE_DELETE	クライアント情報を削除
MF_CLIENT_STATE_EXPIRY	cookie期限切れフィールドで使用するデータ文字列をリターン
MF_CLIENT_STATE_FILE	状態情報を格納するファイルを指定

MF_CLIENT_STATE_PURGE　　条件に合うクライアント情報を削除
MF_CLIENT_STATE_RESTORE　　事前に保存または割り当てられたレコードをリカバリ
MF_CLIENT_STATE_SAVE　　状態ファイルの情報を更新

状態保守ルーチンの使用に関する詳細は、「インターネットアプリケーション（Pipubb03.htm）」を参照してください。

状態保守ルーチンの状態

0　　成功
1　　失敗
2　　クライアントIDが重複しています
3　　次の可能性があります。
クライアントIDが見つかりません（SAVEまたはRESTORE）
一致するキーがないレコードに対してSAVE/RESTORE/DELETEを実行しようとしました。
他の内部エラーが発生しました。（不正なCALLの順番による）
レコードの割当てに問題があります。

8.3.24 テキスト変換

CBL_TOLOWER　　文字列を小文字に変換
CBL_TOUPPER　　文字列を大文字に変換

8.3.25 仮想ヒープ

ヒープは、使用可能なメモリにバッファされたバイトストリームファイルです。

CBL_CLOSE_VFILE　　動的ストリームのクローズ
CBL_OPEN_VFILE　　動的ストリームのオープン
CBL_READ_VFILE　　動的ストリームの読込み
CBL_WRITE_VFILE　　動的ストリームの書出し

各ヒープはヒープが作成されるときに指定された状態語と関連づけられており、ヒープに対する処理が失敗するとこの状態語が書き出されます。このようにして、各ヒープは特定のプログラム、つまり、ヒープの状態語を持っているプログラムに付加されており、そのプログラムが取り消されると、自動的に割付けを解除されます（CBL_CLOSE_VFILEへの呼出しによって、そのヒープが既に明示的に割付け解除されていない限り）。

ヒープは、ヒープ識別語を用いて識別されます。ヒープ識別語はプログラム間で受け渡しされ、任意のヒープが任意のプログラムにより読込み、または、書出しされます。しかし、障害を調べるために、プログラムは関連する状態語にアクセス（例えば、ポインタ変数を用いて、あるいは、通常の連絡節の写像を介して）する必要があります。これに代わる方法として、RETURN-CODEレジスタを調べて一般的なヒープ機能障害を検出できます、あるいは、ON OVERFLOW/EXCEPTION構文をCALL文に使用して、エラーを検出することもできます。これら2つの場合、ヒープ状態語を調べることによってエラーを特定できます。

各ヒープは、必要であれば、「プログラム名 .V*nn*」という名の別のディスクファイルにページングされます。ここで、「プログラム名」はそのヒープが付加されているプログラムのルート名で、「*nn*」は1～99の整数です。いったん「*nn*」が99に達すると、「*V*」は上書きされます。最大許容拡張数は384です。各プログラムは384局所ヒープに制限されています。

読込み/書出しが成功すると、戻り値はゼロとなります。失敗すると、戻り値はゼロ以外となり、ヒープ状態バイトは9となり、その詳細は第2バイトに入ります。

CBL_READ_VFILEとCBL_WRITE_VFILEの両ルーチン用バッファは、データまたは連絡節内の任意の場所に存在できます。このバッファはまたCBL_ALLOC_MEMルーチンを介して動的に割り付けできます。

各ヒープは必要に応じページングされます。ヒープは、ディスク上のsourcename.Vnnという名前のファイルにページングされます。sourcenameはヒープがアタッチされるプログラムの基本名、nnは1から99までの整数です。nnが99ｎ達すると、Vが上書きされます。この拡張子の最大は384です。

ローカルヒープの数は構成によって変えられ、65535まで可能です。

仮想ヒープに関する一般注意事項

COBOLレコード内のバッファを宣言するために、部分参照を使用できます。長さが固定として与えられていれば、これは効率よくコンパイルされます。長さは呼出しインタフェースにより無視されるので、長さを1として与えることができます。

8.3.26 Windows

次のルーチンは、Windows APIアプリケーションを作成するときに使用します。

PC_WIN_INIT	起動値を取得します。

8.4 ライブラリルーチンのアルファベット順リスト

CBL_ABORT_RUN_UNIT	既存のスレッド状態に関わらず現行の実行単位を中断
CBL_ALLOC_DYN_MEM	スレッド格納またはスレッド固有のヒープメモリの割付け
CBL_ALLOC_MEM	動的メモリ割付け
CBL_ALLOC_SHMEM	スレッド格納またはスレッド固有のヒープメモリの割付け
CBL_ALLOC_THREAD_MEM	スレッド格納またはスレッド固有のヒープメモリの割付け
CBL_AND	論理積（AND）
CBL_CANCEL_PROC	プログラム取り消しルーチンを制御
CBL_CHANGE_DIR	現行ディレクトリを変更
CBL_CHECK_FILE_EXIST	ファイルが存在するかどうかチェック
CBL_CLEAR_SCR	画面を消去
CBL_CLOSE_FILE	ファイルのクローズ
CBL_CLOSE_VFILE	動的ストリームのクローズ
CBL_COPY_FILE	ファイルのコピー

CBL_CREATE_DIR　　ディレクトリを作成
CBL_CREATE_FILE　　バイトストリームファイルの作成
CBL_DEBUGBREAK　　デバッガの起動
CBL_DELETE_DIR　　ディレクトリの削除
CBL_DELETE_FILE　　ファイルの削除
CBL_EQ　　論理等価（EQ）
CBL_ERROR_PROC　　利用者エラー手続きの通知および削除
CBL_EVENT_CLEAR　　指定された事象を消去
CBL_EVENT_CLOSE　　与えられた事象ハンドルをクローズ
CBL_EVENT_OPEN_INTRA　　プロセス内事象を作成
CBL_EVENT_POST　　指定した事象を通知
CBL_EVENT_WAIT　　事象の通知を待機
CBL_EXEC_RUN_UNIT　　実行単位を作成
CBL_EXIT_PROC　　閉止手続きの登録
CBL_FILENAME_CONVERT　　スペースおよび空で終わるファイル名形式間の変換
CBL_FILENAME_MAX_LENGTH　　オペレーティングシステムが扱えるファイル名の最大長をリターン
CBL_FLUSH_FILE　　バイトストリームファイルバッファのディスクへのフラッシュ
CBL_FREE_DYN_MEM　　CBL_ALLOC_DYN_MEMによる割付けメモリの解放
CBL_FREE_MEM　　CBL_ALLOC_MEMによる割付けメモリの解放
CBL_FREE_RECORD_LOCK　　ファイルに対するレコードロックの解除
CBL_FREE_SHMEM　　CBL_ALLOC_SHMEMによる割付けメモリの解放
CBL_FREE_THREAD_MEM　　CBL_ALLOC_THREAD_MEMによる割付けメモリの解放
CBL_GET_CSR_POS　　カーソル位置の獲得
CBL_GET_CURRENT_DIR　　現行ディレクトリをリターン
CBL_GET_EXIT_INFO　　閉止手続き呼び出し時の状況通知
CBL_GET_KBD_STATUS　　キーボードでの文字の試験
CBL_GET_MOUSE_MASK　　マウス事象マスクの獲得
CBL_GET_MOUSE_POSITION　　マウス画面座標の獲得
CBL_GET_MOUSE_STATUS　　待ち行列内事象数の獲得
CBL_GET_OS_INFO　　オペレーティングシステム環境情報を獲得
CBL_GET_PROGRAM_INFO　　指定されたプログラムまたは現行呼出しスタック内のプログラムの情報をリターン
CBL_GET_RECORD_LOCK　　ファイルに対するレコードロックの取得
CBL_GET_SCR_GRAPHICS　　図形文字の獲得
CBL_GET_SCR_LINE_DRAW　　線画文字の獲得
CBL_GET_SCR_SIZE　　画面サイズの獲得
CBL_GET_SHMEM_PTR　　名前付き値の読み込み
CBL_HIDE_MOUSE　　マウスポインタを隠す

CBL_IMP 論理暗示（IMP）
CBL_INIT_MOUSE マウス支援を初期化
CBL_JOIN_FILENAME ファイル名の部分・部分を結合
CBL_MONITOR_BROWSE スレッドの走査機能を取得
CBL_MONITOR_BROWSE_TO_READ 走査機能を読み込み機能に変換
CBL_MONITOR_BROWSE_TO_WRITE 走査機能を書き込み機能に変換
CBL_MONITOR_CLOSE 与えられたモニターハンドルをクローズ
CBL_MONITOR_OPEN_INTRA モニターを作成
CBL_MONITOR_READ 読み込み機能の取得
CBL_MONITOR_RELEASE モニターのロックを解除
CBL_MONITOR_UNBROWSE 走査機能を解除
CBL_MONITOR_UNREAD 読み込み機能を解除
CBL_MONITOR_UNWRITE 書き込み機能を解除
CBL_MONITOR_WRITE 書き込み機能の取得
CBL_MONITOR_WRITE_TO_BROWSE モニター書き込み機能を走査機能に変換
CBL_MUTEX_ACQUIRE MUTEXを取得
CBL_MUTEX_CLOSE 指定されたMUTEXをクローズ
CBL_MUTEX_OPEN_INTRA プロセス間MUTEXを作成
CBL_MUTEX_RELEASE 指定されたMUTEXを解除
CBL_NLS_CLOSE_MSG_FILE メッセージファイルをクローズ
CBL_NLS_COMPARE 2つの文字列を比較
CBL_NLS_INFO 国別言語情報を取得または設定
CBL_NLS_OPEN_MSG_FILE メッセージファイルをオープン
CBL_NLS_READ_MSG メッセージファイルからメッセージの読み込み
CBL_NOT 論理否定（NOT）
CBL_OPEN_FILE バイトストリームファイルのオープン
CBL_OPEN_VFILE 動的ストリームのオープン
CBL_OR 論理和（OR）
CBL_PUT_SHMEM_PTR 名前付き値の作成/更新
CBL_READ_FILE バイトストリームファイルの読込み
CBL_READ_KBD_CHAR キーボードからの文字の読込み（表示なし）
CBL_READ_MOUSE_EVENT マウス事象待ち行列の読込み
CBL_READ_SCR_ATTRS 属性列の読込み
CBL_READ_SCR_CHARS 文字列の読込み
CBL_READ_SCR_CHATTRS 文字列および属性列の読込み
CBL_READ_VFILE 動的ストリームの読込み
CBL_RENAME_FILE ファイルの再命名
CBL_SCR_ALLOCATE_COLOR RGB値用カラーの割当て
CBL_SCR_ALLOCATE_VC_COLOR 仮想カラーマップのカラーの割当て

CBL_SCR_CREATE_VC	仮想カラーマップの作成
CBL_SCR_DESTROY_VC	仮想カラーマップの削除
CBL_SCR_GET_ATTR_INFO	属性情報の取得
CBL_SCR_GET_ATTRIBUTES	属性値の取得
CBL_SCR_NAME_TO_RGB	カラー名をRGB値に変換
CBL_SCR_QUERY_COLORMAP	カラーマップエントリの問い合わせ
CBL_SCR_RESTORE_ATTRIBUTES	属性表のリカバリ
CBL_SCR_SAVE_ATTRIBUTES	属性表の保存
CBL_SCR_SET_ATTRIBUTES	属性値の設定
CBL_SCR_SET_PC_ATTRIBUTES	IBM-PC属性パレットをセットアップ
CBL_SEMAPHORE_ACQUIRE	セマフォのリソースの1つを取得
CBL_SEMAPHORE_CLOSE	セマフォをクローズ
CBL_SEMAPHORE_OPEN_INTRA	プロセス間セマフォを作成
CBL_SEMAPHORE_RELEASE	セマフォのリソースの1つを解放
CBL_SET_CSR_POS	カーソル位置の設定
CBL_SET_MOUSE_MASK	マウス事象マスクの設定
CBL_SHOW_MOUSE	マウスポインタの描画
CBL_SPLIT_FILENAME	ファイル名を部分・部分に分割
CBL_SUBSYSTEM	宣言/割付け解除サブシステム
CBL_SWAP_SCR_CHATTRS	文字および属性の交換
CBL_TERM_MOUSE	マウス支援の終了
CBL_TEST_RECORD_LOCK	ファイルに対するレコードロックのテスト
CBL_THREAD_CREATE	名前付き入口点からスレッドを作成
CBL_THREAD_CREATE_P	手続きポインタからスレッドを作成
CBL_THREAD_DETACH	未解放スレッドを解放
CBL_THREAD_EXIT	現行スレッドを終了
CBL_THREAD_IDDATA_ALLOC	IDデータ領域をスレッドに割り付け
CBL_THREAD_IDDATA_GET	IDデータ領域へのポインタを取得
CBL_THREAD_KILL	スレッドを中断
CBL_THREAD_LIST_END	スレッドリスト処理を終了
CBL_THREAD_LIST_NEXT	スレッドリストの次のスレッドを取得
CBL_THREAD_LIST_START	スレッドリスト処理を開始
CBL_THREAD_LOCK	スレッド処理ルーチンをロック
CBL_THREAD_PROG_LOCK	スレッドをロック
CBL_THREAD_PROG_UNLOCK	スレッドのロックを解除
CBL_THREAD_RESUME	サスペンドされたスレッドの再開
CBL_THREAD_SELF	スレッドの識別子を格納
CBL_THREAD_SLEEP	スレッドによるCPU制御を停止
CBL_THREAD_SUSPEND	スレッドをサスペンド

CBL_THREAD_UNLOCK　スレッド処理ルーチンのロックを解除
CBL_THREAD_WAIT　未解放スレッドの終了を待機
CBL_THREAD_YIELD　スレッドのタイムスライスを放棄
CBL_TOLOWER　文字列を小文字に変換
CBL_TOUPPER　文字列を大文字に変換
CBL_TSTORE_CLOSE　スレッド格納領域の割り付けを解除
CBL_TSTORE_CREATE　スレッド格納領域を作成
CBL_TSTORE_GET　スレッド格納領域へのポインタを取得
CBL_WRITE_FILE　バイトストリームファイルの書出し
CBL_WRITE_SCR_ATTRS　属性列の書出し
CBL_WRITE_SCR_CHARS　文字列の書出し
CBL_WRITE_SCR_CHARS_ATTR　属性付き文字列の書出し
CBL_WRITE_SCR_CHATTRS　文字列および属性列の書出し
CBL_WRITE_SCR_N_ATTR　属性書出しの繰返し
CBL_WRITE_SCR_N_CHAR　文字書出しの繰返し
CBL_WRITE_SCR_N_CHATTR　文字および属性書出しの繰返し
CBL_WRITE_SCR_TTY　文字をTTY様式で書出し
CBL_WRITE_VFILE　動的ストリームの書出し
CBL_XOR　排他的論理和（XOR）
CBL_YIELD_RUN_UNIT　現行実行単位を譲渡
MF_CLIENT_STATE_ALLOCATE　クライアント識別子を割り当て
MF_CLIENT_STATE_DELETE　クライアント情報を削除
MF_CLIENT_STATE_EXPIRY　cookie期限切れフィールドで使用するデータ文字列をリターン
MF_CLIENT_STATE_FILE　状態情報を格納するファイルを指定
MF_CLIENT_STATE_PURGE　条件に合うクライアント情報を削除
MF_CLIENT_STATE_RESTORE　事前に保存または割り当てられたレコードをリカバリ
MF_CLIENT_STATE_SAVE　状態ファイルの情報を更新
PC_FIND_DRIVES　有効ドライブの検索
PC_PRINT_FILE　ファイルを印刷
PC_PRINTER_CLOSE　プリンタチャネルをクローズ
PC_PRINTER_CONTROL　プリンタコマンドをプリンタに送信
PC_PRINTER_FREE_BMP　メモリからビットマップを解放
PC_PRINTER_GET_COLOR　プリンタカラーを設定
PC_PRINTER_GET_FONT　プリンタフォントを設定
PC_PRINTER_INFO　プリンタ情報の取得
PC_PRINTER_LOAD_BMP　メモリにビットマップをロード
PC_PRINTER_OPEN　プリンタチャネルをオープン
PC_PRINTER_SET_COLOR　プリンタカラーを設定
PC_PRINTER_SET_DEFAULT　プロセス共通のデフォルトプリンタを設定

PC_PRINTER_SET_FONT　　プリンタフォントを設定
PC_PRINTER_WRITE　　プリンタへのテキスト書き込み
PC_PRINTER_WRITE_BMP　　プリンタへのビットマップ書き込み
PC_READ_DRIVE　　現行ドライブの読込み
PC_SET_DRIVE　　現行ドライブの設定
PC_WIN_CHAR_TO_OEM　　ANSI文字のバッファをOEM文字セットに変換
PC_WIN_INIT　　起動値を取得します。
PC_WIN_OEM_TO_CHAR　　OEM文字のバッファのANSI文字セットに変換

8.4　操作

ライブラリルーチンを使用するには、COBOL CALL文において必要なルーチンの名前を使用します。

CBL_ルーチンに対する呼出しは全てルーチンの名前を大文字として符号化する必要があります。その他のルーチン全てに対する呼出しは、マニュアルに説明してあるように符号化する必要があります。つまり、符号化は一般的には大文字で行いますが、大文字・小文字を使い分ける必要のある例外があります。

8.4.1 ルーチンの説明

名前による呼出しルーチン全てについての説明はアルファベット順になされています。

用語説明

各説明にはルーチン名と機能および下記の記述項（適宜）が網羅されています。

構文:　ルーチンを呼び出すのに使用できるCALL文を示します。定義する必要のあるパラメータはUSING句に列挙されます。

任意選択RETURNING句もまた示してあります。各ルーチンは、処理の結果を示す値を戻します。特に指定のない限り、ゼロは成功を示し、ゼロ以外は失敗を示します。この値はRETURNING句で指定されたデータ項目（このマニュアルでは「状態コード」と呼ばれている）に残っています。この句が省略されると、その値は特殊レジスタRETURN-CODEに残されます。

「状態コード」は、0から65535までの正数を保持できる数字データ項目でなければなりません（たとえば、PIC X(2) COMP-5）。

ルーチンの名前は、大文字で書く必要があります。

パラメータ:　RETURNINGおよびUSING句に示されるパラメータについて説明しています。角型括弧で囲まれたパラメータ、例えば[パラメータ1]は任意選択で、ルーチンの全ての形式について必要となるものではありません。

入口上のパラメータ:　入口で渡されるパラメータを示します。

出口上のパラメータ:　出口で渡されるパラメータを示します。1バイト以上のビットが参照される場合は、ビット0は最下位（最右）ビットです。

説明:　ルーチンを正しく使用するための付加情報を提供します。

CBL_ABORT_RUN_UNIT

　既存のスレッドの状態に関係なく現在の実行単位を中断します。

構文:

```
CALL "CBL_ABORT_RUN_UNIT" USING BY VALUE リターン値
```

パラメータ:

リターン値　　PIC X（4）COMP-5.

入口上のパラメータ:

リターン値　　本実行単位の終了時にオペレーティングシステムに返されるリターンコード

説明:

シングルスレットアプリケーションまたはランタイムシステムでは、本ルーチンは次の構文と同義です。

stop run returning リターン値

マルチスレッドアプリケーションでは、STOP RUNは既存のスレッドがすべて実行単位の終了前に実行されるのを待ちます。ただし、CBL_ABORT_RUN_UNITは、既存のスレッドの実行を待ちません。つまり、すべての実行中のスレッドをキャンセルして実行単位を中断し、制御をオペレーティングシステムに返します。

注:

アプリケーションで重大な問題が検出された場合のみ本ルーチンを使用されることをお勧めします。それは、マルチスレッドアプリケーションで本ルーチンを呼び出すと、スレッドが重要なファイル操作の間に終了してしまうなどの重要な問題が発生することがあるからです。

CBL_ALLOC_MEM

　実行時、動的にメモリを割り付けます。また、割り当てられたヒープデータをスレッドまたは呼出しプログラムに関連付けることができます。

構文:

```
CALL "CBL_ALLOC_MEM" USING  メモリポインタ
                    BY VALUE  メモリサイズ
                              フラグ
                     RETURNING 状態コード
```

パラメータ:

メモリポインタ　　USAGE POINTER.レベル01でなければならない。

メモリサイズ　　PIC X（4）COMP-5.

フラグ　　PIC X（4）COMP-5.

状態コード　　「用語説明」の項を参照してください。

入口上のパラメータ:

メモリサイズ　割り付けるメモリのバイト数

フラグ　　次のビット設定に従って割り付けるメモリの種類を示します。

ビット0　メモリを共有メモリとして割り付ける。

ビット1　予約済み、ゼロに設定する必要がある。

ビット2　どの呼出しプログラムからも独立させてメモリを割り当てる。ビット3が設定されている場合は、呼出しスレッドの終了時に自動的に解放される。ビット3が設定されていない場合は、実行単位の終了時に解放される。

ビット3　スレッドローカルにメモリを割り当てる。ビット2が未設定で直接的または間接的に（混合言語環境において）COBOLプログラムが呼び出されている場合は、呼出しプログラムがキャンセルされるかスレッドが終了したときに解放される。

ビット4-31　予約済み。ゼロに設定する必要がある。

出口上のパラメータ:

メモリポインタ　割り当てられるメモリへのポインタ。割り当てられたメモリは初期化されない。

状態コード　　0　メモリ割当てが正常終了した

157　メモリを割り当てることができない

181　フラグ指定が矛盾している

説明:

割り当てられたメモリは、どの値にも初期化されません。

この機能に割り付けられた任意の共有メモリに対する更新は、ランタイムシステムによって直列化も保護もされません。データの保全性を保持するためにセマフォを使用する必要があります。

メモリがスレッドによって割り当てられている場合は、呼出しスレッドが終了したときに解放されます。

非共有メモリの最大サイズは、（-lランタイムスイッチが設定されていない限りは）オペレーティングシステムだけに制約されます。

共有メモリの最大サイズは、オペレーティングシステムおよびランタイムシステムだけに制約されます。ランタイムチューナーshared_memory_segment_sizeは、最大サイズを設定する場合に使用できます。省略時の最大サイズは65536バイト、最小サイズは8192バイトです。

ビット1、2、または3は、互いに排他的です。排他的でない場合にはチェックするとエラー181が返されます。（混合言語環境において直接的または間接的な）呼出しプログラムがない場合は、ビット2は無視されます。

ビット2が設定されていない場合は、CBL_ALLOC_DYN_MEMによって割り付けられたメモリはそれを割り付けたプログラムが取り消される（論理的または物理的に）ときに解放されます。ただし、これはCOBOLプログラムが直接的または間接的にプログラムの呼出し側である場合です。

CBL_ALLOC_DYN_MEM

動的にメモリを割り付けます。また、そのメモリを呼出しプログラムに関連付けることができます。

構文:

```
CALL "CBL_ALLOC_DYN_MEM" USING  メモリポインタ
                  BY VALUE  メモリサイズ
                            フラグ
                  RETURNING 状態コード
```

パラメータ:

メモリポインタ　USAGE POINTER.レベル01でなければならない。
メモリサイズ　PIC X（4）COMP-5.
フラグ　PIC X（4）COMP-5.
状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

メモリサイズ　割り付けるメモリのバイト数
フラグ　次のビット設定に従って割り付けるメモリの種類を示します。
ビット0-1　予約済み、ゼロに設定する必要がある。
ビット2　どの呼出しプログラムからも独立させてメモリを割り当てる。
ビット4-31　予約済み。ゼロに設定する必要がある。

出口上のパラメータ:

メモリポインタ　割り当てられるメモリへのポインタ。割り当てられたメモリは初期化されない。
状態コード　0　メモリ割当てが正常終了した
157　メモリを割り当てることができない
181　フラグ指定が矛盾している

説明:

割り当てられたメモリは、どの値にも初期化されません。

本ルーチンにより割り当てられるメモリの最大サイズは、（-lランタイムスイッチが設定されていない限りは）オペレーティングシステムだけに制約されます。

ビット2が設定されていない場合は、CBL_ALLOC_DYN_MEMによって割り付けられたメモリはすべてそれを割り付けたプログラムが取り消される（論理的または物理的に）ときに解放されます。ただし、これはCOBOLプログラムが直接的または間接的にプログラムの呼出し側である場合です。

ビット2が設定されている場合は、このメモリは実行単位が終了したときに解放されます。ただし、これはいCBL_FREE_DYN_MEMによってそれまでに解放されていない場合です。

CBL_ALLOC_SHMEM

動的に共有メモリを割り付けます。

構文:

```
CALL "CBL_ALLOC_SHMEM" USING  メモリポインタ
                BY VALUE  メモリサイズ
                 RETURNING 状態コード
```

パラメータ:

メモリポインタ　USAGE POINTER.レベル01でなければならない。

メモリサイズ　PIC X（4）COMP-5.

状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

メモリサイズ　割り付けるメモリのバイト数

出口上のパラメータ:

メモリポインタ　割り当てられるメモリへのポインタ。割り当てられたメモリは初期化されない。

状態コード　0　メモリ割当てが正常終了した

157　メモリを割り当てることができない

説明:

割り当てられたメモリは、どの値にも初期化されません。

この機能に割り付けられた任意の共有メモリに対する更新は、ランタイムシステムによって直列化も保護もされません。データの保全性を保持するためにセマフォを使用する必要があります。

共有メモリの最大サイズは、オペレーティングシステムおよびランタイムシステムだけに制約されます。ランタイムチューナーshared_memory_segment_sizeは、最大サイズを設定する場合に使用できます。省略時の最大サイズは65536バイト、最小サイズは8192バイトです。

CBL_ALLOC_THREAD_MEM

スレッド固有のヒープメモリを割り当てます。

構文:

```
CALL "CBL_ALLOC_THREAD_MEM" USING  メモリポインタ
                BY VALUE  メモリサイズ
                          フラグ
                 RETURNING 状態コード
```

パラメータ:

メモリポインタ　ポインタ

メモリサイズ　picture s9（9）COMP-5

フラグ　picture s9（9）COMP-5

状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

メモリサイズ　割り付けるメモリのバイト数

フラグ　次のビット設定に従って割り付けるメモリの種類を示します。

ビット0-1　予約済み、ゼロに設定する必要がある。

ビット2　どの呼出しプログラムからも独立させてメモリを割り付ける。

ビット4-31　予約済み。ゼロに設定する必要がある。

出口上のパラメータ:

メモリポインタ　割り付けられるメモリへのポインタ。割り付けられたメモリは初期化されない。

状態コード　0　メモリ割当てが正常終了した

157　メモリを割り付けることができない

181　フラグ指定が矛盾している

説明:

ビット2が設定されていない場合は、CBL_ALLOC_DYN_MEMによって割り付けられたメモリはそれを割り付けたプログラムが取り消される（論理的または物理的に）ときに解放されます。ただし、これはCOBOLプログラムが直接的または間接的にプログラムの呼出し側である場合です。

ビット2が設定されている場合は、このメモリを割り付けたスレッドが終了したときに解放されます。ただし、これはいCBL_FREE_THREAD_MEMによってそれまでに解放されていない場合です。

CBL_AND

2つのデータ項目のビット間の論理積をとります。

構文:

```
CALL "CBL_AND" USING 源
                     目標
               BY VALUE 長さ
```

パラメータ:

源　任意のデータ項目

目標　任意のデータ項目

長さ　数字定数またはPIC X（4）COMP-5.

入口上のパラメータ:

源　ANDをとるデータ項目の1つ

目標　ANDをとるもう一方のデータ項目

長さ　ANDをとる源と目標のバイト数

この長さを超えた目標内の位置については変更はありません。

出口上のパラメータ:

目標　結果

説明:

本ルーチンは源と目標の左端から開始し、それらのビットの論理積（AND）をとり、結果を目標に格納します。これの真理値表は次の通りです。

源	目標	結果
0	0	0
0	1	0
1	0	0
1	1	1

CBL_CANCEL_PROC

プログラムの取り消しを制御します。

構文:

```
CALL "CBL_CANCEL_PROC" USING BY VALUE 機能
                 BY REFERENCE パラメータブロック
                 BY VALUE ユーザデータ長
                  RETURNING 状態コード
```

パラメータ:

```
機能                  pic x（4）comp-5.

パラメータブロック  次のように定義された集団項目
                      パラメータバージョン  pic x（4）comp-5 値0
                      フラグ                pic x（4）comp-5.
                      取り消しルーチン  USAGE PROCEDURE-POINTER.
                      ハンドル                USAGE POINTER
                      ユーザデータ            USAGE POINTER
                      導入優先順位  pic x（4）comp-5.
ユーザデータ長      pic x（4）comp-5.
状態コード          「用語説明」の項を参照してください。
```

入口上のパラメータ:

機能　次のいずれか1つを設定します。

0　取り消しルーチンを省略時の優先順位64で導入する。

1　取り消しルーチンを指定された優先順位で導入する。下記の「説明」を参照。

2　前回導入された取り消しルーチンの優先順位をプログラムにより変更できるようにする。

3　取り消しルーチンを呼び出さずに取り消し登録を導入解除する。

4　取り消しルーチンを呼び出した後で取り消し登録を導入解除する。取り消しルーチンは本来の「取り消す」意味ではなく導入解除するために呼び出されたかどうかを検出できることに注意。

上記以外の値は、将来の拡張のために予約されています。

パラメータバージョン　パラメータブロックのバージョン。ここで記述するパラメータブロックは、このフィールドをゼロに設定する必要があります。

フラグ　現時点では未定義。ゼロに設定する必要があります。

取り消しルーチン　設定可能な値は次の通りです。

0, 1　導入する取り消しルーチン
2-4　無視される

ハンドル　設定可能な値は次の通りです。

0, 1　取り消しルーチンに付加されるプログラムのプログラムハンドルまたは空。プログラムハンドルについては、下記の「説明」を参照。
2　登録を修正するための取り消しルーチンのハンドル。
3, 4　機能に0または1が設定された場合に返されるハンドル。取り消しルーチンを登録から削除するためのハンドル。

ユーザデータ　設定可能な値は次の通りです。

0, 1　ユーザ定義のデータ領域。ユーザデータ領域については、下記の「説明」を参照。
2-4　無視される

導入優先順位　設定可能な値は次の通りです。

0　無視される
1　この取り消しルーチンを導入する優先順位
2　取り消しルーチンを設定する優先順位（下記の「説明」を参照）
3, 4 無視される

ユーザデータ長　設定可能な値は次の通りです。

0, 1　ユーザデータのブロック長、またはゼロ。ユーザデータ領域については、下記の「説明」を参照。
2-4　無視される

出口上のパラメータ:

パラメータバージョン　無視される
フラグ　無視される
取り消しルーチン　無視される
ハンドル　設定可能な値は次の通りです。

0, 1　取り消し登録用のハンドル、またはNULL

2　無視される

3, 4　NULL

ユーザデータ　無視される

導入優先順位　無視される

リターンステータス:

0000	成功 - 操作が要求通りに実行されました。
1000	メモリ割り付けエラー処理が要求されました。
1001	不正なハンドルがAPIに渡されました（登録ハンドル）。
1007	システムエラー（ここにリストされていないエラー）
1009	不正なパラメータが（「機能」、「パラメータバージョン」などに）渡されました。

説明:

本ルーチンは、COBOLプログラムがそれ自身のまたは他のCOBOLプログラムが今にも取り消しされそうであるという通知を受けられるような機構を提供します。

そのような通知により、COBOLプログラムはリソースの使用をやめ、割り付けを解除することができます。このリソースはプログラムの実行中にそのプログラムによって割り付けられたものであり、COBOLランタイムシステムまたはオペレーティングによって自動的に扱われるものではありません。これらのリソースは、通常はオペレーティングシステムまたはランタイムシステムによって提供されるものですが、アプリケーション内の他のモジュールによって提供されるリソースもあります。

COBOLランタイムシステムは、取り消されるプログラムに関連するリスト（取り消しリスト）を処理することによって、通知を行います。そのため、通知を可能にするには、COBOLランタイムシステムは取り消しリストに記述項を作成するように命令されなければなりません。取り消しリストの記述項を管理することがCBL_CANCEL_PROCの目的です。

取り消しリストに記述項を作成するために、CBL_CANCEL_PROCは次の3つの事項を知る必要があります。

- どのプログラムがいつ取り消され、通知のきっかけとなるのか。このプログラムは、プログラムハンドル（パラメータ「ハンドル」によって記述項に設定されます）によって識別されます。
- 呼び出す取り消しルーチン。取り消しルーチンは、プログラムが取り消されるときに呼び出される入口点です。
- 取り消し優先順位。これはこの通知用の優先順位です。

パラメータ「機能」に0または1を設定すると、ランタイムシステムがこの情報を提供します。これにより、プログラムハンドルに関連する取り消しリストに記述項を作成することができます。記述項は、通知を受けるときに取り消しルーチンを呼び出すように指定するものです。このリストの位置は、取り消し優先順位の値によって決められます。

出口上の「ハンドル」パラメータで指定する登録呼出しによって返された値を使用すると、作成された取り消しリストの記述項を参照することができます。これにより、プログラムはこのルーチンに2、3、または4を指定した本ルーチンを使用し、後で登録を修正したりリストから削除することができます。

プログラムハンドルの取得

プログラムハンドルは、ユニークな値であり、COBOLランタイムシステムはこれを取り消し可能なものと関連付けます。そのため、あるCOBOLプログラムにおけるすべての入口点は同じプログラムハンドルを共有するので、COBOLプログラムの入口点を取り消すとそのプログラム内のすべての入口点が取り消されます。

CBL_CANCEL_PROCルーチンを使用する基本的な方法が2つあり、そのどちらの方法を必要とするかによって、プログラムが現行のプログラムハンドルを取得するために必要な作業量も違ってきます。以下にその2つの方法を説明します。

ローカル登録

プログラムが取り消されるという通知の受け取りを望んでいる場合には、ローカル登録をすることができます。これは通常そのコードで1回だけ行います。この登録方法では、プログラムはプログラムハンドルとしてNULLを渡しさえすればよいでのです。そうすると、CBL_CANCEL_PROCルーチンは自動的に現行のプログラムハンドルを設定して正しいリストに記述項を作成します。

通常、ローカル登録で指定された取り消しルーチンは、登録プログラムにおける入口点です（その必要はないのですが）。

このようなプログラムの例として、オペレーティングシステムから直接受け取ったいくらかのリソースを割り付けたプログラム､取り消される前にそのリソースの割当てを解除する機会が必要であったプログラムなどがあります。

サービス登録

この方法は、ローカル登録よりも少し複雑な登録方法です。この方法を使用するのは、サービスプログラム（つまり、他のCOBOLプログラムのためにリソースを管理するプログラム）がクライアントが取り消されるという通知の受け取りを必要とする場合です。この登録方法では、プログラムハンドルはCBL_GET_PROGRAM_INFOルーチンを呼び出すことによって取得できます。CBL_GET_PROGRAM_INFOルーチンに渡すパラメータは、どの機能を必要とするかによって違ってきます。最も一般的な使用法は、CBL_GET_PROGRAM_INFOルーチンの機能2を使用して呼出しプログラムのプログラムハンドルを取得する方法です。

通常、サービス登録で指定された取り消しルーチンは、登録プログラムにおける入口点です（その必要はないのですが）。

このようなプログラムの例として、他のCOBOLプログラムのためにファイルを開いたり管理したりするファイルハンドラモジュールなどがあります。指定されたプログラムが取り消されると、そのプログラムのために開かれたファイルはすべてCOBOLの定義に従って自動的に閉じられます。

この例は、ファイルを開くプログラムとファイルを閉じるだけのプログラムにローカル登録を行うことによっても可能であるということに注意してください。ただし、そのようなアプローチではエラーが発生しやすく、COBOLの定義には準拠していません。また、乱雑な方法ともいえます。

CBL_CANCEL_PROCルーチンをサービスプログラムで使用する場合に重要なことは、適切なリソースがクライアントプログラムによって解放されるならばそれに対する通知はそのクライアントプログラムの取り消しリストから削除されるということです。これは、「機能」に3または4を設定すると行われます。これを効率的に行うために、サービスプログラムは「機能」に0または1を設定したときに返されたプログラムハンドルをリソース構造体に格納します。そのため、「機能」に3または4を設定したときに登録ハンドルとして返されるように簡単にプログラムハンドルを配置できます。

サービスプログラムが機能3または4を使用した登録解除に失敗した場合、クライアントプログラムが取り消されるときに仮の通知が発生し、予測できない結果となります。

ユーザデータ

CBL_CANCEL_PROCによって要求された3つの必須情報（つまり、「機能」に0または1を設定することにより得られるプログラムハンドル、取り消しルーチン、および取り消し優先順位）のほかに、オプションでユーザデータも受け入れられます。

「ユーザデータ」パラメータを使用すると、取り消しルーチンを登録するプログラムはプログラムが取り消されたという通知を受けるときに取り消しルーチンに渡すパラメータを指定することができます。登録ごとに異なる「ユーザデータ」パラメータを指定することができ、また複数の登録に同じ「ユーザデータ」パラメータを指定することもできます。さらに、「ユーザデータ長」パラメータにゼロ以外の値を指定することによって、「ユーザデータ」パラメータをコピー（このコピーがパラメータとして取り消しルーチンに渡されます）するように指定することもできます。

この機能は、サービスプログラムでとても役に立ちます。サービスプログラムはクライアントプログラムをサービス「ハンドル」と関連付けるためにリストを管理する必要がありません。ただし、CBL_CANCEL_PROC登録プロセスに「ユーザデータ」パラメータとして「ハンドル」を渡してあるならば、取り消しルーチンに渡すパラメータとしてサービス「ハンドル」を直接受け取ることはできます。

ゼロ以外のユーザデータ長パラメータの指定

「ユーザデータ長」パラメータには、ゼロよりもゼロ以外の値を設定した方が便利です。ただし、これは一般的にはユーザデータ領域が小さい場合のみです。「ユーザデータ長」パラメータにゼロ以外の値を設定すると、「ユーザデータ」パラメータのアドレスから指定されたバイト数がコピーされ、そのコピーはりストの記述項に保持されます。あるプログラムハンドルに関連するプログラムが取り消された場合、取り消しルーチンはパラメータとしてユーザデータ領域のコピーを受け取ります。そのコピーは取り消しルーチンが戻ったときにCOBOLランタイムシステムによって削除されます。

CBL_CANCEL_PROCルーチンは渡されたユーザデータ領域の構造については何も知らず、ただフラットメモリブロックをコピーするだけであるということに注意してください。たとえば、ブロック内に埋め込まれたUSAGE POINTERの項目はコピーされますが、その項目はメモリを指しません。この場合は、アプリケーションがデータ構造全体を割り付けてコピーし、そのコピーを値がゼロのユーザデータ長を使用して渡した方が良いでしょう。

ゼロのユーザデータ長パラメータの指定

「ユーザデータ長」パラメータにゼロを設定すると（これにより登録プロセスに指定された元の「ユーザデータ」パラメータが取り消しルーチンに渡されます。）、取り消しルーチンがブロック内のすべてのデータ構造を削除するかその割り付けを解除します。

この方法を使用して「ユーザデータ」パラメータを渡す場合には、とりわけ同じユーザデータ領域が複数の登録ハンドル（登録ハンドルは、「ハンドル」パラメータによって出口上に返されます）に登録される場合は特に注意が必要です。つまり、次のことに気を付けてください。割り付けられたリソースを解除する場合は1度だけ行うこと、割り付けられたリソースを関連する登録ハンドルの最後が壊れるまで有効な状態に保つこと、この2点です。これを実現するには、通常はユーザデータ領域のどこかに「使用回数」フィールドを置きます。「使用回数」は「ユーザデータ」パラメータが登録ハンドルに登録されるたびに数字を1づつ加算し、取り消しルーチンが「ユーザデータ」パラメータを指定して呼び出されるたびに数字を1づつ原産します。

優先順位

優先順位（機能1を使用すると設定されます）は、ユーザコードに対して0から127までの値でなければなりません。200から209までの値の優先順位は、ファイルハンドラでは逆順の意味に取られます。これ以外の値を優先順位に指定すると、予期できない結果となります。たいていのアプリケーションでは、省略値の64は使用できます。

取り消しルーチンの実行

ここでは、CBL_CANCEL_PROCで導入する場合に適した取り消しルーチンの実行方法について説明します。

```
working-storage section.
01 cancel-status   pic x(4) comp-5.

linkage-section.
01 cancel-reason   pic x(4) comp-5.
01 cancel-flags    pic x(4) comp-5.
01 program-id     usage pointer.
01 user-data      usage pointer.

procedure division.
entry "user-cancel-routine" using by value cancel-reason
                      by value      cancel-flags
                      by value      program-id
                      by value      user-data.
cancel-routine section.
cancel-routine-para.
* This is where the code which handles the CANCEL-notification goes.
* The user-data field specified in the registration is passed to us here,
* along with the program-id of the program being cancelled.
* Iff this routine is handling registrations which specified a
```

```
 * user-data-field length of zero, then the memory pointed to by user-data
 * may also need to be freed before returning.

  exit program returning cancel-status.
```

cancel_reasonパラメータには、取り消しの理由を記述します。次のように設定できます。

ビット0（値1） 取り消しは、STOP RUN処理によるものです。

ビット6（値64） 取り消しは、導入解除（「機能」に4を設定してCBL_CANCEL_PROCを呼び出した場合）によるものです。

これ以外のビットはすべて予約済みであり、設定しても無視されます。

cancel_flagsパラメータは現在予約済みであり、設定しても無視されます。

CBL_CHANGE_DIR

現行ディレクトリを変更します。

構文:

```
CALL "CBL_CHANGE_DIR" USING  パス名
                    RETURNING  状態コード
```

パラメータ:

パス名 PIC X（n).

状態コード 「用語説明」の項を参照してください。

入口上のパラメータ:

パス名 スペースまたは空（x"00"）で終わる相対または絶対パス名。このパス名はオペレーティングシステムにより許容される文字数より長くなってはいけません。また、本ルーチンが呼ばれるとき現行であるディレクトリから有効になっている必要があります。

出口上のパラメータ: なし

CBL_CHECK_FILE_EXIST

ファイルが存在するかどうかチェックし、存在すればその詳細を戻します。

構文:

```
CALL "CBL_CHECK_FILE_EXIST" USING  ファイル名
                                   ファイル詳細
                          RETURNING  状態コード
```

パラメータ:

ファイル名 PIC X（n）.

ファイル詳細 次のように定義された集団項目

ファイルサイズ PIC X（8）COMP-X.

ファイル日付

日 PIC X COMP-X.

月 PIC X COMP-X.

年　　PIC X（2）COMP-X.

ファイル時間

時　　PIC X COMP-X.

分　　PIC X COMP-X.

秒　　PIC X COMP-X.

1/100秒　　PIC X COMP-X.

状態コード　　「用語説明」の項を参照してください。

入口上のパラメータ:

ファイル名　　捜すファイル。名前にはパス名を入れることができ、スペースで終了します。パスが与えられないと、現行ディレクトリと見なされます。本ルーチンは、ワイルドカードを含むファイル名を指定すると動作しません。

出口上のパラメータ:

ファイルサイズ　　ファイルのサイズ（バイト）

ファイル日付　　ファイルが作成された日付

ファイル時間　　ファイルが作成された時間

CBL_CLEAR_SCR

指定された文字と属性に全画面を消去します。

構文:

```
CALL "CBL_CLEAR_SCR" USING  文字
                            属性
                    RETURNING  状態コード
```

パラメータ:

文字　　PIC X.

属性　　PIC X.

状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

文字　　書き出す文字

属性　　書き出す属性。

出口上のパラメータ:　なし

CBL_CLOSE_FILE

バイトストリーム処理用に開かれているファイルを閉じます。

構文:

```
CALL "CBL_CLOSE_FILE" USING  ファイルハンドル
```

パラメータ:

ファイルハンドル　　PIC X（4).

入口上のパラメータ:

ファイルハンドル　　ファイルが開かれたときに返されたファイルハンドル

出口上のパラメータ:　なし

説明:

STOP RUNが実行されるとき開いている任意のバイトストリームファイルが自動的に閉じられます。実行単位のレコードロックは本ルーチンの呼出しにより解除されます。

呼出しが成功しているかどうかは、RETURN-CODEを検査するとわかります。

CBL_CLOSE_VFILE

ヒープを閉じます。

構文:

```
CALL "CBL_CLOSE_VFILE" USING BY VALUE  ヒープid
                       RETURNING  状態コード
```

パラメータ:

ヒープ***id***　　PIC X（2）COMP-5.

状態コード　　「用語説明」の項を参照してください。

入口上のパラメータ:

ヒープ***id***　　ヒープが開かれたときに割り当てられたヒープハンドルが入っています。

出口上のパラメータ:　　なし

CBL_COPY_FILE

ファイルをコピーします。

構文:

```
CALL "CBL_COPY_FILE" USING  ファイル名1
                            ファイル名2
                     RETURNING  状態コード
```

パラメータ:

ファイル名1　　PIC X（n).

ファイル名2　　PIC X（n).

状態コード　　「用語説明」の項を参照してください。

入口上のパラメータ:

ファイル名1　コピーするファイル。名前はパス名を含むことができ、スペースで終了します。パスが与えられてないと、現行ディレクトリと見なされます。

ファイル名2　新しいファイルの名前。名前はパス名を含むことができ、スペースで終了します。パスが与えられてないと、現行ディレクトリと見なされます。

出口上のパラメータ:　なし

説明:

本ルーチンは、ワイルドカードを含むファイル名を指定すると動作しません。

CBL_CREATE_DIR

サブディレクトリを作成します。最後のディレクトリを除く与えられたパス内のディレクトリは全て既に存在している必要があります。

構文:

```
CALL "CBL_CREATE_DIR" USING  パス名
                   RETURNING  状態コード
```

パラメータ:

パス名　PIC X（n）.

状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

パス名　スペースまたは空（x "00"）で終わる相対または絶対パス名。

出口上のパラメータ:　なし

CBL_CREATE_FILE

バイトストリーム処理用の新しいファイルを作成します。

構文:

```
CALL "CBL_CREATE_FILE" USING     ファイル名
                                 アクセスモード
                                 拒否モード
                                 装置
                                 ファイルハンドル
```

パラメータ:

ファイル名　PIC X（n).

アクセスモード　PIC X COMP-X.

拒否モード　PIC X COMP-X.

装置　PIC X COMP-X.

ファイルハンドル　PIC X（4).

入口上のパラメータ:

ファイル名　　開くファイルのスペースまたは空で終了するファイル名

アクセスモード　　アクセスモードを定義します:

1=読込み専用

2=書出し専用（拒否モードは0としておく必要がある）

3=読込み/書出し

拒否モード　　拒否モードを定義します。

0=読込みおよび書出しの両方を拒否（排他的）

1=書出しを拒否

2=読込みを拒否

3=読込み・書出しのどちらも拒否しません

装置　　将来の使用に予約されています（0とする必要がある）。

出口上のパラメータ:

ファイルハンドル　　ファイルを開くことに成功するとファイルハンドルを戻します。

説明:

本呼出しが成功したかどうかは、RETURN－CODEを調べるとわかります。

CBL_DEBUGBREAK

COBOLプログラムがデバッガを起動できるようにします。

構文:

```
CALL "CBL_DEBUGBREAK"
```

パラメータ:　　なし

説明:

本ルーチンは、デバッガを起動するのに使用できます。プログラムがデバッグされていない状態でCBL_DEBUGBREAKを呼び出した場合、デバッガが起動します。それからすでにデバッグを実行している状態でツールバーのStopボタンを押したかのようにデバッガが停止状態になります。

プログラムをデバッグしている状態でCBL_DEBUGBREAKを呼び出した場合は、ツールバーのStopボタンを押したかのようにデバッグが停止します。

本ルーチンは、呼出しが成功した場合は0のRETURN-CODEを返し、デバッガを起動しようとして失敗した場合はゼロ以外のRETURN-CODEを返します。

CBL_DELETE_DIR

ディレクトリを削除します。ディレクトリは空の場合にのみ削除されます。

構文:

```
CALL "CBL_DELETE_DIR" USING  パス名
                   RETURNING  状態コード
```

パラメータ:

パス名　　PIC X（n).

状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

パス名　スペースまたは空（x"00"）で終わる相対または絶対パス名。

出口上のパラメータ:　なし

CBL_DELETE_FILE

ファイルを削除します。

構文:

```
CALL "CBL_DELETE_FILE" USING  ファイル名
                    RETURNING  状態コード
```

パラメータ:

ファイル名　PIC X（n).

状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

ファイル名　削除するファイル。名前はパス名を含むことができ、スペースで終了します。パスが与えられてないと、現行ディレクトリと見なされます。本ルーチンは、ワイルドカードを含むファイル名を指定すると動作しません。

出口上のパラメータ:　なし

CBL_EQ

2つのデータ項目のビット間で論理等価を行います。

構文:

```
CALL "CBL_EQ" USING  源
                     目標
             BY VALUE  長さ
```

パラメータ:

源　　任意のデータ項目

目標　　任意のデータ項目

長さ　　数字定数またはPIC X（4）COMP-5.

入口上のパラメータ:

源　　等価（EQUIVALENCE）をとるデータ項目の内の1つ。

目標　　等価（EQUIVALENCE）をとるもう一方のデータ項目。

長さ　　等価（EQUIVALENCE）をとろうとする源と目標のバイト数。この長さを越えた目標内の位置については変更はありません。

出口上のパラメータ:

目標　　結果

説明:

本ルーチンは源と目標の左端から開始し、それらのビットの等価（EQUIVALENCE）をとり、結果を目標に格納します。これらの真理値表は次の通りです。

源	目標	結果
0	0	1
0	1	0
1	0	0
1	1	1

CBL_ERROR_PROC

ランタイムシステムエラーが発生したときに自動的に呼び出す利用者エラー手続きを導入または削除します。

構文:

```
CALL "CBL_ERROR_PROC" USING  導入フラグ
                             導入アドレス
                     RETURNING  状態コード
```

パラメータ:

導入フラグ　　PIC X COMP-X.

導入アドレス　　USAGE PROCEDURE POINTER

状態コード　　「用語説明」の項を参照してください。

入口上のパラメータ:

導入フラグ　　実行される処理を示します。
0=エラー手続きの導入
1=エラー手続きの導入解除

導入アドレス　　エラー手続きのアドレス

出口上のパラメータ:　なし

説明:

本ルーチンを繰り返し呼び出すことによってアプリケーションにいくつかのエラー手続きを導入できます。

エラー手続きは任意の言語で書くことができます。COBOLで書く場合、「導入アドレス」は入口点のアドレスとする必要があります。次の文を用いてこのアドレスを得ることができます。

set　導入アドレス　**to entry**　入口名

エラーが発生すると、導入された手続きが実行されます。この実行は、最後に導入された手続きで始まり、最初に導入された手続きで終わります。すべての手続きが処理されると、RTSエラー処理手続きが実行されます。

ランタイムシステムエラー処理など他にエラー手続きが必要ない場合は、エラー手続きの終了後に実行されるようにRETURN-CODEをゼロに設定します。

エラー手続きを含むプログラムが取り消された場合は、エラー手続きは削除されます。

あるプログラムに導入されたエラー手続きは、別のプログラムが削除できます。

エラー手続きによりまた別のエラー手続きを導入することができます。

RTSエラーがあるエラー手続きで発生した場合は、そのエラー手続きは終了します。RTSは新しいRTSエラーを処理して次のエラー手続きがある場合はそれを実行します。

手続きを導入する場合は、使用する手続きポインタが有効であるかどうかを確かめる必要があります。

エラー手続きは、それを導入した実行単位に属しています。そのため、そのエラー手続きはそれを導入した実行単位でのみ実行されます。

エラー手続きがあるプログラムの入口点として定義されていてそのプログラム自身にエラーがある場合は、プログラムに局所記憶域節があることを確かめる必要があります。これは、プログラムが再入可能か、エラー手続きの実行時にランタイムシステムがエラー166（「再帰COBOL呼出しが不正です。」）を引き起こさないかどうかを確認するものです。

エラー手続きに渡されるパラメータ

導入されたエラー手続きが呼び出されると、関連するRTSエラーメッセージを含む文字がパラメータとして渡されます。

このパラメータを連絡節にPIC X(325)で定義し、エラー手続きへの入口のUSING指令に含める必要があります。以下にRTSエラーメッセージ文字列の例を示します。

```
Load Error : file 'prog-name'¥n
error code: 173, pc=0, call=-1, seg=0¥n
173  Called program file not found in drive/directory¥n¥0
```

¥nは新しい行の文字列、¥0は空（x"00"）の終了文字です。利用者エラーメッセージはこの形式で記述されています。

本ルーチンは成功した場合RETURN-CODEをゼロに設定し、失敗した場合ゼロ以外に設定します。

エラー手続きの導入例

エラー手続きの導入例およびエラー発生時に呼び出されるエラー手続きの概略を以下に示します。

```
working-storage section
01 install-flag    pic x comp-x value 0.
01 install-address usage procedure-pointer.
01 status-code pic 9(4) comp value zeros.

linkage section.
01 err-msg  pic x(325).

procedure division.
 set install-address to entry "err-proc".
 call "CBL_ERROR_PROC" using  install-flag
  install-address
  returning status-code.

* Error procedure:
 entry "err-proc" using err-msg.

* Process err-msg to find out the error number.
* Act accordingly.
 ...

* Terminate, but allow other error procedures to be executed.
  move 1 to return-code
  exit program
  stop run.
```

次の例は、エラー手続きに渡される情報です。これらの例では、数字の間の空白はわかりやすくするためのものであり、実際のパラメータにはありません。

COBOL PC 033Dのルートセグメントにおける ERRORP.GNT プログラムの RTS エラー163:

```
0 ¥0 163 ¥0 ERRORP.GNT ¥0 RT ¥0 033D ¥0
Illegal character in numeric field  (Error 163) ¥0¥0
```

COBOL PC 008Dのxyzファイルのルートセグメントにおける APE.INT プログラムの RTS エラー013:

```
1 ¥0 013 ¥0 APE.INT ¥0 RT ¥0 0080 ¥0
File not found  (Error 013) ¥0 xyz ¥0¥0
```

COBOL PC 0053の PROG1 プログラムのルートセグメントにおける APE.INT プログラムの RTS エラー173:

```
2 ¥0 173 ¥0 APE.INT ¥0 RT ¥0 0053 ¥0
Called program file not found in drive/directory (Error 173) ¥0 PROG1 ¥0¥0
```

CBL_EVENT_CLEAR

指定された事象を消去します。

構文:

```
CALL "CBL_EVENT_CLEAR" USING BY VALUE 事象ハンドル
```

パラメータ:

事象ハンドル　　ポインタ

入口上のパラメータ:

事象ハンドル　事象ハンドル

出口上のパラメータ:

リターンコード　値0は呼出しが成功したことを意味し、それ以外の値はエラーを意味します。詳細は、「同期ルーチンのRETURN-CODE値」を参照してください。

説明:

本ルーチンは、事象がすでに消去されている場合は効果がありません。

事象ハンドルが不正である場合の動作は不定です。

CBL_EVENT_CLOSE

与えられた事象ハンドルを閉じます。

構文:

```
CALL "CBL_EVENT_CLOSE" USING BY VALUE 事象ハンドル
```

パラメータ:

事象ハンドル　ポインタ

入口上のパラメータ:

事象ハンドル　事象ハンドル

出口上のパラメータ:

リターンコード　値0は呼出しが成功したことを意味し、それ以外の値はエラーを意味します。詳細は、「同期ルーチンのRETURN-CODE値」を参照してください。

説明:

本ルーチンにより事象は壊れます。

事象ハンドルが不正である場合の動作は不定です。

CBL_EVENT_OPEN_INTRA

プロセス内事象を作成します。

構文:

```
CALL "CBL_EVENT_OPEN_INTRA" USING BY REFERENCE 事象ハンドル
                          BY VALUE  オープンフラグ
```

パラメータ:

事象ハンドル　ポインタ

オープンフラグ　PIC X（4）COMP-5

入口上のパラメータ:

オープンフラグ　　事象をどのように作成するかを次のように指定します。

ビット	値	意味
0	0	事象を通知せずに作成します。
	1	事象を通知して作成します。
1～31		予約済み。0に設定する必要があります。

出口上のパラメータ:

事象ハンドル　事象ハンドル

リターンコード　値0は呼出しが成功したことを意味し、それ以外の値はエラーを意味します。詳細は、「同期ルーチンのRETURN-CODE値」を参照してください。

説明:

事象ハンドルはプロセス内で有効です。

CBL_EVENT_POST

指定した事象を通知します。

構文:

```
CALL "CBL_EVENT_POST" USING BY VALUE 事象ハンドル
```

パラメータ:

事象ハンドル　　ポインタ

入口上のパラメータ:

事象ハンドル　事象ハンドル

出口上のパラメータ:

リターンコード　値0は呼出しが成功したことを意味し、それ以外の値はエラーを意味します。詳細は、「同期ルーチンのRETURN-CODE値」を参照してください。

説明:

スレッドがCBL_EVENT_WAIT呼出しで事象が通知されるのを待っている場合、そのスレッドはすべて実行してその後CBL_EVENT_WAIT呼出しから戻ることが許されています。

本呼出しは、事象がすでに通知されている場合は効果がありません。本呼出しは、成功するとゼロを返します。

事象ハンドルが不正である場合の動作は不定です。

CBL_EVENT_WAIT

事象が通知されるのを待ちます。

構文:

```
CALL "CBL_EVENT_OPEN_INTRA" USING BY VALUE 事象ハンドル
                  BY VALUE  待ちなしフラグ
```

パラメータ:

事象ハンドル　　ポインタ

待ちなしフラグ　　PIC X（4）COMP-5

入口上のパラメータ:

事象ハンドル　　事象ハンドル

オープンフラグ　　事象が通知されたかった場合にどのように動作すべきかを次のように指定します。

ビット	値	意味
0	0	スレッドは事象が通知されるまでブロックし、その後戻ります。
	1	呼出しはエラー値を設定して直ちに戻ります。
1〜31		予約済み。0に設定する必要があります。

出口上のパラメータ:

リターンコード　　値0は呼出しが成功したことを意味し、それ以外の値はエラーを意味します。詳細は、「同期ルーチンのRETURN-CODE値」を参照してください。

説明:

本呼出しは、事象がすでに通知されている場合は直ちに戻ります。

事象ハンドルが不正である場合の動作は不定です。

CBL_EXEC_RUN_UNIT

同期実行単位または非同期実行単位を作成します。

構文:

```
CALL "CBL_EXEC_RUN_UNIT" USING        コマンド行
                     BY VALUE     コマンド行長
                     BY REFERENCE 実行単位ID
                     BY VALUE     スタックサイズ
                                  フラグ
                     BY REFERENCE ttyコマンド
                     BY VALUE     ttyコマンド長
                     RETURNING    状態コード
```

パラメータ:

コマンド行　　PIC X（n）

コマンド行長　　PIC X（4）COMP-5.

実行単位ID　　PIC X（8）COMP-5.レベル01にする必要があります。

スタックサイズ　　PIC X（4）COMP-5.

フラグ　　PIC X（4）COMP-5.

ttyコマンド　　PIC X（n）

ttyコマンド長　　PIC X（4）COMP-5.

状態コード　　「用語説明」の項を参照してください。

入口上のパラメータ:

コマンド行　　新しい実行単位に渡されたコマンド。これは、その後ろにどのパラメータも付けられるプログラム名にします。

コマンド行長　　コマンド行の長さ

スタックサイズ　　このパラメータは無視されます。

フラグ　　新しい実行単位をどのように作成するかを次のように設定します。

ビット	値	意味
0	0	CBL_EXEC_RUN_UNITは子プロセスを作成すると直ちに呼出しプロセスに戻ります。
	1	CBL_EXEC_RUN_UNITは呼出しプロセスに戻るまで新しい実行単位が完了するのを待ちます。
1	0	新しく作成された実行単位は、COBOLスイッチ、ランタイムスイッチ、環境変数、およびオープンライブラリが作成時に呼出しプロセス内に存在する場合は、それらをすべて継承します。詳細は、下記の「注」を参照。
	1	新しく作成された実行単位は、作成時に環境変数が呼出しプロセスにある場合は、それらをすべて継承します。
2	0	新しく作成された実行単位は、呼出し側の画面I/O用のコンソールを継承します。一度にこのコンソールにアクセスできるのは、コルーチンの一メンバーだけです。そうしなければ予期できない結果となります。
	1	新しく作成された実行単位は、それに割り当てられたターミナルI/O用の新しいコンソールを持ちます。このコンソールは、親プロセスからは独立していて、画面I/O用に使用することができます。このフラグが指定されていない場合は、ttyコマンドおよびttyコマンド長パラメータを指定する必要があります。
3～31		予約済み。0に設定する必要があります。

ttyコマンド　　「フラグ」にビット2が設定されている場合のみ本パラメータが必要です。ttyコマンドは、新しい実行単位に割り当てるコンソールを識別し、その新しい実行単位用のコンソールを作成するためのコマンドを提供します。オペレーティングシステムが本パラメータの指定を要求しない場合は、ダミー値を指定します。

ttyコマンド長　　ttyコマンドの長さ。Windows95は本パラメータの指定を要求しないので、常に0を設定します。

出口上のパラメータ:

実行単位ID　新しい実行単位を識別するユニークなハンドル

状態コード　0　成功

157　メモリ不足

181　パラメータ不正

200　内部論理エラー

説明:

呼出しを行う実行単位は、親プロセスとして知られています。一方、作成される実行単位は子プロセスとして知られています。子プロセスは、作成されるとその親プロセスからある属性を継承します。子プロセスが継承する属性は、呼出し時点で親プロセスに存在する属性のコピーです。呼出し後に親プロセスの属性に変更された内容は、子プロセスの属性に反映されません。同様に、子プロセスの属性の変更内容は親プロセスの属性に反映されません。

フラグのビット1は、親プロセスのどの属性を子プロセスが継承するかを制御します。

初期の実行単位を含む実行単位と初期実行単位から作成された実行単位のセットは、コルーチンとして知られています。

実行単位を使用する場合は、次の点に気を付けてください。

- 子プロセスのCOBOLプログラムはすべてその初期状態と同じ状態です。
- CBL_GET_SHMEM_PTR、CBL_PUT_SHMEM_PTR、またはCBL_ALLOC_MEMを使用して明示的に指定されていない限り、データを実行単位で共有することはできません。
- オープンされているファイルおよびヒープは、親プロセスから継承されません。

CBL_EXIT_PROC

アプリケーションが終了するとき自動的に呼び出される閉止手続きを導入または削除します。

構文:

```
CALL "CBL_EXIT_PROC" USING  導入フラグ
                            導入パラメータ
                  RETURNING  状態コード
```

パラメータ:

導入フラグ　PIC X COMP-X.

導入パラメータ

導入アドレス　USAGE PROCEDURE POINTER

導入優先順位　PIC X COMP-X.

状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

導入フラグ　実行される処理を示します:

0=省略時の優先順位64の閉止手続きの導入

1=閉止手続きの導入解除

2=導入された閉止手続きの優先順位問い合わせ

3=指定した優先順位での閉止手続きの導入

導入アドレス　導入、導入解除、または問い合わせをする閉止手続きのアドレス

導入優先順位　導入フラグが3の場合、導入されている閉止手続きの0から127までの範囲の優先順位が入る。それ以外の優先順位は無視される。

出口上のパラメータ:

導入優先順位　導入フラグが2の場合、選択された閉止手続きの優先順位を返す。

説明:

アプリケーションが正常に終了しようと（STOP RUNにより）、異常終了しようと（中断のためのキー順（Ctrl+Break）、RTSエラー等による）、手続きは実行されます。本ルーチンを繰返し呼び出すことにより、アプリケーション用にいくつかの閉止手続きを導入できます。

閉止手続きは任意の言語で書くことができます。COBOLで書く場合、導入アドレスは入口点のアドレスとする必要があります。次の文を用いてこのアドレスを得ることができます。

set　導入アドレス　**to entry**　入口名

COBOLで書かれた閉止手続きは、CALL文を含む任意の正当なCOBOLを含むことができます。手続きの主プログラムがEXIT PROGRAM/GOBACKを行うと、あるいは、STOP RUN文が実行されると、閉止手続きは終了します。

閉止手続きの呼び出し時はパラメータが渡されません。

導入されている閉止手続きすべてに優先順位がつけられています。この優先順位は、手続きを実行する順番を決めるものです。つまり、最も小さい値の手続きが最初に処理されます。同じ優先順位を持つ手続きがいくつかある場合は、導入の順序と逆の順序で実行されます。つまり、最後に導入された手続きが最初に実行されます。

優先順を指定せずに（導入フラグ=0）導入された手続きには64という省略時の優先順位が与えられます。優先順位を設定するには、導入フラグに3を設定して優先順位に0から127までの値を設定します。128から256の優先順位は、COBOLシステムでは逆順に解釈されます。したがって、127よりも高い数字を優先順位に設定しないでください。ただし、プログラムがユーザが書いたファイルハンドラの場合は例外で、この場合は200を設定する必要があります。

閉止手続きがすでに導入されている場合に、再導入しようとすると失敗します。ただし、再導入の際に使用される優先順位が最初に導入したときの優先順位と異なっていれば、その優先順位が変更されます。この場合、RETURN-CODEはゼロに設定されます。

それまでに導入されている手続きの優先順位を調べるには、導入フラグに2を設定します。これにより、見つかれば導入優先順位にその優先順位が返されます。手続きは導入アドレスによって識別されます。手続きが見つからない場合は、RETURN-CODEにはゼロ以外の値が設定されます。

CBL_FILENAME_CONVERT

スペースで終わっているファイル名とNULLで終わっているファイル名間の変換を行います。

構文:

```
CALL "CBL_FILENAME_CONVERT" USING        ファイル名フラグ
                                         入力ファイル名
                                         出力ファイル名
                                 BY VALUE    入力ファイル名長
                                 BY VALUE    出力ファイル名長
```

パラメータ:

ファイル名機能　PIC X COMP-X.

入力ファイル名　PIC X（n）.

出力ファイル名　PIC X（n）.

入力ファイル名長　PIC X（4）COMP-5.

出力ファイル名長　PIC X（4）COMP-5.

入口上のパラメータ:

ファイル名フラグ　新しい実行単位をどのように作成するかを次のように設定します。

ビット	値	意味
0	0	スペースで終わるファイル名からNULLで終わるファイル名に変換します。
	1	NULLで終わるファイル名からスペースで終わるファイル名に変換します。
1	0	ビット0が0に設定されている場合、入力ファイル名の長さはファイル名の最後にスペース終了文字を使用して決められます。
	1	ビット0が0に設定されている場合、入力ファイル名の長さは「入力ファイル名長」により決められます。文字はすべてファイル名の一部として扱われます。
2	0	ファイル名を大文字にフォールドしません。
	1	ファイル名を大文字にフォールドします。
3	0	ビット0に2を設定することによって決められた動作
	1	ビット0に2を設定しても無視され、入力ファイル名の長さは、ファイル名をスペースで終わるファイル名として走査

することにより決められます。また、下記で説明するように他の終了文字のリストを指定することもできます。

4-7　予約済み。ゼロを設定する必要があります。

入力ファイル名　入力するファイル名

出力ファイル名　ファイル名フラグのビット3に1が設定されている場合、ファイル名が引用符で囲まれていなければその文字列のリスト。追加文字がなければNULLポインタが渡されます。

入力ファイル名長　入力するファイル名の長さ

出力ファイル名長　ファイル名フラグのビット3に0が設定されている場合、変換されたファイル名用のバッファサイズ。

ファイル名フラグのビット3に1が設定されている場合、出力ファイル名（指定されている場合は）の文字数。

出口上のパラメータ:

出力ファイル名　ファイル名フラグのビット3に0が設定されている場合、変換されたファイル名。入力ファイル名が不正な値である場合、本パラメータには空のファイル名が設定されます。（ビット0が0に設定されている場合はNULL、それ以外の場合はスペースが入ります。）

リターンコード　0の場合、エラーが発生。0以外の場合、入力または出力されるスペースで終わる名前の長さ。

説明:

スペースで終わるファイル名とは、スペース文字で終端されるファイル名です。ファイル名に埋め込みスペースが含まれている場合、二重引用符でファイル名を囲むことによってこれを設定することができます。

NULLで終わるファイル名では、二重引用符は無視されますが、埋め込みスペースを保持します。

CBL_FILENAME_MAX_LENGTH

ランタイムシステムおよび使用しているオペレーティングシステムが処理できるファイル名の最大サイズを返します。このサイズには、NULLまたはスペースなどの終了文字は含まれません。

構文:

```
CALL "CBL_FILENAME_MAX_LENGTH"
```

パラメータ:　なし

CBL_FLUSH_FILE

ファイル用のファイルバッファすべてをディスクに書き込まれるようにします。このバッファには、ランタイムシステムバッファおよびオペレーティングシステムバッファの両方がふくまれます。

構文:

```
CALL "CBL_FLUSH_FILE" USING  ファイルハンドル
```

パラメータ:

ファイルハンドル　　PIC X（4）

入口上のパラメータ:

ファイルハンドル　　（CBL_OPEN_FILEによって）ファイルがオープンされたときに返されるファイルハンドル

出口上のパラメータ:

RETURN-CODE　ルーチンが成功したかどうかを示します。

0　成功
-1　バッファがフラッシュした
その他　エラーを示す2バイトのファイル状態コード

説明:

オペレーティングシステムの中には、ファイルバッファをフラッシュできないものもあります。この場合、適切な状態コードが返されます。

呼出しが成功したかどうかは、RETURN-CODEを調べるとわかります。

CBL_FREE_DYN_MEM

CBL_ALLOC_DYN_MENルーチンにより動的に割り付けられたメモリを解放します。

構文:

```
CALL "CBL_FREE_DYN_MEM" USING BY VALUE  メモリポインタ
                        RETURNING  状態コード
```

パラメータ:

メモリポインタ　　USAGE POINTER.レベル01にする必要があります。
状態コード　　「用語説明」の項を参照してください。

入口上のパラメータ:

メモリポインタ　　メモリがCBL_ALLOC_DYN_MEMを用いて割り付けられたとき戻されたポインタ

出口上のパラメータ:　なし

説明:

本ルーチンはCBL_ALLOC_DYN_MEMルーチンを介して割り付けられたメモリを解放します。

CBL_FREE_MEM

動的に割り付けられたメモリを解放します。

構文:

```
CALL "CBL_FREE_MEM" USING BY VALUE  メモリポインタ
                    RETURNING  状態コード
```

パラメータ:

メモリポインタ　USAGE POINTER.

状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

メモリポインタ　メモリがCBL_ALLOC_MEMを用いて割り付けられたとき戻されたポインタ

出口上のパラメータ:　なし

説明:

本ルーチンはCBL_ALLOC_MEMルーチンを介して割り付けられたメモリを解放します。

CBL_FREE_SHMEM

動的に割り付けられた共有メモリを解放します。

構文:

```
CALL "CBL_FREE_SHMEM" USING BY VALUE  メモリポインタ
                      RETURNING  状態コード
```

パラメータ:

メモリポインタ　USAGE POINTER.

状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

メモリポインタ　メモリがCBL_ALLOC_SHMEMを用いて割り付けられたとき戻されたポインタ

出口上のパラメータ:　なし

説明:

本ルーチンはCBL_ALLOC_SHMEMルーチンを介して割り付けられたメモリを解放します。

CBL_FREE_RECORD_LOCK

ファイルに対するレコードロックを解除します。

構文:

```
CALL "CBL_FREE_RECORD_LOCK" USING  ファイルハンドル
                                   レコード相対番地
                                   レコード長
                                   予約済み
```

パラメータ:

ファイルハンドル　　PIC X（4）COMP-X.
レコード相対番地　　PIC X（4）COMP-X.
レコード長　　PIC X（4）COMP-X.
予約済み　　PIC X（2）COMP-X.
RETURN-CODE　　PIC X（2）COMP-X.

入口上のパラメータ:

ファイルハンドル　　CBL_OPEN_FILEまたはCBL_CREATE_FILE呼出しが成功した場合に返されるハンドル
レコード相対番地　　ロックを解除する先頭バイトに対するファイルの先頭の相対位置
レコード長　　ロックを解除するバイトの数。レコード相対番地から数えます。
予約済み　　予約済み。ゼロに設定する必要があります。

出口上のパラメータ:

RETURN-CODE　　0　ロックの解除が成功しました。
9/nnn　エラー状態を示すファイル状態コード

CBL_FREE_THREAD_MEM

動的に割り付けられたメモリを解放します。

構文:

```
CALL "CBL_FREE_THREAD_MEM" USING BY VALUE メモリポインタ
                    RETURNING 状態コード
```

パラメータ:

メモリポインタ　　USAGE POINTER.
状態コード　　「用語説明」の項を参照してください。

入口上のパラメータ:

メモリポインタ　　CBL_ALLOC_THREAD_MEMによってメモリが割り付けられた場合に返されるポインタ

出口上のパラメータ:　なし

説明:

本ルーチンは、CBL_ALLOC_THREAD_MEMルーチンによって割り付けられたメモリを解放します。

CBL_GET_CSR_POS

カーソル位置を戻します。

構文:

```
CALL "CBL_GET_CSR_POS" USING 画面位置
                    RETURNING 状態コード
```

パラメータ:

画面位置　次のように定義された集団項目
行番号　PIC X COMP-X.
カラム番号　PIC X COMP-X.
状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:　なし

出口上のパラメータ:

画面位置　カーソルの画面位置。左上の角が行0、カラム0です。カーソルが見えない場合は、行番号とカラム番号両方とも255に設定されています。

CBL_GET_CURRENT_DIR

カレントディレクトリを返します。

構文:

```
CALL "CBL_GET_CURRENT_DIR" USING BY VALUE  フラグ
                           BY VALUE     名前長
                           BY REFERENCE  ディレクトリ名
                     RETURNING  状態コード
```

パラメータ:

フラグ　PIC X（4）COMP-5.
名前長　PIC X（4）COMP-5.
ディレクトリ名　X（n）
状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:　なし

フラグ　予約済み。ゼロに設定する必要があります。
名前長　「ディレクトリ名」パラメータのバイト単位のサイズ。本パラメータには、返すことのできる最長のディレクトリ名のサイズを設定する必要があります。

出口上のパラメータ:

ディレクトリ名　スペースで終わる形式の現行作業ディレクトリが入ります。ディレクトリ名に埋め込みスペースが含まれている場合、二重引用符で囲まれて返されます。本パラメータは、これらの引用符も保持できるほどの長さでなければなりません。
状態コード　次のコードが返されます。
0　成功。ディレクトリ名には現行作業ディレクトリが入ります。
128　「名前長」で指定されたディレクトリ名のサイズが足りないため、現行作業ディレクトリの名前を入れることができません。

説明:

「名前長」パラメータの適合性をチェックするのは、呼出しプログラムの役割です。つまり、本ルーチンではパラメータをチェックしません。このパラメータを渡すには、「BY VALUE LENGTH OF ディレクトリ名 SIZE 4」のように使用してください。

CBL_GET_EXIT_INFO

アプリケーションに終了手続きが呼び出された状況を知らせます。

構文:

```
CALL "CBL_GET_EXIT_INFO" USING      パラメータブロック
                         RETURNING  状態コード
```

パラメータ:

パラメータブロック　次のように定義された集団項目

Pブロックサイズ　PIC X（4）COMP-5.値16

Pリターンコード　PIC X（4）COMP-5.

PRTSエラー　PIC X（4）COMP-5.

P終了フラグ　PIC X（4）COMP-5.

状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

Pブロックサイズ　本フィールドを含むパラメータブロックのサイズ（16を設定する必要があります）

出口上のパラメータ:

Pリターンコード　RETURN-CODEの現在の値

PRTSエラー　終了の原因となったランタイムシステムエラーの番号（エラーがなければ0が入る）

P終了フラグ　次のようにビットを設定すると、追加終了情報が提供されます。

ビット	値	意味
0	0	アニメータ環境で実行されていません。
	1	アニメータ環境で実行されています。
1	0	
	1	アニメータからの抜けるためです。
2	0	
	1	STOP RUNによる終了です。
3	0	COBOLプログラムにおける終了です。
	1	COBOL以外のプログラムにおける終了です。
4	0	
	1	Ctrl+Break、Ctrl+C、または同様のプログラムを終了させる機構による終了です。
5	0	

1　　スレッドの終了中に呼び出された終了手続きによる終了です。

6-31　　予約済み。ゼロを設定する必要があります。

状態コード　　0　成功

1006　APIが終了手続きの外から呼ばれました。

1009　APIに渡されたパラメータが不正です。

説明:

本ルーチンは、CBL_EXIT_PROCを使用して導入された終了手続き内からしか呼び出すことができません。

本ルーチンは一般的に終了手続きがどのような状況で呼び出されたかを知る必要があるデータアプリケーションにおいて使用されます。データアプリケーションは、一般的には終了手続きが呼び出されたのは、正常なランタイムシステムの終了によるものか、または異常なランタイムシステムの終了によりものかを知る必要があります。本ルーチンにより、そのようなデータベースアプリケーションはコミットするかロールバックするかを決めることができます。

CBL_GET_KBD_STATUS

キーボードから読み込まれるのを待っている文字があるかどうかチェックします。

構文:

```
CALL "CBL_GET_KBD_STATUS" USING  キー状態
                       RETURNING  状態コード
```

パラメータ:

キー状態　　PIC X COMP-X.

状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:　なし

出口上のパラメータ:

キー状態　　0=文字なし

1=文字あり

説明:

CBL_キーボードルーチンをx"AF"キーボードルーチンと共に使用することはできません。

「キーボードルーチンの使用例」

CBL_GET_MOUSE_MASK

マウス事象マスクを戻します。

構文:

```
CALL "CBL_GET_MOUSE_MASK" USING  マウスハンドル
                                 事象マスク
```

RETURNING **状態コード**

パラメータ:

マウスハンドル PIC X（4）COMP-X.
事象マスク PIC X（2）COMP-X.
状態コード 「用語説明」の項を参照してください。

入口上のパラメータ:

マウスハンドル マウス一意名で、CBL_INIT_MOUSEに対する前の呼出しにより獲得されます

出口上のパラメータ:

事象マスク 「カテゴリー別ルーチン」の項の「マウス」を参照してください。

説明:

CBL_マウスルーチンをx"AF"マウスルーチンと共に使用することはできません。

CBL_GET_MOUSE_POSITION

マウスポインタの画面位置を戻します。

構文:

```
CALL "CBL_GET_MOUSE_POSITION" USING マウスハンドル
                                    マウス位置
                              RETURNING 状態コード
```

パラメータ:

マウスハンドル PIC X（4）COMP-X.
マウス位置 次のように定義された集団項目:
マウス行 PIC X（2）COMP-X.
マウスカラム PIC X（2）COMP-X.
状態コード 「用語説明」の項を参照してください。

入口上のパラメータ:

マウスハンドル マウス一意名で、CBL_INIT_MOUSEに対する前の呼出しにより獲得されます。

出口上のパラメータ:

マウス位置 マウスポインタの画面位置

説明:

CBL_マウスルーチンをx"AF"マウスルーチンと共に使用することはできません。

CBL_GET_MOUSE_STATUS

待ち行列内の事象数を戻します。

構文:

```
CALL "CBL_GET_MOUSE_STATUS" USING  マウスハンドル
                                   待機事象数
                         RETURNING  状態コード
```

パラメータ:

マウスハンドル　PIC X（4）COMP-X.

待機事象数　PIC X（2）COMP-X.

状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

マウスハンドル　マウス一意名で、CBL_INIT_MOUSEに対する前の呼出しにより獲得されます。

出口上のパラメータ:

待機事象数　待ち行列内の事象数

説明:

CBL_マウスルーチンをx"AF"マウスルーチンと共に使用することはできません。

CBL_GET_OS_INFO

オペレーティングシステム環境についての情報を返します。

構文:

```
CALL "CBL_GET_OS_INFO" USING  パラメータブロック
                    RETURNING  状態コード
```

パラメータ:

パラメータブロック　次のように定義された集団項目:

- **パラメータサイズ**　PIC X（2）COMP=X VALUE 15.
- ***p-OS*種類**　PIC X COMP-X.
- ***p-OS*バージョン**　PIC X（4）COMP-X.
- ***p-DBCS*サポート**　PIC X COMP-X.
- ***p*-文字コード**　PIC X COMP-X.
- ***p*-国*ID***　PIC X（2）COMP-X.
- ***p*-コードページ**　PIC X（2）COMP-X.
- ***p*-処理の種類**　PIC X COMP-X.
- ***p*-RTS機能**　PIC X COMP-X.

状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:　なし

出口上のパラメータ:

***p-OS*種類**　131=Windows 95およびWindows NT

***p-OS*バージョン**　用途はオペレーティングシステムに固有です。チップ種類やオペレーティングシステムバージョン番号などの情報を含むこともあります。DOSおよびOS/2の場合、第3および第4バイトには副および主リリースのオペレーティングシステムバージョン番号がそれぞれ入ります。

***p-DBCS*サポート**　ビット0=0-DBCS妥当性検査がサポートされてない場合
=1-DBCS妥当性検査がサポートされている場合
ビット1=0-Micro Focus PIC Nデータ型がサポートされていない場合
=1-Micro Focus PIC Nデータ型がサポートされている場合

***p*-文字コード**　0=ASCII
1=シフトJIS
2=EUC

p*-国*ID　国番号。詳細については、オペレーティングシステムのマニュアルを参照してください。

***p*-コードページ**　コードページ。詳細については、オペレーティングシステムのマニュアルを参照してください。

***p*-処理の種類**　0=全画面セッションで実行する処理
3=真の図形アプリケーションとして実行する処理

***p*-RTS機能**　ビット0=0-ランタイムシステムがスレッドシステムである場合
=1-ランタイムシステムがスレッドシステムでない場合

CBL_GET_PROGRAM_INFO

指定されたプログラム、または現在呼出しスタック内にあるプログラムの情報を返します。

構文:

```
CALL "CBL_GET_PROGRAM_INFO" USING BY VALUE 機能
                  BY REFERENCE  パラメータブロック
                  BY REFERENCE  名前バッファ
                  BY REFERENCE  名前長
                  RETURNING  状態コード
```

パラメータ:

機能 pic x（4）comp-5.

パラメータブロック　次のように定義された集団項目

```
    サイズ              pic x（4）comp-5.
    フラグ                  pic x（4）comp-5.
    ハンドル                    ポインタ
    プログラムハンドル          ポインタ
    属性                        pic x（4）comp-5.
```

名前バッファ　pic x（n）.
ユーザデータ長　pic x（4）comp-5.
状態コード　**「用語説明」の項を参照してください。**

入口上のパラメータ:

機能　次のいずれか1つを設定します。

0　現在のプログラムの状態情報を返します。このプログラムに関する詳細な情報を取り出したい場合には、この機能が「ハンドル」を返すように「フラグ」パラメータのビット0を設定する必要があります。「ハンドル」は、機能2～7の入力として使用されるものです。「ハンドル」に関連する初期化プログラムが現在のプログラムです。

1　指定されたプログラムの状態情報を返します。このプログラムに関する詳細な情報を取り出したい場合には、この機能が「ハンドル」を返すように「フラグ」パラメータのビット0を設定する必要があります。「ハンドル」は、機能2～7の入力として使用されるものです。「ハンドル」に関連する初期化プログラムが指定されたプログラムです。

2　現在「ハンドル」に関連するプログラムを呼び出したプログラムの状態情報を返します。この機能は、機能0または1で設定された「ハンドル」を必要とします。「ハンドル」に関連するプログラムは、呼出しプログラムに更新されます。

3　前回作成された「ハンドル」をクローズします。この機能は、機能0または1で設定された「ハンドル」を必要とします。この機能による呼出しは、「ハンドル」が終了した時点行う必要があります。

4　現在「ハンドル」に関連するプログラムの入口点を探します。「ハンドル」は、機能0または1で設定する必要があります。この機能により一度呼出しを行うと、機能5により本ルーチンを繰り返し使用してプログラム内の残りの入口点をすべて取り出すことができます。必要な入口点がすべて一度返されると、機能6により本ルーチンを使用して入口点の検索を終了することができます。

5　現在「ハンドル」に関連するプログラムの次の入口点を探します。この機能は、機能0または1で設定された有効な「ハンドル」と機能4用に最初または次の入口点の検索シーケンスを初期化するために使用された「ハンドル」を必要とします。

6　入口点の検索を終了します。この機能は、機能0または1で設定された有効な「ハンドル」と機能4用に最初または次の入口点の検索シーケンスを初期化するために呼び出された「ハンドル」を必要とします。

7　現在「ハンドル」に関連するプログラムの完全なプログラム名を返します。この機能は、機能0または1で設定された有効な「ハンドル」を必要とします。

サイズ　本フィールドを含むパラメータブロックのサイズ。20を設定する必要があります。

フラグ　次のビットを設定すると必要な情報が得られます。

ビット	意味
0	機能0および1が「ハンドル」を返すかどうかを決めます。他の機能では、このビットは無視されます。 0　「ハンドル」を返しません。（機能0ま

たは1の場合）

1 「ハンドル」を返します。（機能0または1の場合）

1 機能0および2がプログラム状態情報の一部として、名前バッファにプログラムの基本名を返すかどうかを決めます。

0 プログラムの基本名を返しません。（機能0または2の場合）

1 プログラムの基本名を返します。（機能0または2の場合）

2 名前バッファに入力される、または返される基本名および完全なプログラム名がNULLまたはスペースのどちらで終わる形式にするか決めます。

0 入出力名はすべてスペースで終わります。

1 入出力名はすべてNULLで終わります。

3 機能0、1、および2がプログラムの属性を「プログラム属性」に返すかどうかを決めます。

0 プログラム属性を返しません。

1 プログラム属性を返します。

4-31 将来のために予約済み。0を設定する必要があります。

ハンドル 機能0または1によって作成されるハンドル

名前バッファ 機能1の場合に、情報を必要とするプログラムの基本名。

名前長 名前バッファの長さ。この長さが小さすぎて情報が返されない場合、ルーチンは失敗します。

出口上のパラメータ:

ハンドル 機能0または1によって返されるハンドル。これは、他のすべての機能でも使用する必要があります。

プログラムハンドル 現在「ハンドル」に関連するプログラムのユニークな識別子

プログラム属性 現在「ハンドル」に関連するプログラムの属性

ビット	値	意味
0	0	
	1	AMODE(24)コンパイラ指令（この指令は、現在NetExpressでは使用できません）を使用してコンパイルされたプログラム
1	0	
	1	AMODE(31)コンパイラ指令（この指令は、現在NetExpressでは使用できません）を使用してコンパイルされたプログラム
2	0	CHARSET(ASCII)コンパイラ指令を使用してコンパイルされたプログラム
	1	CHARSET(EBCDIC)コンパイラ指令を使用してコンパイルされたプログラム

3	0	
	1	ANS85コンパイラ指令を使用してコンパイルされたプログラム
4	0	
	1	VSC2コンパイラ指令（この指令は、現在NetExpressでは使用できません）を使用してコンパイルされたプログラム
5	0	
	1	OSVSコンパイラ指令（この指令は、現在NetExpressでは使用できません）を使用してコンパイルされたプログラム
6-27	0	将来のために予約済み。0を設定する必要があります。
	1	
28	0	
	1	サブシステムのメンバーであるプログラム
29	0	現在の呼出しスタックにあるプログラム
	1	現在の呼出しスタックにないプログラム
30	0	
	1	プログラムは取り消されました
31	0	COBOLプログラム
	1	COBOL以外のプログラム

名前バッファ　要求された基本名、入口点の名前、または完全なプログラム名。

名前長　名前バッファに返された名前の長さ、または状態コードが1013の場合に要求されたバッファの長さ。

状態コード

0	成功
500	情報の終り（現在関連するプログラムの呼出しプログラムがない場合に機能2によって返されるか、これ以上入口点がない場合に機能5によって返される）
1001	入力された「ハンドル」が不正です
1009	パラメータが不正です
1013	バッファが小さすぎて情報を返すことができません

CBL_GET_RECORD_LOCK

ファイルに対してレコードロックをかけます。

構文:

```
CALL "CBL_GET_RECORD_LOCK" USING  ファイルハンドル
                                  レコード相対番地
                                  レコード長
                                      予約済み
```

パラメータ:

ファイルハンドル　PIC X（4）COMP-X

レコード相対番地　　PIC X（4）COMP-X

レコード長　　PIC X（4）COMP-X

予約済み　　PIC X（2）COMP-X

RETURN-CODE　　PIC X（2）COMP-X

入口上のパラメータ:

ファイルハンドル　　CBL_OPEN_FILEまたはCBL_CREATE_FILE呼出しが成功した場合に返されるハンドル

レコード相対番地　　ロックする先頭バイトに対するファイルの先頭の相対位置

レコード長　　ロックするバイトの数。レコード相対番地から数えます。

予約済み　　予約済み。ゼロに設定する必要があります。

出口上のパラメータ:

RETURN-CODE　　0　　ロックが成功しました。

1　　この実行単位ではロックがすでにかけられています。

9/nnn　　エラー状態を示すファイル状態コード

CBL_GET_SCR_GRAPHICS

図形文字用オペレーティングシステム独立コードを戻します。

構文:

```
CALL "CBL_GET_SCR_GRAPHICS" USING  図形パラメータ
                          RETURNING  状態コード
```

パラメータ:

図形パラメータ　　次のように定義された集団項目:

　図形フラグ　　PIC X COMP-X.

　図形バッファサイズ　　PIC X（2）COMP-X.

　図形バッファ　　次のように定義された集団項目:

　　dbcs上矢印文字　　次のように定義された集団項目:

　　　上矢印文字*dbcs*フラグ　　PIC X COMP-X.

　　　上矢印文字　　PIC X.

　　***dbcs*下矢印文字**　　次のように定義された集団項目:

　　　下矢印文字*dbcs*フラグ　　PIC X COMP-X.

　　　下矢印文字　　PIC X.

　　***dbcs*右矢印文字**　　次のように定義された集団項目:

　　　右矢印文字*dbcs*フラグ　　PIC X COMP-X.

　　　右矢印文字　　PIC X.

　　***dbcs*左矢印文字**　　次のように定義された集団項目:

　　　左矢印文字*dbcs*フラグ　　PIC X COMP-X.

　　　左矢印文字　　PIC X.

***dbcs*最大化文字**	次のように定義された集団項目:
最大化文字*dbcs*フラグ	PIC X COMP-X.
最大化文字	PIC X.
***dbcs*最小化文字**	次のように定義された集団項目:
最小化文字*dbcs*フラグ	PIC X COMP-X.
最小化文字	PIC X.
***dbcs*チェック文字**	次のように定義された集団項目:
チェック文字*dbcs*フラグ	PIC X COMP-X.
チェック文字	PIC X.
***dbcs*復元文字**	次のように定義された集団項目:
復元文字*dbcs*フラグ	PIC X COMP-X.
復元文字	PIC X.
***dbcs*無線文字**	次のように定義された集団項目:
無線文字*dbcs*フラグ	PIC X COMP-X.
無線文字	PIC X.
***dbcs*スクロール文字**	次のように定義された集団項目:
スクロール文字*dbcs*フラグ	PIC X COMP-X.
スクロール文字	PIC X.
***dbcs*ラバーバンド文字**	次のように定義された集団項目:
ラバーバンド文字*dbcs*フラグ	PIC X COMP-X.
ラバーバンド文字	PIC X.
***dbcs*システムメニュー文字**	次のように定義された集団項目:
システムメニュー文字*dbcs*フラグ	PIC X COMP-X.
システムメニュー文字	PIC X.
***dbcs*エディタ図形**	次のように定義された集団項目:
エディタ図形*dbcs*フラグ	PIC X COMP-X.
エディタフラグ	PIC X.
***dbcs-hyhelp*タブ文字**	次のように定義された集団項目:
***hyhelp*タブ文字*dbcs*フラグ**	PIC X COMP-X.
***hyhelp*タブ文字**	PIC X.
***dbcs-hyhelp-btab*文字**	次のように定義された集団項目:
***hyhelp-btab*文字*dbcs*フラグ**	PIC X COMP-X.
***hyhelp-btab*文字**	PIC X.
状態コード	「用語説明」の項を参照してください。

入口上のパラメータ:

図形フラグ	戻す文字コードの種類: 0-ホスト環境に適した1バイトまたは2バイト文字集合 1-1バイト文字集合（SBCS）のみ

2-2バイト文字集合（DBCS）のみ

図形バッファサイズ　図形バッファの長さ（バイト）。より小さな値を指定することによって、必要とする図形値のみを要求することはできますが、図形バッファサイズの通常の値は30です。

図形バッファサイズを図形バッファの長さより大きく設定してはいけません。図形バッファサイズが図形バッファの長さを越えると、ランタイムシステムは宣言されているバッファサイズを越えてメモリへの書出しを継続します。これは他のデータの損傷を招くことがあります。

出口上のパラメータ:

図形バッファサイズ　充てんされるバイト数

図形バッファ　図形バッファサイズバイト数に完全に含まれている各記述項ごとの文字コード。部分的にしかそのバイト数以内に納まらない任意の文字コードの内容は不定です。完全にそのバイト数外にある文字コードはどれも無視されます。

*dbcs*フラグデータ項目の場合、ゼロは1バイト文字コードが返されたことを示しており、ゼロ以外の値は返される2バイト文字の第1バイトです。

状態コード　戻り状態:

0=成功

1=要求したコードが必ずしも全て利用できたわけではなかったことを示します。図形バッファサイズには戻されたバイト数が入ります。

2=図形バッファが指定された文字コード（SBCSまたはDBCS）で充てんできなかったことを示します。この場合、あたかも図形フラグはゼロに設定されているかのように、利用できなかった文字は充てんされました。

3=状態コード=1および2によって知らされるエラーが両方とも発生しました。

CBL_GET_SCR_LINE_DRAW

与えられた線画文字を正しく表示するために、ランタイムシステム画面出力ルーチンにより要求された文字値を定義する値の表を戻します。

構文:

```
CALL "CBL_GET_SCR_LINE_DRAW" USING  関数コード
                                    線画表
                            RETURNING  状態コード
```

パラメータ:

関数コード　PIC X COMP-X.

次のサブルーチン番号のどれか1つが入っています:

0=1バイト線画表を要求

1=2バイト線画表を要求

2=個々の1バイト線画コードを要求

3=個々の2バイト線画コードを要求

線画表　次のように定義された集団項目

関数コード=0用

線画コード　PIC X COMP-X.

線画文字　PIC X OCCURS 256 TIMES.

関数コード=1用

線画コード　PIC X COMP-X.

dbcs線画文字　PIC X（2）OCCURS 256 TIMES.

関数コード=2用

線画コード　PIC X COMP-X.

線画文字　PIC X.

関数コード=3用

線画コード　PIC X COMP-X.

dbcs線画文字　PIC X（2）.

状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

線画コード　2または3

線画コード　文字の相対番地（形状記述）

出口上のパラメータ:

線画コード　以下のような表内で発生した任意の写像を確定するためにランタイムシステムにより充てんされるバイト。

ビット0=単厚の線はサポートされていません-写像されています

ビット1=二重厚の線はサポートされていません-写像されています

ビット2=拡張型の線はサポートされていません-写像されています

ビット3=予約済み（0に設定）

ビット4=予約済み（0に設定）

ビット5=予約済み（0に設定）

ビット6=予約済み（0に設定）

ビット7=予約済み（0に設定）

ビット0～2については、線種は単厚、二重厚、拡張あるいはASCII文字のうち最初に使用可能となったものに写像されます。

線画文字

関数コード=0用　1バイト線画表。表内のバイトは次のように配列されています。

ビット0と1=東方向の線

ビット2と3=西方向の線

ビット4と5=南方向の線

ビット6と7=北方向の線

関数コード=1用　　2バイト線画表。表内のバイトは次のように配列されています。

ビット0と1=東方向の線

ビット2と3=西方向の線

ビット4と5=南方向の線

ビット6と7=北方向の線

関数コード=2用　　要求された個々の1バイト線画コード

関数コード=3用　　要求された個々の2バイト線画コード

説明:

個々の線画文字（関数コード=2または3）を要求している場合、どの文字を必要としているのかをランタイムシステムに知らせるのに、線画コードパラメータが使用されます。与えられた線画形状用の線画コードパラメータは、次のように計算できます。

バイトを4ビットペアに分割することによって値は得られます。各ビットペアは、4方向の内のいずれかの方向に引く線の種類を定義します。

ビット:　76|54|32|10（最下位ビット）

北|南|西|東

方向は文字の中心から定義されます。

各ビットペアは次の値のいずれかをとります:

00-与えられた方向には線がないことを示します。

01-単厚の線があることを示します。

10-二重厚の線があることを示します。

11-拡張型の線があることを示します。

点線のような付加的な種類の線引きがあるシステムにおいてのみ、拡張線種は使用可能です。システムに2線種以上ある場合、一度には1種類しか使用できません。

例

側線が単厚で底線が二重厚の箱の左下を定義する画面ハンドラの線用文字コードを知りたい場合は、次のような計算をします。

ビットペア:

北:01（単）

南:00（空白）

西:00（空白）

東:10（二重）

これは2進値01000010または10進値66を意味します。従って、この形状のための正しい文字を得るには、CBL_GET_SCR_LINE_DRAWを呼び出す前に66を「線画コード」に転記することになります。

完全な線引き表を得るために関数コード=0または1を使用すると、与えられた文字は上記のビットペアアルゴリズムを使用して表から見つけられます。このビットペアアルゴリズムは与えられた形状用の数値を生成します。しかし、省略時は、COBOL添字はゼロではなく1で始まります。従って、

```
03  線画文字          PIC X OCCURS 256 TIMES.
```

完全な線画表を得るために上記のような構成体を使用すると、正しい文字を見つけるために、ビットペアアルゴリズムの結果に1を加える必要があります（そして、これにより、1バイトより大きな添字が必要になります）。

しかし、完全な線画表を得るために次のような構成体を使用できます。

関数コード=0として:

```
03  filler          PIC X.
03  線画文字        PIC X OCCURS 255 TIMES.
```

関数コード=1として:

```
03  filler          PIC X（2）.
03  DBCS線画文字    PIC X（2） OCCURS 255 TIMES.
```

NOBOUNDコンパイラ指令が設定されて、ビットペアアルゴリズムの結果に直接写像する添字0～255を使用できるようにしておく必要があります。

CBL_GET_SCR_SIZE

画面のサイズについての情報を戻します。

構文:

```
CALL "CBL_GET_SCR_SIZE" USING  縦長さ
                               横幅
                     RETURNING  状態コード
```

パラメータ:

縦長さ	PIC X COMP-X.
横幅	PIC X COMP-X.
状態コード	「用語説明」の項を参照してください。

入口上のパラメータ: なし

出口上のパラメータ:

縦長さ	行数
横幅	カラム数

CBL_GET_SHMEM_PTR

名前付き値を読み込みます。

構文:

```
CALL "CBL_GET_SHMEM_PTR" USING  ノード値
                                ノード名
```

RETURNING **状態コード**

パラメータ:

ノード値 USAGE POINTER

ノード名 次のように定義された集団項目

名前長 PIC X COMP-5 値n

名前 PIC X(n) 値"名前"

状態コード 「用語説明」の項を参照してください。

入口上のパラメータ:

ノード名 名前の長さ

名前 ノード名に割り当てられた値

出口上のパラメータ:

ノード値 名前付き値の値

説明

名前付き値は、実行時に同意を得た名前を使用して異なる実行単位間でポインタを渡す方法を提供します。コルーチン内のすべての実行単位は名前付き値を同時に読み込むことができます。それは、ランタイムシステムが更新を一切禁止し、逐次化しているからです。名前付き値の最大数は、マシンのメモリサイズによって違ってきます。

CBL_HIDE_MOUSE

マウスポインタを見えないようにします。

構文:

```
CALL "CBL_HIDE_MOUSE" USING  マウスハンドル
                     RETURNING  状態コード
```

パラメータ:

マウスハンドル PIC X（4）COMP-X.

状態コード 「用語説明」の項を参照してください。

入口上のパラメータ:

マウスハンドル マウス一意名で、CBL_INIT_MOUSEに対する前の呼出しにより獲得されます。

出口上のパラメータ: なし

説明:

本ルーチンが呼び出された後も、なお、マウス事象は発生しますが、マウスポインタは表示されません。

CBL_マウスルーチンをx"AF"マウスルーチンと共に使用することはできません。

CBL_IMP

2つのデータ項目のビット間の論理暗示（IMPLIES）を行います。

構文:

```
CALL "CBL_IMP" USING 源
                     目標
               BY VALUE 長さ
```

パラメータ:

源　　任意のデータ項目

目標　　任意のデータ項目

長さ　　数字定数またはPIC X（4）COMP-5.

入口上のパラメータ:

源　　暗示（IMPLIES）をとるデータ項目の内の1つ

目標　　暗示（IMPLIES）をとるもう一方のデータ項目

長さ　　暗示（IMPLIES）をとろうとする源と目標のバイト数。この長さを越えた目標内の位置については変更はありません。

出口上のパラメータ:

目標　　結果

説明:

本ルーチンは源と目標の左端から開始し、それらのビットの暗示（IMPLIES）をとり、結果を目標に格納します。これの真理値表は次の通りです。

源	目標	結果
0	0	1
0	1	1
1	0	0
1	1	1

CBL_INIT_MOUSE

マウス支援を初期化します。本ルーチンを呼び出さないと他のマウスルーチンを呼び出すことはできません。

構文:

```
CALL "CBL_INIT_MOUSE" USING マウスハンドル
                            マウスボタン
                      RETURNING 状態コード
```

パラメータ:

マウスハンドル　　PIC X（4）COMP-X.

マウスボタン　　PIC X（2）COMP-X.

状態コード　　「用語説明」の項を参照してください。

入口上のパラメータ:　なし

出口上のパラメータ:

マウスハンドル　マウス一意名で、この後呼び出す任意のマウスルーチンに渡します。

マウスボタン　マウス上のボタン数

説明:

このルーチンから返された状態コード（またはRETURN-CODE）の値がゼロ以外の場合、マウスハンドルは使用できません。また、どのマウスルーチンも呼び出せません。

CBL_マウスルーチンをx"AF"マウスルーチンと共に使用することはできません。

CBL_JOIN_FILENAME

ファイル名の構成要素、つまり、パス名、基本名および拡張子を結合してファイル名を形成します。

構文:

```
CALL "CBL_JOIN_FILENAME" USING  分割・結合パラメータ
                                結合バッファ
                                パスバッファ
                                基本名バッファ
                                拡張子バッファ
                          RETURNING 状態コード
```

パラメータ:

分割・結合パラメータ	次のように定義された集団項目
パラメータ長	PIC X（2）COMP-X.
分割・結合フラグ1	PIC X COMP-X.
分割・結合フラグ2	PIC X COMP-X.
パス相対番地	PIC X（2）COMP-X.
パス長	PIC X（2）COMP-X.
基本名相対番地	PIC X（2）COMP-X.
基本名長	PIC X（2）COMP-X.
拡張子相対番地	PIC X(2)COMP-X.
拡張子長	PIC X（2）COMP-X.
全長	PIC X（2）COMP-X.
分割バッファ長	PIC X（2）COMP-X.
結合バッファ長	PIC X（2）COMP-X.
第1パス構成要素長	PIC X（2）COMP-X.
結合バッファ	PIC X（n）.
パスバッファ	PIC X（n）.
基本名バッファ	PIC X（n）.
拡張子バッファ	PIC X（n）.
状態コード	「用語説明」の項を参照してください。

入口上のパラメータ:

パラメータ長　　分割・結合パラメータの長さ（バイト）で、パラメータ長用の2バイトを含みます。パラメータ長の通常の値は24です。

分割・結合フラグ1　次のように設定します

ビット	値	意味
1	1	文字列は空終了
	0	文字列はスペース終了
0	1	新しいファイル名は大文字にフォールドされる
1	1	元の大文字・小文字がそのまま保存される

分割・結合フラグ2　次のように設定します

ビット	値	意味
2	1	分割・結合フラグ1のビット1が0に設定されている場合、各ファイル名の構成要素の長さが、パス長、基本名長、および拡張子長により正確に与えられていることを意味する。このフラグは、ファイル名の構成要素の1つに既知の意味のあるスペースがある場合に使用できる。分割・結合フラグ1のビット1が1に設定されている場合、この設定は無視される。
	0	ファイル名の構成要素の長さは、分割・結合フラグ1のビット1によって決まる。
1		予約済み。
0		予約済み。

パス相対番地　パスバッファ内でのパス名の先頭の相対番地（1から指標付け）

パス長　スペース終了でも空終了でもない場合の装置名の長さ

基本名相対番地　基本名バッファ内での基本名の先頭の相対番地（1から指標付け）

基本名長　スペース終了でも空終了でもない場合の基本名長

拡張子相対番地　拡張子バッファ内での拡張子の先頭の相対番地（1から指標付け）

拡張子長　スペース終了でも空終了でもない場合の拡張子長

パスバッファ　パス名

基本名バッファ　基本名

拡張子バッファ　拡張子

結合バッファ長　結合バッファの長さ

出口上のパラメータ:

全長　ファイル名の合計文字数

結合バッファ　結合ファイル名

状態コード　　戻り状態:

0=成功

1=結合バッファには大きすぎるファイル名

4=違法ファイル名

説明:

新しいファイル名は以下の項目を結合して形成されます。

- 装置バッファの最初のパス長バイト（装置相対番地から開始）
- 基本名バッファの最初の基本名長（基本名相対番地から開始）
- 拡張子バッファの最初の拡張子長（拡張子相対番地から開始）

そして、全長の長さの結合バッファに格納されます。

分割・結合フラグ1の最下位のビット（ビット0）を設定することによって、本ルーチンを大文字にフォールドするようにできます。このビットが設定されていないと、元の大文字・小文字はそのまま保存されます。

本ルーチンは空で終了する文字列・スペースで終了する文字列のいずれも受け取ります。分割・結合フラグ1の最下位ビットの次のビット（ビット1）を設定すると、ルーチンは空で終了する文字列を受け取るものと予想します。このビットが設定されてないと、スペースで終了する文字列を受け取るものと予想します。

パス名、基本名、および拡張子の各項目は、それらがスペースまたは空（分割・結合フラグ1のビット1の設定に依る）で終了していれば、それぞれパス長、基本名長、および拡張子長で指定された長さより短くできます。

パスバッファ、基本名バッファ、拡張子バッファおよび結合バッファは4つの別個のバッファとする必要はありません。これは、ファイル名の1つの構成要素を置き換えるのに、本ルーチンをCBL_SPLIT_FILENAMEと一緒に使用できることを意味しています。

パスバッファが空でなく、後続円記号（¥）または斜線（/）またはコロン（:）が入っておらず、さらに基本名バッファが空でない場合、本ルーチンは結合バッファ内のパス名と基本名の間に¥を挿入します。

拡張子がピリオド（.）の場合、結合バッファに入れて戻される文字列はスペースの拡張子を持っています。つまり、ファイル名には終了ピリオドが付きます。

全長が結合バッファ長より短いと、ファイル名の終わりに続く文字列は分割・結合フラグ1のビット1により空列またはスペース列となります。

パス名が有効なドライブ文字で構成されているがコロンがない場合、本ルーチンがコロンを追加します。コロンを必要としない装置（例えばLPT1）については追加は行いません。装置（例えば、ドライブ文字に対立するものとしてのLPT1）を空でない基本名に結合することはできません。

「CBL_JOIN_FILENAMEの使用例」

CBL_LOCATE_FILE

本ルーチンには2つの用法があります。ファイル指定においては環境変数にいくつかのパスのリストが入っており、この環境変数を拡張するのに本ルーチンを使用できます。また、特定ファイル指定を使用するOPEN INPUTがファイルをライブラリ（ .LBRファイル）内に見つけるか、あるいは、別のディスクファイルとして見つけるかを、本ルーチンで決定できます。

構文:

```
CALL "CBL_LOCATE_FILE" USING  利用者ファイル指定
                              利用者モード
                              実ファイル指定
                              存在フラグ
                              パスフラグ
                      RETURNING  状態コード
```

パラメータ:

利用者ファイル指定	PIC X(n).
利用者モード	PIC X COMP-X.
実ファイル指定	次のように定義された集団項目
バッファ長	PIC X(2) COMP-X.
バッファ	PIC X(n).
存在フラグ	PIC X COMP-X.
パスフラグ	PIC X COMP-X.
状態コード	「用語説明」の項を参照してください。

入口上のパラメータ:

利用者ファイル指定　ファイル名指定が入ります。これは埋め込み環境変数またはライブラリ名を含むことができます。

利用者モード　利用者ファイル指定をどうするかを指定します:

0=ファイルがライブラリに、あるいは、別のディスクファイルとして存在するかどうかチェックします。

利用者ファイル指定が埋め込みライブラリ名を含んでいる場合、そのライブラリは開かれて（存在していれば）、ファイルを捜すために探索されます。その後、そのライブラリは開かれたままです。

利用者ファイル指定が埋め込み環境変数を含んでいる場合、その変数に指定されている各パスに沿ってファイルが捜されます。そのファイルが見つかると、出口上の実ファイル指定は、環境変数を成功したパスに展開したファイル指定を格納します。

そうでない場合、出口上の実ファイル指定は、環境変数をそれに含まれている最初のパスに展開したファイル指定を格納します。

1=利用者ファイル指定が環境変数を含んでいる場合、出口上の実ファイル指定は、環境変数をそれに含まれている最初のパスに展開したファイル指定を格納します。ファイルの探索は行われません。

2=利用者ファイル指定が環境変数を含んでいる場合、出口上の実ファイル指定は、環境変数をそれに含まれている次のパスに展開したファイル指定を格納します。ファイルの探索は行われません。このオプションの使用は、利用者モード=1または2とした呼出しが成功した後に限定する必要があります。次のパスフラグを参照してください。

パスフラグ　　利用者モード=2の場合、このデータ項目には、前の利用者モード=1または2の呼出しから本項目に入れて戻された値が入っている必要があります。

バッファ長　　バッファのサイズ

出口上のパラメータ:

バッファ　　利用者モードのところで説明したように、解決されたファイル指定を格納するバッファ。解決されたファイル指定がバッファ長で指定されたサイズよりも大きい場合、バッファの内容は変更されずに状態コードはその内容に従って設定されます。

存在フラグ　　利用者モード=0の場合、出口上の本データ項目は、利用者ファイル指定に指定されたファイルが存在するかどうか示します。

0=ファイルが見つかりません、または探索されていません
1=既に開かれていたライブラリにファイルが見つかりました。
2=利用者ファイル指定に指定されているライブラリにファイルが見つかりました。
3=別のディスクファイルとしてファイルが見つかりました
利用者モードが0でなければ、本データ項目は出口上で常に0となっています

パスフラグ　　利用者ファイル指定が次のような実ファイル指定に展開された埋め込み環境変数を格納したかどうかを示します。

0=実ファイル指定は展開された環境変数を含みません。
1以上=実ファイル指定は展開された環境変数を含みます。

状態コード　　戻り状態:

0=成功
1=環境変数が存在しません
2=次のパスがありません。
3=解決されたファイル名がバッファサイズを超えています。
4=生成されたファイル名が違法です
255=その他のエラー

例:

利用者ファイル指定は、次のような形式をとることができます。

標準ファイル名:

パス名¥ファイル名 .拡張子

埋め込み環境変数:

$envname¥ファイル名 .拡張子

埋め込みライブラリ名:

パス名¥ライブラリ名 .LBR¥ファイル名 .拡張子

CBL_MONITOR_BROWSE

現行スレッド用に指定されたモニターの走査機能を取得します。本ルーチンは、その機能が取得されるまでスレッドをブロックします。

構文:

```
CALL "CBL_MONITOR_BROWSE" USING BY VALUE モニターハンドル
```

パラメータ:

モニターハンドル　　ポインタ

入口上のパラメータ:

モニターハンドル　　モニターハンドル

出口上のパラメータ:

リターンコード　　値0は呼出しが成功したことを意味し、それ以外の値はエラーを意味します。詳細は、「同期ルーチンのRETURN-CODE値」を参照してください。

説明:

スレッドが異常終了した場合、そのスレッドが取得したモニターはすべて解除されます。スレッドがモニターを取得したまま正常終了した場合、そのモニターが解除されるかランタイムシステムエラーが発生します。これは、スレッドがCBL_THREAD_CREATEにより作成されたときのフラグ設定に依存します。

モニターハンドルが不正な場合の動作は不定です。

CBL_MONITOR_BROWSE_TO_READ

現行スレッドによってすでに取得されたモニターの走査機能を読み込み機能に変換します。

構文:

```
CALL "CBL_MONITOR_BROWSE_TO_READ" USING BY VALUE モニターハンドル
```

パラメータ:

モニターハンドル　　ポインタ

入口上のパラメータ:

モニターハンドル　　モニターハンドル

出口上のパラメータ:

リターンコード　　値0は呼出しが成功したことを意味し、それ以外の値はエラーを意味します。詳細は、「同期ルーチンのRETURN-CODE値」を参照してください。

説明:

次のような場合の動作は不定です。

- 指定したモニター上でそれまでにこのスレッドによってかけられた走査ロックがない。
- モニターハンドルの値が不正である。

CBL_MONITOR_BROWSE_TO_WRITE

現行スレッドによってすでに取得されたモニターの走査機能を書き込み機能に変換します。本ルーチンは、これが成功するまでスレッドをブロックします。

構文:

```
CALL "CBL_MONITOR_BROWSE_TO_WRITE" USING BY VALUE モニターハンドル
```

パラメータ:

モニターハンドル　　ポインタ

入口上のパラメータ:

モニターハンドル　　モニターハンドル

出口上のパラメータ:

リターンコード　　値0は呼出しが成功したことを意味し、それ以外の値はエラーを意味します。詳細は、「同期ルーチンのRETURN-CODE値」を参照してください。

説明:

次のような場合の動作は不定です。

- 指定したモニター上でそれまでにこのスレッドによってかけられた走査ロックがない。
- モニターハンドルの値が不正である。

CBL_MONITOR_CLOSE

与えられたモニターハンドルを閉じます。

構文:

```
CALL "CBL_MONITOR_CLOSE" USING BY VALUE モニターハンドル
```

パラメータ:

モニターハンドル　　ポインタ

入口上のパラメータ:

モニターハンドル　　モニターハンドル

出口上のパラメータ:

リターンコード　　値0は呼出しが成功したことを意味し、それ以外の値はエラーを意味します。詳細は、「同期ルーチンのRETURN-CODE値」を参照してください。

説明:

このプロセスがモニターへのハンドルを持つ最後のプロセスであった場合、モニターは壊れます。

モニターハンドルの値が不正である場合の動作は不定です。

CBL_MONITOR_OPEN_INTRA

1つのプロセス内にスレッド同期化用のモニター同期オブジェクトを作成します。

構文:

```
CALL "CBL_MONITOR_OPEN_INTRA" USING BY REFERENCE モニターハンドル
                          BY VALUE オープンフラグ
```

パラメータ:

モニターハンドル　　ポインタ

オープンフラグ　　PIC X(4) COMP-5

入口上のパラメータ:

オープンフラグ　　ロックを認める場合に使用する優先順位アルゴリズムを次のように設定します。

ビット	値	意味
0	0	読み書きの優先順位がインターリーブされています。つまり、書き込み要求が出された場合、他の読み込みは最初の書き込み要求が許可されて終了するまではブロック状態になります。
	1	読み込み要求に常に優先権があります。これにより、書き込みを行えなくなる可能性があります。
1-31		予約済み。ゼロに設定する必要があります。

出口上のパラメータ:

モニターハンドル　　モニターハンドル

リターンコード　　値0は呼出しが成功したことを意味し、それ以外の値はエラーを意味します。詳細は、「同期ルーチンのRETURN-CODE値」を参照してください。

説明:

オープンフラグのビット0が0に設定されていて読み込みロックが入れ子になって要求されている場合、シングルスレッドがデッドロックする可能性があります。

CBL_MONITOR_READ

現行のスレッドまたはプロセス用に指定されたモニターの読み込み機能を取得します。本ルーチンは、その機能が取得されるまでスレッドをブロックします。

構文:

```
CALL "CBL_MONITOR_READ" USING BY VALUE モニターハンドル
```

パラメータ:

モニターハンドル　　ポインタ

入口上のパラメータ:

モニターハンドル　　モニターハンドル

出口上のパラメータ:

リターンコード　　値0は呼出しが成功したことを意味し、それ以外の値はエラーを意味します。詳細は、「同期ルーチンのRETURN-CODE値」を参照してください。

説明:

スレッドが異常終了した場合、そのスレッドが取得したモニターはすべて解除されます。スレッドがモニターを取得したまま正常終了した場合、そのモニターが解除されるかランタイムシステムエラーが発生します。これは、スレッドがCBL_THREAD_CREATEにより作成されたときのフラグ設定に依存します。

モニターハンドルの値が不正である場合の動作は不定です。

CBL_MONITOR_RELEASE

CBL_MONITOR_WRITE、CBL_MONITOR_READ、またはCBL_MONITOR_BROWSEによって指定されたモニターに現在かけられているロックを解除します。

構文:

```
CALL "CBL_MONITOR_RELEASE" USING BY VALUE モニターハンドル
```

パラメータ:

モニターハンドル　　ポインタ

入口上のパラメータ:

モニターハンドル　　モニターハンドル

出口上のパラメータ:

リターンコード　　値0は呼出しが成功したことを意味し、それ以外の値はエラーを意味します。詳細は、「同期ルーチンのRETURN-CODE値」を参照してください。

説明:

次のような場合の動作は不定です。

- 指定したモニター上でそれまでにこのスレッドによってかけられたロックがない。
- モニターハンドルの値が不正である。

CBL_MONITOR_UNBROWSE

CBL_MONITOR_BROWSEまたはCBL_MONITOR_WRITE_TO_BROWSEによって取得されたモニターの走査機能を解除します。

構文:

```
CALL "CBL_MONITOR_UNBROWSE" USING BY VALUE モニターハンドル
```

パラメータ:

モニターハンドル　　ポインタ

入口上のパラメータ:

モニターハンドル　　スペースまたはNULLで終わるモニター名

出口上のパラメータ:

リターンコード　　値0は呼出しが成功したことを意味し、それ以外の値はエラーを意味します。詳細は、「同期ルーチンのRETURN-CODE値」を参照してください。

説明:

次のような場合の動作は不定です。

- 指定したモニター上でそれまでにこのスレッドによってかけられた走査ロックがない。
- モニターハンドルの値が不正である。

CBL_MONITOR_UNREAD

CBL_MONITOR_READまたはCBL_MONITOR_BROWSE_TO_READによって取得されたモニターの読み込み機能を解除します。

構文:

```
CALL "CBL_MONITOR_UNREAD" USING BY VALUE モニターハンドル
```

パラメータ:

モニターハンドル　　ポインタ

入口上のパラメータ:

モニターハンドル　　スペースまたはNULLで終わるモニター名

出口上のパラメータ:

リターンコード　　値0は呼出しが成功したことを意味し、それ以外の値はエラーを意味します。詳細は、「同期ルーチンのRETURN-CODE値」を参照してください。

説明:

次のような場合の動作は不定です。

- 指定したモニター上でそれまでにこのスレッドによってかけられた読み込みロックがない。
- モニターハンドルの値が不正である。

CBL_MONITOR_UNWRITE

CBL_MONITOR_WRITEまたはCBL_MONITOR_BROWSE_TO_WRITEによって取得されたモニターの書き込み機能を解除します。

構文:

```
CALL "CBL_MONITOR_UNWRITE" USING BY VALUE モニターハンドル
```

パラメータ:

モニターハンドル　　ポインタ

入口上のパラメータ:

モニターハンドル　　スペースまたはNULLで終わるモニター名

出口上のパラメータ:

リターンコード　　値0は呼出しが成功したことを意味し、それ以外の値はエラーを意味します。詳細は、「同期ルーチンのRETURN-CODE値」を参照してください。

説明:

次のような場合の動作は不定です。

- 指定したモニター上でそれまでにこのスレッドによってかけられた書き込みロックがない。
- モニターハンドルの値が不正である。

CBL_MONITOR_WRITE

現在のスレッド用に指定されたモニターの書き込み機能を取得します。本ルーチンは、書き込み機能が取得できるまでスレッドをブロックします。

構文:

```
CALL "CBL_MONITOR_WRITE" USING BY VALUE モニターハンドル
```

パラメータ:

モニターハンドル　　ポインタ

入口上のパラメータ:

モニターハンドル　　スペースまたはNULLで終わるモニター名

出口上のパラメータ:

リターンコード　　値0は呼出しが成功したことを意味し、それ以外の値はエラーを意味します。詳細は、「同期ルーチンのRETURN-CODE値」を参照してください。

説明:

次のような場合の動作は不定です。

スレッドが異常終了した場合、そのスレッドが取得したモニターはすべて解除されます。スレッドがモニターを取得したまま正常終了した場合、そのモニターが解除されるかランタイムシステムエラーが発生します。これは、スレッドがCBL_THREAD_CREATEにより作成されたときのフラグ設定に依存します。

モニターハンドルの値が不正である場合の動作は不定です。

CBL_MONITOR_WRITE_TO_BROUSE

現在のスレッドによってすでに取得されたモニター書き込み機能を走査機能に変換します。

構文:

```
CALL "CBL_MONITOR_WRITE_TO_BROUSE" USING BY VALUE モニターハンドル
```

パラメータ:

モニターハンドル　　ポインタ

入口上のパラメータ:

モニターハンドル　　スペースまたはNULLで終わるモニター名

出口上のパラメータ:

リターンコード　　値0は呼出しが成功したことを意味し、それ以外の値はエラーを意味します。詳細は、「同期ルーチンのRETURN-CODE値」を参照してください。

説明:

次のような場合の動作は不定です。

- 指定したモニター上でそれまでにこのスレッドによってかけられた書き込みロックがない。
- モニターハンドルの値が不正である。

CBL_MUTEX_ACQUIRE

指定されたスレッドのMUTEXを取得します。

構文:

```
CALL "CBL_MUTEX_ACQUIRE" USING BY VALUE MUTEXハンドル
                         BY VALUE 待ちなしフラグ
```

パラメータ:

MUTEXハンドル　　ポインタ
待ちなしフラグ　　PIC X(4) COMP-5

入口上のパラメータ:

MUTEXハンドル　　MUTEXハンドル
待ちなしフラグ　　MUTEXを取得できない場合に何をすべきかを次のように設定します。

ビット	値	意味
0	0	ルーチンは、MUTEXを取得できるまでスレッドまたはプロセス

をブロックします。

1　ルーチンは直ちにゼロ以外の値を返します。

1-31　予約済み。ゼロに設定する必要があります。

出口上のパラメータ:

リターンコード　値0は呼出しが成功したことを意味し、それ以外の値はエラーを意味します。詳細は、「同期ルーチンのRETURN-CODE値」を参照してください。

説明:

MUTEXハンドルの値が不正である場合の動作は不定です。

CBL_MUTEX_CLOSE

指定されたMUTEXをクローズします。

構文:

```
CALL "CBL_MUTEX_CLOSE" USING BY VALUE MUTEXハンドル
```

パラメータ:

MUTEXハンドル　ポインタ

入口上のパラメータ:

MUTEXハンドル　MUTEXハンドル

出口上のパラメータ:

リターンコード　値0は呼出しが成功したことを意味し、それ以外の値はエラーを意味します。詳細は、「同期ルーチンのRETURN-CODE値」を参照してください。

説明:

本ルーチンを呼び出すと、MUTEXは壊れます。

MUTEXハンドルの値が不正である場合の動作は不定です。

CBL_MUTEX_OPEN_INTRA

プロセス間MUTEXを作成します。

構文:

```
CALL "CBL_MUTEX_OPEN_INTRA" USING BY VALUE MUTEXハンドル
                          BY VALUE オープンフラグ
```

パラメータ:

MUTEXハンドル　ポインタ

オープンフラグ　PIC X(4) COMP-5

入口上のパラメータ:

オープンフラグ　MUTEXのオープン方法を次のように指定します。

ビット	値	意味
0	0	MUTEXは、作成時に現行のスレッドによって取得されません。
	1	MUTEXは、作成時に現行のスレッドによって取得されます。
1-32		予約済み。ゼロに設定する必要があります。

出口上のパラメータ:

MUTEXハンドル　MUTEXハンドル

リターンコード　値0は呼出しが成功したことを意味し、それ以外の値はエラーを意味します。詳細は、「同期ルーチンのRETURN-CODE値」を参照してください。

説明:

MUTEXハンドルは、プロセス内で有効です。

CBL_MUTEX_RELEASE

指定されたMUTEXを解除します。

構文:

```
CALL "CBL_MUTEX_RELEASE" USING BY VALUE MUTEXハンドル
```

パラメータ:

MUTEXハンドル　ポインタ

入口上のパラメータ:

MUTEXハンドル　MUTEXハンドル

出口上のパラメータ:

リターンコード　値0は呼出しが成功したことを意味し、それ以外の値はエラーを意味します。詳細は、「同期ルーチンのRETURN-CODE値」を参照してください。

説明:

1つ以上のスレッドがこのMUTEXを待つCBL_MUTEX_ACQUIREルーチンでブロックされている場合、それらの1つがブロックを解除されてMUTEXを取得した状態で呼出しから戻ります。

次のような場合の動作は不定です。

- 現行のスレッドがまだMUTEXを取得していない。
- MUTEXハンドルの値が不正である。

CBL_NLS_CLOSE_MSG_FILE

国別言語サポート（NLS）のメッセージファイルを閉じます。

構文:

```
CALL "CBL_NLS_CLOSE_MSG_FILE" USING BY VALUE メッセージファイルハンドル
                    RETURNING  状態コード
```

パラメータ:

メッセージファイルハンドル　　PIC X(4)

状態コード　　「用語説明」の項を参照してください。

入口上のパラメータ:

メッセージファイルハンドル　　メッセージファイルがオープンされたときに返される識別ハンドル

出口上のパラメータ:

状態コード　　ルーチンが成功したかどうかを次のように示します。

0　　成功

40　　NLSモジュールが初期化されていません

404　　メッセージファイルハンドルの値が不正です

状態コードがこれ以外の値の場合、その値はランタイムエラーメッセージの番号です。

説明:

本ルーチンにより、CBL_NLS_OPEN_MSG_FILEルーチンを使用してそれまで開かれていた国別言語サポート（NLS）のメッセージファイルを閉じることができます。

CBL_NLS_COMPARE

2つの文字列を比較します。

構文:

```
CALL "CBL_NLS_COMPARE" USING         文字列1
                                     文字列2
                       BY VALUE      文字列1長
                       BY VALUE      文字列2長
                       BY REFERENCE 結果バイト
                       RETURNING    状態コード
```

パラメータ:

文字列1	PIC X(n).
文字列2	PIC X(n).
文字列1**長**	PIC X(4)COMP-5.
文字列2**長**	PIC X(4)COMP-5.
結果バイト	PIC s9 COMP-5.
状態コード	**「用語説明」の項を参照してください。**

入口上のパラメータ:

文字列1	1**つ目の文字列**
文字列2	2**つ目の文字列**

文字列1長　　1**つ目の文字列の長さ**
文字列2長　　2**つ目の文字列の長さ**

出口上のパラメータ:

結果バイト　**比較の結果**
0　2**つの文字列が同じ長さである**
-1　**文字列2の方が長い**
+1　**文字列1の方が長い**

状態コード　ルーチンが成功したかどうかを次のように示します。

0　成功

40　NLSモジュールが初期化されていません

説明:

本ルーチンは、NLSコンパイラ指令でコンパイルされたプログラムだけが使用できます。

CBL_NLS_INFO

国別言語情報を取得または設定します。

構文:

```
CALL "CBL_NLS_INFO" USING      機能コード
                           情報カテゴリー
                              情報バッファ
                    RETURNING  状態コード
```

パラメータ:

機能コード　PIC X COMP-X. **次のどちらかが入ります。**
1　**国別言語情報を取得します。**
2　**国別言語情報を設定します。**
情報カテゴリー　PIC X COMP-X.
情報バッファ　PIC X(n).
状態コード　**「用語説明」の項を参照してください。**

入口上のパラメータ:

機能コードが1の場合
情報カテゴリー　NLS**モジュールから取得する情報のカテゴリー**
1　**通貨記号**
2　**桁区切り**
3　**小数点**
機能コードが2の場合
情報カテゴリー　**設定する情報のカテゴリー**
1　**通貨記号**
2　**桁区切り**
3　**小数点**
文字バッファ　設定する情報（NULL で終わる）。桁区切りおよび小数点は、それぞれ1文字の長さです。
通貨記号の長さは、10文字までです。

出口上のパラメータ:

機能コードが1の場合
情報バッファ　要求された情報
状態コード　　ルーチンが成功したかどうかを次のように示します。
0　**成功**
40　NLS**モジュールが初期化されていません。**
機能コードが2の場合
状態コード　　ルーチンが成功したかどうかを次のように示します。
0　**成功**
40　NLS**モジュールが初期化されていません。**
405 **失敗**

説明:

本ルーチンにより、国別言語に関する情報を取得することも設定することもできます。機能コードに2（情報の設定）を指定すると、変更内容は呼出しを行ったプログラムだけに適用されます。

本ルーチンは、NLSコンパイラ指令でコンパイルされたプログラムだけが使用できます。

CBL_NLS_OPEN_MSG_FILE

国別言語サポート（NLS）のメッセージファイルをオープンします。

構文:

```
CALL "CBL_NLS_OPEN_MSG_FILE" USING        メッセージファイル名
                                  メッセージファイル名長
                                     メッセージファイルハンドル
                           RETURNING     状態コード
```

パラメータ:

メッセージファイル名　　PIC X(n).
メッセージファイル名長　　PIC X COMP-X.
メッセージファイルハンドル　　PIC X(4).
状態コード　　**「用語説明」の項を参照してください。**

入口上のパラメータ:

メッセージファイル名　　**オープンするメッセージファイルの名前**
メッセージファイル名長　　**メッセージファイル名の長さ。このパラメータが0に設定されると、「メッセージファイル名」の値に関わらず省略時のメッセージファイルがオープンされます。**

出口上のパラメータ:

メッセージファイルハンドル　　**識別ハンドル**
状態コード　　**ルーチンが成功したかどうかを次のように示します。**
0　**成功**
40　NLS**モジュールが初期化されていません。**
401　**メッセージセットが見つかりません。**
402　**メッセージがセットにありません。**
403　**メッセージが長すぎてメッセージテキストバッファに入りません。状態コードがこれ以外の値の場合、その値はランタイムエラーメッセージの番号です。**

説明:

本ルーチンは、NLSメッセージファイルを開きます。そしてCBL_NLS_READ_MSGおよびCBL_NLS_CLOSE_MSG_FILEルーチンで使用できる識別ハンドルを返します。同じ呼出しを使用して適切な国別言語における各メッセージにアクセスし、プログラムに記述する各国別言語に対して異なるメッセージを作成できます。省略時のメッセージファイルを使用することもできるし、ユーザ固有のメッセージファイルを使用することもできます。

CBL_NLS_READ_MSG

国別言語サポート（NLS）のメッセージファイルをからメッセージを読み込みます。

構文:

```
CALL "CBL_NLS_READ_MSG" USING       メッセージファイルハンドル
                              フルメッセージ名
                                 メッセージ挿入構造
                                 メッセージバッファ
                           RETURNING    状態コード
```

パラメータ:

メッセージファイルハンドル	PIC X(4).
フルメッセージ名	**次のように定義された集団項目**
	メッセージセット番号 PIC X(2) COMP-X.
	メッセージ番号 PIC X(2) COMP-X.
メッセージ挿入構造	**次のように定義された集団項目**
	挿入数 PIC X(2) COMP-X.
	挿入ポインタ USAGE POINTER**は**n**回発生します。**
メッセージバッファ	**次のように定義された集団項目**
	メッセージバッファ長 PIC X(2) COMP-X.
	メッセージバッファテキスト PIC X(n).
状態コード	**「用語説明」の項を参照してください。**

入口上のパラメータ:

メッセージファイルハンドル	**メッセージを取り出すメッセージファイルの識別ハンドル**
メッセージセット番号	**メッセージを取り出すメッセージファイルにおけるセット**
メッセージ番号	**メッセージを取り出すメッセージセットにおけるメッセージ番号**
挿入数	**メッセージに挿入するテキスト部分の数**
挿入ポインタ	**メッセージに挿入するNULL終わるテキスト部分へのポインタ**
メッセージバッファ長	**メッセージバッファテキストの長さ**

出口上のパラメータ:

メッセージバッファテキスト	**返されたテキスト（NULL終わる）**
状態コード	**ルーチンが成功したかどうかを次のように示します。** 0 **成功** 40 NLS**モジュールが初期化されていません。** 401 **メッセージセットが見つかりません。** 402 **メッセージがセットにありません。** 403 **メッセージが長すぎてメッセージテキストバッファに入りません。** **状態コードがこれ以外の値の場合、その値はランタイムエラーメッセージの番号です。**

説明:

各メッセージファイルにおいて、メッセージはセットに分けられます。これにより、必要であればユーザ固有のメッセージセットを省略時のメッセージファイルに定義することができます。

また、本ルーチンはによってテキスト部分をメッセージファイルから取り出されたメッセージに挿入することができます。その順番は、国別言語の文法に従います。

CBL_NOT

データ項目のビットに対して論理否定（NOT）を行います。

構文:

```
CALL "CBL_NOT" USING  目標
             BY VALUE  長さ
```

パラメータ:

目標　　任意のデータ項目

長さ　　数字定数またはPIC X(4) COMP-5.

入口上のパラメータ:

目標　演算を行うデータ

出口上のパラメータ:

目標　ビットを反転したデータ

長さ　変更する目標のバイト数。
これを越えた位置については変更はありません。

説明:

本ルーチンは目標の左端から開始し、ビットを反転します。この真理値表は次の通りです。

前	後
0	1
1	0

CBL_OPEN_FILE

バイトストリーム処理のために現存するファイルを開きます。

構文:

```
CALL "CBL_OPEN_FILE" USING  ファイル名
                            呼出し法
                            拒否モード
                            装置
                            ファイルハンドル
```

パラメータ:

ファイル名　　PIC X(n).

呼出し法　　PIC X COMP-X.

拒否モード　　PIC X COMP-X.

装置　　PIC X COMP-X.

ファイルハンドル　　PIC X(4).

入口上のパラメータ:

ファイル名　　開こうとしているファイルのスペースまたは空終了のファイル名。

呼出し法　　呼出し法を定義します:

1=読込み専用

2=書出し専用（拒否モードは0とする必要があります）

3=読込み／書出し

拒否モード　　拒否モードを定義します:

0=読込み・書出しの両方とも拒否（排他的）

1=書出しを拒否

2=読込みを拒否

3=読込み・書出しのいずれも拒否しない

装置　　将来の使用に予約されています（0とする必要があります）。

出口上のパラメータ:

ファイルハンドル　　ファイルを開くことに成功したものについてファイルハンドルを戻します。

説明:

呼出しが成功したかどうかは、RETURN-CODEを調べればわかります。

CBL_OPEN_VFILE

ヒープを開きます。

構文:

```
CALL "CBL_OPEN_VFILE" USING  ヒープID
                             状態語
                       RETURNING  状態コード
```

パラメータ:

ヒープ***ID***　　PIC X(2) COMP-5.

状態語　　PIC X(2).

状態コード　　「用語説明」の項を参照してください。

入口上のパラメータ:　　なし

出口上のパラメータ:

ヒープ*ID*　　割り付けられたヒープハンドルが入ります。ヒープハンドルがゼロの場合、ヒープを開けなかったことを意味します。

状態語　　ヒープの状態語で、開くときにゼロに設定されます。状態語（1:1）=9のとき、第2バイトの2進値は次のようになります:

000=利用者の要求によりヒープが閉じられました。
001=ヒープアクセスが失敗しました-バッファ外。
002=プログラムがアクティブでないときヒープが割付け解除されました。
014=ファイルのバックアップが失敗しました-ファイル数が多すぎます。
037=ファイルのバックアップが失敗しました-ファイルアクセスが拒否されました。
201=ファイルのバックアップが失敗しました-入出力の失敗
ヒープが閉じられるまで、状態語はヒープと関連をとられたままです。ヒープを開く（OPEN）ことに成功すると、第1バイトはASCIIゼロに設定されます。後続のヒープ読込み（READ）または書出し（WRITE）または閉じる動作（CLOSE）が割付けまたは入出力エラー（動作が成功してもゼロにリセットはされない）を検出すると、状態語はファイル状態データとして書き出されます。
ヒープ状態語が実際存在する（つまり、連絡節内ではない）プログラムが取り消されるとされると、状態語がそのプログラム内にあるヒープは全て自動的に取り消され、そのヒープ一意名（他のプログラムに渡された可能性がある）をもはや使用してはいけません。

CBL_OR

2つのデータ項目のビット間の論理和（OR）をとります。

構文:

```
CALL "CBL_OR" USING  源
                     目標
               BY VALUE  長さ
```

パラメータ:

源　　任意のデータ項目
目標　　任意のデータ項目
長さ　　数字定数またはPIC X(4) COMP-5.

入口上のパラメータ:

源　　論理和（OR）をとるデータ項目の内の1つ
目標　　論理和（OR）をとるもう一方のデータ項目
長さ　　論理和（OR）をとろうとする源と目標のバイト数。この長さを越えた目標内の位置については変更はありません。

出口上のパラメータ:

目標　結果

説明:

本ルーチンは源と目標の左端から開始し、それらのビットの論理和（OR）をとり、結果を目標に格納します。これの真理値表は次の通りです。

源	目標	結果
0	0	0
0	1	1
1	0	1
1	1	1

CBL_PUT_SHMEM_PTR

名前付き値を作成または更新します。

構文:

```
CALL "CBL_PUT_SHMEM_PTR" USING BY VALUE  ノード値
                                         ノード名
                         RETURNING  状態コード
```

パラメータ:

ノード値　USAGE POINTER

ノード名　次のように定義された集団項目

名前長　PIC X COMP-5 値n

名前　PIC X(n) 値"名前"

状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

ノード値　作成/更新された名前付き値に割り当てられた値

名前長　名前の長さ

名前　名前付き値の名前

出口上のパラメータ:　なし

説明

名前付き値は、実行時に同意を得た名前を使用して異なる実行単位間でポインタを渡す方法を提供します。コルーチン内のすべての実行単位は名前付き値を同時に読み込むことができます。それは、ランタイムシステムが更新を一切禁止し、逐次化しているからです。名前付き値の最大数は、マシンのメモリサイズによって違ってきます。

CBL_READ_FILE

ファイルからバイト列を読み出します。

構文:

```
CALL "CBL_READ_FILE" USING  ファイルハンドル
                            ファイル相対番地
                            バイト数
                            フラグ
                            バッファ
```

パラメータ:

ファイルハンドル	PIC X(4).
ファイル相対番地	PIC X(8) COMP-X.
バイト数	PIC X(4) COMP-X.
フラグ	PIC X COMP-X.
バッファ	PIC X(n).

入口上のパラメータ:

ファイルハンドル　ファイルが開かれたとき戻されたファイルハンドル。

ファイル相対番地　読込みを行うファイル内の相対番地。本項目は現在最大値x "00FFFFFFFF"に制限されています。

バイト数　読み込むバイト数。本項目は現在最大値x "00FFFF"に制限されています。

フラグ　本パラメータは次の値をとります:

0　標準の読込み用

128　現ファイルサイズをファイル相対番地フィールドに入れて戻させます。

出口上のパラメータ:

ファイル相対番地　フラグパラメータが入口上で128に設定されていると、戻される現ファイルサイズが入ります。

バッファ　バイトカラムが読み込まれるバッファ。利用者の責任において、読み込まれるバイト数に足るバッファサイズを確保してください。

説明:

呼出しが成功したかどうかは、RETURN-CODEを調べるとわかります。

.lbrファイルに含まれているファイルを読み込むために本ルーチンを使用する場合には、ファイル終了状態が返されます。

目的のファイルを確実に読み込むには、まずファイルサイズを取り出し（フラグに128を設定します）、そのサイズまでファイルを読み込みます。

CBL_READ_KBD_CHAR

文字がタイプされるまで待ち、表示なしでそれを読み込みます。

構文:

```
CALL "CBL_READ_KBD_CHAR" USING  文字
                         RETURNING  状態コード
```

パラメータ:

文字　　PIC X.

状態コード　　「用語説明」の項を参照してください。

入口上のパラメータ:　なし

出口上のパラメータ:

文字　　タイプされたASCII文字

説明:

CBL_キーボードルーチンをx"AF"キーボードルーチンと共に使用することはできません。

「キーボードルーチンの使用例」

CBL_READ_MOUSE_EVENT

マウス事象待ち行列を読込み、事象についての情報を戻します。

構文:

```
CALL "CBL_READ_MOUSE_EVENT" USING  マウスハンドル
                                   事象データ
                                   読込み種類
                            RETURNING  状態コード
```

パラメータ:

マウスハンドル　　PIC X(4) COMP-X.

事象データ　　本付録において前述した「カテゴリー別ルーチン」の項の「マウス」を参照してください。

読込み種類　　PIC X COMP-X.

状態コード　　「用語説明」の項を参照してください。

入口上のパラメータ:

マウスハンドル　　マウス一意名で、前のCBL_INIT_MOUSEに対する呼出しにより獲得されます。

読込み種類　　待ち行カラム内に事象がない場合どうするかを示します。

0=即座に戻る

1=事象を待って、それから戻る

出口上のパラメータ:

事象データ　　本付録において前述した「カテゴリー別ルーチン」の項の「マウス」を参照してください。

説明:

事象待ち行カラムに事象がない場合、本ルーチンからの戻りは読込み種類の値に依ります。読込み種類がゼロの場合、事象データ内の値を全てゼロとして本ルーチンは即座に戻ります。読込み種類が値1を持っていると、事象が待ち行列に入ってしまうまで戻りは遅らされます。

CBL_マウスルーチンをx"AF"マウスルーチンと共に使用することはできません。

CBL_READ_SCR_ATTRS

画面から属性の列を読み込みます。

構文:

```
CALL "CBL_READ_SCR_ATTRS" USING  画面位置
                                 属性バッファ
                                 文字列長
                       RETURNING 状態コード
```

パラメータ:

画面位置　次のように定義される集団項目:
- **行番号**　PIC X COMP-X.
- **カラム番号**　PIC X COMP-X.

属性バッファ　PIC X(n)

文字列長　PIC X(2) COMP-X.

状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

画面位置　読込みを開始する画面位置。左上の角が行0、カラム0です。本付録において先に出てきた「カテゴリー別ルーチン」の項の「画面」を参照してください。

文字列長　読み込む文字列の長さ

出口上のパラメータ:

属性バッファ　画面から読み込まれる属性。このデータ項目は少なくとも文字列長により指定された長さをしている必要があります。この長さを越えたデータ項目内の位置については変更はありません。

文字列長　画面の終わりに到達すると、読み込まれた長さがここで返されます。

CBL_READ_SCR_CHARS

画面から文字列を読み込みます。

構文:

```
CALL "CBL_READ_SCR_CHARS" USING  画面位置
                                 文字バッファ
                                 文字列長
                       RETURNING 状態コード
```

パラメータ:

画面位置　次のように定義される集団項目:
　行番号　PIC X COMP-X.
　カラム番号　PIC X COMP-X.
文字バッファ　PIC X(n).
文字列長　PIC X(2) COMP-X.
状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

画面位置　読込みを開始する画面位置。左上の角が行0、カラム0です。本付録において前述した「カテゴリー別ルーチン」の項の「画面」を参照してください。
文字列長　読み込む文字列の長さ

出口上のパラメータ:

文字バッファ　画面から読み込まれる文字。このデータ項目は少なくとも文字列長により指定された長さをしている必要があります。この長さを越えたデータ項目内の位置については変更はありません。
文字列長　画面の終わりに到達すると、読み込まれた長さがここで返されます。

CBL_READ_SCR_CHATTRS

画面から文字とそれらの属性の列を読み込みます。

構文:

```
CALL "CBL_READ_SCR_CHATTRS" USING  画面位置
                                   文字バッファ
                                   属性バッファ
                                   文字列長
                             RETURNING  状態コード
```

パラメータ:

画面位置　次のように定義される集団項目:
　行番号　PIC X COMP-X.　カラム番号　PIC X COMP-X.
文字バッファ　PIC X(n).
属性バッファ　PIC X(n).
文字列長　PIC X(2) COMP-X.
状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

画面位置　読込みを開始する画面位置。左上の角が行0、カラム0です。本付録において先に出てきた「カテゴリー別ルーチン」の項の「画面」を参照してください。

文字列長　読み込む文字列の長さ

出口上のパラメータ:

文字バッファ　画面から読み込まれる文字。このデータ項目は少なくとも文字列長により指定された長さをしている必要があります。この長さを越えたデータ項目内の位置については変更はありません。

属性バッファ　画面から読み込まれる属性。このデータ項目は少なくとも文字列長により指定された長さをしている必要があります。この長さを越えたデータ項目内の位置については変更はありません。

文字列長　画面の終わりに到達すると、読み込まれた長さ（セル数、つまり、文字-属性ペア数）がここで返されます。

CBL_READ_VFILE

ヒープからバイト列を読み込みます。

構文:

```
CALL "CBL_READ_VFILE" USING BY VALUE  ヒープID
                            ヒープ参照
                            ヒープ長
                            BY REFERENCE  ヒープバッファ
                            RETURNING  状態コード
```

パラメータ:

ヒープ*ID*　PIC X(2) COMP-5.

ヒープ参照　PIC X(4) COMP-5.

ヒープ長　PIC X(4) COMP-5.

ヒープバッファ　PIC X(n).

状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

ヒープ***ID***　ヒープが開かれたときに割り当てられたヒープハンドルが入ります。

ヒープ参照　読込みを開始するヒープ内の相対番地

ヒープ長　読込むバイト数

出口上のパラメータ:

ヒープバッファ　バイト列が読み込まれるバッファ。利用者の責任において、読み込まれるバイト数に足るバッファサイズを確保してください。

説明:

まだ書き出されてないヒープの領域からデータを読み込もうとすると、バッファに不定のデータが戻されます。

CBL_RENAME_FILE

ファイルの名前を変更します。

構文:

```
CALL "CBL_RENAME_FILE" USING  旧ファイル名
                              新ファイル名
                        RETURNING  状態コード
```

パラメータ:

旧ファイル名　PIC X(n).
新ファイル名　PIC X(n).
状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

旧ファイル名　再命名するファイル。名前はパス名を含むことができ、スペースで終了します。パスが与えられていないと、現行ディレクトリとみなされます。本ルーチンは、ワイルドカードを含むファイル名を指定すると動作しません。

新ファイル名　新しい名前で、スペースで終了します。旧ファイル名にパス名が入っている場合、本項目は同じパス名を含んでいる必要があります。オペレーティングシステムによっては、新ファイル名に指定された名前のファイルがすでに存在する場合にファイルの名前を付け替えることを許さないものもあります。

出口上のパラメータ:　なし

CBL_SCR_ALLOCATE_COLOR

1つ以上の与えられたRGB値のそれぞれに対してカラーマップのあるエントリーを探します。

構文:

```
CALL "CBL_SCR_ALLOCATE_COLOR" USING       表数
                                  RGB値
                                  カラー属性
                                  不完全一致
                      RETURNING   状態コード
```

パラメータ:

表数　PIC X(2) COMP-X.
RGB**値**　**次のように定義された集団項目。**n**回繰り返す。**
　赤　PIC X(2) COMP-X.
　緑　PIC X(2) COMP-X.
　青　PIC X(2) COMP-X.
　充填　PIC X(2).
カラー属性　PIC X(4) COMP-X**を**n**回繰り返す。**
不完全一致　PIC X COMP-X**を**n**回繰り返す。**
状態コード　**「用語説明」の項を参照してください。**

入口上のパラメータ:

表数　**要求されたカラーの数**
RGB**値**　**探す、またはカラーマップに追加するカラーの表。**

出口上のパラメータ:

カラー属性　**要求されたカラーに対応するカラーマップの索引の表。これは、既存のカラーマップエントリーか、要求されたカラーが設定された**
　割り当てられていない読み書きカラーマップエントリーです。

不完全一致　対応するカラー属性が完全に一致しているかどうかを示すフラグの表。完全に一致している場合はゼロ、それ以外はゼロ以外
の値が入ります。
表数　　割当てに成功したカラーの数。通常、要求された数が入りますが、呼出しが完全に成功しなかった場合はそれよりも小さい
値が入ります。
RGB**値　　検索中に実際に使用された**RGB**値の表。環境にあわせて切り取られます。**
充填　　使用されていません。

説明:

RGB値は、定数として指定したりCBL_SCR_NAME_TO_RGBルーチンの呼出しにより取得することができます。もっとも似ているものを決めるアルゴリズムは、環境によって定義されます。

CBL_SCR_ALLOCATE_VC_COLOR

1つ以上の指定された仮想カラーマップを提供されているRGB値に設定します。

構文:

```
CALL "CBL_SCR_ALLOCATE_VC_COLOR" USING       表数
                                       カラー属性
                                       RGB値
                               RETURNING     状態コード
```

パラメータ:

**表数　　**PIC X(2) COMP-X.
カラー属性　PIC X(4) COMP-X**を**n**回繰り返す。**
RGB**値　　次のように定義された集団項目。**n**回繰り返す。**
赤　　PIC X(2) COMP-X.
緑　　PIC X(2) COMP-X.
青　　PIC X(2) COMP-X.
充填　　PIC X(2).
状態コード　　「用語説明」の項を参照してください。

入口上のパラメータ:

表数　　割り当てるカラーの数
カラー属性　仮想カラーマップにおける配置の索引の表。この表には、指定されたRGB**値を格納します。**
RGB**値　　仮想カラーマップに追加するカラーの表。**

出口上のパラメータ:

表数　　割当てに成功したカラーの数。通常、要求された数が入りますが、呼出しが完全に成功しなかった場合はそれよりも小さい
値が入ります。
RGB**値　　検索中に実際に使用された**RGB**値の表。環境にあわせて切り取られます。**
充填　　使用されていません。

説明:

RGB値は、定数として指定したりCBL_SCR_NAME_TO_RGBルーチンの呼出しにより取得することができます。

CBL_SCR_CREATE_VC

仮想カラーマップを作成して、そのカラーマップにシステムカラーマップの内容をコピーします。

構文:

```
CALL "CBL_SCR_CREATE_VC" RETURNING    状態コード
```

パラメータ:

状態コード　　「用語説明」の項を参照してください。

入口上のパラメータ:　なし

出口上のパラメータ:　なし

説明:

ランタイムシステムは、仮想カラーマップを1つだけ持つことができます。その仮想カラーマップが存在する限りは、システムカラーマップに優先して使用されます。

CBL_SCR_DESTROY_VC

仮想カラーマップを作成して、そのカラーマップにシステムカラーマップの内容をコピーします。

構文:

```
CALL "CBL_SCR_DESTROY_VC" RETURNING    状態コード
```

パラメータ:

状態コード　　「用語説明」の項を参照してください。

入口上のパラメータ:　なし

出口上のパラメータ:　なし

CBL_SCR_GET_ATTR_INFO

環境用の属性に関する情報を返します。

構文:

```
CALL "CBL_SCR_GET_ATTR_INFO" USING       属性情報
                         RETURNING    状態コード
```

パラメータ:

属性情報　　**次のように定義された集団項目。**

```
クラス                          PIC X COMP-X.
カラーマップサイズ    PIC X(2) COMP-X.
カラーマップフラグ    PIC X COMP-X.
```

ビット／gun PIC X COMP-X.
状態コード **「用語説明」の項を参照してください。**

入口上のパラメータ: なし

出口上のパラメータ:

クラス **環境のクラス**
1 **白黒**
2 **読み込み専用カラーマップ**
3 **読み書き専用カラーマップ**
4 **読み込み専用分解カラーマップ**
5 **読み書き専用分解カラーマップ**
カラーマップサイズ **カラーマップ上のエントリーの数**
カラーフラグ **次のように設定されたビットが入ります。**

ビット	値	意味
0	1	**カラーマップは読み書き専用**
	0	**カラーマップは読み込み専用**
1	1	**仮想カラーマップを作成できる環境**
	0	**仮想カラーマップを作成できない環境**
2	1	**属性を読み込み直してかかれた値を読み込み直します。**
	0	**現在画面上にあるカラーマップを読み込み直します。**

ビット/gun **ハードウェアを動かすために実際に使用されたRGB値の有功ビット数。ゼロはその環境ではビット数がわからないことを意味します。**
各ガンが異なる有功ビット数である場合、最下位ビットが使用されます。

CBL_SCR_GET_ATTRIBUTES

属性表の1つ以上のエントリー用の文字属性コード、および前景と背景のカラーマップ索引を取得します。

構文:

```
CALL "CBL_SCR_GET_ATTRIBUTES" USING      表数
                                         属性索引
                                         属性値
                            RETURNING    状態コード
```

パラメータ:

表数 PIC X(2) COMP-X.
属性索引 PIC X COMP-X**をn回繰り返す。**
属性値 **次のように定義された集団項目。n回繰り返す。**
前景索引 PIC X(4) COMP-X.
背景索引 PIC X(4) COMP-X.
文字属性 PIC X COMP-X.
充填 PIC X(3).
状態コード **「用語説明」の項を参照してください。**

入口上のパラメータ:

表数　　**問い合わせる属性表エントリーの数**
属性索引　　**問い合わせるエントリーの索引表**

出口上のパラメータ:

表数　　**問い合わせに成功したエントリーの数。通常、要求された数が入りますが、呼出しが完全に成功しなかった場合はそれよりも小さい値が入ります。**
属性値　　**提供されている索引に対応する属性値の表**
充填　　**使用されていません。**

CBL_SCR_NAME_TO_RGB

単独の名前付きカラーをRGB値に変換します。

構文:

```
CALL "CBL_SCR_NAME_TO_RGB" USING       カラー名
                                       RGB値
                           RETURNING   状態コード
```

パラメータ:

カラー名　　**次のように定義された集団項目。**
カラー名長　PIC X COMP-X.
カラー名バッファ　PIC X(n).
RGB値　　**次のように定義された集団項目。**
赤　PIC X(2) COMP-X.
緑　PIC X(2) COMP-X.
青　PIC X(2) COMP-X.
充填　PIC X(2).
状態コード　　**「用語説明」の項を参照してください。**

入口上のパラメータ:

カラー名長　　**名前の文字数**
カラー名バッファ　　**変換されるカラー名**

出口上のパラメータ:

RGB値　　**名前付きカラーの変換先のRGB値**
充填　　**使用されていません。**

CBL_SCR_QUERY_COLORMAP

1つ以上のカラーマップエントリー用のRGB値を返します。

構文:

```
CALL "CBL_SCR_QUERY_COLORMAP" USING       表数
                                          カラー属性
                                          RGB値
                              RETURNING   状態コード
```

パラメータ:

```
表数                    PIC X(2) COMP-X.
カラー属性    PIC X(4) COMP-Xをn回繰り返す。
RGB値                   次のように定義された集団項目。
                        赤        PIC X(2) COMP-X.
                        緑        PIC X(2) COMP-X.
                        青        PIC X(2) COMP-X.
                        充填      PIC X(2).
状態コード      「用語説明」の項を参照してください。
```

入口上のパラメータ:

```
表数                    問い合わせるエントリーの数
カラー属性    問い合わせる1つ以上のカラーマップ索引の表
```

出口上のパラメータ:

```
RGB値                   表索引に対応するRGB値の表
充填                    使用されていません。
```

CBL_SCR_RESTORE_ATTRIBUTES

属性表を指定されたバッファハンドルに関連する保存された値に戻します。

構文:

```
CALL "CBL_SCR_RESTORE_ATTRIBUTES" USING       ハンドル
                    RETURNING    状態コード
```

パラメータ:

```
ハンドル                ポインタ
状態コード      「用語説明」の項を参照してください。
```

入口上のパラメータ:

```
ハンドル                CBL_SCR_SAVE_ATTRIBUTESに関連する呼出しで返されたハンドル
```

出口上のパラメータ: なし

CBL_SCR_SAVE_ATTRIBUTES

属性表の現在の設定を保存します。

構文:

```
CALL "CBL_SCR_SAVE_ATTRIBUTES" USING       ハンドル
                    RETURNING    状態コード
```

パラメータ:

```
ハンドル                ポインタ
状態コード      「用語説明」の項を参照してください。
```

入口上のパラメータ:　なし

出口上のパラメータ:

ハンドル　　属性を保存するのに使用された内部バッファのハンドル

CBL_SCR_SET_ATTRIBUTES

1つ以上のCOBOL属性値（属性表索引）のそれぞれに関連する文字属性および前景・背景カラーマップを定義します。

構文:

```
CALL "CBL_SCR_SET_ATTRIBUTES" USING       表数
                                          属性索引
                                          属性値
                            RETURNING     状態コード
```

パラメータ:

表数　　PIC X(2) COMP-X.
属性索引　　PIC X COMP-X**をn回繰り返す。**
属性値　　**次のように定義された集団項目。**
前景索引　PIC X(4) COMP-X.
背景索引　PIC X(4) COMP-X.
文字属性　PIC X COMP-X.
充填　PIC X(3).
状態コード　**「用語説明」の項を参照してください。**

入口上のパラメータ:

表数　**設定する属性の数**
属性索引　**属性表における索引の表**
属性値　**索引表に関連する属性記述の表**

出口上のパラメータ:

表数　**設定に成功した属性の数。通常は要求された数が入りますが、呼出しが完全に成功しなかった場合はそれより小さい値が入ります。**
充填　**使用されていません。**

CBL_SCR_SET_PC_ATTRIBUTES

できるだけIBM-PCの属性表に近づけるように現行のCOBOL属性表を設定します。

構文:

```
CALL "CBL_SCR_SET_PC_ATTRIBUTES" RETURNING    状態コード
```

パラメータ:

状態コード　**「用語説明」の項を参照してください。**

入口上のパラメータ:

表数　　**設定する属性の数**
属性索引　　**属性表における索引の表**
属性値　　**索引表に関連する属性記述の表**

出口上のパラメータ:

状態コード　　**「用語説明」の項を参照してください。**
0　**成功**
1　**成功したが、1つ以上のカラーが正確ではない。**
2　**失敗**

説明:

本ルーチンを使用することと一般的な属性ルーチンを使用してIBM-PCの属性表をすべて設定することは同義です。本ルーチンは、属性処理を移植可能にすることによって、既存のプログラムに一般的な属性を適用させることができるようにするために提供されています。ただし、本ルーチンは、あらゆる可能性のある属性を定義しているため、効率は低くなってしまいます。新しいアプリケーションには、一般的な属性ルーチンを使用して必要な属性だけ定義してください。

CBL_SEMAPHORE_ACQUIRE

セマフォが持っているリソースの1つを関連するカウントを減らしていくことによって確保します。

構文:

```
CALL "CBL_SEMAPHORE_ACQUIRE" USING BY VALUE      セマフォハンドル
                         BY VALUE 待ちなしフラグ
```

パラメータ:

セマフォハンドル　　**ポインタ**
待ちなしフラグ　　PIC X(4) COMP-5

入口上のパラメータ:

セマフォハンドル　　**セマフォハンドル**
待ちなしフラグ　　**カウントがゼロの場合にルーチンの動作を決定するために次のようにビットを設定します。**

ビット	値	意味
0	0	カウントがゼロ以外になるまでスレッドまたはプロセスをブロックし、その後カウントは減少して呼出しが戻ります。
	1	カウントを減らさず直ちにゼロ以外の値を返します。
1-31		予約済み。ゼロに設定する必要があります。

出口上のパラメータ:

リターンコード　値0は呼出しが成功したことを意味し、それ以外の値はエラーを意味します。詳細は、「同期ルーチンのRETURN-CODE値」を参照してください。

説明:

セマフォハンドルの値が不正である場合の動作は不定です。

CBL_SEMAPHORE_CLOSE

指定されたセマフォハンドルを閉じます。

構文:

```
CALL "CBL_SEMAPHORE_CLOSE" USING BY VALUE セマフォハンドル
```

パラメータ:

セマフォハンドル　ポインタ

入口上のパラメータ:

セマフォハンドル　セマフォハンドル

出口上のパラメータ:

リターンコード　値0は呼出しが成功したことを意味し、それ以外の値はエラーを意味します。詳細は、「同期ルーチンのRETURN-CODE値」を参照してください。

説明:

本ルーチンにより、セマフォは壊れます。

セマフォハンドルの値が不正である場合の動作は不定です。

CBL_SEMAPHORE_OPEN_INTRA

プロセス間セマフォを作成します。

構文:

```
CALL "CBL_SEMAPHORE_OPEN_INTRA" USING BY REFERENCE セマフォハンドル
                                BY VALUE セマフォ開始
                                BY VALUE オープンフラグ
```

パラメータ:

セマフォハンドル　ポインタ

セマフォ開始　PIC X(4) COMP-5

オープンフラグ　PIC X(4) COMP-5

入口上のパラメータ:

セマフォ開始　セマフォ用の初期化カウント

オープンフラグ　予約済み。ゼロに設定する必要があります。

出口上のパラメータ:

セマフォハンドル　セマフォハンドル

リターンコード　値0は呼出しが成功したことを意味し、それ以外の値はエラーを意味します。詳細は、「同期ルーチンのRETURN-CODE値」を参照してください。

CBL_SEMAPHORE_RELEASE

関連するカウントを増加させることによってセマフォが持っているリソースの1つを解放します。

構文:

```
CALL "CBL_SEMAPHORE_RELEASE" USING BY VALUE セマフォハンドル
```

パラメータ:

セマフォハンドル　ポインタ

入口上のパラメータ:

セマフォハンドル　セマフォハンドル

出口上のパラメータ:

リターンコード　値0は呼出しが成功したことを意味し、それ以外の値はエラーを意味します。詳細は、「同期ルーチンのRETURN-CODE値」を参照してください。

説明:

カウントが増加される前にゼロである場合、およびカウントがゼロ以外の値になるのを待っているCBL_SEMAPHORE_ACQUIRE呼出しで他のスレッドがブロックされている場合、そのスレッドの1つが解放されます。そのため、スレッドはカウントを減らすことができ、CBL_SEMAPHORE_ACQUIREから戻ります。

セマフォハンドルの値が不正である場合の動作は不定です。

CBL_SET_CSR_POS

カーソルを移動します。

構文:

```
CALL "CBL_SET_CSR_POS" USING  画面位置
                    RETURNING  状態コード
```

パラメータ:

画面位置　次のように定義される集団項目:

行番号　PIC X COMP-X.

カラム番号　PIC X COMP-X.

状態コード　　「用語説明」の項を参照してください。

入口上のパラメータ:

画面位置　　カーソルの移動先である画面位置。左上の角が行0、カラム0です。本付録において前述した「カテゴリー別ルーチン」の項の「画面」を参照してください。

出口上のパラメータ:　なし

説明:

カーソルを見えないようにするには行番号とカラム番号を255に設定します。他の正当なオンスクリーン値はどれもカーソルを表示させます。

CBL_SET_MOUSE_MASK

マウス事象マスクを設定します。

構文:

```
CALL "CBL_SET_MOUSE_MASK" USING  マウスハンドル
                                 事象マスク
                       RETURNING  状態コード
```

パラメータ:

マウスハンドル　PIC X(4) COMP-X.
事象マスク　PIC X(2) COMP-X.
状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

マウスハンドル　マウス一意名で、前のCBL_INIT_MOUSEに対する呼出しにより獲得されます。
事象マスク　本付録において前述した「カテゴリー別ルーチン」の項の「マウス」を参照してください。

出口上のパラメータ:　なし

説明:

どの事象が可能となっているか見つけるために、CBL_GET_MOUSE_MASKを最初に呼び出す必要があります。

CBL_マウスルーチンをx"AF"マウスルーチンと共に使用することはできません。

CBL_SHOW_MOUSE

マウスポインタを見えるようにします。

構文:

```
CALL "CBL_SHOW_MOUSE" USING  マウスハンドル
                     RETURNING  状態コード
```

パラメータ:

マウスハンドル　PIC X(4) COMP-X.
状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

マウスハンドル　マウス一意名で、前のCBL_INIT_MOUSEに対する呼出しにより獲得されます。

出口上のパラメータ:　なし

説明:

マウス支援がCBL_INIT_MOUSE呼出しにより初期化されると、本ルーチンが呼び出されるまでポインタは表示されません。この呼出し後、マウスを隠すかマウス支援を終了するルーチンが呼び出されるまで、システムはマウスポインタを表示します。PC_SET_MOUSE_HIDE_AREAにより前に定義された任意の衝突領域を、本ルーチンは取り消します。

CBL_マウスルーチンをx"AF"マウスルーチンと共に使用することはできません。

CBL_SPLIT_FILENAME

ファイル名をその構成要素、つまり、パス名、基本名および拡張子に分割します。

構文:

```
CALL "CBL_SPLIT_FILENAME" USING  分割・結合パラメータ
                                 分割バッファ
                       RETURNING  状態コード
```

パラメータ:

分割・結合パラメータ　次のように定義された集団項目

パラメータ長	PIC X(2) COMP-X.
分割・結合フラグ1	PIC X COMP-X.
分割・結合フラグ2	PIC X COMP-X.
パス相対番地	PIC X(2) COMP-X.
パス長	PIC X(2) COMP-X.
基本名相対番地	PIC X(2) COMP-X.
基本名長	PIC X(2) COMP-X.
拡張子相対番地	PIC X(2) COMP-X.
拡張子長	PIC X(2) COMP-X.
全長	PIC X(2) COMP-X.
分割バッファ長	PIC X(2) COMP-X.
結合バッファ長	PIC X(2) COMP-X.
第1パス構成要素長	PIC X(2) COMP-X.
分割バッファ	PIC X(n).

状態コード　　「用語説明」の項を参照してください。

入口上のパラメータ:

パラメータ長　　分割・結合パラメータの長さ（バイト）で、パラメータ長の2バイトを含む。パラメータ長の通常の値は24です。

分割・結合フラグ1　　ビット1-設定されると、ファイル名は空終了であり、設定されないと、スペース終了であることを指定します。

ビット0-設定されると、新しい文字列は大文字にフォールドされ、設定されないと元の大文字・小文字がそのまま保存されることを指定します。

分割バッファ長　　分割バッファの長さ

分割バッファ　　分割する文字列

出口上のパラメータ:

分割・結合フラグ2　　ビット2-ファイル名に意味のあるスペースが入っている場合設定されます。

ビット1-ワイルドカードがパス名に入っている場合設定されます。

ビット0-基本名または拡張子にワイルドカードが入っている場合設定されます。

パス相対番地　　分割バッファ内での装置名の先頭で、1から始まります。

パス長　　パス名の長さ。何も入ってない場合はゼロ。これは後に続くコロンを含みます。

基本名相対番地　　分割バッファ内での基本名の先頭で、1から始まります。

基本名長　　基本名の長さ。何も入ってない場合はゼロ。これは後に続くピリオドを含みません。

拡張子相対番地　　分割バッファ内での拡張子の先頭で、1から始まります。

拡張子長　　拡張子の長さ。何も入ってない場合はゼロ。これは先行するピリオドを含みません。

全長　　文字列内の合計文字数

第1パス構成要素長　　最初の¥または/またはコロン（含む）までの文字数。分割バッファにこれらのいずれも含まれてない場合、このフィールド=パス長

分割バッファ　　大文字へフォールドされるとき、分割・結合フラグ1のビット1が設定されてない限り、変化なし。

状態コード　　0=成功

4=違法ファイル名

説明:

分割・結合フラグ1の最下位のビット（ビット0）を設定することによって、本ルーチンを大文字にフォールドするようにできます。このビットが設定されていないと、元の大文字・小文字はそのまま保存されます。

本ルーチンは空で終了する文字列・スペースで終了する文字列のいずれも受け取ります。分割・結合フラグ1の最下位ビットの次のビット（ビット1）を設定すると、ルーチンは空で終了する文字列を受け取るものと予想します。このビットが設定されてないと、スペースで終了する文字列を受け取るものと予想します。

2個またはそれ以上のドットがファイル名に含まれていると（パス名に含まれるドットは数えない）、戻される拡張子は最後のドットとファイル名の終わりの間の文字列で構成されます。基本名は最後のドット（ドットは含まない）までの全てを含みます。

拡張子のないファイル名と拡張子がスペースであるファイル名（つまり、最後の文字がドットである基本名）を区別するためには、拡張子がスペースの場合は、拡張子長は1で拡張子相対番地は最後のドットを指します。

「CBL_SPLIT_FILNAMEの使用例」

CBL_SUBSYSTEM

サブシステムを宣言または再割付けします。

構文:

```
CALL "CBL_SUBSYSTEM" USING  関数コード
                            パラメータ
                    RETURNING  状態コード
```

パラメータ:

関数コード　PIC X COMP-X.
次の副関数番号の1つが入ります:
0=サブシステムの宣言
1=サブシステムの取り消し
2=サブシステムからの削除

パラメータ

関数コード=0用　次のように定義される集団項目:

***ss*ハンドル**　PIC X(2) COMP-X.

***ss*名長**　PIC X(2) COMP-X.

***ss*名**　PIC X(n).

関数コード=1用

***ss*ハンドル**　PIC X(2) COMP-X.

関数コード=2用

ダミーパラメータ　PIC X(2) COMP-X VALUE 0.

状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

関数コード　副関数番号。

***ss*名長**　サブシステムプログラム名項目の長さ（関数コード=0の場合）。

***ss*名**　サブシステムプログラム名-スペース終了（関数コード=0の場合）。これはCOBOL（ .int）または（ .gnt）モジュールとする必要があります。

***ss*ハンドル**　関数0呼出しにより戻されたサブシステムハンドル（関数コード=1の場合）

ダミーパラメータ　値0（関数コード=2の場合）

出口上のパラメータ:

***ss*ハンドル**　　サブシステムハンドル（関数コード=0の場合）

説明:

サブシステムはアプリケーション内の指定プログラム、および他のサブシステムにまだ属していないサブシステム内に既にあるプログラムによりその後呼び出される任意の副プログラムに定義されます。

関数コード=0の場合

この関数はサブシステムを宣言します。本ルーチンはサブシステムハンドルを戻します。プログラムがまだロードされていないと、本関数がロードします。プログラムを見つけるときまたはロードするときエラーが発生すると、サブシステムハンドルゼロが戻されます。

サブシステムに属するプログラムは、CANCEL動詞により取り消されるか、プログラムが関数コード=1を用いてサブシステム全体を取り消すか、あるいはアプリケーションがSTOP RUNまたはCHAIN文を実行するとき、割付けを解除されるだけです（つまり、メモリから削除される）。サブシステムの主プログラムは、そのサブシステム内の他のプログラムが全て取り消されてしまっていない限り、CANCEL文で取り消されてはいけません。

関数コード=1の場合

本関数は指定されたサブシステム内の全てのプログラムを取り消します。サブシステム内の任意のプログラムがまだアクティブであると、そのプログラムはサブシステムから解放され、取り消されません。

関数コード=2の場合

本関数は、プログラムが入っている任意の副プログラムからその関数を呼びだしたプログラムを削除します。プログラムがどのサブシステムにも確実に含まれないようにするには、プログラムの各要素の開始時に本関数を呼び出すようにします。

CBL_SWAP_SCR_CHATTRS

文字とそれらの属性の列を画面からの列に交換します。

構文:

```
CALL "CBL_SWAP_SCR_CHATTRS" USING  画面位置
                                   文字バッファ
                                   属性バッファ
                                   文字列長
                             RETURNING  状態コード
```

パラメータ:

画面位置	次のように定義される集団項目:
行番号	PIC X COMP-X.
カラム番号	PIC X COMP-X.
文字バッファ	PIC X(n).
属性バッファ	PIC X(n).
文字列長	PIC X(2) COMP-X.

状態コード　　「用語説明」の項を参照してください。

入口上のパラメータ:

画面位置　　書出しを開始する画面位置。左上の角が行0、カラム0です。本付録において前述した「カテゴリー別ルーチン」の項の「画面」を参照してください。

文字バッファ　書き出す文字列。

属性バッファ　書き出す属性列。

文字列長　　書き出す文字列の長さ。この書出しが画面の終わりからはみ出す場合、書出しは画面の終わりで終了します。

出口上のパラメータ:

文字バッファ　画面から読み込まれる文字。このデータ項目は少なくとも文字列長により指定された長さをしている必要があります。この長さを超えたデータ項目内の位置については変更はありません。

属性バッファ　画面から読み込まれる属性。このデータ項目は少なくとも文字列長により指定された長さをしている必要があります。この長さを超えたデータ項目内の位置については変更はありません。利用者の環境における属性バイトのレイアウトについての詳細は、「リリースノート」を参照してください。

文字列長　　画面の終わりに到達すると、スワッピングされた長さ（セル数、つまり、文字-属性ペア数）がここで返されます。

CBL_TERM_MOUSE

マウス支援を終了して、内部資源を開放します。

構文:

```
CALL "CBL_TERM_MOUSE" USING  マウスハンドル
                     RETURNING  状態コード
```

パラメータ:

マウスハンドル　PIC X(4) COMP-X.

状態コード　　「用語説明」の項を参照してください。

入口上のパラメータ:

マウスハンドル　マウスの一意名で、前のCBL_INIT_MOUSEに対する呼出しにより獲得されます。

出口上のパラメータ:　なし

説明:

CBL_INIT_MOUSEにより割り付けられた内部資源を本ルーチンは開放します。本ルーチンの後では、マウスハンドルはもはや有効でなく、CBL_INIT_MOUSE以外のマウスルーチンを呼び出すとエラーとなります。

CBL_マウスルーチンをx"AF"マウスルーチンと共に使用することはできません。

CBL_TEST_RECORD_LOCK

ファイルに対する既存のレコードロックをテストします。

構文:

```
CALL "CBL_TEST_RECORD_LOCK" USING  ファイルハンドル
                                   レコード相対番地
                                   レコード長
                                   予約済み
```

パラメータ:

ファイルハンドル	PIC X（4）COMP-X.
レコード相対番地	PIC X（4）COMP-X.
レコード長	PIC X（4）COMP-X.
予約済み	PIC X（2）COMP-X.
RETURN-CODE	PIC X（2）COMP-X.

入口上のパラメータ:

ファイルハンドル	CBL_OPEN_FILEまたはCBL_CREATE_FILE呼出しが成功した場合に返されるハンドル
レコード相対番地	ロックをテストする先頭バイトに対するファイルの先頭の相対位置
レコード長	ロックをテストするバイトの数。レコード相対番地から数えます。
予約済み	予約済み。ゼロに設定する必要があります。

出口上のパラメータ:

RETURN-CODE	0	レコードがロックされていません。
	1	ロックはすでに実行単位が所有しています。
	9/nnn	エラー状態を示すファイル状態コード

CBL_THREAD_CREATE

名前付き入口点からスレッドを作成します。

構文:

```
CALL "CBL_THREAD_CREATE" USING BY REFERENCE  入口名
                               BY REFERENCE  スレッドパラメータ
                           BY VALUE  パラメータサイズ
                           BY VALUE  フラグ
                           BY VALUE  優先順位
                           BY VALUE  スタックサイズ
                           BY REFERENCE スレッドID
```

パラメータ:

入口名	PIC X(n)	**スペースまたはNULLで終わる**
スレッドパラメータ	**任意**	
パラメータサイズ	PIC X(4) COMP-5	
フラグ	PIC X(4) COMP-5	
優先順位	PIC s9(9) COMP-5	

スタックサイズ　　PIC X(4) COMP-5
スレッドID　　**ポインタ**

入口上のパラメータ:

入口名　　**スペースまたはNULLで終わるスレッドの名前**
スレッドパラメータ　　**新しいスレッドに渡されるパラメータ**
パラメータサイズ　　0　　**何もデータ値をコピーせずに「スレッドパラメータ」のアドレスだけがパラメータとして新しいスレッドの入口点に渡されます。**
0より大きい値　　**「スレッドパラメータ」の値がパラメータサイズの長さ用にコピーされ、このコピー領域のアドレスが新しいスレッドに渡されます。「スレッドパラメータ」の値は、作成中のスレッドがCBL_THREAD_CREATE呼出しの直後に「スレッドパラメータ」を変更するか割当てを解除しても作成されたスレッドのためそのままで残されます。**
フラグ　　**次のようにビットを設定できます。**

ビット	値	意味
0	0	スレッドが終了すると、直ちにスレッドを切り離します。
	1	スレッドを直ちに切り離しません。スレッドが終了しても、そのリソースはスレッドが切り離されるまでは解放されません。
1	0	相対的なスレッド優先順位（-100～+100）
	1	絶対的なスレッド優先順位（0～100）
2	0	スレッドが終了したときにこのスレッドによってロックされているモニターは、ランタイムシステムエラーを起こす可能性があります。
	1	スレッドが終了したときにモニターがロックされていてもエラーは発生しません。
3	0	スレッドをアクティブにします。
	1	スレッドをサスペンドします。
4-31		予約済み

優先順位　　**優先順位を設定します。通常、優先順位は-100から+100までの値で現行のスレッド優先順位と相対的な値です。-1および+1は、スレッドに呼出し側よりも低い/高い優先順位を与える最小の減少/増加を示します（呼出し側にすでに最低/最高優先順位が設定されてない場合）。負または正の値により実際に設定できる優先順位よりも低い/高い優先順位が設定された場合、可能な範囲内で最小/最大の値が使用されます。「フラグ」のビット1に1が設定された場合、優先順位は絶対値0から100までの値を取ります。**

スタックサイズ　　ランタイムシステムに、スタックのサイズを新しいスレッドに通知させます。0を設定すると、省略値が使用されます。パラメータが不正なことがわかると、ランタイムシステムはそれを無視するか不正な値が設定されたことを示すエラーを返します。

出口上のパラメータ:

スレッドID スレッド識別子

リターンコード　　成功したかどうかを示す値。詳細は、「同期ルーチンのRETURN-CODE値」を参照してください。

説明:

指定された入口点は、どの言語でも実行できます。

呼出しが成功して新しいスレッドが作成された場合、そのスレッドの識別子は「スレッドID」に格納され、RETURN-CODEは0に設定されます。これは、スレッドが作成時に切り離された場合でも行われます。ただし、この場合、呼出し側は他の呼出しでそのスレッドIDが使用されるまでは新しいスレッドがそのまま存在するように気を付ける必要があります。

呼出しが失敗した場合、スレッドIDにはNULLが設定され、エラー番号がRETURN-CODEに設定されます。

CBL_THREAD_CREATE_P

手続きポインタからスレッドを作成します。

構文:

```
CALL "CBL_THREAD_CREATE_P" USING BY VALUE 入口名
                           BY REFERENCE  スレッドパラメータ
                         BY VALUE パラメータサイズ
                         BY VALUE フラグ
                         BY VALUE 優先順位
                         BY VALUE スタックサイズ
                         BY REFERENCE スレッドID
```

パラメータ:

入口点	**手続きポインタ**
スレッドパラメータ	**任意**
パラメータサイズ	PIC X(4) COMP-5
フラグ	PIC X(4) COMP-5
優先順位	PIC s9(9) COMP-5
スタックサイズ	PIC X(4) COMP-5
スレッドID	**ポインタ**

入口上のパラメータ:

入口点　　**実行するプログラムの適切な入口点に設定された手続きポインタ**

スレッドパラメータ　　**新しいスレッドに渡されるパラメータ**

パラメータサイズ　　0　　**何もデータ値をコピーせずに「スレッドパラメータ」のアドレスだけがパラメータとして新しいスレッドの**

入口点に渡されます。
0より大きい値　「スレッドパラメータ」の値がパラメータサイズの長さ用にコピーされ、このコピー領域のアドレスが
新しいスレッドに渡されます。「スレッドパラメータ」の値は、作成中のスレッドがCBL_THREAD_CREATE呼出し
の直後に「スレッドパラメータ」を変更するか割当てを解除しても作成されたスレッドのためそのままで残さ
れます。
フラグ　次のようにビットを設定できます。

ビット	値	意味
0	0	スレッドが終了すると、直ちにスレッドを切り離します。
	1	スレッドを直ちに切り離しません。スレッドが終了しても、そのリソースはスレッドが切り離されるまでは解放されません。
1	0	相対的なスレッド優先順位（-100～+100）
	1	絶対的なスレッド優先順位（0～100）
2	0	スレッドが終了したときにこのスレッドによってロックされているモニターは、ランタイムシステムエラーを起こす可能性があります。
	1	スレッドが終了したときにモニターがロックされていてもエラーは発生しません。
3	0	スレッドをアクティブにします。
	1	スレッドをサスペンドします。
4-31		予約済み

優先順位　優先順位を設定します。通常、優先順位は-100から+100までの値で現行のスレッド優先順位と相対的な値です。-1および+1は、スレッドに
呼出し側よりも低い/高い優先順位を与える最小の減少/増加を示します（呼出し側にすでに最低/最高優先順位が設定されてない場合）。
負または正の値により実際に設定できる優先順位よりも低い/高い優先順位が設定された場合、可能な範囲内で最小/最大の値が使用されます。
「フラグ」のビット1に1が設定された場合、優先順位は絶対値0から100までの値を取ります。

スタックサイズ　ランタイムシステムに、スタックのサイズを新しいスレッドに通知させます。0を設定すると、省略値が使用されます。パラメータが不正な
ことがわかると、ランタイムシステムはそれを無視するか不正な値が設定されたことを示すエラーを返します。

出口上のパラメータ:

スレッドID スレッド識別子

リターンコード　成功したかどうかを示す値。詳細は、「スレッド制御ルーチンのRETURN-CODE値」を参照してください。

説明:

指定された入口点は、どの言語でも実行できます。

呼出しが成功して新しいスレッドが作成された場合、そのスレッドの識別子は「スレッドID」に格納され、RETURN-CODEは0に設定されます。これは、スレッドが作成時に切り離された場合でも行われます。ただし、この場合、呼出し側は他の呼出しでそのスレッドIDが使用されるまでは新しいスレッドがそのまま存在するように気を付ける必要があります。

呼出しが失敗した場合、スレッドIDにはNULLが設定され、エラー番号がRETURN-CODEに設定されます。

CBL_THREAD_DETACH

スレッドの終了時にスレッドのリソースを解放します。

構文:

```
CALL "CBL_THREAD_DETACH" USING BY VALUE  スレッドID
```

パラメータ:

スレッドID　ポインタ

入口上のパラメータ:

スレッドID　スレッド識別子へのポインタ

出口上のパラメータ:

リターンコード　成功したかどうかを示す値。詳細は、「スレッド制御ルーチンのRETURN-CODE値」を参照してください。

説明:

CBL_THREAD_DETACHにより、スレッドが終了したときにスレッドのリソースを解放することができます。スレッドが作成されると（切り離しなし）、ハンドルが作成側に返されます。そのハンドルは、スレッドを待ったり、スレッドからリターンコードを取り出したり、スレッドの状態を検査したりするために使用できます。スレッドが終了しても、そのハンドルは待機または切り離しが終わるまで有効です。つまり、リソースが（終了した）スレッドにまだ割り付けられています。スレッドが切り離されたときには、終了したスレッドからすべてのリソースが直ちに解放されます。スレッドがアクティブな状態である場合は、そのリソースはスレッドが終了したときに解放されるようにマークされます。

次のような場合の動作は不定です。

- スレッドIDが定義されていない。

- 2つのスレッドが動じに同じスレッドに対してCBL_THREAD_WAITまたはCBL_THREAD_DETACHを呼び出そうとした。

本ルーチンが成功した場合、RETURN-CODEに0が設定されます。失敗した場合は、エラー番号を返そうとします。ただし、動作が不定な場合もあるため、呼出しが失敗するとランタイムシステムエラーになります。

CBL_THREAD_EXIT

現行のスレッドを終了します。

構文:

```
CALL "CBL_THREAD_EXIT" USING BY VALUE リターン値
```

パラメータ:

リターン値 ポインタ

入口上のパラメータ: なし

出口上のパラメータ:

リターン値 スレッド処理から返された値

リターンコード 成功したかどうかを示す値。詳細は、「スレッド制御ルーチンのRETURN-CODE値」を参照してください。

説明:

CBL_THREAD_CREATEにより作成されたスレッドでは、本呼出しの結果は、COBOLプログラムからSTOP RUN RETURNING "リターン値"を実行したとき、または他の言語のプログラムからcobexit()を実行したときと同じになります。

この関数は、他のプログラム言語で記述したプログラムからもcobexit()に優先して呼び出すことができます。

この呼出しが失敗すると、現行のスレッドがCBL_THREAD_CREATEで作成されていない場合はエラーコードを返します。

CBL_THREAD_IDDATA_ALLOC

現行のスレッド用にIDデータ領域を割り付けます。

構文:

```
CALL "CBL_THREAD_IDDATA_ALLOC" USING BY REFERENCE IDデータ
                                     BY VALUE IDデータサイズ
```

パラメータ:

IDデータ PIC X(n)

IDデータサイズ PIC X(4) COMP-5

入口上のパラメータ:

IDデータ　現行スレッド用のIDデータ領域

IDデータサイズ　IDデータ領域のサイズ。ゼロの場合、スレッドにIDデータ領域はありません。

出口上のパラメータ:

リターンコード　成功したかどうかを示す値。詳細は、「スレッド制御ルーチンのRETURN-CODE値」を参照してください。

説明:

本呼出しは、現行スレッド用にIDデータ領域を割り付けます。そしてそれを登録し、そのスレッドのハンドルを用いてさらにCBL_THREAD_IDDATA_GETを呼び出し、このデータ領域にポインタを返すようにします。CBL_THREAD_LIST_STARTまたはCBL_THREAD_LIST_NEXTは、このデータ領域へのポインタを取り出すのに使用できます。

すでにこのスレッド用のIDデータ領域がある場合、その領域が解放されて新しい領域に置きかえられます。ただし、呼出しが失敗した場合にはこの置き換えは行われません。つまり、一般的にはメモリ不足により新しいIDデータ領域を割り付けることができないような場合に、この置き換えが行われます。

「IDデータ」パラメータに値0が設定されて渡されると、割り付けられていた領域が少ない領域に初期化されます。0以外の場合は、割り付けられていたメモリ領域用に「IDデータ」により初期化データが与えられます。

CBL_THREAD_IDDATA_GET

指定されたスレッド用にIDデータ領域へのポインタを返します。

構文:

```
CALL "CBL_THREAD_IDDATA_GET" USING BY REFERENCE  IDデータポインタ
                                   BY VALUE  スレッドID
```

パラメータ:

IDデータポインタ　ポインタ

スレッドID　ポインタ

入口上のパラメータ:

IDデータポインタ　指定されたスレッド用のIDデータ領域へのポインタ

スレッドID　スレッド識別子へのポインタ

出口上のパラメータ:

リターンコード　成功したかどうかを示す値。詳細は、「スレッド制御ルーチンのRETURN-CODE値」を参照してください。

説明:

本呼出しは、指定されたスレッド用にIDデータ領域へのポインタを返します。また、IDデータ領域がこれまで割り付けられていない場合は、NULLを返します。

現行スレッドのIDデータ領域へのポインタが返された場合は、「スレッドID」がNULLである可能性があります。

ほかのスレッドのIDデータ領域がこの関数によって取り出された場合、ユーザは必ずすべてのスレッドのIDデータ領域へのアクセスをロックなどによって防ぐようにする必要があります（たとえば、CBL_THREAD_LOCKおよびCBL_THREAD_UNLOCKを使用してその領域へのアクセスを制限するなど）。

CBL_THREAD_KILL

指定されたスレッドを中断し、そのスレッドを異常終了させて関連するリソースをすべて解放します。

構文:

```
CALL "CBL_THREAD_KILL" USING BY VALUE スレッドID
```

パラメータ:

スレッドID　　ポインタ

入口上のパラメータ:

スレッドID　　スレッド識別子へのポインタ

出口上のパラメータ:

リターンコード　　成功したかどうかを示す値。詳細は、「スレッド制御ルーチンのRETURN-CODE値」を参照してください。

説明:

本ルーチンは、CBL_THREAD_CREATEで作成されたスレッドを中断するためにだけ使用でき、それ以外の方法で作成されたスレッドは中断しません。

目標のスレッドがすでに完了しているが切り離されていない場合、本呼出しはそのスレッドの切り離しだけを行い、リターン値を破棄します。

本ルーチンには、現行のスレッドに設定されたスレッドIDを使用することができます。この場合、その結果はNULLリターン値が設定されたCBL_THREAD_EXITと同じになります。

次のような場合の動作は不定です。

- スレッドIDの値が不正である場合。
- スレッドが中断されたスレッドに対してCBL_THREAD_WAIT呼出しを行う場合。

CBL_THREAD_LIST_END

CBL_THREAD_LIST_STARTおよびCBL_THREAD_LIST_NEXTと組み合わせて使用すると、ランタイムシステムが認識している現行スレッドすべてのリストを取得できます。本ルーチンは、CBL_THREAD_LIST_STARTを終了するのに必要です。

構文:

```
CALL "CBL_THREAD_LIST_END"
```

パラメータ: なし

入口上のパラメータ: なし

出口上のパラメータ:

リターンコード 成功したかどうかを示す値。詳細は、「スレッド制御ルーチンのRETURN-CODE値」を参照してください。

説明:

本ルーチンは、失敗するとRETURN-CODEにエラー番号を返します。ただし、動作が不定な場合もあるため、本ルーチンの呼出しに失敗するとランタイムシステムエラーとなることもあります。

CBL_THREAD_LIST_NEXT

CBL_THREAD_LIST_STARTおよびCBL_THREAD_LIST_ENDと組み合わせて使用すると、ランタイムシステムが認識している現行スレッドすべてのリストを取得できます。本ルーチンは、スレッドのリスト中にある次の項目を返します。

構文:

```
CALL "CBL_THREAD_LIST_NEXT" USING BY REFERENCE スレッドID
                                  BY REFERENCE スレッド状態
                                  BY REFERENCE スレッドIDデータ
```

パラメータ:

スレッドID	**ポインタ**
スレッド状態	PIC X(4) COMP-5
スレッドIDデータ	**ポインタ**

入口上のパラメータ: なし

出口上のパラメータ:

スレッドID	**本ルーチンの内部リストにある先頭のスレッド識別子**
スレッド状態	**次のようにスレッド状態が設定されます。**

ビット	値	意味
0	0	スレッドは切り離されます。
	1	スレッドは切り離されません。
1	0	スレッドはサスペンドされません。
	1	スレッドはCBL_THREAD_SUSPENDによってサスペンドされます。
2	0	スレッドはメインスレッドではありません。
	1	スレッドは他言語のプログラムであり、ランタイムシステムに認識されています。または、メインスレッドです。
3-31		未定義。どの値も設定されます。

スレッドIDデータ　　スレッドのIDデータ領域が存在する場合はその領域へのポインタ（それ以外はNULL）。IDデータはCBL_THREAD_ALLOC_IDDATAルーチンによって指定されます。
リターンコード　　成功したかどうかを示す値。詳細は、「スレッド制御ルーチンのRETURN-CODE値」を参照してください。

説明:

既存のスレッド、つまり終了していないスレッドに関する情報だけが返されます。本ルーチンは、終了しているが切り離されていないスレッドに対しては、スレッドIDを返しません。

スレッドIDがNULLの場合、リストの先頭の項目が返されます（ただし、この場合はCBL_THREAD_LIST_START呼出しがその前に行われているはずです）。

スレッドIDの値が不正である場合の動作は不定です。

CBL_THREAD_LIST_START

CBL_THREAD_LIST_NEXTおよびCBL_THREAD_LIST_ENDと組み合わせて使用すると、ランタイムシステムが認識している現行スレッドすべてのリストを取得できます。

構文:

```
CALL "CBL_THREAD_LIST_START" USING BY REFERENCE スレッドID
                         BY REFERENCE  スレッド状態
                      BY REFERENCE スレッドIDデータ
```

パラメータ:

スレッドID	**ポインタ**
スレッド状態	PIC X(4) COMP-5
スレッドIDデータ	**ポインタ**

入口上のパラメータ: なし

出口上のパラメータ:

スレッドID　　本ルーチンの内部リストにある先頭のスレッド識別子
スレッド状態　　次のようにスレッド状態が設定されます。

ビット	値	意味
0	0	スレッドは切り離されます。
	1	スレッドは切り離されません。
1	0	スレッドはサスペンドされません。
	1	スレッドはCBL_THREAD_SUSPENDによってサスペンドされます。
2	0	スレッドはメインスレッドではありません。
	1	スレッドは他言語のプログラムであり、ランタイムシステムに認識されています。または、メインスレッドです。
3-31		未定義。どの値も設定されます。

スレッドIDデータ　　スレッドのIDデータ領域が存在する場合はその領域へのポインタ（それ以外はNULL）。IDデータはCBL_THREAD_ALLOC_IDDATAルーチンによって指定されます。

リターンコード　　成功したかどうかを示す値。詳細は、「スレッド制御ルーチンのRETURN-CODE値」を参照してください。

説明:

既存のスレッド、つまり終了していないスレッドに関する情報だけが返されます。本ルーチンは、終了しているが切り離されていないスレッドに対しては、スレッドIDを返しません。

CBL_THREAD_LOCK

スレッド処理ルーチンのほとんどの関数をロックします。

構文:

```
CALL "CBL_THREAD_LOCK"
```

パラメータ: なし

入口上のパラメータ: なし

出口上のパラメータ:

リターンコード　成功したかどうかを示す値。詳細は、「スレッド制御ルーチンのRETURN-CODE値」を参照してください。

説明:

本ルーチンは、グローバルデータまたは外部データ、IDデータ、およびエラー処理などへのアクセスを同期化するためにも使用できます。

CBL_THREAD_LOCKを呼び出すスレッドは、CBL_TREAD_UNLOCKを行うスレッドの前にあってはなりません。

CBL_THREAD_PROG_LOCK

CBL_THREAD_PROG_UNLOCKと組み合わせて使用すると、単独の事前に初期化された同期化オブジェクトをCOBOLプログラムの呼出しごとに提供します。

構文:

```
CALL "CBL_THREAD_PROG_LOCK"
```

パラメータ:　なし

入口上のパラメータ:　なし

出口上のパラメータ:

リターンコード　成功したかどうかを示す値。詳細は、「スレッド制御ルーチンのRETURN-CODE値」を参照してください。

説明:

この関数は、CBL_THREAD_PROG_UNLOCKと組み合わせて使用すると、単独の事前に初期化された同期化オブジェクトをCOBOLプログラムの呼出しごとに提供します。これは、プログラムローカルデータ項目または同期化オブジェクトなどを初期化する場合に特に便利です。例

これらの関数は、COBOLプログラムから直接または間接的に呼び出す必要があります。呼出しCOBOLプログラムにはロックが関連付けられています。

CBL_THREAD_PROG_UNLOCK

CBL_THREAD_PROG_LOCKと組み合わせて使用すると、単独の事前に初期化された同期化オブジェクトをCOBOLプログラムの呼出しごとに提供します。

構文:

```
CALL "CBL_THREAD_PROG_UNLOCK"
```

パラメータ:　なし

入口上のパラメータ:　なし

出口上のパラメータ:

リターンコード　成功したかどうかを示す値。詳細は、「スレッド制御ルーチンのRETURN-CODE値」を参照してください。

説明:

この関数は、CBL_THREAD_PROG_LOCKと組み合わせて使用すると、単独の事前に初期化された同期化オブジェクトをCOBOLプログラムの呼出しごとに提供します。これは、プログラムローカルデータ項目または同期化オブジェクトなどを初期化する場合に特に便利です。例

これらの関数は、COBOLプログラムから直接または間接的に呼び出す必要があります。呼出しCOBOLプログラムにはロックが関連付けられています。

CBL_THREAD_RESUME

CBL_THREAD_SUSPENDによってサスペンドされた、またはサスペンドされるスレッドを再開します。

構文:

```
CALL "CBL_THREAD_RESUME" USING BY VALUE スレッドID
```

パラメータ:

スレッドID　ポインタ

入口上のパラメータ:

スレッドID　スレッド識別子へのポインタ

出口上のパラメータ:

リターンコード　成功したかどうかを示す値。詳細は、「スレッド制御ルーチンのRETURN-CODE値」を参照してください。

説明:

本ルーチンの呼び出し時に指定されたスレッドがサスペンドされている場合、RETURN-CODEに0が設定されます。サスペンドされていなかった場合には、RETURN－CODEには負の値が設定されます。この値は、対象のスレッドが実際にサスペンドされる前にCBL_THREAD_SUSPENDを呼び出す必要がある回数です。

スレッドIDの値が不正である場合の動作は不定です。

CBL_THREAD_SELF

現行スレッドのスレッド識別子を格納します。

構文:

```
CALL "CBL_THREAD_SELF" USING BY REFERENCE スレッドID
```

パラメータ:

スレッドID　ポインタ

入口上のパラメータ:

スレッドID　スレッド識別子へのポインタ

出口上のパラメータ:

リターンコード　成功したかどうかを示す値。詳細は、「スレッド制御ルーチンのRETURN-CODE値」を参照してください。

説明:

ランタイムシステムはスレッド識別子を返し、それを「スレッドID」に格納します。

本ルーチンを呼び出すことによって、プログラムが実行されているランタイムシステムがマルチスレッドをサポートしているかどうかを確認できます。

マルチスレッドランタイムシステムを確認する場合のCBL_THREAD_SELFの使用例

CBL_THREAD_SLEEP

呼出しスレッドに指定されたミリ秒間だけプロセッサを制御するのを止めさせます。

構文:

```
CALL "CBL_THREAD_SLEEP" USING BY VALUE ミリ秒
```

パラメータ:

値　ミリ秒

入口上のパラメータ:

値　ミリ秒数

出口上のパラメータ:

リターンコード　成功したかどうかを示す値。詳細は、「スレッド制御ルーチンのRETURN-CODE値」を参照してください。

説明:

オペレーティングシステムがミリ秒間という単位でスレッドを放棄することをサポートしていない場合は、その時間の単位はオペレーティングシステムがサポートする単位に調整されます。その調整は、オペレーティングシステムがサポートする単位で最も近い値に丸め込むという方法です。

CBL_THREAD_SUSPEND

現行スレッドをサスペンドします。

構文:

```
CALL "CBL_THREAD_SUSPEND" USING BY VALUE スレッドID
```

パラメータ:

スレッドID　ポインタ

入口上のパラメータ:

スレッドID　スレッド識別子へのポインタ

出口上のパラメータ:

リターンコード　成功したかどうかを示す値。詳細は、「スレッド制御ルーチンのRETURN-CODE値」を参照してください。

説明:

スレッドIDがNULLまたは現行のスレッドである場合には、本ルーチンは他のスレッドがCBL_THREAD_RESUMEを現行のスレッドに対して呼び出すまでその現行スレッドをサスペンドします。本ルーチンは、成功するとRETURN-CODEに0を設定します。

1つ以上のCBL_THREAD_RESUMEルーチンがすでにこのスレッドに対して呼び出されている場合、このスレッドはRETURN-CODEに負の値を設定します。RETURN－CODEには負の値が設定されます。この値は、対象のスレッドが実際にサスペンドされる前にCBL_THREAD_SUSPENDを呼び出す必要がある回数です。

それ以外の場合は、ゼロ以外の（正の）エラーコードを返します。

現行スレッド以外のスレッドをサスペンドすることはできません。

スレッドIDの値が不正である場合の動作は不定です。

CBL_THREAD_UNLOCK

スレッド処理ルーチンのほとんどの関数のロックを解除します。

構文:

```
CALL "CBL_THREAD_UNLOCK"
```

パラメータ:　なし

入口上のパラメータ:　なし

出口上のパラメータ:

リターンコード　成功したかどうかを示す値。詳細は、「スレッド制御ルーチンのRETURN-CODE値」を参照してください。

説明:

本ルーチンは、グローバルデータまたは外部データ、IDデータ、およびエラー処理などへのアクセスを同期化するためにも使用できます。

CBL_THREAD_LOCKを呼び出すスレッドは、CBL_TREAD_UNLOCKを行うスレッドの前にあってはなりません。

CBL_THREAD_WAIT

指定された切り離されていないスレッドの完了を待ちます。スレッドがすでに完了している場合は直ちに戻ります。

構文:

```
CALL "CBL_THREAD_WAIT" USING BY VALUE スレッドID
                BY REFERENCE リターン値
```

パラメータ:

スレッドID　　ポインタ
リターン値　　ポインタ

入口上のパラメータ:

スレッドID　　スレッド識別子へのポインタ

出口上のパラメータ:

リターンコード　　スレッドのリターン値。詳細は、「スレッド制御ルーチンのRETURN-CODE値」を参照してください。

説明:

次のような場合の動作は不定です。

- スレッドIDの値が不正である場合。
- スレッドIDがそれまでに切り離されたスレッドを指定している場合。
- 2つのスレッドが同時に同じスレッドに対してCBL_THREAD_WAITまたはCBL_THREAD_DETACHを呼び出そうとした場合。
- CBL_THREAD_WAITの処理中に対象のスレッドがCBL_THREAD_KILLによって中断された場合。

本ルーチンは、成功するとスレッドのリターン値を「リターン値」に格納し、対象のスレッドが切り離されると0を返します。

CBL_THREAD_YIELD

現行のスレッドのタイムスライスの残り時間を放棄します。

構文:

```
CALL "CBL_THREAD_YIELD"
```

パラメータ:　なし

入口上のパラメータ:　なし

出口上のパラメータ:

リターンコード　成功したかどうかを示す値。詳細は、「スレッド制御ルーチンのRETURN-CODE値」を参照してください。

説明:

本ルーチンは可能であれば同じプロセス内の他のスレッドに時間を譲りますが、できなければ別のプロセス内のスレッドに譲ります。

CBL_TOLOWER

文字の列を小文字に変換します。

構文:

```
CALL "CBL_TOLOWER" USING  文字列
                BY VALUE  長さ
                RETURNING  状態コード
```

パラメータ:

文字列　PIC X(n).

長さ　PIC X(4) COMP-5.

状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

文字列　変換する文字列。

長さ　変更する文字列のバイト数で、これを超えた位置については変更はありません。

出口上のパラメータ:

文字列　変換された文字列。

説明:

本ルーチンは文字列の左端で開始し、1バイト文字を小文字に変換します（小文字へのフォールドとも呼ばれています）。

CBL_TOUPPER

文字の列を大文字に変換します。

構文:

```
CALL "CBL_TOUPPER" USING  文字列
                BY VALUE  長さ
                RETURNING  状態コード
```

パラメータ:

文字列　PIC X(n).

長さ　PIC X(4) COMP-5.

状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

文字列　変換する文字列。

長さ　変更する文字列のバイト数で、これを超えた位置については変更はありません。

出口上のパラメータ:

文字列　変換された文字列。

説明:

本ルーチンは文字列の左端で開始し、1バイト文字を大文字に変換します（大文字へのフォールドとも呼ばれています）。

CBL_TSOTRE_CLOSE

CBL_TSTORE_CREATEによって返されたハンドルを閉じます。

構文:

```
CALL "CBL_TSOTRE_CLOSE" USING BY VALUE スレッド格納ハンドル
```

パラメータ:

スレッド格納ハンドル　ポインタ

入口上のパラメータ:

スレッド格納ハンドル　スレッド格納ハンドル

出口上のパラメータ:　なし

説明:

本ルーチンは、CBL_TSOTRE_CREATEによって返されたスレッド格納領域へのハンドルを閉じます。そして全スレッドに割り付けられていたメモリをすべて解放します。

CBL_TSOTRE_CREATE

スレッド領域用のハンドルを作成し、スレッド領域のサイズを指定します。これらの値は、後続のCBL_TSTORE_GETの呼出しで使用できます。

構文:

```
CALL "CBL_TSOTRE_CREATE" USING スレッド格納ハンドル
                     BY VALUE スレッド格納サイズ
                     BY VALUE スレッド格納フラグ
```

パラメータ:

スレッド格納ハンドル	**ポインタ**
スレッド格納サイズ	PIC s9(9) COMP-5
スレッド格納フラグ	PIC s9(9) COMP-5

入口上のパラメータ:なし

出口上のパラメータ:

スレッド格納サイズ　返されたハンドルを使用してCBL_TSTORE_GET**を呼び出すスレッドに返されるスレッド格納領域のサイズ**
フラグ　次のようにスレッド格納領域の種類をビットで設定できます。

ビット	値	意味
0	0	**予約済み。**0**を設定する必要があります。**
1	0	**予約済み。**0**を設定する必要があります。**
2	0	**スレッド格納ハンドルを割り当てます。ハンドルは呼出しプログラムと関連があり、呼出しプログラムが中断されると自動的に割当てを解除されます。** **どの呼出しプログラムとも関連のないスレッド格納ハンドルを割り当てます。このビットが設定されると、スレッド格納ハンドルは実行単位の最後に閉じられます。**
3-31	0	**予約済み。**0**を設定する必要があります。**

出口上のパラメータ:

スレッド格納ハンドル　スレッド格納ハンドル
リターンコード　成功したかどうかを示します。
0　**割当て成功**
1000　**スレッド格納ハンドルを割り当てられません。**

説明:

CBL_TSTORE_CREATEは、後続のCBL_TSTORE_GET呼出しによって返されるメモリブロックサイズを指定します。また、オプションでそのハンドルを呼出しプログラムと関連付け、返されたハンドルが呼出しプログラムが中断されたときに自動的に閉じられるようにします。どちらの場合も、返されたハンドルは実行単位またはCBL_TSTORE_CLOSE呼出しの終了時に閉じられます。

このスレッド格納ハンドルは、プログラムと関連を持っていない場合は実行単位の最後に閉じられます。

本ルーチンは次の方法によってシングルスレッドモードで呼び出す必要があります。それは、モニターロック内でシングルスレッド動作を無理に行う方法か、シングルスレッドモードでアプリケーションを初期化する場合に本ルーチンを呼び出す方法です。

CBL_TSOTRE_GET

CBL_TSTORE_CREATEによって指定されたスレッド固有のメモリへのポインタを取得します。

構文:

```
CALL "CBL_TSOTRE_GET" USING BY VALUE スレッド格納ハンドル
                                     スレッド格納ポインタ
```

パラメータ:

スレッド格納ハンドル　　USAGE POINTER

スレッド格納ポインタ　　USAGE POINTER

入口上のパラメータ:

スレッド格納ハンドル　　NULL以外のハンドル。このハンドルは、CBL_TSOTRE_CREATEによって返されたハンドルです。

出口上のパラメータ:

スレッド格納ポインタ　　CBL_TSTORE_CREATEによって指定されたサイズのメモリブロックへのポインタ。

リターンコード　　成功したかどうかを示します。

0 割当て成功

1000 要求されたメモリブロックを取得できません。

1001 ハンドルの値が不正です。

1002 ハンドルが閉じています。

説明:

本ルーチンは、CBL_TSOTRE_CREATEによって指定されたサイズのメモリブロックへのポインタを返します。このポインタは、各呼出しプロセスに対してユニークです。

値はRETURN-CODEに返されます。これはルーチンが成功したかどうかを示します。

CBL_WRITE_FILE

ファイルへバイト列を書き出します。

構文:

```
CALL "CBL_WRITE_FILE" USING  ファイルハンドル
                             ファイル相対番地
                             バイト数
                             フラグ
                             バッファ
```

パラメータ:

ファイルハンドル　　PIC X(4).

ファイル相対番地　　PIC X(8) COMP-X.

バイト数　　PIC X(4) COMP-X.

フラグ　　PIC X COMP-X.

バッファ　　PIC X(n).

入口上のパラメータ:

ファイルハンドル　ファイルが開かれたとき戻されたファイルハンドル

ファイル相対番地　書出しを行うファイル内の相対番地。本項目は現在最大値x "00FFFFFFFF"に制限されています。

バイト数　書き出すバイト数。本項目は現在最大値x "00FFFF"に制限されています。
値ゼロを本フィールドに入れると、ファイルはファイル相対番地項目で指定されたサイズに打ち切られるか、あるいは、拡張されます。

フラグ　本パラメータは次の値をとります:
0　標準の書出し用

バッファ　そこからバイト列が書き出されるバッファ。利用者の責任において、書き出されるバイト数に足るバッファサイズを確保してください。

出口上のパラメータ:　なし

説明:

呼出しが成功したかどうかは、RETURN-CODEを調べればわかります。

CBL_WRITE_SCR_ATTRS

画面へ属性の列を書き出します。

構文:

```
CALL "CBL_WRITE_SCR_ATTRS" USING  画面位置
                                  属性バッファ
                                  文字列長
                            RETURNING  状態コード
```

パラメータ:

画面位置　次のように定義される集団項目:
　行番号　PIC X COMP-X.
　カラム番号　PIC X COMP-X.

属性バッファ　PIC X(n).

文字列長　PIC X(2) COMP-X.

状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

画面位置　書出しを開始する画面位置。左上の角が行0、カラム0です。本付録において前述した「カテゴリー別ルーチン」の項の「画面」を参照してください。

属性バッファ　書き出す属性。利用者の環境における属性バイトのレイアウトについての詳細は、「リリースノート」を参照してください。

文字列長　書き出す文字列の長さ。この書出しが画面の終わりをはみ出す場合、書出しは画面の終わりで終了します。

出口上のパラメータ:　なし

CBL_WRITE_SCR_CHARS

画面へ文字列を書出します。

構文:

```
CALL "CBL_WRITE_SCR_CHARS" USING  画面位置
                                  文字バッファ
                                  文字列長
                          RETURNING 状態コード
```

パラメータ:

画面位置　次のように定義される集団項目:
　行番号　PIC X COMP-X.
　カラム番号　PIC X COMP-X.
文字バッファ　PIC X(n).
文字列長　PIC X(2) COMP-X.
状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

画面位置　書出しを開始する画面位置。左上の角が行0、カラム0です。本付録において前述した「カテゴリー別ルーチン」の項の「画面」を参照してください。
文字バッファ　書き出す文字列。
文字列長　書き出す文字列の長さ。この書出しが画面の終わりをはみ出す場合、書出しは画面の終わりで終了します。

出口上のパラメータ:　なし

CBL_WRITE_SCR_CHARS_ATTR

画面へ文字列を書出し、それらの文字全てに同じ属性を与えます。

構文:

```
CALL "CBL_WRITE_SCR_CHARS_ATTR" USING  画面位置
                                       文字バッファ
                                       文字列長
                                       属性
                               RETURNING 状態コード
```

パラメータ:

画面位置　次のように定義される集団項目:
　行番号　PIC X COMP-X.
　カラム番号　PIC X COMP-X.
文字バッファ　PIC X(n).
属性　PIC X.

文字列長　　PIC X(2) COMP-X.

状態コード　　「用語説明」の項を参照してください。

入口上のパラメータ:

画面位置　　書出しを開始する画面位置。左上の角が行0、カラム0です。本付録において前述した「カテゴリー別ルーチン」の項の「画面」を参照してください。

文字バッファ　書き出す文字列

属性　　書き出す属性。

文字列長　　書き出す文字列の長さ。この書出しが画面の終わりをはみ出す場合、書出しは画面の終わりで終了します。

出口上のパラメータ:　なし

CBL_WRITE_SCR_CHATTRS

画面へ文字とそれらの属性の列を書出します。

構文:

```
CALL "CBL_WRITE_SCR_CHATTRS" USING  画面位置
                                    文字バッファ
                                    属性バッファ
                                    文字列長
                             RETURNING  状態コード
```

パラメータ:

画面位置　　次のように定義される集団項目:

　行番号　　PIC X COMP-X.

　カラム番号　　PIC X COMP-X.

文字バッファ　PIC X(n).

属性バッファ　PIC X(n).

文字列長　　PIC X(2) COMP-X.

状態コード　　「用語説明」の項を参照してください。

入口上のパラメータ:

画面位置　　書出しを開始する画面位置。左上の角が行0、カラム0です。本付録において前述した「カテゴリー別ルーチン」の項の「画面」を参照してください。

文字バッファ　書き出す文字列

属性バッファ　書き出す属性列。

文字列長　　書き出す文字列の長さ。この書出しが画面の終わりをはみ出す場合、書出しは画面の終わりで終了します。

出口上のパラメータ:　なし

CBL_WRITE_SCR_N_ATTR

画面上の位置列へ指定された属性を書き出します。

構文:

```
CALL "CBL_WRITE_SCR_N_ATTR" USING  画面位置
                                   属性
                                   文字列長
                         RETURNING  状態コード
```

パラメータ:

画面位置　次のように定義される集団項目:
　行番号　PIC X COMP-X.
　カラム番号　PIC X COMP-X.
属性　PIC X.
文字列長　PIC X(2) COMP-X.
状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

画面位置　書出しを開始する画面位置。左上の角が行0、カラム0です。本付録において前述した「カテゴリー別ルーチン」の項の「画面」を参照してください。
属性　書き出す属性
文字列長　属性を書き出す画面位置の個数。この書出しが画面の終わりをはみ出す場合、書出しは画面の終わりで終了します。

出口上のパラメータ:　なし

CBL_WRITE_SCR_N_CHAR

画面上の位置列へ指定された文字を書き出します。

構文:

```
CALL "CBL_WRITE_SCR_N_CHAR" USING  画面位置
                                   文字
                                   文字列長
                         RETURNING  状態コード
```

パラメータ:

画面位置　次のように定義される集団項目:
　行番号　PIC X COMP-X.
　カラム番号　PIC X COMP-X.
文字　PIC X.
文字列長　PIC X(2) COMP-X.
状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

画面位置　書出しを開始する画面位置。左上の角が行0、カラム0です。本付録において前述した「カテゴリー別ルーチン」の項の「画面」を参照してください。

文字　書き出す文字

文字列長　文字を書き出す画面位置の個数。この書出しが画面の終わりをはみ出す場合、書出しは画面の終わりで終了します。

出口上のパラメータ:　なし

CBL_WRITE_SCR_N_CHATTR

画面上の位置列へ指定された文字と属性を書き出します。

構文:

```
CALL "CBL_WRITE_SCR_N_CHATTR" USING  画面位置
                                     文字
                                     属性
                                     文字列長
                             RETURNING  状態コード
```

パラメータ:

画面位置　次のように定義される集団項目:

　行番号　PIC X COMP-X.

　カラム番号　PIC X COMP-X.

文字　PIC X.

属性　PIC X.

文字列長　PIC X(2) COMP-X.

状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

画面位置　書出しを開始する画面位置。左上の角が行0、カラム0です。本付録において前述した「カテゴリー別ルーチン」の項の「画面」を参照してください。

文字　書き出す文字

属性　書き出す属性。

文字列長　文字－属性ペアを書き出す画面位置の個数。この書出しが画面の終わりをはみ出す場合、書出しは画面の終わりで終了します。

出口上のパラメータ:　なし

CBL_WRITE_SCR_TTY

文字列を現在位置から開始して画面に書出し、スクロールします。

構文:

```
CALL "CBL_WRITE_SCR_TTY" USING  文字バッファ
                                文字列長
                          RETURNING  状態コード
```

パラメータ:

文字バッファ　PIC X(n).

文字列長　PIC X(2) COMP-X.

状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

文字バッファ　書き出す文字列

文字列長　書き出す文字列の長さ。この書出しが画面の端をはみ出す場合、画面は上に1行スクロールし、書出しはいちばん下の行で継続します。

出口上のパラメータ:　なし

CBL_WRITE_VFILE

バイト列をヒープに書き出します。

構文:

```
CALL "CBL_WRITE_VFILE" USING BY VALUE  ヒープID
                                       ヒープ参照
                                       ヒープ長
                             BY REFERENCE  ヒープバッファ
                             RETURNING  状態コード
```

パラメータ:

ヒープ***ID***　PIC X(2)COMP-5.

ヒープ参照　PIC X(4)COMP-5.

ヒープ長　PIC X(4)COMP-5.

ヒープバッファ　PIC X(n).

状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

ヒープ*ID*　ヒープが開かれたときに割り当てられたヒープハンドルが入ります。

ヒープ参照　書出しを開始するヒープ内の相対番地

ヒープ長　書き出すバイト数

ヒープバッファ　バイト列が書き出されるバッファ。利用者の責任において、書き出されるバイト数に足るバッファサイズを確保してください。

出口上のパラメータ:　なし

CBL_XOR

2つのデータ項目のビット間の排他的論理和（OR）をとります。

構文:

```
CALL "CBL_XOR" USING 源
                     目標
             BY VALUE 長さ
```

パラメータ:

源　　任意のデータ項目

目標　　任意のデータ項目

長さ　　数字定数またはPIC X(4) COMP-5.

入口上のパラメータ:

源　　排他的論理和（XOR）をとるデータ項目の内の1つ

目標　　排他的論理和（XOR）をとるもう一方のデータ項目

長さ　　排他的論理和（XOR）をとろうとする源と目標のバイト数。この長さを超えた目標内の位置については変更はありません。

出口上のパラメータ:

目標　　結果

説明:

本ルーチンは源と目標の左端から開始し、それらのビットの排他的論理和（XOR）をとり、結果を目標に格納します。これの真理値表は次の通りです。

源	目標	結果
0	0	0
0	1	1
1	0	1
1	1	0

CBL_YIELD_RUN_UNIT

2つのデータ項目のビット間の排他的論理和（OR）をとります。

構文:

```
CALL "CBL_YIELD_RUN_UNIT"
```

パラメータ:　なし

説明:

実行単位のタイムスライスの残り時間は、指定されていない実行単位に譲られます。

MF_CLIENT_STATE_ALLOCATE

クライアント識別子を割り当てます。

構文:

```
CALL "MF_CLIENT_STATE_ALLOCATE" USING  クライアントID
                                       クライアント長
                                       サーバ状態
```

パラメータ:

クライアントID　PIC X(30)

クライアント長　PIC X(4) COMP-X.

サーバ状態　PIC X COMP-X.

入口上のパラメータ:

クライアントID　クライアント識別子。下記の「説明」を参照。

クライアント長　状態情報用の空のレコード長

出口上のパラメータ:

クライアントID　クライアント識別子。下記の「説明」を参照。

サーバ状態　操作の状態。「状態管理ルーチン」を参照

説明:

本ルーチンは、次の操作を行います。

- 状態情報用に「クライアント長」で指定された長さの新しい空のレコードを割り当てます。
- 新しい「クライアントID」を返し、「クライアントID」に渡された値にスペースが設定されている場合は空のレコードを割り当てます。それ以外の場合で「クライアントID」に有効なクライアント識別子が設定されていれば、そのクライアント識別子を空のレコードに割り当てます。

例:

```
working-storage section.
...
01 client-id              pic x(30).
01 client-length           pic xxxx comp-x.
01 state-status            pic x comp-x.
...
procedure division.
...
call "MF_CLIENT_STATE_ALLOCATE"
     using client-id client-length state-status
...
```

MF_CLIENT_STATE_DELETE

既存のクライアント識別子とそれに関連するクライアント情報を削除します。

構文:

```
CALL "MF_CLIENT_STATE_DELETE" USING  クライアントID
                                     サーバ状態
```

パラメータ:

クライアントID　PIC X(30).

サーバ状態　PIC X COMP-X.

入口上のパラメータ:

クライアントID　クライアント識別子

出口上のパラメータ:

サーバ状態　操作の状態。「状態管理ルーチン」を参照

MF_CLIENT_STATE_EXPIRY

cookieの期限切れフィールドで使用するための日付文字列を返します。

構文:

```
CALL "MF_CLIENT_STATE_EXPIRY" USING  日付基準
                                     期限切れ文字列
```

パラメータ:

日付基準　PIC X(4) COMP-X.

期限切れ文字列　PIC X(50).

入口上のパラメータ:

日付基準　何日後にクライアントレコードを期限切れにするかを設定する。

出口上のパラメータ:

期限切れ文字列　cookieの期限切れフィールドで使用する有効期限

説明:

「期限切れ文字列」は、値を日付形式*dd-month-yyyy hh:mm:ss* GMTで返します。

MF_CLIENT_STATE_FILE

状態情報を保存するためにサーバ上で使用するファイルを指定します。

構文:

```
CALL "MF_CLIENT_STATE_FILE" USING  ファイル名
                                   サーバ状態
```

パラメータ:

ファイル名　PIC X(255).

サーバ状態　PIC X COMP-X.

入口上のパラメータ:

ファイル名　ファイル名

出口上のパラメータ:

サーバ状態　　操作の状態。「状態管理ルーチン」を参照

例:

```
working-storage section.
...
01 state-status           pic x comp-x.
01 state-filename          pic x(255)
            value "MF-STATE-SAVE.DAT".
...
procedure division.
...
call "MF_CLIENT_STATE_FILE" using state-filename
                            server-status
...
```

MF_CLIENT_STATE_PURGE

指定された日数よりも古いクライアント情報を削除します。

構文:

```
CALL "MF_CLIENT_STATE_PURGE" USING  日付基準
                                    サーバ状態
```

パラメータ:

日付基準　　PIC X(4) COMP-X.

サーバ状態　　PIC X COMP-X.

入口上のパラメータ:

日付基準　　何日前からのクライアント情報を残すか設定する

出口上のパラメータ:

サーバ状態　　操作の状態。「状態管理ルーチン」を参照

MF_CLIENT_STATE_RESTORE

それまでに保存または割り当てられ、クライアント識別子に一致するレコードを戻します。

構文:

```
CALL "MF_CLIENT_STATE_RESTORE" USING  クライアントID
                                      クライアント状態
                                      クライアント長
                                      サーバ状態
```

パラメータ:

クライアントID　　PIC X(30)

クライアント状態　PIC X(4) COMP-X.

クライアント長　　下記の「説明」を参照

サーバ状態　　PIC X COMP-X.

入口上のパラメータ:

クライアントID　クライアント識別子。
クライアント状態　保存する状態情報。下記の「説明」を参照。
クライアント長　状態情報用の空のレコード長

出口上のパラメータ:

クライアント状態　保存する状態情報。下記の「説明」を参照。
サーバ状態　操作の状態。「状態管理ルーチン」を参照

説明:

「クライアント長」は、入口上はバッファの長さであり、出口上はリカバリされたデータの長さを返します。

「クライアント状態」レコードには任意の形式を定義することができます。「クライアント長」フィールドには、「クライアント状態」レコードの長さを定義してください。

例:

```
 working-storage section.
 ...
 01 client-id              pic x(30).
 01 client-length           pic xxxx comp-x.
 01 state-status            pic x comp-x.
01 client-state.
   03 user-preferences      pic x(10).
   03 user-selection-list    pic x(80).
...
 procedure division.
 ...
 call "MF_CLIENT_STATE_RESTORE"
         using client-id client-state
              client-length state-status
 ...
```

MF_CLIENT_STATE_SAVE

状態ファイルの情報を更新します。

構文:

```
CALL "MF_CLIENT_STATE_SAVE" USING  クライアントID
                                   クライアント状態
                                   クライアント長
                                   サーバ状態
```

パラメータ:

クライアントID　PIC X(30)
クライアント状態　PIC X(4) COMP-X.
クライアント長　下記の「説明」を参照
サーバ状態　PIC X COMP-X.

状態コード　　「用語説明」の項を参照してください。

入口上のパラメータ:

クライアントID　クライアント識別子。
クライアント状態　保存する状態情報。下記の「説明」を参照。
クライアント長　状態情報用の空のレコード長

出口上のパラメータ:

サーバ状態　　操作の状態。「状態管理ルーチン」を参照

説明:

本ルーチンは、これまでに割り当てられた「クライアントID」と「クライアント状態」を保存します。またはこれまでに保存された「クライアントID」および「クライアント状態」を書き換えます。「クライアント長」は更新できます。

「クライアント状態」レコードには任意の形式を定義することができます。「クライアント長」フィールドには、「クライアント状態」レコードの長さを定義してください。

例:

```
 working-storage section.
...
01 client-id             pic x(30).
01 client-length          pic x(4) comp-x.
01 state-status           pic x comp-x.
01 client-state.
  03 user-preferences      pic x(10).
  03 user-selection-list    pic x(80).
...
procedure division.
...
call "MF_CLIENT_STATE_SAVE"
      using client-id client-state
           client-length state-status
...
```

PC_FIND_DRIVES

パソコン上の全ての有効ドライブを戻します。

構文:

```
CALL "PC_FIND_DRIVES" USING  ドライブ情報
                   RETURNING  状態コード
```

パラメータ:

ドライブ情報　PIC X(4) COMP-X.
状態コード　　「用語説明」の項を参照してください。

入口上のパラメータ:　なし

出口上のパラメータ:

ドライブ情報　現存する全てのドライブ

状態コード　状態コード

PC_PRINT_FILE

指定したファイルの内容を出力します。オプションで、出力開始前にプリンタ制御、フォント、および進行状況ダイアログボックスなどを表示します。

構文:

```
CALL "PC_PRINT_FILE" USING       ファイル名
                                 ドキュメントタイトル
                           BY VALUE  フラグ
                           BY VALUE  ウィンドウハンドル
                 RETURNING    状態コード
```

パラメータ:

ファイル名　次のように定義された集団項目
　　ファイル長　PIC X(2) COMP-5
　　ファイルテキスト　PIC X(n)
ドキュメントタイトル　次のように定義された集団項目
　　タイトル長　PIC X(2) COMP-5
　　タイトルテキスト　PIC X(n)
フラグ　数字定数または PIC X(4) COMP-5
ウィンドウハンドル　数字定数または PIC X(4) COMP-5
状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

ファイル名　出力するファイルの名前（NULLまたはスペースで終わる））
タイトル長　ドキュメントタイトルの長さ
タイトルテキスト　出力するドキュメントのタイトル
フラグ　プリンタオプションを定義するビットの組
　　ビット0 プリンタ制御ダイアログを表示してプリンタ機能を定義できるようにします。
　　（ビット2またはビット3と一緒に使用することはできません）
　　ビット1 フォントダイアログボックスを表示してドキュメントの省略時のフォントを定義できるようにします。
　　ビット2 縦置きで出力します。（ビット0またはビット3と一緒に使用することはできません）
　　ビット3 横置きで出力します。（ビット0またはビット2と一緒に使用することはできません）
　　ビット4 進行状況表示ダイアログボックスを表示します。
　　上記以外のビットは、将来のために予約済みであるため、設定できません。
ウィンドウハンドル　親ウィンドウのハンドル。親ウィンドウは、ダイアログボックスを配置するための参照用として使用されます。
　　ウィンドウハンドルが0の場合、ダイアログボックスの画面上の位置はシステム依存になります。

出口上のパラメータ:

状態コード 状態コード

PC_PRINTER_CLOSE

それまでに開かれていたプリンタチャネルを閉じます。

構文:

```
CALL "PC_PRINTER_CLOSE" USING プリンタハンドル
                    RETURNING    状態コード
```

パラメータ:

プリンタハンドル PIC X(4) COMP-5
状態コード **「用語説明」の項を参照してください。**

入口上のパラメータ:

プリンタハンドル プリンタが開かれたときに返されたプリンタハンドル

出口上のパラメータ: なし

PC_PRINTER_CONTROL

出力コマンドをプリンタに送ります。

構文:

```
CALL "PC_PRINTER_CONTROL" USING プリンタハンドル
                       BY VALUE     出力コマンド
                     RETURNING    状態コード
```

パラメータ:

プリンタハンドル PIC X(4) COMP-5
出力コマンド **数字定数または**PIC X(4) COMP-5
状態コード **「用語説明」の項を参照してください。**

入口上のパラメータ:

プリンタハンドル **プリンタが開かれたときに返されたプリンタハンドル**
出力コマンド **プリンタに送られるコマンド**
1 **ドキュメントの出力を中断し、プリンタを閉じます。**
2 **ページをとばします。**
3 **出力バッファをフラッシュします。**
4 **復帰改行します。**

出口上のパラメータ:

状態コード プリンタ状態コード

PC_PRINTER_FREE_BMP

メモリからビットマップを解放し、そのイメージとパレットをオペレーティングシステムに解放します。

構文:

```
CALL "PC_PRINTER_FREE_BMP" USING  プリンタハンドル
                    BY VALUE  BMP-ID
                    RETURNING  状態コード
```

パラメータ:

プリンタハンドル　PIC X(4) COMP-5.

BMP-***ID***　PIC X(4) COMP-5.

状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

プリンタハンドル　プリンタがオープンされたときに返されたプリンタのハンドル

BMP-***ID***　ビットマップがロードされたときに返される、ロードされたビットマップの一意の識別子

出口上のパラメータ:

状態コード　プリンタの状態コード:

0　成功

3　プリンタ装置がオープンしていません

20　ビットマップを解放できませんでした

例:

```
CALL "PC_PRINTER_FREE_BMP" USING  プリンタハンドル
                    BY VALUE  BMP-IDロゴ
                    RETURNING  プリンタretコード
```

PC_PRINTER_GET_COLOR

プリンタに対して色を設定します。

構文:

```
CALL "PC_PRINTER_GET_COLOR" USING  プリンタハンドル
                     BY VALUE  前景／背景
                     BY VALUE  赤
                     BY VALUE  緑
                     BY VALUE  青
                     RETURNING  状態コード
```

パラメータ:

プリンタハンドル　PIC X(4) COMP-5.

前景／背景　PIC X(2) COMP-5.

赤　PIC X(2) COMP-5.

緑　PIC X(2) COMP-5.

青　PIC X(2) COMP-5.

状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

プリンタハンドル　プリンタがオープンされた時に返されるプリンタのハンドル。

前景／背景　前景か背景のいずれかを設定します:

1　前景

2　背景

出口上のパラメータ:

赤　　0から255までの値; 赤の濃度
緑　　0から255までの値; 緑の濃度
青　　0から255までの値; 青の濃度
状態コード　プリンタ状態コード

0　成功

3　プリンタ装置がオープンしていません

20　ビットマップを解放できませんでした

説明:

このルーチンを使用する前に、プリンタをオープンしなければなりません。

PC_PRINTER_INFOルーチンを使用すると、プリンタがカラー印刷をサポートしているかどうか確かめることができます。

PC_PRINTER_GET_FONT

プリンタに対してフォントを設定します。

構文:

```
CALL "PC_PRINTER_GET_FONT" USING  プリンタハンドル
                                  フォント名
                                  フォントサイズ
                                  フォントスタイル
                          RETURNING  状態コード
```

パラメータ:

プリンタハンドル　PIC X(4) COMP-5.
フォント名　次のように定義された集団項目
　名前長　PIC X(2) COMP-5.
　名前テキスト　PIC X(n).
フォントサイズ　PIC X(2) COMP-5.
フォントスタイル　PIC X(2) COMP-5.
状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

プリンタハンドル　プリンタがオープンされた時に返されるプリンタのハンドル。
名前長　フォント名用のバッファの長さ。

出口上のパラメータ:

名前長　　フォント名用のバッファの長さ。ゼロが設定された場合はフォントが設定されていないため、オペレーティングシステムの省略値が使用されます（この場合フォントサイズとフォントスタイルは無効です）。

名前テキスト　　使用されるフォントの名前（Courier、Helvetica、Timesなど）

フォントサイズ　　フォントのポイント数

フォントスタイル　　要求されたフォントスタイル

bit 3　太字

bit 2　ストライクアウト

bit 1　下線

bit 0　斜体

これら以外のbitは、将来のために予約されているため使用できません。

例:

```
working-storage section.

01.
  03 document-title.
    05 title-len          pic x(2) comp-5.
    05 title-text         pic x(20).

  03 font-family.
    05 font-family-namelen  pic x(2) comp-5 value 80.
    05 font-family-name     pic x(80).

  03 font-size             pic x(4) comp-5.
  03 font-style            pic x(4) comp-5.

  03 abort                 pic x(4) comp-5 value 1.
  03 control               pic x(4) comp-5 value 2.
  03 flags                 pic x(4) comp-5 value 3.
  03 handle                pic x(4) comp-5.

procedure division.
move 17 to title-len
move "Printer Info Test" to title-text

call "PC_PRINTER_OPEN" using by reference handle
                        by reference document-title
                        by value flags
                        by value 0
end-call

if return-code = zero
    call "PC_PRINTER_GET_FONT" using by reference handle
                                by reference font-family
                                by reference font-size
                                by reference font-style
    end-call
    if return-code equal zero
        if font-size equal zero
            display "Current Font   : no font selected"
        else
            display "Current Font   : "
            font-family-name(1:font-family-namelen)
```

```
            display "Font size     : " font-size
            display "Raw Font Style : " font-style
         end-if
      else
         display "PC_PRINTER_GET_FONT failed"
      end-if
   end-if

   perform close-down-printer
   .

   close-down-printer section.
   call "PC_PRINTER_CONTROL" using by reference handle
                             by value abort
   end-call

   call "PC_PRINTER_CLOSE" using by reference handle
   end-call
   .
```

PC_PRINTER_INFO

Windowsのグラフィック関数ディレクトリを呼び出すことによって、ドキュメント印刷機能を向上させることができるWindowsデバイスハンドル(HDC)およびその他の情報を返します。

構文:

```
CALL "PC_PRINTER_INFO" USING  プリンタハンドル
                              printer-info-struct
                      RETURNING  状態コード
```

パラメータ:

プリンタハンドル　　PIC X(2) COMP-5.

printer-info-struct　　以下を含む集団項目:

```
   pi-struct-size            pic x(4) comp-5.
   printer-hdc               pic x(4) comp-5.
   printer-hps               pic x(4) comp-5.
   printer-orientation                pic x(4) comp-5.
   printer-rows              pic x(4) comp-5.
   printer-cols              pic x(4) comp-5.
   printer-rows-left                  pic x(4) comp-5.
   printer-max-horiz                  pic x(4) comp-5.
   printer-max-vert pic x(4) comp-5.
   printer-min-horiz                  picx (4) comp-5.
   printer-min-vert pic x(4) comp-5.
   printer-curr-horiz                 pic x(4) comp-5.
   printer-curr-vert                  pic x(4) comp-5.
   printer-copies   pic 9(4) comp-5.
   printer-quality  pic 9(4) comp-5.
   printer-color    pic 99 comp-5.
   reserved-item1   pic 99 comp-5.
   printer-device-ver                 pic 9(4) comp-5.
   printer-name     以下を含む集団項目：
          printer-pname-size          pic x(4) comp-5.
          printer-pname               pointer.
   printer-type     以下を含む集団項目：
          printer-ptype-size          pic x(4) comp-5.
```

```
        printer-ptype                pointer.
    printer-device    以下を含む集団項目：
        printer-pdevice-size         pic x(4) comp-5.
        printer-plocation            pointer.
    p-comment         以下を含む集団項目：
        printer-pcomment-size        pic x(4) comp-5.
        printer-pcomment             pointer.
    printer-type      以下を含む集団項目：
        printer-papersize-size       pic x(4) comp-5.
        printer-papersize            pointer.
```

状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

プリンタハンドル　プリンタがオープンされた時に返されるプリンタのハンドル。

pi-struct-size　構造体のサイズ。

printer-papersize 関連するバッファのサイズ

printer-pcomment バッファのアドレス

出口上のパラメータ:

printer-hdc　Windowsデバイスハンドル。

printer-hps　予約済み。

printer-orientation　印刷方向：

1　横

2　縦

printer-rows　現在のフォントで現在のページ上にある行数の合計

printer-cols　現在のフォントでの平均カラム数

printer-rows-left　現在のフォントで、現在のページに残されている行の数

printer-max-horiz　グラフィック座標の水平軸の最大値

printer-max-vert　グラフィック座標の垂直軸の最大値

printer-min-horiz　グラフィック座標の水平軸の最小値

printer-min-vert　グラフィック座標の垂直軸の最小値

printer-curr-horiz　グラフィック座標のX座標

printer-curr-vert　グラフィック座標のY座標

printer-copies　コピーの数

printer-quality　プリンタのクォリティが設定されます。値は、次のいずれか1つです。

0　ドラフト

1　低

2　中

3　高

4　プリンタの省略値

上記以外はDPIが返されます。

print-color　プリンタのカラー出力

0　白黒

1　カラー

printer-version	プリンタドライバのバージョン
printer-pname-size	プリンタ名のサイズ
printer-pname	プリンタ名へのポインタ
printer-ptype-size	プリンタタイプのサイズ
printer-ptype	プリンタタイプへのポインタ
printer-pdevice-size	プリンタデバイスのサイズ
printer-plocation	プリンタデバイスへのポインタ
printer-pcomment-size	プリンタのコメントサイズ
printer-pcomment	プリンタのコメントへのポインタ
printer-papersize-size	プリンタのペーパーサイズ
printer-papersize	プリンタのペーパーサイズへのポインタ
状態コード	プリンタの状態コード:

説明:

printer-typeは、HP、Canonなどのプリンタタイプを指定します。

printer-deviceは、lpt1:、:file、:cm1などのプリンタデバイスを指定します。このパラメータに複数のデバイスを指定することもできます。その場合、各デバイスをコンマで区切ります（例：lpt1:,lpt2）。

printer-commentには、プリンタに関するコメントを記述します（例：「会計用プリンタ」）。

paper-sizeは、ペーパーサイズを指定します（例：A4、Letter）。

例:

```
$set remove(control)
 working-storage section.
 01 PRT-INFO-1 is typedef.
   03 pi-struct-size     pic x(4) comp-5.
   03 hdc               pic x(4) comp-5.
   03 hps               pic x(4) comp-5.
   03 orientation        pic x(4) comp-5.
   03 rows              pic x(4) comp-5.
   03 cols              pic x(4) comp-5.
   03 rows-left          pic x(4) comp-5.
   03 max-horiz          pic x(4) comp-5.
   03 max-vert           pic x(4) comp-5.
   03 min-horiz          pic x(4) comp-5.
   03 min-vert           pic x(4) comp-5.
   03 curr-horiz         pic x(4) comp-5.
   03 curr-vert          pic x(4) comp-5.
   03 copies            pic 9(4) comp-5.
   03 quality           pic 9(4) comp-5.
   03 color             pic 99 comp-5.
   03 reserved1          pic x comp-5.
   03 driver-ver         pic 9(4) comp-5.
   03 pname.
     05 cbsize          pic x(4) comp-5.
     05 buffer          pointer.
   03 ptype.
     05 cbsize          pic x(4) comp-5.
     05 buffer          pointer.
   03 pdevice.
```

```
    05 cbsize          pic x(4) comp-5.
    05 buffer          pointer.
  03 plocation.
    05 cbsize          pic x(4) comp-5.
    05 buffer          pointer.
  03 pcomment.
    05 cbsize          pic x(4) comp-5.
    05 buffer          pointer.
  03 ppapersize.
    05 cbsize          pic x(4) comp-5.
    05 buffer          pointer.
01.
  03 document-title.
    05 title-len       pic x(2) comp-5.
    05 title-text      pic x(20).
      03  font-family.
    05 font-family-namelen  pic x(2) comp-5 value 80.
    05 font-family-name    pic x(80).
  03 print-info        PRT-INFO-1.
  03 abort             pic x(4) comp-5 value 1.
  03 control           pic x(4) comp-5 value 2.
  03 flags             pic x(4) comp-5 value 3.
  03 handle            pic x(4) comp-5.
01 printer-name        pic x(255).
01 printer-type        pic x(255).
01 printer-device      pic x(255).
01 printer-location    pic x(255).
01 printer-comment     pic x(255).
01 printer-papersize   pic x(255).

procedure division.
move 17 to title-len
move "Printer Info Test" to title-text

call "PC_PRINTER_OPEN" using by reference handle
                        by reference document-title
                        by value flags
                        by value 0
end-call

if return-code = zero
   move length of print-info to pi-struct-size
   set buffer of pname of print-info
      to address of printer-name
   move 255 to cbsize of pname of print-info
   set buffer of ptype of print-info
      to address of printer-type
   move 255 to cbsize of ptype of print-info
   set buffer of pdevice of print-info
       to address of printer-device
   move 255 to cbsize of pdevice of print-info
   set buffer of plocation of print-info
      to address of printer-location
   move 255 to cbsize of plocation of print-info
   set buffer of pcomment of print-info
      to address of printer-comment
   move 255 to cbsize of pcomment of print-info
   set buffer of ppapersize of print-info
      to address of printer-papersize
   move 255 to cbsize of ppapersize of print-info
   call "PC_PRINTER_INFO" using by reference handle
```

```
                        by reference print-info
end-call
if return-code not equal zero
    display "PC_PRINTER_INFO failed (return-code)"
    display "   === " return-code
    perform close-down-printer
   stop run
end-if
display "Orientation   : " orientation of print-info
display "Rows         : " rows of print-info
display "Cols         : " cols of print-info
display "Rows Left    : " rows-left of print-info
display "Max horz     : " max-horiz of print-info
display "Max vert     : " max-vert of print-info
display "Min horz     : " min-horiz of print-info
display "Min vert     : " min-vert of print-info
display "Current horz  : " curr-horiz of print-info
display "Current vert  : " min-vert of print-info
display "Copies       : " copies of print-info
display "Quality      : " no advancing
evaluate quality of print-info
    when 0 display "Draft"
    when 1 display "Low"
    when 2 display "Medium"
    when 3 display "High"
    when 4 display "printers default used"
    when other display quality of print-info " DPI"
end-evaluate
display "Color        : " no advancing
if color of print-info equals 0
    display "Mono Chrome"
else
    display "Color"
end-if
if cbsize of pname of print-info equal 0
    display "Printer name  : not available"
else
    display "Printer name  : "
       printer-name(1: cbsize of pname of print-info)
   display "Printer name  size : " cbsize of pname of print-info
end-if
if cbsize of ptype of print-info equal 0
    display "Printer type  : not available"
else
    display "Printer type  : "
        printer-type(1: cbsize of ptype of print-info)
    display "Printer type size : " cbsize of ptype of print-info
end-if
if cbsize of pdevice of print-info equal 0
    display "Printer device: not available"
else
    display "Printer device(s): "
       printer-device(1: cbsize of pdevice of print-info)
    display "Printer device size : " cbsize of pdevice of print-info
end-if
if cbsize of plocation of print-info equal 0
    display "Printer location: not available"
else
    display "Printer location: "
       printer-location(1: cbsize of plocation of print-info)
    display "Printer location size : " cbsize of plocation of print-info
```

```
        end-if
        if cbsize of pcomment of print-info equal 0
           display "Printer comment: not available"
        else
           display "Printer comment: "
              printer-comment(1: cbsize of pcomment of print-info)
           display "Printer comment size : " cbsize of pcomment of print-info
        end-if
        if cbsize of ppapersize of print-info equal 0
           display "Printer papersize: not available"
        else
           display "Printer papersize: "
              printer-papersize(1: cbsize of ppapersize of print-info)
           display "Printer papersize size : " cbsize of ppapersize of print-info
        end-if
        display "Driver version : "driver-ver of print-info
    end-if
    perform close-down-printer.

    close-down-printer section.
    call "PC_PRINTER_CONTROL" using by reference handle
                                    by value abort
    end-call

    call "PC_PRINTER_CLOSE" using by reference handle
    end-call.
```

PC_PRINTER_LOAD_BMP

PC_PRINTER_WRITE_BMP用のメモリにビットマップをロードします。

構文:

```
CALL "PC_PRINTER_FREE_BMP" USING  プリンタハンドル
                     BY REFERENCE  BMPファイル名
                     BY REFERENCE  BMP-ID
                     RETURNING  状態コード
```

パラメータ:

プリンタハンドル　PIC X(4) COMP-5.

BMPファイル名　PIC X(n)

BMP-***ID***　PIC X(4) COMP-5.

状態コード　「用語説明」の項を参照してください。

入口上のパラメータ:

プリンタハンドル　プリンタがオープンされたときに返されたプリンタのハンドル

BMPファイル名　ロードするビットマップファイルの名前（名前はNULLまたはスペースで終わる必要があります。）

出口上のパラメータ:

BMP-***ID***　ロードされたビットマップの一意の識別子

状態コード　プリンタの状態コード:

0　成功

3　プリンタ装置がオープンしていません

18　ビットマップをロードできませんでした

19　ビットマップ識別子の値が不正です。

説明:

本ルーチンは、Windows環境でのみ使用できます。

例:

```
call "pc_printer_load_bmp" using printer-handle
                         by reference      "mflogo.bmp" & x"0"
                         by reference      bmp-id
                         returning                  printer-retcode.
```

PC_PRINTER_OPEN

プリンタチャネルを開き、それにドキュメントタイトルを渡します。オプションで、プリンタ制御、フォント、および進行状況ダイアログボックスなどを表示します。

構文:

```
CALL "PC_PRINTER_OPEN" USING       プリンタハンドル
                                       ドキュメントタイトル
                             BY VALUE  フラグ
                             BY VALUE  ウィンドウハンドル
                       RETURNING    状態コード
```

パラメータ:

プリンタハンドル　　PIC X(4) COMP-5.
ドキュメントタイトル　次のように定義された集団項目
　　タイトル長　　PIC X(2) COMP-5
　　タイトルテキスト　PIC X(n)
フラグ　　**数字定数または**PIC X(4) COMP-5
ウィンドウハンドル数字定数または**PIC X(4) COMP-5**
状態コード　　**「用語説明」の項を参照してください。**

入口上のパラメータ:

タイトル長　ドキュメントタイトルの長さ
タイトルテキスト　　出力するドキュメントのタイトル
フラグ　　プリンタオプションを定義するビットの組
　　ビット0 **プリンタ制御ダイアログを表示してプリンタ機能を定義できるようにします。**
　　（ビット2**またはビット**3**と一緒に使用することはできません）**
　　ビット1 **フォントダイアログボックスを表示してドキュメントの省略時のフォントを定義できるようにします。**
　　ビット2 **縦置きで出力します。（ビット**0**またはビット**3**と一緒に使用することはできません）**
　　ビット3 **横置きで出力します。（ビット**0**またはビット**2**と一緒に使用することはできません）**
　　ビット4 **進行状況表示ダイアログボックスを表示します。**
　　上記以外のビットは、将来のために予約済みであるため、設定できません。

ウィンドウハンドル親ウィンドウのハンドル。親ウィンドウは、ダイアログボックスを配置するための参照用として使用されます。
ウィンドウハンドルが0の場合、ダイアログボックスの画面上の位置はシステム依存になります。

出口上のパラメータ:

プリンタハンドル 成功した場合に次のプリンタ操作で使用するハンドルを返します。

状態コード 状態コード

PC_PRINTER_SET_COLOR

プリンタに対して色を設定します。

構文:

```
CALL "PC_PRINTER_GET_COLOR" USING  プリンタハンドル
                      BY VALUE  前景／背景
                      BY VALUE  赤
                      BY VALUE  緑
                      BY VALUE  青
                      RETURNING  状態コード
```

パラメータ:

プリンタハンドル	PIC X(4) COMP-5.
前景／背景	PIC X(2) COMP-5.
赤	PIC X(2) COMP-5.
緑	PIC X(2) COMP-5.
青	PIC X(2) COMP-5.
状態コード	「用語説明」の項を参照してください。

入口上のパラメータ:

プリンタハンドル	プリンタがオープンされた時に返されるプリンタのハンドル。
前景／背景	前景か背景のいずれかを設定します: 1　前景 2　背景
赤	0から255までの値; 赤の濃度
緑	0から255までの値; 緑の濃度
青	0から255までの値; 青の濃度

出口上のパラメータ:

状態コード	本ルーチンは、状態コードを返しません。

説明:

このルーチンを使用する前に、プリンタをオープンしなければなりません。

このルーチンを使用する前に、プリンタをオープンしなければなりません。主要３原色の濃度を示す値の組合わせによって希望する色を指定することができます。例として、次の表に色とそれに対応するRGB値の設定を示します。

色	Red	Green	Blue
赤	255	0	0
黄色	255	255	0
緑	0	255	0
シアン	0	255	255
青	0	0	255
マゼンタ	255	0	255
白	255	255	255
黒	0	0	0
その他の色			
ダークオレンジ	255	140	0
ネービーブルー	0	0	128
紫	238	120	238
ベージュ	245	245	220

ユーザのプリンタがカラーをサポートしない場合、色の変換方法はプリンタデバイスドライバによって異なります。たとえば、グレースケールで印刷するものもあれば、各色を黒か白で印刷するものもあります。

PC_PRINTER_SET_DEFAULT

プリンタがオープンされる前にプリンタルーチンにデフォルトプリンタを設定します。

構文:

```
CALL "PC_PRINTER_SET_DEFAULT" USING BY VALUE デフォルトオプション
                          BY REFERENCE   default-struct
                    RETURNING  状態コード
```

パラメータ:

デフォルトオプション　PIC X(4) COMP-5.

以下を含む集団項目:

```
デフォルトオプションが1の場合　　次のように定義された集団項目：
printer-name　　以下を含む集団項目：
        printer-name-len  pic x(2) comp-5.
        printer-name      pic x(printer-name-len)
デフォルトオプションが2の場合　　次のように定義された集団項目：
        broser-hwnd       pic x(4) comp-5.
        printer-name-len  pic x(2) comp-5.
        reserved pic x(2) comp-5.
        printer-name      pic x(printer-name-len)
```

状態コード　　**「用語説明」の項を参照してください。**

入口上のパラメータ:

デフォルトオプション　**デフォルトオプションを指定します。**

1　**使用するデフォルトプリンタを設定します。**

2　**デフォルトプリンタを走査します。（本オプションは、**WindowsNT**でのみ使用できます。）**

デフォルトオプションが1**の場合：**

printer-name-len　**プリンタ名の長さ**

printer-name　　**プリンタの名前**

デフォルトオプションが2**の場合：**

***broser-hwnd*プリンタ走査用のハンドル**

出口上のパラメータ:

デフォルトオプションが2**の場合：**

printer-name-len　**プリンタ名の長さ**

printer-name　　**プリンタの名前**

reserved　**予約済み**

状態コード　**状態コード**:

説明:

デフォルトオプションに2**を設定すると、本関数は利用者が選択したプリンタを返しますが、それをデフォルトプリンタとして設定しません。**

例1:

次の例は、PC_PRINTER_SET_DEFAULT**を使用してデフォルトプリンタを設定する方法を示しています。**

```
working-storage section.
01 MyDocumentInfo.
  03 filename.
    05 len            pic x(2) comp-5.
    05 body           pic x(128).
  03 document.
    05 len            pic x(2) comp-5.
    05 body           pic x(128).
```

```
  03 document-flags    pic x(4) comp-5.
  03 window-hwnd       pic x(4) comp-5.
01 Printer-RetCode pic  9(4) comp-5.

78 USE-OPEN-DIALOG     value 1.
78 USE-FONT-DIALOG     value 2.
78 USE-FORCE-PORTRAIT   value 4.
78 USE-FORCE-LANDSCAPE  value 8.
78 USE-PROGRESS-DIALOG  value 16.

01 default-info.
  03 option           pic x(4) comp-5.
  03 ourprinter.
    05 len            pic x(2) comp-5.
    05 body           pic x(128).

78 SET-DEFAULT-PRINTER  value h"0001".
procedure division.

move "My Color PS" to body of ourprinter of default-info
move 11 to len of ourprinter of default-info
move SET-DEFAULT-PRINTER to option of default-info

call "PC_PRINTER_SET_DEFAULT" using
                          by value option of default-info,
                          by reference ourprinter of default-info
                          returning Printer-RetCode
end-call

if Printer-RetCode not equal zero
   display "Unable to setup a default printer"
   display " + Retcode = " Printer-RetCode
   stop run
end-if

move "c:¥config.sys" to body of filename
move 13 to len of filename

move "My Config.sys" to body of document
move 13 to len of document

move USE-PROGRESS-DIALOG to document-flags
move zero to window-hwnd

call "PC_PRINT_FILE" using by reference filename
                        by reference document
                        by value document-flags
                        by value window-hwnd
                        returning Printer-RetCode
end-call

if Printer-RetCode not equal zero
   display "PC_PRINT_FILE failed.. " Printer-RetCode
else
   display "PC_PRINT_FILE: OK"
end-if
```

例2:

次の例は、PC_PRINTER_SET_DEFAULT**を使用してプリンタを走査する方法を示しています。この例は、**Windows NT**上でのみ動作します。**

```
   01 MyDocumentInfo.
    03 filename.
```

```
    05 len              pic x(2) comp-5.
    05 body             pic x(128).
  03 document.
    05 len              pic x(2) comp-5.
    05 body             pic x(128).
  03 document-flags       pic x(4) comp-5.
  03 window-handle        pic x(4) comp-5.

01 MyDefaultPrinter.
  03 hwnd               pic x(4) comp-5.
  03 len                pic x(2) comp-5.
  03 reserved            pic x(2) comp-5.
  03 body               pic x(128).

01 Printer-RetCode        pic 9(4) comp-5.

78 USE-OPEN-DIALOG        value 1.
78 USE-FONT-DIALOG        value 2.
78 USE-FORCE-PORTRAIT     value 4.
78 USE-FORCE-LANDSCAPE    value 8.
78 USE-PROGRESS-DIALOG    value 16.

01 default-info.
  03 option             pic x(4) comp-5.
  03 ourprinter.
    05 len              pic x(2) comp-5.
    05 body             pic x(128).

78 SET-DEFAULT-PRINTER    value h"0001".
78 BROWSE-DEFAULT-PRINTER  value h"0002".
linkage section.
01 lnk-hwnd               pic x(4) comp-5.

procedure division using by value lnk-hwnd.
move BROWSE-DEFAULT-PRINTER to option of default-info
move lnk-hwnd to hwnd of MyDefaultPrinter
move 127 to len of MyDefaultPrinter

call "PC_PRINTER_SET_DEFAULT" using
                          by value option of default-info
                          by reference hwnd
                          returning Printer-RetCode
end-call

if Printer-RetCode not equal zero
   display "Unable to browse for a default printer"
   display " + Retcode = " Printer-RetCode
   stop run
end-if

display "Browse Printer : "
   body of MyDefaultPrinter(1:len of MyDefaultPrinter)

move SET-DEFAULT-PRINTER to option of default-info
move len of MyDefaultPrinter to len of ourprinter
move body of MyDefaultPrinter(1:len of MyDefaultPrinter)
   to body of ourprinter

call "PC_PRINTER_SET_DEFAULT" using
                          by value option of default-info
                          by reference ourprinter of default-info
```

```
                              returning Printer-RetCode
   end-call

   if Printer-RetCode not equal zero
      display "Unable to set a default printer"
      display " + Retcode = " Printer-RetCode
      stop run
   end-if

   move "c:¥config.sys" to body of filename
   move 13 to len of filename

   move "My Config.sys" to body of document
   move 13 to len of document

   move USE-PROGRESS-DIALOG to document-flags
   move zero to window-handle

   call "PC_PRINT_FILE" using by reference filename
                         by reference document
                         by value document-flags
                         by value window-handle
                         returning Printer-RetCode
   end-call

   if Printer-RetCode not equal zero
      display "PC_PRINT_FILE failed.. " Printer-RetCode
   else
      display "PC_PRINT_FILE: OK"
   end-if
```

PC_PRINTER_SET_FONT

プリンタに対して現在のフォントを設定します。

構文:

```
  CALL "PC_PRINTER_SET_FONT" USING  プリンタハンドル
                                    フォント名
                        BY VALUE      フォントサイズ
                        BY VALUE      フォントスタイル
                        RETURNING  状態コード
```

パラメータ:

プリンタハンドル PIC X(4) COMP-5.

フォント名 **次のように定義された集団項目**

名前長 PIC X(2) COMP-5.

名前テキスト PIC X(n).

フォントサイズ **数字定数または**PIC X(4) COMP-5.

フォントスタイル **数字定数または**PIC X(4) COMP-5.

状態コード **「用語説明」の項を参照してください。**

入口上のパラメータ:

プリンタハンドル **プリンタがオープンされた時に返されるプリンタのハンドル。**

名前長 **フォント名の長さ**

名前テキスト **使用されるフォントの名前（**Courier、Helvetica、Times、Symbol、Times New Roman、Roman**など）**

フォントサイズ **フォントのポイント数**

フォントスタイル **要求されたフォントスタイル**

bit 3 **太字**

bit 2 **ストライクアウト**

bit 1 **下線**

bit 0 **斜体**

これら以外のbit**は、将来のために予約されているため使用できません。**

出口上のパラメータ:

状態コード **状態コード**

PC_PRINTER_WRITE

プリンタチャネルを開き、それにドキュメントタイトルを渡します。オプションで、プリンタ制御、フォント、および進行状況ダイアログボックスなどを表示します。

構文:

```
CALL "PC_PRINTER_WRITE" USING       プリンタハンドル
                                    出力バッファ
                              BY VALUE  出力バッファ長
                    RETURNING    状態コード
```

パラメータ:

プリンタハンドル PIC X(4) COMP-5.
出力バッファ PIC X(n).
出力バッファ長 数字定数または PIC X(4) COMP-5
状態コード 「用語説明」の項を参照してください。

入口上のパラメータ:

プリンタハンドル プリンタがオープンされたときに渡されたプリンタバッファ
出力バッファ バイトを出力するバッファ
出力バッファ長 出力するバイト数

出口上のパラメータ:

状態コード 状態コード

PC_PRINTER_WRITE_BMP

ロードされたビットマップを指定サイズで指定位置にプリンタに書き込みます。

構文:

```
CALL "PC_PRINTER_WRITE_BMP" USING  プリンタハンドル
                     BY REFERENCE  BMP-ID
                        BY VALUE  reserved
                     BY VALUE  BMProw
                     BY VALUE  BMPcol
                     BY VALUE  BMP幅
                     BY VALUE  BMP高さ
                     RETURNING  状態コード
```

パラメータ:

プリンタハンドル PIC X(4) COMP-5.
reserved PIC X(4) COMP-5. **値** 0
BMP-ID PIC X(4) COMP-5.
BMProw PIC X(4) COMP-5.
BMPcol PIC X(4) COMP-5.
BMP**幅** PIC X(4) COMP-5.
BMP**高さ** PIC X(4) COMP-5.
状態コード 「用語説明」の項を参照してください。

入口上のパラメータ:

プリンタハンドル プリンタがオープンされた時に返されるプリンタのハンドル。
BMP-ID 出力するビットマップの一意の識別子。
BMProw、BMPcol、BMP**幅**、BMP**高さ**
印刷時のビットマップの位置と大きさ。
状態コード プリンタの状態コード:
0 成功
3 プリンタ装置がオープンしていません
4 印刷中にメモリ不足になりました
5 ファイルのスプール中にディスクが一杯になりました
7 印刷ジョブが破棄されました。プリントマネージャにスプールされているファイルはありません
11 書き込みエラーです
21 ビットマップを印刷できませんでした

出口上：

状態コード プリンタの状態コード:

説明:

このルーチンは Windows 環境でのみ使用できます。

制限事項:

このルーチンは PostScript および HP PCL プリンタで動作します。
このルーチンは HP Deskjet またはドットマトリックスプリンタをサポートしていません。

例:

```
  CALL "PC_PRINTER_WRITE_BMP" USING printer-handle
     BY VALUE bmp-id-logo
     BY VALUE  0 SIZE 4
     BY VALUE  7 SIZE 4
     BY VALUE  5 SIZE 4
     BY VALUE 25 SIZE 4
     BY VALUE 15 SIZE 4
     RETURNING printer-retcode
  END-CALL
```

PC_READ_DRIVE

現省略値ドライブ文字を戻します。

構文:

```
  CALL "PC_READ_DRIVE" USING  ドライブ
                    RETURNING  状態コード
```

パラメータ:

ドライブ　　PIC X.
状態コード　**「用語説明」の項を参照してください。**

入口上のパラメータ:　**なし**

出口上のパラメータ:

ドライブ　**ドライブ文字で、大文字または小文字**

PC_SET_DRIVE

省略値ドライブ文字を設定します。

構文:

```
  CALL "PC_SET_DRIVE" USING  ドライブ
                   RETURNING  状態コード
```

パラメータ:

ドライブ　　PIC X.
状態コード　**「用語説明」の項を参照してください。**

入口上のパラメータ:

ドライブ　**ドライブ文字で、大文字または小文字**

出口上のパラメータ:　**なし**

PC_WIN_CHAR_TO_OEM

ANSI**文字のバッファを**OEM**文字セットに変換します。**

構文:

```
  CALL "PC_WIN_CHAR_TO_OEM" USING  文字列バッファ
                                   ターゲットバッファ
                                    BY VALUE  文字列長
                                   ターゲット長
                                 RETURNING  状態コード
```

パラメータ:

文字列バッファ　PIC X(n).
ターゲットバッファ　PIC X(n).
文字列長　PIC X(4) COMP-5.
ターゲット長　PIC X(4) COMP-5.
状態コード　**「用語説明」の項を参照してください。**

入口上のパラメータ:

文字列バッファ　**変換する**ANSI**文字のバッファ**
文字列長　**文字列バッファの長さ。本パラメータがゼロの場合、文字列バッファは**NULL**で終わる文字列になりこれが変換されます。**
ターゲット長　**ターゲットバッファの長さ**

出口上のパラメータ：

ターゲットバッファ　**入力された**ANSI**文字から変換される**OEM**文字のバッファ。本パラメータは、少なくとも文字列バッファの文字数以上の長さを持つ必要があります。文字列バッファとターゲットバッファが同じバッファである場合、文字列は正確にバッファ内で変換されます。**

説明:

状態コードがゼロ以外の場合、呼出しはターゲットバッファのサイズが足りなかったために失敗しました。

PC_WIN_INIT

Windows**の起動値を返します。**

構文:

```
  CALL "PC_WIN_INIT" USING  hInst
```

```
                        hPrevInstance
                                hPrevInstance
                        lpszCmdLine
                                nCmdShow
                      RETURNING  状態コード
```

パラメータ:

```
hInst pic x(4) comp-5.
hPrevInstance pic x(4) comp-5.
lpszCmdLine pointer.
nCmdShow pic x(4) comp-5.
```

状態コード　　「用語説明」の項を参照してください。

入口上のパラメータ: なし

出口上のパラメータ:

`hInst` **このインスタンスのハンドル**
`hPrevInstance` **前回のインスタンスのハンドル**
`lpszCmdLine` **コマンド行へのポインタ。**`COBOL`**用ではない。**
`nCmdShow` **作成されたウィンドウの属性を示すフラグ**

説明:

`Windows`**アプリケーションを起動すると、**`Windows`**は**`4`**つの起動値を返します。これれは、多くの**`Windows API`**ルーチンを呼び出すときに必要になります。共有ランタイムシステムを使用している場合は、**`Windows API`**を使用できるように本ルーチンを呼び出して起動値を取得する必要があります。**
本ルーチンにより渡されたパラメータをコマンド行で使用しないでください。代りに、`ACCEPT...FROM COMMAND-LINE`**を使用してください。**

`PC_WIN_CHAR_TO_OEM`

`OEM`**文字のバッファを**`ANSI`**文字セットに変換します。**

構文:

```
CALL "PC_WIN_OEM_TO_CHAR" USING  文字列バッファ
                                 ターゲットバッファ
                                  BY VALUE  文字列長
                                 ターゲット長
                               RETURNING  状態コード
```

パラメータ:

文字列バッファ　　`PIC X(n).`
ターゲットバッファ　　`PIC X(n).`
文字列長　　`PIC X(4) COMP-5.`
ターゲット長　　`PIC X(4) COMP-5.`
状態コード　　「用語説明」の項を参照してください。

入口上のパラメータ:

文字列バッファ　　変換する`ANSI`**文字のバッファ**
文字列長　　文字列バッファの長さ。本パラメータがゼロの場合、文字列バッファは`NULL`**で終わる文字列になりこれが変換されます。**
ターゲット長　　ターゲットバッファの長さ

出口上のパラメータ:

ターゲットバッファ　　入力された`OEM`**文字から変換される**`ANSI`**文字のバッファ。本パラメータは、少なくとも文字列バッファの文字数以上の長さを持つ必要があります。文字列バッファとターゲットバッファが同じバッファである場合、文字列は正確にバッファ内で変換されます。**

説明:

状態コードがゼロ以外の場合、呼出しはターゲットバッファのサイズが足りなかったために失敗しました。

スレッドロックルーチンの使用法

次のコードをプログラムに組み込むと、最初のルーチンのみデータを初期化し、その後に続くスレッドは事前に初期化された使用可能なデータを所有するということが確かめられます。

```
if  first-time = 0
     call "cbl_thread_prog_lock"
    if  first-time = 0
        initialize my-data-division
       move 1 to first-time
    end-if
    call "cbl_thread_prog_unlock"
end-if
```

first-time変数がダブルチェックされていることに注目してください。最初のチェックは、最適化のためです。つまり、データがすでに初期化されていることが確かな場合にプログラムロックのオーバーヘッドを避けるためです。

オペレーティングシステムをチェックするためのCBL_THERAD_SELFの使用法

オペレーティングシステムがマルチスレッドをサポートしているかどうかをチェックするために、次のようにCBL_THERAD_SELFを使用することができます。

```
call 'cbl_thread_self' using thread-id
     on exception
        * no cbl_thread api support
     end-call
     if  return-code == 1008
        * running in a single threaded only rts
     end-if
```

スレッド固有データ処理ルーチンの使用法

このプログラムは、スレッド格納ハンドルを初期化し、その後それぞれがハンドルを使用してスレッドローカルデータにアクセスするいくつかのスレッドを起動します。各スレッドの終了および実行単位の終了時にts-testへの各入口内で一貫性のチェックが行われます。このプログラムは、THREAD-LOCAL-STORAGE節、外部データ項目、閉止手続き、およびこの関数を実行するための基本的な同期化を利用しています。

ソースコード:

```
$set + reentrant
*****************************************************************************
*                                                                           *
*  tstore-main.                                                             *
*  Main routine to initialize tables and kick off threads.                  *
*                                                                           *
*****************************************************************************
 program-id. 'tstore'.
 environment division.
 special-names.     command-line is cmdln.
 working-storage section.

  78 THREAD-COUNT VALUE 10.

  01  tstore-handle      usage pointer external.
  01  procptr           usage procedure-pointer.
  01  c-0               pic x comp-x value 0.
  01  foo-item          pic 9(9) value 0.
  01  thredid           pic xxxx comp-5.
  01  thread-handle      usage pointer.

 thread-local-storage section.
  01  filler.
      05  tl-count       pic x value 'x'.
      05  tl-ptr         usage pointer.

 linkage section.
  01  tstore-item.
      05  filler              pic x.
          88  TSTORE-INIT       VALUE 'Y'.
      05  tstore-count    pic 999.

 procedure division.
    *
    * Initialize thread table and set up for clean exit
    *
    call "CBL_TSTORE_CREATE"   using tstore-handle
                        by value length tstore-item
                        by value h'04'
    *
    * Set up for clean exit
    *
```

```
    set procptr to entry 'exitproc'
    move 0 to c-0
    call 'CBL_EXIT_PROC'      using c-0 procptr
    call 'ts-get'            using tl-ptr
    set address of tstore-item    to tl-ptr
    move THREAD-COUNT           to tstore-count

    set procptr                 to entry "ts-entry".

    move 1                     to thredid
    perform THREAD-COUNT times
       call "CBL_THREAD_CREATE_P" using
                           by value    procptr
                           by reference thredid
                           by value    length of thredid
                           by value    0
                           by value    0
                           by value    0
                           by reference thread-handle
       if return-code not = 0
          call 'CBL_THREAD_PROG_LOCK'
          display "FAIL: Cannot create thread"
          call 'CBL_THREAD_PROG_UNLOCK'
          stop run
       end-if
       add 1 to thredid
    end-perform
    display 'just exiting'
    stop run.

 entry "exitproc"
    call "CBL_TSTORE_GET"     using
                        by value tstore-handle
                    by reference tl-ptr
    set address of tstore-item    to tl-ptr

    if  not TSTORE-INIT
    or  tstore-count not = THREAD-COUNT
       display "FAIL: TSTORE not initialized properly!"
    else
       display "PASS: Main thread has count " tstore-count
    end-if

    call "CBL_TSTORE_CLOSE"    using
                        by value tstore-handle
    exit program.


 end program 'tstore'.

*****************************************************************************
*                                                                           *
*  ts-entry.                                                                *
*  Root entry point for threads created by application.                     *
*                                                                           *
*****************************************************************************

   program-id. 'ts-entry'.

   working-storage section.
   78 REP-COUNT         VALUE 5.
```

```
   01  tl-ptr          usage pointer.

   linkage section.
   01  lnk-thredid      pic xxxx comp-5.

   01  tstore-item
     . 05  filler               pic x.
         88  TSTORE-INIT        VALUE 'Y'.
       05  tstore-count          pic 999.

   procedure division using lnk-thredid.
   thread-section.

       perform REP-COUNT times
         call 'ts-test'         using lnk-thredid
       end-perform

       call 'ts-get'            using tl-ptr
       set address of tstore-item    to tl-ptr

       call "CBL_THREAD_PROG_LOCK"
       if  tstore-count not = REP-COUNT
          display "FAIL: Thread storage rep-count BAD"
       else
          display "PASS: Thread storage rep-count good"
       end-if
       call "CBL_THREAD_PROG_UNLOCK"
       exit program.
   end program 'ts-entry'.

*****************************************************************************
*                                                                           *
*  ts-test.                                                                 *
*  Routine to get a thread storage area and increment its count             *
*                                                                           *
*****************************************************************************

   program-id. 'ts-test'.

   working-storage section.
   01  global-count     pic 99999 value 0.

   thread-local-storage section.
   01  tl-ptr          usage pointer.
   01  tl-count        pic 999 value 0.

   linkage section.
   01  lnk-thredid      pic xxxx comp-5.

   01  tstore-item.
       05  filler               pic x.
          88  TSTORE-INIT        VALUE 'Y'.
       05  tstore-count          pic 999.

   procedure division using lnk-thredid.
   thread-section.
       call 'ts-get'            using tl-ptr
       set address of tstore-item    to tl-ptr
       add 1                      to tstore-count
       add 1                      to tl-count
```

```
      if tstore-count not = tl-count
         display "ERROR: inconsistent thread local data"
         stop run
      end-if

      call "CBL_THREAD_PROG_LOCK"
      add 1                     to global-count
      display "MESSAGE: thread-test has been called "
             tstore-count
             " by thread "
             lnk-thredid
      display "MESSAGE: thread-test has been called "
             global-count
             " globally "
      call "CBL_THREAD_PROG_UNLOCK"

      exit program.

   end program 'ts-test'.

*****************************************************************************
*                                                                           *
*  ts-get.                                                                  *
*  Common routine to get and initialize the thread storage                  *
*   area allocated by CBL_TSTORE_GET.                                       *
*                                                                           *
*****************************************************************************

   program-id. 'ts-get'.
   data division.
   working-storage section.
   01  tstore-handle   usage pointer external.

   thread-local-storage section.
   01  tl-ptr                   usage pointer.

   linkage section.
   01  tstore-item.
       05  filler               pic x.
           88  TSTORE-INIT       VALUE 'Y'.
       05  tstore-count          pic 999.

   01  lnk-ptr                  usage pointer.

   procedure division using lnk-ptr.
       call "CBL_TSTORE_GET"     using
                   by value tstore-handle
                by reference tl-ptr
       if  tl-ptr = NULL
           display "FAIL: Error in getting thread storage data"
           stop run
       end-if
       set address of tstore-item    to tl-ptr
       if  not TSTORE-INIT
           move 0                 to tstore-count
       end-if
       set tstore-init             to true
       set lnk-ptr                to tl-ptr
       exit program.
   end program 'ts-get'.
```

第9章　ライブラリルーチン（数字による呼出し）

本製品と一緒に提供されるランタイムシステムは、COBOL CALL文を用いて使用可能なルーチンのライブラリを持っています。これらのルーチンは一般にCOBOL構文を用いてアクセスできない機能を提供します。16進数を呼び出すことによりアクセスされるこれらのルーチンは、数字による呼出しルーチンと呼ばれています。

9.1　概説

本COBOLシステムの初期のバージョンでは、全てのシステムライブラリルーチンは、ただ1個の16進数を呼び出すことでアクセスされていました。これらのルーチンは数字による呼出しルーチンとして知られており、名前を呼び出してアクセスされるルーチン（名前による呼出しルーチン）によって置き換えられています。ディスクマニュアルに含まれているリストは、各名前による呼出しルーチンごとに、置き換えた数字による呼出しルーチンがあれば、その数字を示しています。名前による呼出しルーチンが追加機能を持っている場合もあります。移植性の観点から、数字による呼出しルーチンよりは名前による呼出しルーチンを使用されることをお勧めします。本COBOLシステムで利用できる名前による呼出しルーチンの詳細については、「第8章 ライブラリルーチン（名前による呼出し）」を参照してください。

数字による呼出しルーチンに対するパラメータは、プログラムの連絡節または局所記憶節で定義してはいけません。また、データ部の最初の64Kバイト内にある必要があります。

9.1.1 使用可能なルーチン

次の機能を実行するための数字による呼出しルーチンがあります。

x "84"	DOS割込みの実行
x "85"	1バイト位置の検査
x "86"	1バイト位置の設定
x "87"	ハードウェアポートからの読込み
x "88"	ハードウェアポートに1バイト出力
x "91"関数11	COBOLプログラムスイッチの設定
x "91"関数12	COBOLプログラムスイッチの読込み
x "91"関数13	ランタイムスイッチの設定
x "91"関数14	ランタイムスイッチの読込み
x "91"関数15	プログラムの存在をチェック
x "91"関数16	連結パラメータ数の獲得
x "91"関数35	プログラムの呼出し（DOS"4B"呼出しと同様）
x "91"関数46	空文字列の挿入を可能とする
x "91"関数47	空文字列の挿入を禁止とする

x "91"関数48	タブ挿入を可能とする
x "91"関数49	タブ挿入を禁止とする
x "91"関数69	スキャンディレクトリ
x "94"	2バイト位置の検査
x "95"	2バイト位置への書出し
x "96"	ハードウェアポートから2バイト入力
x "97"	ハードウェアポートへ2バイト出力
x "A7"関数6	現行ユーザー属性の読込み
x "A7"関数7	現行ユーザー属性の設定
x "A7"関数16	ユーザー属性のオン/オフ
x "A7"関数18	操作卓I/Oをリダイレクト可能とする
x "A7"関数20/21	システム属性の読込み・設定
x "A8"	複数リールファイル見出しレコードの設定
x "AF"	ADIS構成を変更する
x "B0"関数0	ファンクションキーテーブルの設定
x "B0"関数2	シフト型キー状態の試験
x "B0"関数4	「中断」のキー順を使用禁止とする
x "E5"	警告音を鳴らす
x "F4"	データをバイトにパックする
x "F5"	データをバイトにアンパックする

次の付加的数字による呼出しルーチンはToolset構成要素のランタイム環境でのみ利用可能です:

x "91"関数60/61	メモリ内データの固定

9.2 操作

9.2.1 ルーチンの説明

数字による呼出しルーチン全てについての説明はそれらの16進数の順序でなされています。各ルーチンはそれを呼出すのに使用できるCALL文に示されています。必要とするパラメータはUSING句に示されています。

9.2.1.1 DOS割込みの実行

CALL x "84" USING **割込み番号,フラグ,*AX*パラメータ,*BX*パラメータ,*CX*パラメータ,*DX*パラメータ**

ここで、

割込み番号 DOS割込み番号の値を格納しているPIC X COMP-X項目

フラグ レジスタAX、BX、CX、およびDXに何を書き込むかにより0または1に設定される8ビットのPIC X COMP-X項目

***AX*、*BX*、*CX*、および*DX*パラメータ**	集団項目で、それぞれ、16ビットレジスタの上位および下位に対応する2個のPIC X COMP-Xデータ項目で構成されています。これらの項目には、対応するフラグビットが0の場合レジスタによりアドレス指定される値が、フラグビットが1の場合レジスタに書き込まれる値が入ります。

AX、BX、CXおよびDXレジスタを介してパラメータを受け取る割込み用にのみ、本ルーチンは使用できます。本制約外のサポートを必要とする任意の割込み（DSおよびESの制御、または、割込み後のスタックに対する調整など）は、アセンブラ副プログラミングを用いてアクセスされる必要があります。

本ルーチンはOS/2またはWindows上では使用できません。

9.2.1.2 フラグバイト

フラグバイトの最下位4ビットは、入口上のAX、BX、CX、およびDXの内容を制御します。最下位ビットが設定されていると、AXにはAXパラメータの16ビット値がロードされます。最下位ビットが設定されてないと、DS:AXはAXパラメータのアドレス指定をします。フラグバイトの最上位4ビットは、出口上のAX、BX、CXおよびDXの内容を制御します。最上位ビットが設定されていると、割込みによりDXに入れて戻される値はDXパラメータに入ります。

割込み呼出しへの入口では、DS=ES=プログラムアドレススペースとなります。

CALL x "84"は、AX、BX、CXおよびDXレジスタを介してパラメータを受け取る割込みについてのみ使用できます。本制約外のサポートを必要とする任意の割込み（DSおよびESの制御、または、割込み後のスタックに対する調整など）は、アセンブラサブプログラミング中にアクセスされる必要があります。

例

次の例はファイルを開くためのDOS FCBインタフェースを使用します。

```
* dos interrupt number=21h
 01 msdos       pic x comp-x value h"21".

* the contents of the parameter funct are to be placed in and
* taken from AX before and after the interrupt.
 01 flag        pic x comp-x value h"11".

 01 funct.
     02 funct-ah    pic x comp-x.
     02 funct-al    pic x comp-x.
 01 fcb.
     02 filler    pic x value x"00".
     02 filler    pic x (8)  value "myfile".
     02 filler    pic x (3)  value "dat".
     02 filler    pic x (25) .

 01 null-param    pic x.
    ...
```

```
    move 15 to funct-ah
* ah = dos function, open fcb
    coll x"84" using msdos, flag, funct,
                     null-param, null-param, fcb.

* al = result
     if funct-al is not = 0
        go to file-not-found
     end-if.
```

9.2.1.3 1バイトまたは2バイト位置からの読込み

```
     { x "85" }
CALL {        } USING 区分,相対番地,データ値
     { x "94" }
```

ここで、

区分　検査する区分が入っているPIC 9（5）項目

相対番地　区分内の相対番地が入っているPIC 9（5）項目

データ値　x "85"の場合はPIC X項目、x "94"の場合はPIC XX項目で、戻り値が格納されます。

CALL x "85"は1バイト値を戻し、CALL x "94"は2バイト値を戻します。指定する場所の内容がプログラムのデータ値項目にコピーされます。

これらのルーチンはDOS上で使用するように設計されています。OS/2またはWindows上では使用しないでください。

共有ランタイムシステムの使用時、またはランタイム環境で、これらのルーチンをCOBOLデータ領域内（COBOLシステムおよびCOBOLアプリケーションを含む）でアドレス処理に使用してはいけません。これらのランタイムシステムが行うメモリ管理により問題となるデータが再配置されることがあるからです。

9.2.1.4 1バイトまたは2バイト位置への書出し

```
     { x "86" }
CALL {        } USING 区分,相対番地,データ値
     { x "95" }
```

ここで、

区分　書出しをする区分が入っているPIC 9（5）項目

相対番地　区分内の相対番地が入っているPIC 9（5）項目

データ値　CALL x "86"の場合はPIC X項目、CALL x "95"の場合はPIC XX項目で、書き出すデータが格納されます。

CALL x "86"は1バイト位置へ書出し、CALL x "95"は2バイト位置へ書き出します。データ値内のデータは指定された位置へコピーされます。

これらのルーチンはDOS上で使用するように設計されています。OS/2またはWindows上では使用しないでください。

共有ランタイムシステムの使用時、またはランタイム環境で、これらのルーチンをCOBOLデータ領域内（COBOLシステムおよびCOBOLアプリケーションを含む）でアドレス処理に使用してはいけません。これらのランタイムシステムが行うメモリ管理により問題となるデータが再配置されることがあるからです。

9.2.1.5 ハードウェアポートからの1バイト値または2バイト値の読込み

```
        { x "87" }
CALL    {        } USING ポート,データ値
        { x "96" }
```

ここで、

ポート　そこから入力するポートが入っているPIC 9（5）項目

データ値　CALL x "87"の場合はPIC X項目、CALL x "96"の場合はPIC XX項目で、戻される値が格納されます。

CALL x "87"は1バイト値を戻し、CALL x "96"は2バイト値を戻します。

9.2.1.6 ハードウェアポートへの1バイト値または2バイト値の書出し

```
        { x "88" }
CALL    {        } USING ポート,データ値
        { x "97" }
```

ここで、

ポート　そこへ書き出すハードウェアポートが入っているPIC 9（5）項目

データ値　CALL x "88"の場合はPIC X項目、CALL x "97" の場合はPIC XX項目で、戻される値が格納されます。

CALL x "88"は1バイト値を書出し、CALL x "97"は2バイト値を書出します。ポートは利用者のプログラムから出力されます。

9.2.1.7 プログラム間関数呼出し

```
CALL x "91" USING 結果,関数コード,パラメータ
```

ここで、

結果　PIC X COMP-X項目で、呼出しが成功するとゼロを格納し、成功しなかった場合はゼロ以外の値を格納します。成功しなかった場合戻される番号は関数コードにより変わります。

関数コード　以下に列挙する値の1つを格納するPIC X COMP-X項目で、どの関数を必要とするかによりその値が変わります。

パラメータの形式は関数に依存します。

関数コードに格納できる値は次の通りです。

11　COBOLプログラムスイッチの設定
12　COBOLプログラムスイッチの読込み
13　ランタイムスイッチの設定
14　ランタイムスイッチの読込み
15　プログラムの存在をチェック
16　連結パラメータ数の獲得
35　プログラムの呼出し（DOS "4B"呼出しと同様）
46　空文字列の挿入を可能とする
47　空文字列の挿入を不能とする
48　タブ挿入を可能とする
49　タブ挿入を不能とする
69　ディレクトリを探索して与えられたファイル指定を捜す

X"91" 関数11

COBOLプログラムスイッチを設定します。

構文:

```
CALL X"91"  USING 結果
                  関数コード
                  パラメータ
```

パラメータ:

結果	PIC X COMP-X
関数コード	PIC X COMP-X
パラメータ	次のように定義された集団項目
	スイッチフラグ　PIC X COMP-X OCCURS 8
	デバッグフラグ　PIC X COMP-X

入口上のパラメータ:

関数コード	値11
スイッチフラグ	スイッチ0～7に設定する値（0または1）
デバッグフラグ	1に設定するとANSIデバッグモジュールが使用可能になります。

出口上のパラメータ:

結果　　呼出しに成功した場合はゼロが設定され、そうでなければゼロ以外が設定されます。

説明:

他のスイッチに影響を及ぼさずに1個のスイッチを更新するには、関数12でスイッチを読込み、パラメータを更新し、それから本機能を呼び出す必要があります。

X"91" 関数12

COBOLプログラムスイッチを読み込みます。

構文:

```
CALL X"91" USING 結果
                 関数コード
                 パラメータ
```

パラメータ:

結果　　PIC X COMP-X

関数コード　　PIC X COMP-X

パラメータ　　次のように定義された集団項目

スイッチフラグ　　PIC X COMP-X OCCURS 8

デバッグフラグ　　PIC X COMP-X

入口上のパラメータ:

関数コード　　値12

出口上のパラメータ:

結果　　呼出しに成功した場合はゼロが設定され、そうでなければゼロ以外が設定されます。

スイッチフラグ　　スイッチ0～7の値

デバッグフラグ　　1に設定するとANSIデバッグモジュールが使用可能になります。

X"91" 関数13

ランタイムスイッチを設定します。

構文:

```
CALL X"91" USING 結果
                 関数コード
                 パラメータ
```

パラメータ:

結果　　PIC X COMP-X

関数コード　　PIC X COMP-X

パラメータ　　PIC X COMP-X OCCURS 26

入口上のパラメータ:

関数コード　　値13

パラメータ　　各バイトはスイッチの文字の１つを表します。各バイトの個々のビットはスイッチ名に続く数字に対応します。例えば、パラメータ(1)のビット0はスイッチA（またはA0）を表し、パラメータ(19)のビット5はスイッチS5を表します。

出口上のパラメータ:

結果　　呼出しに成功した場合はゼロが設定され、そうでなければゼロ以外が設定されます。

説明:

他のスイッチに影響を及ぼさずに1個のスイッチを更新するには、関数14でスイッチを読込み、パラメータを更新し、それから本機能を呼び出す必要があります。

X"91" 関数14

ランタイムスイッチを読み込みます。

構文:

```
CALL X"91"  USING 結果
                  関数コード
                  パラメータ
```

パラメータ:

結果　　PIC X COMP-X

関数コード　　PIC X COMP-X

パラメータ　　PIC X COMP-X OCCURS 26

入口上のパラメータ:

関数コード　　値14

出口上のパラメータ:

パラメータ　　各バイトはスイッチの文字の１つを表します。各バイトの個々のビットはスイッチ名に続く数字に対応します。例えば、パラメータ(1)のビット0はスイッチA（またはA0）を表し、パラメータ(19)のビット5はスイッチS5を表します。

結果　　呼出しに成功した場合はゼロが設定され、そうでなければゼロ以外が設定されます。

X"91" 関数15

指定されたプログラムが存在するかどうかチェックします。

構文:

```
CALL X"91"  USING 結果
                  関数コード
                  パラメータ
```

パラメータ:

結果　　PIC X COMP-X

関数コード　　PIC X COMP-X

パラメータ　　次のように定義された集団項目

名前長　　PIC X COMP-X
プログラム名　　PIC X(n)

入口上のパラメータ:

関数コード　　値15
名前長　　ファイル名の長さ
プログラム名　　探すプログラム名。本パラメータは少なくとも65文字の長さが必要です。

出口上のパラメータ:

プログラム名　　プログラムが見つかった場合は、そのフルネームが入ります。
結果　　プログラムが見つかった場合はゼロ、そうでなければゼロ以外の値が入ります。

説明:

プログラムファイルが見つかった場合、戻されたファイル名を用いて、そのプログラムを呼び出す必要があります。

データファイルの存在を調べるのにこの呼出しを使用してはいけません。CBL_CHECK_FILE_EXISTを使用してください。

X"91" 関数16

呼ぶプログラムのCALL文内から呼ばれたプログラムに転送されたパラメータの個数を示します。

構文:

```
CALL X"91"  USING 結果
                  関数コード
                  パラメータ
```

パラメータ:

結果　　PIC X COMP-X
関数コード　　PIC X COMP-X
パラメータ　　PIC X COMP-X

入口上のパラメータ:

関数コード　　値16

出口上のパラメータ:

パラメータ　　パラメータの数
結果　　呼出しに成功した場合はゼロが設定され、そうでなければゼロ以外が設定されます。

説明:

本関数は呼ばれるプログラムで使用されます。生成されたコード内で働く関数の場合、プログラムはPARAMCOUNTCHECK指令を設定してコンパイルする必要があります。

本関数が使用されるのは、プログラムがCOBOLプログラムから呼ばれる場合のみです。

ある状況では、呼出しによって返される数が、呼出しプログラムのCALL文によって転送されたパラメータの数よりも大きい場合があります。その例を以下に示します。

- 呼び出されたサブプログラムに連絡節項目がある場合、呼出しプログラムから転送された項目以外の項目は、NULL以外の値が設定されたアドレスを持ちます。そのような項目は、返される数を１づつ加算します。
- 呼び出されたプログラムに外部データ項目がある場合、返される値は各外部データレコード（01レベル）ごとに1づつ加算されます
- SYSIN指令によりプログラムをコンパイルした場合、返される数は2づつ加算されます。また、GNTANLZ指令でコンパイルした場合は1づつ加算されます。

X"91" 関数35

指定されたプログラムファイルに対してEXEC呼出し（DOS 4B呼出しと同様）を実行します。

構文:

```
CALL X"91"  USING 結果
                  関数コード
                  パラメータ
```

パラメータ:

結果　PIC X COMP-X

関数コード　PIC X COMP-X

パラメータ　次のように定義された集団項目

名前長	PIC X COMP-X
プログラム名	PIC X(n)

入口上のパラメータ:

関数コード　値35

名前長　プログラム名の文字数。ゼロを設定すると、コマンド行で以前に指定したものが実行されます。

プログラム名　実行するプログラムのファイル名。

出口上のパラメータ:

結果　EXEC呼出しが成功した場合はゼロ、そうでなければゼロ以外の値が入ります。失敗の原因がオペレーティングシステムエラーで255よりも小さい番号である場合、その番号が返されます。それ以外は255が返されます。

説明:

名前付きのプログラムではなくコマンド行から実行するには、「名前長」にゼロを設定し、DISPLAY...UPON COMMAND-LINE構文を使用してコマンド行を指定してください。

X"91" 関数46

行順ファイルにおいて、その値がx "20"未満のデータ文字の前に空文字x "00"を挿入できるようにします。

構文:

```
CALL X"91"  USING 結果
                  関数コード
                  パラメータ
```

パラメータ:

結果　　PIC X COMP-X

関数コード　　PIC X COMP-X

パラメータ　　ファイルのFD名

入口上のパラメータ:

関数コード　　値46

パラメータ　　ファイル記述（FD）で指定されたファイル識別子。現在開いている行順ファイルを参照する必要があります。

出口上のパラメータ:

結果　　呼出しが成功した場合はゼロ、そうでなければゼロ以外の値が入ります。

説明:

ASCII以外のデータをファイルに含ませたい場合、空挿入を使用可能とする必要があります。本関数を用いれば、ランタイムスイッチNの設定にかかわらず、個々のファイルについて空挿入を可能とすることができます。

例:

```
fd payroll-file
    ...
    call x"91" using result, func-num,
                             payroll-file
```

X"91" 関数47

行順ファイルにおいて、その値がx "20"未満のデータ文字の前に空文字x "00"を挿入することを禁止します。

構文:

```
CALL X"91"  USING 結果
                  関数コード
                  パラメータ
```

パラメータ:

結果　　PIC X COMP-X

関数コード　　PIC X COMP-X

パラメータ　　ファイルのFD名

入口上のパラメータ:

関数コード　　値47

パラメータ　　ファイル記述（FD）で指定されたファイル識別子。現在開いている行順ファイルを参照する必要があります。

出口上のパラメータ:

結果　　呼出しが成功した場合はゼロ、そうでなければゼロ以外の値が入ります。

説明:

本関数を用いれば、ランタイムスイッチNの設定にかかわらず、個々のファイルについて空挿入を禁止することができます。

ファイルがEXTERNALとして宣言されているとこの関数を使用することはできません。

X"91" 関数48

行順ファイルにおいて、タブ文字の挿入を可能とします。

構文:

```
CALL X"91"  USING 結果
                  関数コード
                  パラメータ
```

パラメータ:

結果　　PIC X COMP-X

関数コード　　PIC X COMP-X

パラメータ　　ファイルのFD名

入口上のパラメータ:

関数コード　　値48

パラメータ　　ファイル記述（FD）で指定されたファイル識別子。現在開いている行順ファイルを参照する必要があります。

出口上のパラメータ:

結果　　呼出しが成功した場合はゼロ、そうでなければゼロ以外の値が入ります。

説明:

入力時、全てのタブ文字が正しい個数のスペースに展開されます。一方、ディスクへの出力時、タブ位置の前の複数のスペースがタブ文字に短縮されます。本関数を用いれば、ランタイムスイッチTの設定にかかわらず、個々のファイルについてタブ挿入を可能とすることができます。

X"91" 関数49

行順ファイルにおいて、タブ文字の挿入を可能とします。

構文:

```
CALL X"91"  USING 結果
                  関数コード
                  パラメータ
```

パラメータ:

結果　　PIC X COMP-X

関数コード　PIC X COMP-X

パラメータ　ファイルのFD名

入口上のパラメータ:

関数コード　　値49

パラメータ　　ファイル記述（FD）で指定されたファイル識別子。現在開いている行順ファイルを参照する必要があります。

出口上のパラメータ:

結果　　呼出しが成功した場合はゼロ、そうでなければゼロ以外の値が入ります。

説明:

入力時、全てのタブ文字が正しい個数のスペースに展開されます。ディスクへの出力時、スペースはスペースとして出力されます。本関数を用いれば、ランタイムスイッチTの設定にかかわらず、個々のファイルについてタブ挿入を禁止することができます。

X"91" 関数69

ディレクトリを探索して与えられたファイル指定を捜します。

構文:

```
CALL X"91"  USING 結果
                  関数コード
                  パラメータ
```

パラメータ:

結果　　次のように定義された集団項目:

fーエラー　　PIC X COMP-X.

fーハンドル　　PIC X (2) COMP-X.

fー属性***out***　　PIC X COMP-X.

fー時刻　　PIC X (2) COMP-X.

fー日付　　PIC X (2) COMP-X.

fーサイズ　　PIC X (4) COMP-X.

fーファイル***out***　PIC X (n) .

関数コード　　PIC X COMP-X.

パラメータ　　次のように定義された集団項目:

fー動作　　PIC X COMP-X.

fー属性***in***　　PIC X COMP-X.

fーファイル***in***　　PIC X (n) .

入口上のパラメータ:

関数コード　69が入ります。

***f*ー動作**
実行する動作を定義します:
0=最初の一致ファイルを見つける（最初を発見）
1=次の一致ファイルを見つける（次を発見）
2=早めに探索を終了（探索終了）
3=1個の一致ファイルを見つける（1個発見）

f*ー属性*in
属性バイト。設定されたビットは、これらの属性を持ったファイルのみが含まれることを示します。
ビット7-未使用
ビット6-未使用
ビット5-所定期間保存ファイル
ビット4-サブディレクトリ
ビット3-ボリュームラベル
ビット2-システムファイル
ビット1-隠れたファイル
ビット0-読出し専用ファイル

f*ーファイル*in
要求されたファイルのスペース終了ファイル名指定。このファイル名指定はドライブ/ディレクトリまたは任意のワイルドカード文字を含む場合があります。

出口上のパラメータ:

***f*ーエラー**
結果の状態が入ります。
0=成功/ファイルが見つかりました
1=これ以上ファイルがありません
2=エラー

***f*ーハンドル**
検索ハンドル。本項目は「最初を発見」機能（上記 f-動作を参照）により設定され、対応する探索終了が行われるまで変更してはいけません。

f*ー属性*out
見つかったファイルの属性バイト:
ビット7-未使用
ビット6-未使用
ビット5-所定期間保存ファイル
ビット4-サブディレクトリ
ビット3-ボリュームラベル
ビット2-システムファイル
ビット1-隠れたファイル
ビット0-読出し専用ファイル

***f*ー時刻**
時刻ファイルがDOS形式で作成されました。
ビット15〜11 - 時 0〜23
ビット10〜5 - 分 0〜59

ビット4〜0 - 2秒単位 0〜29

***f*ー日付**

日付ファイルがDOS形式で作成されました。
ビット15〜9 - 年 0〜119（1980〜2099）
ビット8〜5 - 月1〜12
ビット4〜0 - 日1〜31

***f*ーサイズ**　ファイルのサイズ（バイト）

f*ーファイル*out　見つけたファイルの名前で、スペース終了

説明:

1個のファイルを見つけるには、「1個発見」機能を使用します。

いくつかのファイルを見つけるには、「最初を発見」機能を使用し、次に「次を発見」機能を使用します。そして、これを繰り返します。一致するファイルがそれ以上なくなると、「次を発見」はf-エラーに1を入れて戻ります。これが発生するまで「次を発見」を呼び続けることはしない場合、「探索終了」機能を呼び出して終了する必要があります。

「最初を発見」または「1個発見」機能を実行する前に、f-ハンドル項目をゼロに設定する必要があります。その後、最初/次/終了シーケンスを使用しているのであれば、探索終了後まではこの項目を変更してはいけません。「1個発見」機能を使用しているのであれば、この項目はその呼出しの後で自動的にゼロにリセットされます。

任意の時点でf-エラーが1または2を返すと、探索の終了が自動的に実行され、「探索の終了」機能を実行する必要はありません。

本ルーチンは、¥¥server1¥share1のようなネットワーク共有資源があるかどうかをチェックするためには使用できません。しかし、¥¥server1¥share1¥*のような共有資源にファイルまたはディレクトリを置くために使用することはできます。

9.2.1.8 画面の制御

X"A7" 関数6、7

利用者属性の読み込みと設定を行います。

構文:

```
CALL x "A7" USING 関数コード
                  利用者属性
      RETURNING 状態コード
```

パラメータ:

関数コード　PIC X COMP-X.

パラメータ　　PIC X COMP-X.

状態コード　　「用語説明」の項を参照してください。

入口上のパラメータ:

関数コード　処理を指定します。

6　利用者属性を読み込みます。

7　利用者属性を設定します。

利用者属性　関数7用。利用者属性を設定する値。

出口上のパラメータ:

利用者属性　関数6用。現在の利用者属性の値。

説明:

本ルーチンを使用する場合、他のCOBOL構文を使用する場合や標準のCOBOL構文を使用する場合と同じように注意が必要です。

X"A7" 関数16

利用者属性をオンまたはオフにします。

構文:

```
CALL x "A7" USING 関数コード
                  利用者属性
```

パラメータ:

関数コード　　PIC X COMP-X.

パラメータ　　PIC X COMP-X.

入口上のパラメータ:

関数コード　値16

パラメータ　希望の処理を示します。

0　利用者属性をオン

1　利用者属性をオフ

出口上のパラメータ:　なし

X"A7" 関数17

カーソルタイプを設定します。

構文:

```
CALL x "A7" USING 関数コード
                  カーソルタイプ
```

パラメータ:

関数コード　　PIC X COMP-X.

カーソルタイプ　次のように定義された集団項目

カーソル終了　PIC X COMP-X.
カーソル開始　PIC X COMP-X.

入口上のパラメータ:

関数コード　　値17
カーソル終了　ラインカーソルの終了を調べます。
カーソル開始　ラインカーソルの開始を調べます。

出口上のパラメータ:　なし

説明:

カーソルの開始および終了行の詳細については、「IBM BIOSテクニカルリファレンスマニュアル」を参照してください。

X"A7" 関数18

コンソール入出力をリダイレクト可能にします。

構文:

```
CALL x "A7" USING 関数コード
                  パラメータ
```

パラメータ:

関数コード　　PIC X COMP-X.
パラメータ　　PIC X COMP-X.

入口上のパラメータ:

関数コード　　値18
パラメータ　　実行する処理
　　0　DOS型コンソール入出力を使用禁止にします。
　　1　DOS型コンソール入出力を使用可能にします。

出口上のパラメータ:　なし

説明:

通常、ランタイムシステムはキーボードを読み込むのにBIOSを使用することになり、全ての表示についてビデオマップに直接書き出します。本ルーチンはオペレーティングシステム機能が代わりに使用されるようにします。これはオペレータがキーボード入力および画面出力のリダイレクトを行うオペレーティングシステム機能を使用できることを意味しています。

X"A7" 関数20、21

システム属性の読込み・設定を行います。

構文:

```
CALL X "A7" USING 関数コード
```

システム属性配列
RETURNING **状態コード**

パラメータ:

関数コード	PIC X COMP-X.
システム属性配列	次のように定義された集団項目:
システム属性	PIC X COMP-X OCCURS 16 TIMES.
状態コード	「用語説明」の項を参照してください。

入口上のパラメータ:

関数コード

動作を定義します。

20=システム属性の読込み
21=システム属性の設定

システム属性配列

設定するシステム属性が入ります。属性の使用目的は次の通りです。

システム属性1=見えないようにする
システム属性2=反転
システム属性3=通常
システム属性4=強調表示
システム属性5=通常の下線
システム属性6=強調表示の下線
システム属性7=システム通常
システム属性8=点滅強調表示
システム属性9=予備
システム属性10=利用者反転
システム属性11=利用者通常
システム属性12=利用者強調表示
システム属性13=予備
システム属性14=予備
システム属性15=予備
ステム属性16=オペレーティングシステム通常

出口上のパラメータ:

システム属性配列　読み込まれたシステム属性が入ります

説明:

システム属性は、画面を各種の色あるいは陰影（画面の種類による）で表示するために、システムのある部分により使用されます。これは、システム属性を変えることによって、ある特定の状態に表示を色づけできることを意味しています。属性値を自身のパラメータの一部として、あるいは、プラットホーム間アプリケーションの開発のために必要とするPanels（オプション製品に提供されている）などの構成要素を開発者が使用している場合、これらの呼出しを使用することがあります。

属性のどれかを変更しようとする場合、先ず、属性の現在設定を読込み、希望する変更を加えて属性配列を更新し、それから、呼出しに属性を設定させます。このようにすることによって、他の関係ない属性の設定に影響しないですみます。

X"A7" 関数25

現在の画面タイプをランタイムシステムが認識できるように返します。

構文:

```
CALL X "A7" USING 関数コード
                  パラメータ
```

パラメータ:

関数コード PIC X COMP-X.

パラメータ PIC X COMP-X.

入口上のパラメータ:

関数コード 値25

出口上のパラメータ:

次のように画面タイプを示す設定が入ります。

パラメータ

ビット0 設定しない。白黒画面
ビット1 予約済み
ビット2 予約済み
ビット3 設定する。EGAタイプ画面
ビット4 設定する。VGAタイプ画面
ビット5 予約済み
ビット6 予約済み
ビット7 予約済み

9.2.1.9 複数リールファイル見出しレコードの設定

```
CALL x "A8" USING 利用者ラベル
```

利用者ラベル　　見出しに入れる情報の入ったPIC X（44）項目

注：

- この領域にいれることのできるのは、ASCII印字可能文字のみです。本ルーチンがいったん使用されると、複数リールファイル上の各後続OPEN OUTPUTは、その見出しに利用者ラベルの内容を保持します。
- 初期のバージョンのMicro Focus COBOLにより作成された複数リールファイルを読み込むプログラムを実行しているときは、-Vランタイムスイッチを使用する必要があります。これにより、見出しファイルのチェックが使用禁止になります。
- これ以上の情報については、本マニュアルの「第8章　COBOLファイル処理」を参照してください。

この呼出しは、外部ファイル・ハンドラ（MFFH）でアクセスするファイルにはサポートされません。これには、索引ファイルおよび外部ファイルまたはCALLFH指令を指定してコンパイルされたプログラムからアクセスされる全てのファイルが該当します。

ADIS構成

CALL x"AF" USING ビットペアの設定　パラメータブロック

X"AF" 関数1

ADISの各種構成可能な機能を設定します。この機能には、個別利用者機能、ADISやデータキー、実行時の連続キーなどが含まれます。

構文:

CALL X "AF" USING **関数コード**
パラメータブロック

パラメータ:

関数コード　PIC X COMP-X.

パラメータブロック　次のように定義された集団項目

キー設定	PIC X COMP-X
副関数コード	PIC X COMP-X
先頭キーID	PIC X COMP-X
キー数	PIC X COMP-X

入口上のパラメータ:

関数コード　値1

次のいずれかの値を設定します。

副関数コード

1　利用者定義関数キーおよび互換キーを使用可能／使用禁止にする
2　ADIS制御キーおよび他の構成可能なキーを使用可能／使用禁止にする
3　データキーを使用可能／使用禁止にする
4　シフトキーを使用可能／使用禁止にする
5　ロックキーを使用可能／使用禁止にする

出口上のパラメータ:

パラメータ　次のように画面タイプを示すビット設定が入ります。

説明:

機能は、別のX"AF"の呼び出しによって明示的に変更されるまで、またはアプリケーションが終了するまでは、使用可能にされたり使用禁止にされたりします。利用者定義キーの初期状態は、ランタイムオペレーティングシステムによって異なります（詳細は、ADISの章を参照してください）。

関数キーを使用可能または使用禁止にする呼出しは、付加的なものです。たとえば、関数キーF1を使用可能にするのにX"AF"を呼び出し、F10を使用可能にするのに別の呼出しを行う場合、両方のキーが使用可能になります。

ADISの章はその機能とパラメータについて詳細に記述しています。

X"AF" 関数18

画面上の現在のカーソル位置に文字を表示します。

構文:

```
CALL X "AF" USING 関数コード
                  文字
```

パラメータ:

関数コード PIC X COMP-X.
文字 PIC X.

入口上のパラメータ:

関数コード 値18

出口上のパラメータ: なし

X"AF" 関数22

ターミナルアラームを発します。

構文:

```
CALL X "AF" USING 関数コード
                  パラメータ
```

パラメータ:

関数コード PIC X COMP-X.
パラメータ PIC X.

入口上のパラメータ:

関数コード 値22
パラメータ 予約済み

出口上のパラメータ: なし

X"AF" 関数26

キーボードから文字を取得します。

構文:

CALL X "AF" USING **関数コード**
キー状態

パラメータ:

関数コード　PIC X COMP-X.

次のように定義された集団項目

キー状態

キータイプ	PIC X.
キーコード1	PIC X COMP-X.
キーコード2	PIC X COMP-X.

入口上のパラメータ:

関数コード　値26

出口上のパラメータ:

次のいずれかの値を設定します。

キータイプ

1	利用者定義関数キー
2	ADIS関数キー
3	データキー
9	エラー

キーコード1　キータイプが1または2の場合、キーの番号が入ります。番号は、利用者定義関数キー用は0～127、ADIS用は0～39です。関数キーの詳細については、「Programmer's Guide to Creating User Interfaces」の「ADIS構成ユーティリティ（ADIS CF）」を参照してください。
キータイプが3の場合、押されたキーのASCIIコードが入ります。キータイプが9の場合、次のエラーコードが入ります。

8	使用不可能な文字が入力されました。キーコード2にその文字が入っています。
9	（2バイト以上の）キーストロークが不正です。

説明:

本ルーチンは、COBOL画面操作システムを起動します。詳細については、「Programmer's Guide to Creating User Interfaces」の「Comparison of Screen Handling Methods」を参照してください。

マウス支援

マウス支援用の各機能については、「プログラマーガイド(PC編)」の「第5章　COBOLにおけるマウス支援」を参照してください。

9.2.1.10 ファンクションキーと中断キー

X"B0" 関数0

ファンクションキーテーブルを作成します。

構文:

```
CALL X "B0" USING 関数コード
                  キーテーブル
```

パラメータ:

関数コード PIC X COMP-X.

次のように定義された集団項目

キーテーブル

キープレス	PIC X COMP-X.	
キーエントリリスト	検出する各キーに対して次のように定義された集団項目	
	キーエントリ長	PIC X COMP-X.
	キーエントリ	PIC X(n).
	キーリストエンド	PIC X COMP-X.

入口上のパラメータ:

関数コード 値0

キーエントリ長 キーエントリのバイト単位の長さ（1または2）

キーエントリ 要求されたキーにより作成されたコードシーケンス

キーリストエンド 値0

出口上のパラメータ:

キープレス 使用されるキーに一致するテーブルエントリ、または一致するものがない場合はゼロ。

説明:

本ルーチンは、キーコード順（認識される各追加キーごとに1つのコード順）の表を宣言します。各キーにより送られるコードシーケンスの詳細は、現在使用しているパソコンの説明書をご参照ください。

ルーチンが呼び出された後、ACCEPT文はEnterキー、または、そのコード順が表の記述項に一致するキーのどれか1つにより終了されます。表の第1バイトには、次に、表のどの記述項が一致したかを示す番号が入ります。ゼロはEnterキーが使用されたことを示します。Enterキーが表内に記述項を持つと、ゼロではなくそのキーの記述項番号が戻されます。

本ルーチンは必要となるたびに異なる表と一緒に呼び出すことができます。現在の表が変更されると（例えば、キーが変更されるか、あるいは、長さを増加する）、そのルーチンをもう一度呼び出す必要はありません。

ある特定のプログラムにより使用される表は、本ルーチンに対する別の呼出しによりその表が抑制されるまでそのプログラムについて使用可能となったままです。これは、本ルーチンが利用者の組内の他のプログラムにより何度呼び出されるかは関係ありません。

本機能を使用するプログラムの組内の各プログラムごとに表を明示的に設定する必要があります。表を連絡節に入れて渡すだけでは十分ではありません。

送信されたキーコードを利用者が認識している限り、任意のキーを表に定義できます。例えば、長さが1でPIC X VALUE"C"というキーコードを持った表記述項は、文字CによりACCEPTを終了させます。従って、そのようなキーをACCEPTを「抜ける」ために使用し、利用者自身の処理を実行し、そして、ACCEPT文に戻るということが可能になります。この効果は、その特定のキーの効果を再定義したということです。

定義された表内のキーが押されると、ACCEPTの終了時およびルーチンCBL_READ_KBD_CHARに対する個々の呼出し時の両方において、結果バイトが設定されます。これが発生すると、x "0D"がCBL_READ_KBD_CHARルーチンに戻され、表内のどのキーが押されたかを、結果バイトが示します。これは、ファンクションキーを含む任意の単一キー入力を読み込むのに、CBL_READ_KBD_CHARを使用できることを意味しています。

キーによって送られるコードシーケンスの全リストについては、ご利用のパーソナルコンピュータの「テクニカルリファレンスマニュアル」を参照してください。

本呼出しにより定義されたキーはADISCFで定義されたキーに優先します。

ADISCFにより設定され、CALL x "AF"により使用されるファンクションキーテーブルは、CALL x "B0"により使用されるファンクションキーテーブルとは異なります。両テーブルとも、x "B0"テーブル内の全てのキー順に対して復帰改行文字x "0D"をADISに戻させるようにして、同時に使用可能にできます。

「キーボードルーチンの使用例」

X"B0" 関数2

Shit、Alt、Ctrl、およびLockキーの状態をテストします。

構文:

```
CALL X "B0" USING 関数コード
                  状態ブロック
```

パラメータ:

関数コード PIC X COMP-X.

状態ブロック 次のように定義された集団項目

状態	PIC X
状態ID	PIC X COMP-X.
状態予約	PIC X(6)

入口上のパラメータ:

関数コード 値2

状態ID 2を設定する必要があります。呼出しにより上書きされるので、呼出しの前に2にリセットします。

状態予約　　予約済み

出口上のパラメータ:

次の順番で、キーの状態を表示します（1はキーが押された、0はキーが押されなかった）。

状態

ビット7	Ins
ビット6	Caps Lock
ビット5	Num Lock
ビット4	Scroll Lock
ビット3	Alt
ビット2	Ctrl
ビット1	Shift（左側のみ）
ビット0	Shift（右側のみ）

説明:

Shit、Alt、Ctrlキーに対しては、本ルーチンの呼び出し時にキーが実際に押された場合のみ1を返します。その他のキーに対しては、キーが押されるたびに適切なビットの設定と設定解除が交互に行われます。

「キーボードルーチンの使用例」

X"AF"関数4

中断のキー順（通常Ctrl+Break）を使用禁止とします。

構文:

CALL X"B0" USING **関数コード**
パラメータ

パラメータ:

関数コード　　PIC X COMP-X.
パラメータ　　PIC X COMP-X.

入口上のパラメータ:

関数コード　　4が入ります
パラメータ　　1が入ります

出口上のパラメータ:　　なし

説明:

本ルーチンはアプリケーションの残りの部分について中断のキー順を使用禁止にします。任意の副プログラムが呼び出される前に本ルーチンを呼び出す必要があります。

「キーボードルーチンの使用例」

9.2.1.11 警告音を鳴らす

X"E5"

本ルーチンは警告音を鳴らします。

構文:

```
CALL x "E5"
```

パラメータ: なし

説明:

この鳴動時間および間隔は利用者の環境により変化する場合があります。

9.2.1.12 バイトをパックする

X"F4"

8個のバイトの最下位ビットをシングルバイトにパックします。

構文:

```
CALL x "F4" USING バイト,配列
```

パラメータ:

バイト　　PIC X COMP-X.
配列　　PIC X COMP-X OCCURS 8

入口上のパラメータ:

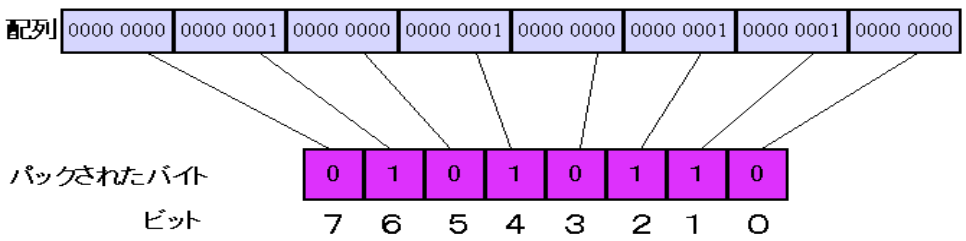
配列　　パックされるビット

出口上のパラメータ:

バイト　パックされたバイト

説明:

本ルーチンは配列から8個の項目を取り出し、各バイトの最下位ビットを用いてバイトを形成します。配列内に最初に出現するのはバイトの最上位ビット（ビット7）です。



J-1　バイトのパック機能の例

9.2.1.13 バイトをアンパックする

X"F5"

1個のバイトのビットを8バイトにアンパックします。

構文:

CALL x "F5" USING **バイト**,**配列**

パラメータ:

バイト　　PIC X COMP-X.

配列　　PIC X COMP-X OCCURS 8

入口上のパラメータ:

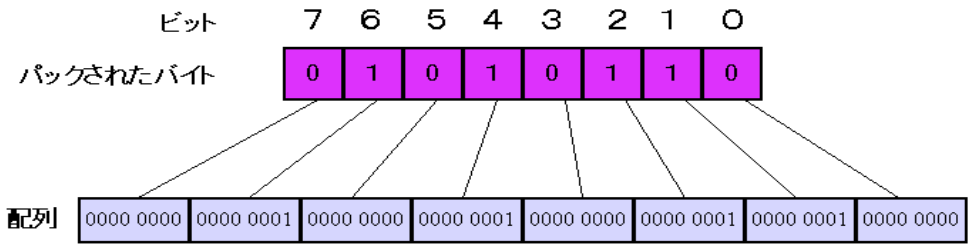
バイト　アンパックされるバイト

出口上のパラメータ:

配列　　アンパックされたビット

説明:

本ルーチンはバイトの8個のビットを取り込み、それらを配列内の対応する出現に転記します。



J-2　バイトのアンパック機能の例

付録A:　文字集合

現在使用されているパーソナルコンピュータは標準のASCII文字集合以外の文字も扱うことができます。扱える完全な文字集合については、コンピュータの解説書を参照してください。これらの文字は全てCOBOLシステムで使用できます。

次の表はASCII文字集合と日本語環境でのカタカナと漢字の第1バイト値についてのコード範囲を示します。

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0				0	@	P	´	p								
1			!	1	A	Q	a	q								
2			"	2	B	R	b	r								
3			#	3	C	S	c	s								
4			$	4	D	T	d	t								
5			%	5	E	U	e	u								
6			&	6	F	V	f	v								
7			,	7	G	W	g	w								
8			(	8	H	X	h	x								
9			)	9	I	Y	i	y								
A			*	:	J	Z	j	z								
B			+	;	K	[	k	{								
C			,	<	L	¥	l	|								
D			-	=	M	]	m	}								
E			.	>	N	^	n	～								
F			/	?	O	-	o	del								

2バイト漢字コード第1バイト範囲

1バイトカタカナコード

2バイト漢字コード第1バイト範囲