IBM

eserver

iSeries

WebSphere® Development Studio: ILE RPG プログラマーの手引き

バージョン 5

SD88-5042-04
(英文原典：SC09-2507-05)

IBM

@server

iSeries

WebSphere® Development Studio: ILE RPG プログラマーの手引き

バージョン 5

SD88-5042-04
(英文原典：SC09-2507-05)

ご注意

本書および本書で紹介する製品をご使用になる前に、 523 ページの『特記事項』に記載されている情報をお読みください。

本書は IBM WebSphere Development Studio for iSeries™ (5722-WDS) のバージョン 5 リリース 3 モディフィケーション・レベル 0 の ILE RPG コンパイラーに適用されます。また、改訂版などで特に断りのないかぎり、これ以降のすべてのリリースおよびモディフィケーションにも適用されます。本書は、RISC システムにのみ適用されます。

本書は SD88-5042-03 の改訂版です。

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

http://www.ibm.com/jp/manuals/main/mail.html

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

http://www.ibm.com/jp/manuals/ の「ご注文について」をご覧ください。

(URL は、変更になる場合があります)

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

原　典： SC09-2507-05
iSeries
WebSphere(R) Development Studio: ILE RPG Programmer's Guide
Version 5

発　行： 日本アイ・ビー・エム株式会社

担　当： ナショナル・ランゲージ・サポート

第1刷 2004.4

この文書では、平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、および平成角ゴシック体™W7を使用しています。この(書体*)は、(財) 日本規格協会と使用契約を締結し使用しているものです。フォントとして無断複製することは禁止されています。

注* 平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、
平成角ゴシック体™W5、平成角ゴシック体™W7

目次

本書について

本書は、ILE における ILE RPG コンパイラー (ILE RPG) の使用法を説明しています。ILE RPG を導入することにより、OS/400 オペレーティング・システムを使用する iSeries サーバー上で RPG IV 言語が使えるようになります。RPG IV をソースとした ILE アプリケーション・プログラムの作成および実行には、本書をご使用ください。

本書では、下記の事柄を行う方法について記述しております。

- RPG IV ソース・ステートメントの入力
- モジュールの作成
- モジュールのバインド
- ILE プログラムの実行
- 他のオブジェクトの呼び出し
- ILE プログラムのデバッグ
- 例外の処理
- ファイルの定義および処理
- 装置のアクセス
- RPG III 形式から RPG IV 形式へのプログラムの変換
- コンパイラー・リストの解釈

本書の対象読者

本書は、RPG プログラミング言語を熟知された、ILE フレームワークでの使用方法について学習したいとお考えの、プログラマーを対象としています。このほか、RPG III から RPG IV 形式へのプログラム変換をご希望のプログラマーも対象としています。本書は、iSeries システム上で ILE RPG コンパイラーを使用するための手引きとなるように書かれています。

本書は、ILE フレームワークでの RPG IV の使用方法を示していますが、RPG IV 仕様書および命令の詳細な説明を提供するものではありません。言語の詳しい説明は、「*WebSphere Development Studio: ILE RPG 解説書*, SD88-5043-04」をご参照ください。

本書をお使いになる前に、次のことが必要です。

- 適用できる iSeries サーバーのメニューと表示、または制御言語 (CL) コマンドの使用法を知っていること。
- ここで説明されている CL コマンドおよびオブジェクトに対する適切な権限をもっていること。
- 「*ILE 概念*, SD88-5033-06」に詳述の ILE について確実に理解していること。

前提条件および関連情報

iSeries および AS/400e 技術情報を検索する手始めとして、iSeries Information Center を使用します。Information Center には次の 2 つの方法でアクセスできます。

- 次の Web サイト:

  `http://www.ibm.com/eserver/iseries/infocenter`

- OS/400 の注文により出荷される次の CD-ROM:

  「*iSeries V5R3 Information Center*, SK88-8055-01」。このパッケージには、iSeries マニュアルの PDF バージョンである「*iSeries V5R2 Information Center: 補足資料*, SK88-8056-01」も含まれます。これは、ソフトコピー・ライブラリー CD-ROM に代わるものです。

iSeries Information Center には、CL コマンド、システム・アプリケーション・プログラミング・インターフェース (API)、論理区画、クラスター化、Java、TCP/IP、Web 機能、および保護されたネットワークといった推奨事項および重要なトピックが含まれています。また、関連する IBM レッドブックへのリンクや、Technical Studio および IBM ホーム・ページなどの他の IBM Web サイトへのインターネット・リンクも含まれています。

ILE RPG コンパイラー に最も関連の深い資料については、 527 ページの『参考文献』にリストされています。

ご意見をお寄せください

提供する情報の正確性と品質を高めるためには、お客様からのご意見のフィードバックは重要です。本書や他の iSeries の資料に関するご意見をお持ちの際は、本書の中表紙の裏に記載されているアドレスにご意見をご送付ください。

以下の項目の記入をお願いします。

- 資料名。
- 資料番号。
- ご意見の内容に該当するページ番号またはトピック。

新着情報

RPG IV には、最初の V3R1 リリース以来いくつかのリリースが存在しています。以下は、V3R1 から現行リリースに至るまでの各リリースにおいて行なわれた機能強化を列挙したものです。

- xi ページの『本リリースの新機能』
- xvi ページの『V5R2 での変更点』
- xix ページの『V5R1 での変更点』
- xxv ページの『V4R4 での変更点』
- xxix ページの『V4R2 での変更点』
- xxxiv ページの『V3R7 での変更点』
- xxxix ページの『V3R6/V3R2 での変更点』

このセクションは、RPG IV の新機能へリンクし習得するために使用することができます。

注: 当製品に関する情報は、RPG IV の V5R3 リリースで最新のものです。コンパイラーの前のリリースを使用している場合は、ユーザーのシステムでどの機能がサポートされるかを判断する必要があります。例えば、V5R1 システムを使用している場合、V5R3 リリースでの新機能はサポートされません。

V5R2 および V5R1 以降このガイドに加えられた変更

本書 V5R3 の「*WebSphere Development Studio: ILE RPG プログラマーの手引き*」(SD88-5042-04) は、多くの点で V5R2 (SD88-5042-03) および V5R1 (SD88-5042-03) とは異なります。変更の大部分は、前のリリース以降に行われた機能強化と関連していますが、小さな技術的訂正を反映している個所もあります。本書を利用しやすいように、技術的変更点および拡張機能については、余白に以下の記号が付けてあります。

- V5R3 での拡張には垂直バー (|)。
- V5R2 での拡張にはポンド記号 (#)。

注: 本書に含まれる例の多くは、「伝統的な」コーディング・スタイルではなく「自由形式」に変更されました。この、例の変更については記号は付いていません。この 2 つのコーディング・スタイル間の相違点についての説明は、「*WebSphere Development Studio: ILE RPG 解説書*」を参照してください。

本リリースの新機能

以下に、V5R3 の ILE RPG における機能強化について説明します。

- **新規組み込み関数 %SUBARR:**

  新規組み込み関数 %SUBARR を使用すると、副配列への代入または値として副配列を返すことができます。

  既存の %LOOKUP 組み込み関数とともに、この拡張では要素数が変化する動的サイズ変更配列の実装が可能になりました。

  %SUBARR(array : start) は、array(start) から配列の末尾までの配列の要素を指定します。

  %SUBARR(array : start : num) は、array(start) から array(start + num - 1) までの配列の要素を指定します。

  例:

```
// 配列の一部を別の配列にコピー:
resultArr = %subarr(array1:start:num);
// 配列の一部を別の配列の一部にコピー:
%subarr(Array1:x:y) = %subarr(Array2:m:n);
// 配列の一部をソート
sorta %subarr(Array3:x:y);

// 配列の一部の合計
sum = %xfoot(%subarr(Array4:x:y));
```

- **SORTA 命令コードは部分配列のソートが可能なように拡張されました。**

  演算項目 2 として %SUBARR が指定されると、%SUBARR 組み込み関数で示された部分配列のみがソートされます。

- **%DEC を使用した、date/time/timestamp から数値への直接変換:**

  %DEC は、最初のパラメーターに日付、時刻またはタイム・スタンプを指定し、オプションの 2 番目のパラメーターで結果の数値のフォーマットを指定できるように拡張されました。

  例:

```
D numDdMmYy       s              6p 0
D date            s               d    datfmt(*jul)
 date = D'2003-08-21';
 numDdMmYy = %dec(date : *dmy);
 // now numDdMmYy = 210803
```

- **実行時の文字データを正しく変換するための制御仕様 CCSID(*CHAR : *JOBRUN):**

  制御仕様書 CCSID キーワードは最初のパラメーターとして *CHAR を許容するように拡張されました。最初のパラメーターが *CHAR の場合、2 番目のパラメーターは *JOBRUN である必要があります。 CCSID(*CHAR : *JOBRUN) は、実行時に文字データが UCS-2 に変換される方法を制御します。
  CCSID(*CHAR:*JOBRUN) が指定されると、文字データはジョブ CCSID 内にあると見なされ、CCSID(*CHAR : *JOBRUN) が指定されない場合は、文字データはジョブ CCSID に関連する混合バイト CCSID 内にあるものと見なされます。

- **%TRIM、%TRIMR および %TRIML の 2 番目のパラメーターは以下のように、どの文字をトリムするかを示します。**

  %TRIM には、トリムする文字のリストを指定する、オプションの 2 番目のパラメーターを取るように拡張されました。

  例:

```
trimchars = '*-.';
data = '***a-b-c-.'
result = %trim(data : trimchars);
// 結果 = 'a-b-c'。すべての * - および . はデータの末尾からトリムされました。
```

- **トリムされたパラメーターを渡すための新規プロトタイプ・オプション OPTIONS(*TRIM):**

  プロトタイプ・パラメーターに OPTIONS(*TRIM) が指定されると、渡されるデータの先頭および末尾のブランクはトリムされます。 OPTIONS(*TRIM) は文字、UCS-2 および CONST または VALUE で定義されたグラフィック・パラメーターに対して有効です。また、OPTIONS(*STRING) で定義されたポインター・パラメーターに対しても有効です。 OPTIONS(*STRING : *TRIM) が指定されると、ポインターが呼び出しで渡される場合であっても、渡されるデータはトリムされます。

  例:

```
D proc            pr
D   parm1                       5a    const options(*trim)
D   parm2                       5a    const options(*trim : *rightadj)
D   parm3                       5a    const varying options(*trim)
D   parm4                        *    value options(*string : *trim)
D   parm5                        *    value options(*string : *trim)
D ptr             s              *
D data            s             10a
D fld1            s              5a

 /free
   data = ' rst ' + x'00';
   ptr = %addr(data);
```

```
proc (' xyz ' : ' @#$ ' : ' 123 ' : ' abc ' : ptr);
// 呼び出されたプロシージャーは以下のパラメーターを受け取ります
//    parm1 = 'xyz  '
//    parm2 = '  @#$'
//    parm3 = '123'
//    parm4 = a pointer to 'abc.' (where . is x'00')
//    parm5 = a pointer to 'rst.' (where . is x'00')
```

- **63 桁のパック 10 進数値およびゾーン 10 進数値のサポート**

  パック・データおよびゾーン・データを 63 桁および小数点以下 63 桁まで定義できます。以前の限界は 31 桁でした。

- **I/O の結果データ構造として外部記述されたファイルおよびレコード様式を使用する規則の緩和**

  – レコード様式に対する I/O のための結果データ構造は、外部記述されたデータ構造にすることができます。

  – データ構造は、I/O の結果フィールド内で、命令コード CHAIN、READ、READE、READP および READPE の外部記述ファイル名に指定することができます。

  例:

  1. 以下のプログラムは外部記述されたデータ構造を使用してレコード様式への書き込みを行います。

```
Foutfile    o    e           k disk
D outrecDs      e ds                    extname(outfile) prefix(O_)
/free
  O_FLD1 = 'ABCDE';
  O_FLD2 = 7;
  write outrec outrecDs;
  *inlr = *on;
/end-free
```

  2. 以下のプログラムはマルチフォーマット論理ファイルから、それぞれのレコード様式のフィールドを保持する 2 つの重複したサブフィールドを含むデータ構造 INPUT への読み取りを行います。

```
Flog       if   e           k disk    infds(infds)
D infds           ds
D   recname             261    270
D input           ds                  qualified
D   rec1                              likerec(rec1) overlay(input)
D   rec2                              likerec(rec2) overlay(input)
 /free
  read log input;
  dow not %eof(log);
   dsply recname;
   if recname = 'REC1';
    // ハンドル rec1
   elseif recname = 'REC2';
    // ハンドル rec2
   endif;
   read log input;
  enddo;
  *inlr = *on;
 /end-free
```

- **Java メソッドを呼び出す RPG プログラムで使用する新規環境変数のサポート**

  – **QIBM_RPG_JAVA_PROPERTIES** を使用すると RPG ユーザーは、JVM の開始に使用する java プロパティーを明示的に設定可能です。

この環境変数は RPG プログラムがジョブ内で Java メソッドを呼び出す前に設定する必要があります。

この環境変数には、オプション文字列内に現れない文字で分離および終了された Java オプションが含まれます。一般にセミコロンが適切です。

例:

1. **単一のオプションを指定:** システムのデフォルトの JDK が 1.3 であり、RPG プログラムで JDK 1.4 を使用する場合は、環境変数 QIBM_RPG_JAVA_PROPERTIES に次の値を設定します。

```
'-Djava.version=1.4;'
```

なお、オプションが 1 つのみの場合でも、終了文字が必要であることに注意してください。この例ではセミコロンを使用しています。

2. **複数のオプションを指定:** os400.stdout オプションもデフォルト以外の値を設定する場合は、環境変数を次の値に設定します。

```
'-Djava.version=1.4!-Dos400.stdout=file:mystdout.txt!'
```

この例では分離文字/終了文字として感嘆符を使用しています。注: この機能は PTF 適用済みの V5R1 および V5R2 でもサポートされています。 V5R1: SI10069、V5R2: SI10101。

– **QIBM_RPG_JAVA_EXCP_TRACE** を使用すると RPG ユーザーは、Java メソッドへの RPG 呼び出しが例外で終了した場合に例外トレースを取得できます。

この環境変数はいつでも設定、変更、または除去できます。

この環境変数に値 'Y' が含まれる場合、RPG からの Java メソッド呼び出し中に Java 例外が発生するか、呼び出された Java メソッドが例外を呼び出し元にスローすると、例外の Java トレースが出力されます。デフォルトでは、画面に出力され、読み取れない場合もあります。ファイルに出力するには、Java オプション os400.stderr を設定します。 (これは新規ジョブに対して行う必要があり、 QIBM_RPG_JAVA_PROPERTIES 環境変数を次の値に設定することで行います。

```
'-Dos400.stderr=file:stderr.txt;'
```

- **RPG プリプロセッサーにより、SQL プリプロセッサーにおける条件コンパイルおよびネストした /COPY の処理が可能**

RPG コンパイラーは、パラメーター PPGENOPT に *NONE 以外の値を指定して呼び出された場合、RPG プリプロセッサーとして動作します。この場合、プログラムが生成されるのではなく、新しいソース・ファイルが生成されます。新しいソース・ファイルには、/DEFINE や /IF などの条件コンパイル・ディレクティブによって受け入れられたオリジナルのソース行が含まれます。また、/COPY ステートメントで組み込まれたファイルのソース行、およびオプションで /INCLUDE ステートメントで組み込まれたソース行も含まれます。PPGENOPT(*DFT) または PPGENOPT(*NORMVCOMMENT) が指定された場合、新しいソース・ファイルにはオリジナル・ソース・ファイルからのコメントも含まれます。SQL プリコンパイラーが新規パラメーター RPGPPOPT に *NONE 以外の値を指定して呼び出されると、プリコンパイラーは /COPY、条件コンパイル・ディレクティブ、そしておそらくは /INCLUDE ディレクティブを処理するた

めに、この RPG プリプロセッサーを使用します。これにより、SQLRPGLE ソースでネストされた /COPY ステートメントおよび条件付きで使用されるステートメントを使用できます。

表 1. V5R2 以降に変更された言語要素

言語単位	要素	説明
制御仕様書キーワード	CCSID(*GRAPH:parameter| *UCS2:number| *CHAR:*JOBRUN)	実行時の文字データの扱いを制御するために、最初のパラメーターとして *CHAR、2 番目のパラメーターとして *JOBRUN を指定できるようになりました。
組み込み関数	%DEC(expression {format})	タイプ Date、Time または Timestamp のパラメーターを使用できるようになりました。
	%TRIM(expression:expression)	トリムする文字のセットを示す 2 番目のパラメーターを指定できるようになりました。
定義仕様書キーワード	OPTIONS(*TRIM)	渡されたパラメーターからブランクをトリムすることを示します。
定義仕様書	長さおよび桁数の入力	パック・フィールドおよびゾーン・フィールドでは、長さおよび小数点以下の桁数として 63 桁まで指定できます。
入力仕様書	長さ入力	パック・フィールドの長さとして 32 まで、ゾーン・フィールドの長さとして 63 まで指定できます。
	小数部の桁数入力	パック・フィールドおよびゾーン・フィールドでは、小数点以下の桁数として 63 桁まで指定できます。
演算仕様書	長さおよび桁数の入力	パック・フィールドおよびゾーン・フィールドでは、長さおよび小数点以下の桁数として 63 桁まで指定できます。
	CHAIN、READ、READE、READP、および READPE 操作	演算項目 2 が外部記述ファイルの名前である場合に、結果フィールドにデータ構造を指定できます。
	CHAIN、READ、READC、READE、READP、READPE、WRITE、UPDATE 操作	演算項目 2 が外部記述レコード様式の名前である場合に、結果フィールドに外部記述データ構造を指定できます。
	SORTA 操作	演算項目 2 が拡張され、%SUBARR を指定できるようになりました。

表 2. V5R2 以降の新しい言語要素

言語単位	要素	説明
組み込み関数	%SUBARR(array:starting element {:number of elements})	配列のセクションを返すか、配列のセクションを変更できます。

V5R2 での変更点

次に、ILE RPG V5R2 における機能強化について説明します。

- 文字から数値への変換

  組み込み関数 %DEC、%DECH、%INT、%INTH、%UNS、%UNSH、および %FLOAT が、文字パラメーターを利用できるように拡張されました。例えば、%DEC('-12345.67' : 7 : 2) と指定すると、数値 -12345.67 が戻されます。

- ビット単位の論理組み込み関数

  %BITAND、%BITOR、%BITXOR、および %BITNOT で、RPG 式内での直接ビット操作が認められるようになりました。

- 複雑なデータ構造

  データ構造定義が拡張され、データ構造の配列、およびそれ自身がデータ構造である LIKEDS で定義されたデータ構造のサブフィールドが認められようになりした。これにより、配列の配列、つまり構造のサブ配列が入っている構造の配列など、複雑な構造のコーディングを行えるようになりました。

```
Example:   family(f).child(i).hobbyInfo.pets(p).type = 'dog';
           family(f).child(i).hobbyInfo.pets(p).name = 'Spot';
```

  さらに、レコード様式と同じように、新しい LIKEREC キーワードを使用してデータ構造を定義できるようになりました。

- 外部記述データ構造の拡張

  外部記述データ構造に、プログラマーが選択した入力、出力、入出力、キー、またはすべてのフィールドを保持できるようになりました。現在、外部記述データ構造に保持できるのは入力フィールドだけです。

- キーによる入出力の拡張

  キーによる入出力操作での検索引き数を /FREE 計算で新たに 2 つの方法で指定できるようになりました。

  1. リストで検索引き数 (式も可能) を指定する
  2. 検索引き数が入っているデータ構造を指定する

```
Examples: D custkeyDS    e ds        extname(custfile:*key)
          /free
             CHAIN  (keyA : keyB : key3) custrec;
             CHAIN  %KDS(custkeyDS) custrec;
```

- 外部記述ファイルのデータ構造結果

  外部記述ファイルに入出力操作を使うときに、結果フィールドにデータ構造を指定できるようになりました。この方法は、V5R2 より前ではプログラム記述ファイルでしか行えませんでした。ファイル内のフィールド数が多い場合は、データ構造を使用するとパフォーマンスを向上させることができます。

- 選択フィールドのみを更新する UPDATE 操作

UPDATE 操作により、更新するフィールドのリストを指定できるようになりました。これは、V5R2 より前では例外出力を使用しなければ実行できませんでした。

例: `update record %fields(salary:status).`

- 31 桁のサポート

  最大 31 桁のパックおよびゾーン数データをサポートします。これは、DDS でサポートされる最大長さです。V5R2 より前では 30 桁しかサポートされませんでした。

- FEOD のパフォーマンス・オプション

  FEOD 操作は、ローカルでブロック・バッファーへの書き込みしか実行せず、コストのかかるディスク書き込みは実行しないことを示す、拡張 N をサポートすることで拡張されました。

- データ域アクセスの拡張

  DTAARA キーワードが拡張され、データ域の名前およびライブラリーを実行時に指定できるようになりました。

- 新しい代入演算子

  新しい代入演算子 +=、-=、*=、/=、**= により、変数を古い値に基づいてより簡潔に変更できるようになりました。

  ```
  Example: totals(current_customer) += count;
  ```

  上記のステートメントは、現在 ″totals(current_customer)″ に入っている値に ″count″ を追加します。 ″totals(current_customer)″ を 2 度コーディングする必要はありません。

- IFS ソース・ファイル

  ILE RPG コンパイラーは、メイン・ソース・ファイルと /COPY ファイルの両方を IFS からコンパイルできるようになりました。 /COPY 指示および /INCLUDE 指示が IFS ファイル名をサポートするように拡張されました。

- プログラム呼び出しマークアップ言語 (PCML)

  ILE RPG コンパイラーは、PCML が入っている IFS ファイルを生成し、プログラム (CRTBNDRPG) またはエクスポートされたプロシージャー (CRTRPGMOD) に対するパラメーターを表します。

表 3. V5R1 以降に変更された言語要素

言語単位	要素	説明
組み込み関数	%DEC(expression)	型を表す文字のパラメーターを取れるようになった。
	%DECH(expression)	
	%FLOAT(expression)	
	%INT(expression)	
	%INTH(expression)	
	%UNS(expression)	
	%UNSH(expression)	

表 3. V5R1 以降に変更された言語要素 (続き)

言語単位	要素	説明
定義仕様書キーワード	DTAARA({*VAR:}data-area-name)	データ域名として、名前、'LIBRARY/NAME' を指定する文字リテラル、または実行時に実際のデータ域を指定する文字変数を指定できる。
	DIM	データ構造の指定が可能。
	LIKEDS	サブフィールドの指定が可能。
	EXTNAME(filename{:extrecname}{:*ALL|*INPUT|*OUTPUT|*KEY})	オプションの ″type″ パラメーターを使って、外部記述データ構造の場合にどのタイプのフィールドを取り出すかを制御する。
定義仕様書	長さおよび桁数の入力	パック・フィールドおよびゾーン・フィールドでは、長さおよび桁数として 31 まで指定できる。
命令コード	CHAIN、DELETEREADE、READPE、SETGT、SETLL	自由形式演算では、Factor 1 にキー値のリストを指定できる。
	CHAIN、READ、READC、READE、READP、READPE、 UPDATE、WRITE	外部記述ファイルまたはレコード様式で使用する場合は、結果フィールドにデータ構造を指定できる。
	UPDATE	自由形式演算では、最後の引き数に、更新するフィールドのリストを指定できる。
	FEOD	命令拡張 N を指定できる。これは、まだ書き込まれていないバッファーはデータベースには書き込む必要があるが、必ずしもディスクには書き込む必要がないことを示す。
演算仕様書	長さおよび桁数の入力	パック・フィールドおよびゾーン・フィールドでは、長さおよび桁数として 31 まで指定できる。

表 4. V5R1 以降の新しい言語要素

言語単位	要素	説明
式	代入演算子 += -= *= /= **=	代入演算子を使用する場合、演算のターゲットは演算の第 1 オペランドでもある。
制御仕様書キーワード	DECPREC(30|31)	表示の場合 (例えば、%EDITC および %EDITW) の 10 進数中間値の精度を制御する。
定義仕様書キーワード	LIKEREC(intrecname{:*ALL|*INPUT|*OUTPUT|*KEY})	サブフィールドがレコード様式と同じであるデータ構造を定義する。

表 4. V5R1 以降の新しい言語要素 (続き)

言語単位	要素	説明
組み込み関数	%BITAND(expression : expression)	オペランドの対応するビットが両方ともオンの場合にオンであるビットの結果を戻す。
	%BITNOT(expression)	引き数のビットとは逆のビットの結果を戻す。
	%BITOR(expression : expression)	オペランドの対応するビットのいずれかがオンの場合にオンであるビットの結果を戻す。
	%BITXOR(expression : expression)	オペランドの対応するビットの 1 つだけがオンである場合にオンであるビットの結果を戻す。
	%FIELDS(name{:name...})	自由形式の UPDATE で、更新するフィールドを指定する場合に使用。
	%KDS(data structure)	自由形式のキー付き命令コード CHAIN、SETLL、SETGT、READE、および READPE で、操作のキーがデータ構造内にあることを示す場合に使用。

V5R1 での変更点

ILE RPG コンパイラーは、新たに C/C++ および COBOL コンパイラーが組み込まれた IBM WebSphere Development Studio 製品の一部であり、アプリケーション開発ツールセット・ツールです。

V4R4 以降の RPG IV の主要な機能強化は、 Java との簡単なインターフェース接続、新しい組み込み関数、自由形式演算仕様書、オープンにするファイルの制御、修飾サブフィールド名、およびエラー処理の拡張です。

以下に、これらの強化機能について説明します。

- Java と ILE RPG 間の呼び出しのサポートが、Java Native Interface (JNI) の使用により改善されました。
  - 新しいデータ・タイプ: オブジェクト
  - 新しい定義仕様書キーワード: CLASS
  - LIKE 定義仕様書キーワードが拡張され、オブジェクトをサポートするようになりました。
  - EXTPROC 定義仕様書キーワードが拡張され、 Java プロシージャーをサポートするようになりました。
  - 新しい状況コード。
- 新しい組み込み関数。
  - 数字を期間に変換する次の関数。これらは算術式の中で使用することができます:

    %MSECONDS、%SECONDS、%MINUTES、%HOURS、%DAYS、%MONTHS、および %YEARS。
  - %DIFF 関数。ある日付、時刻、またはタイム・スタンプの値を、もう 1 つの値から減算します。

  - 文字ストリング (または日付またはタイム・スタンプ) を日付、時刻、またはタイム・スタンプに変換する次の関数: %DATE、%TIME、および %TIMESTAMP。
  - %SUBDT 関数。日付、時刻、またはタイム・スタンプのサブセットを取り出します。
  - 記憶域の割り振りまたは再割り振りを行なう関数: %ALLOC および %REALLOC。
  - 配列の要素を検索する次の関数: %LOOKUP、%LOOKUPGT、%LOOKUPGE、%LOOKUPLT、および %LOOKUPLE。
  - テーブルの要素を検索する次の関数: %TLOOKUP、%TLOOKUPGT、%TLOOKUPGE、%TLOOKUPLT、および %TLOOKUPLE。
  - 指定された文字だけを含むストリングを検査する関数: %CHECK および %CHECKR。
  - %XLATE 関数。変換元文字と変換先文字のリストに基づいてストリングの変換を行います。
  - %OCCUR 関数。複数オカレンス・データ構造において現行のオカレンスを入手したり設定したりします。
  - %SHTDN 関数。オペレーターがシャットダウンを要求したかどうかを判断します。
  - %SQRT 関数。数値の平方根を計算します。
- 演算仕様書の新たな自由形式構文。自由形式演算仕様書のブロックは、コンパイラー指示 /FREE および /END-FREE で区切ります。
- ファイル仕様書に EXTFILE キーワードおよび EXTMBR キーワードを指定して、ファイルがオープンされる時にどの外部ファイルが使用されるかを制御することができます。
- データ構造の中で次の通り修飾名がサポートされます。
  - 新しい定義仕様書キーワード: QUALIFIED。このキーワードは、サブフィールド名がデータ構造名によって修飾されることを示します。
  - 新しい定義仕様書キーワード: LIKEDS。このキーワードは、サブフィールドが別のデータ構造から複製されることを指定します。サブフィールド名は新しいデータ構造名によって修飾されます。LIKEDS はプロトタイプ化されたパラメーターに使用できます。このキーワードにより、パラメーターのサブフィールドを直接に呼び出し先プロシージャーで使用することができます。
  - INZ 定義仕様書キーワードが拡張され、データ構造をその親データ構造を基にして初期化することができるようになりました。
- 次のような、エラー処理の機能拡張。
  - 新しい 3 つの演算コード (MONITOR、ON-ERROR、および ENDMON) によって、状況コードに合わせた条件付きエラー処理を伴う命令グループを定義することができるようになりました。

このリリースについては、この他にも、次のような機能強化がなされました。

- パラメーターを持たないプロシージャー呼び出しにおいて、括弧を指定することができます。
- プロシージャーが ILE C 呼び出し規則または ILE CL 呼び出し規則を使用することを、EXTPROC 定義仕様書キーワードで指定することができます。
- 次の /DEFINE 名が事前定義されています:
  *VnRnMn、*ILERPG、*CRTBNDRPG、および *CRTRPGMOD。
- %SCAN 命令における検索ストリングが、検索対象となるストリングよりも長くてもかまいません。 (そのストリングは検索されないことになりますが、これによってエラー条件が生成されることはなくなりました)。
- DIM キーワード、OCCURS キーワード、および PERRCD キーワードのパラメーターは、事前に定義されている必要がなくなりました。
- %PADDR 組み込み関数は、その引き数に応じて、プロトタイプ名または入り口点名のいずれかをとることができるようになりました。
- 新しい命令コード ELSEIF は、ELSE 命令コードと IF 命令コードを結合したもので、追加の ENDIF は必要ありません。
- DUMP 命令コードは、DEBUG(*NO) が指定されていても必ずダンプが生成されるという意味の A 拡張を、新しくサポートするようになりました。
- 新しい指示 /INCLUDE は、/INCLUDE が SQL プリプロセッサーによって展開されない点を除けば、/COPY と同等です。組み込みファイルは、組み込み SQL またはホスト変数を含むことはできません。
- OFLIND ファイル仕様書キーワードは、名前付き標識を含むすべての標識を、引き数として取ることができるようになりました。
- LICOPT (ライセンス内部コード・オプション) キーワードが、CRTRPGMOD コマンドおよび CRTBNDRPG コマンドで使用可能になりました。
- PREFIX ファイル記述キーワードが、引き数として上段シフト文字リテラルを取ることができるようになりました。リテラルはピリオドで終わることができ、ファイルはこのピリオドがあれば修飾付きサブフィールドで使用できるようになります。
- PREFIX 定義仕様書キーワードも、引き数として上段シフト文字リテラルを取ることができるようになりました。このリテラルはピリオドで終わることはできません。

以下の表に、影響を受ける言語の部分別に、変更のあった言語要素と新しい言語要素を要約します。

表 5. V4R4 以降に変更された言語要素

言語単位	要素	説明
組み込み関数	%CHAR(式{:形式})	オプションである 2 番目のパラメーターには、使用したい日付、時刻、またはタイム・スタンプの形式を指定します。結果は、指定された形式と形式のセパレーターを使用し、入力の形式とセパレーターは使用しません。
	%PADDR(プロトタイプ名)	この関数は、その引き数に応じて、プロトタイプ名または入り口点名のいずれかをとることができるようになりました。

表 5. V4R4 以降に変更された言語要素 (続き)

言語単位	要素	説明
定義仕様書キーワード	EXTPROC(*JAVA:クラス名:プロシージャー名)	これを指定すると、Java メソッドが呼び出されます。
	EXTPROC(*CL:プロシージャー名)	プロシージャーが戻り値に ILE CL 規則を使用することを指定します。
	EXTPROC(*CWIDEN:プロシージャー名)	プロシージャーが、パラメーター拡大付きで ILE C 規則を使用することを指定します。
	EXTPROC(*CNOWIDEN:プロシージャー名)	プロシージャーが、パラメーター拡大なしで ILE C 規則を使用することを指定します。
	INZ(*LIKEDS)	LIKEDS キーワードを指定して定義されたデータ構造が、その親データ構造から初期設定を継承することを指定します。
	LIKE(オブジェクト名)	オブジェクトが、別のオブジェクトと同じクラスを持つことを指定します。
	PREFIX(文字リテラル{:数値})	指定された文字リテラルでサブフィールドに接頭語を付け、任意指定で指定された文字数を置換します。
ファイル仕様書のキーワード	OFLIND(名前)	このキーワードは、任意の名前付き標識をパラメーターとしてとることができるようになりました。
	PREFIX(文字リテラル{:数値})	指定された文字リテラルでサブフィールドに接頭語を付け、任意指定で指定された文字数を置換します。
命令コード	DUMP (A)	この命令コードは、DEBUG(*NO) が指定されていてもダンプが生成される、 A 拡張を今後取ることができるようになりました。

表 6. V4R4 以降の新しい言語要素

言語単位	要素	説明
データ・タイプ	オブジェクト	Java オブジェクトに使用されます。
コンパイラー指示	/FREE ... /END-FREE	/FREE... /END-FREE コンパイラー指示は、自由形式演算仕様書ブロックを示します。
	/INCLUDE	SQL プリプロセッサーによって展開されない点を除き、/COPY と同等です。コピーされたファイルの中にある、ネストされたファイルを組み込むために使用できます。コピーされたファイルに、埋め込み SQlL または埋め込みホスト変数を入れることはできません。
定義仕様書キーワード	CLASS(*JAVA:クラス名)	オブジェクトのクラスを指定します。
	LIKEDS(データ構造名)	データ構造、プロトタイプ化されたパラメーター、あるいは戻り値が、別のデータ構造のサブフィールドを継承することを指定します。
	QUALIFIED	データ構造内のサブフィールド名が、データ構造名によって修飾されることを示します。

表 6. V4R4 以降の新しい言語要素 (続き)

言語単位	要素	説明
ファイル仕様書のキーワード	EXTFILE(ファイル名)	どのファイルをオープンするか指定します。値はリテラルまたは変数を指定できます。デフォルトのファイル名は、ファイル仕様書の位置 7 で指定されている名前になります。デフォルトのライブラリーは *LIBL です。
	EXTMBR(メンバー名)	どのメンバーをオープンするか指定します。値はリテラルまたは変数を指定できます。デフォルト値は *FIRST です。

表 6. V4R4 以降の新しい言語要素 (続き)

言語単位	要素	説明
組み込み関数	%ALLOC(数値)	指定された容量の記憶域を割り振ります。
	%CHECK(コンパレーター:基本{:開始})	コンパレーターの中になく基本ストリングの中にある先頭文字を検索します。
	%CHECKR(コンパレーター:基本{:開始})	コンパレーターの中になく基本ストリングの中にある末尾文字を検索します。
	%DATE(式{:日付の形式})	式を日付に変換します。
	%DAYS(数値)	数値を日単位の期間に変換します。
	%DIFF(オプション1:オプション2:単位)	2 つの日付、時刻、またはタイム・スタンプの値の間の差 (期間) を、指定された単位で計算します。
	%HOURS(数値)	数値を時間単位の期間に変換します。
	%LOOKUPxx(引き数:配列{:開始索引{:要素の数}})	指定された引き数に近い引き数、または指定されたタイプに近いタイプを、指定された配列内で検索します。
	%MINUTES(数値)	数値を分単位の期間に変換します。
	%MONTHS(数値)	数値を月単位の期間に変換します。
	%MSECONDS(数値)	数値をマイクロ秒単位の期間に変換します。
	%OCCUR(データ構造名)	複数オカレンス・データ構造の現在位置を設定するか、または入手します。
	%REALLOC(ポインター:数値)	指定された容量の記憶域を、指定されたポインター用に再割り振りします。
	%SECONDS(数値)	数値を秒単位の期間に変換します。
	%SHTDN	システム・オペレーターがシャットダウンを要求しているかどうかをチェックします。
	%SQRT(数値式)	指定された数値の平方根を計算します。
	%SUBDT(値:単位)	日付、時刻、またはタイム・スタンプの値から、指定された部分を抽出します。
	%THIS	代わりに固有メソッドが呼び出されるクラス・インスタンスへの参照を含むオブジェクト値を戻します。
	%TIME(式{:時刻形式})	式を時刻に変換します。
	%TIMESTAMP(式{:*ISO|*ISO0})	式をタイム・スタンプに変換します。
	%TLOOKUP(引き数:検索テーブル{:代替テーブル})	指定された引き数に近い引き数、または指定されたタイプに近いタイプを、指定されたテーブル内で検索します。
	%XLATE(変換元:変換先:ストリング{:開始位置})	指定されたストリングを、変換元ストリングから変換先ストリングにもとづいて変換します。
	%YEARS(数値)	数値を年単位の期間に変換します。

表 6. V4R4 以降の新しい言語要素 (続き)

言語単位	要素	説明
命令コード	MONITOR	条件付きエラー処理を持つ命令のグループを開始します。
	ON-ERROR	状況コードに基づいて、条件付きエラー処理を実行します。
	ENDMON	条件付きエラー処理を持つ命令のグループを終了します。
	ELSEIF	ELSE 命令コードに IF 命令コードを続けたものと同等です。
CRTBNDRPG キーワードおよび CRTRPGMOD キーワード	LICOPT(オプション)	ライセンス内部コード・オプションを指定します。

V4R4 での変更点

V4R2 以降 RPG IV に施された機能強化の主だったものとして、スレッド環境で ILE RPG モジュールを安全に実行するためのサポート、3 桁および 20 桁の符号付きおよび符号なしの新しい数字データ・タイプ、新しい汎用文字セット・バージョン 2 (UCS-2) データ・タイプおよび UCS-2 フィールドとグラフィックまたは単一バイト文字フィールド間の変換についてのサポートが挙げられます。

以下に、これらの強化機能について説明します。

- Domino™ あるいは Java™ のように、スレッド化されたアプリケーションから ILE RPG プロシージャーを呼び出すためのサポート。
  - 新しい制御仕様書キーワード THREAD(*SERIALIZE) は、マルチスレッド環境で実行できるようになったモジュールを識別するものです。モジュール内のプロシージャーへのアクセスは、順番に行われます。
- 新しい 1 バイトおよび 8 バイト数字データ・タイプのサポート。3I および 20I 符号付き整数。
  - これらの新しい数字データ・タイプにより、これまでより広範囲な整数値が使用できるようになるため、数字計算のパフォーマンスも向上し、64 ビット AS/400 RISC プロセッサーをフルに活用できます。
  - 新しい 3U タイプにより、単一バイト文字 (CHAR) の戻りタイプおよびパラメーターが値別に渡される ILE C プロシージャーとの通信がさらに容易になります。
  - 新しい INTPREC 制御仕様書キーワードにより、式に入っている整数および符号なし 2 進算術演算の中間値について 20 桁の精度を指定することができます。
  - 整数の除算と剰余演算をサポートする組み込み関数 %DIV および %REM が追加されました。
- 新しい汎用文字セット・バージョン 2 (UCS-2) つまり Unicode データ・タイプのサポート

  - UCS-2 (Unicode) 文字セットは、多くの作成言語について文字をエンコードすることができます。フィールドは、2 バイトの長さの文字をもつ文字フィールドです。
  - Unicode についてのサポートが追加されたことにより、多国間企業を対象に開発するアプリケーションが 1 つで済むため、コード・ページ変換を行う必要がありません。Unicode を使用すると、整合性を失うことなく、複数のスクリプトで文字を処理できます。
  - MOVEL 演算と、新しい %UCS2 および %GRAPH 組み込み関数を使用した、UCS-2 フィールドとグラフィック・フィールドまたは単一バイト文字フィールド間の変換についてのサポート。
  - EVAL、MOVE、MOVEL 演算と、新しい %UCS2 組み込み関数を使用した、別のコード化文字セット識別コード (CCSID) をもつ UCS-2 フィールドまたはグラフィック・フィールド間の変換についてのサポート。

このリリースについては、この他にも、次のような機能強化がなされました。

- OPTION 制御仕様書キーワードおよび作成コマンドの新規パラメーター
  - *SRCSTMT は、コンパイラー・リスト内のソース ID および SEU 順序番号からデバッグするためにステートメント番号を割り当てられるようにします。(ステートメント番号は、デバッガーがコンパイラー・リスト内でエラーを識別し、実行時エラーが発生したステートメントを特定するのに使用されます。)*NOSRCSTMT は、ステートメント番号がリストの行番号と関連付けられ、数字が順番に割り当てられるよう指定します。
  - *NODEBUGIO を使って、デバッグ・ビュー内の入出力仕様書について停止点を生成しないよう選択できるようになりました。このオプションを選択した場合、デバッガー内の READ ステートメントの STEP は、入力仕様書のステップを実行するのではなく、次の計算ステップに進むことを指示します。
- 次のような、INZ 定義仕様書キーワードの新しい特殊語
  - INZ(*EXTDFT)。外部記述データ構造サブフィールドを初期化するために、DDS でデフォルト値を使用できるようにします。
  - INZ(*USER) によって初期化された文字変数は、現行ユーザー・プロファイルの名前に初期設定されます。
- 新しい %XFOOT 組み込み関数。指定された配列式のすべての要素を合計します。
- 新しい EVALR 命令コード。式を評価して、その結果を固定長文字またはグラフィック結果に割り当てます。この割り当てにより、データは、結果内で右そろえされます。
- 新しい FOR 命令コード。対話式ループを実行して、初期値、増分値、および限界値について自由形式を使用できるようにします。
- 新しい LEAVESR 命令コード。これを使用すると、サブルーチン内のどのポイントでも終了することができます。
- OVERLAY(名前:*NEXT) キーワードの新しい *NEXT パラメーター。これを使用すると、あるサブフィールドが、次の使用可能位置で別のサブフィールドをオーバーレイするよう指示できます。
- SETLL 命令コードの新しい *START 値および *END 値は、ファイルの先頭または終端に位置付けます。

- 初期化演算および自由形式演算 (例えば、EVAL、IF など) で整数フィールドおよび符号なし整数フィールドをもつ 16 進リテラルが使用可能。
- 新しい制御仕様書キーワード OPENOPT{(*NOINZOFL | *INZOFL)}。ファイルがオープンされたときにオーバーフロー標識を *OFF にリセットするかどうかを指示します。
- テラバイト・スペース内、つまり、1 つの割り振りで 16 メガバイトを超える連続する記憶域を許すメモリー・モデルでのポインターの許容。

以下の表に、影響を受ける言語の部分別に、変更のあった言語要素と新しい言語要素を要約します。

表 7. V4R2 以降変更された言語要素

言語単位	要素	説明
制御仕様書キーワード	OPTION(*{NO}SRCSTMT)	*SRCSTMT は、デバッグのためにステートメント番号を生成する際にコンパイラーが SEU 順序番号およびソース ID を使用するよう要求できるようにします。*NOSRCSTMT の場合は、ステートメント番号はリストの行番号と関連付けられ、数字が順番に割り当てられます。
	OPTION(*{NO}DEBUGIO)	*{NO}DEBUGIO は、入出力仕様書について停止点を生成するかどうかを決定します。
定義仕様書キーワード	INZ(*EXTDFT)	外部記述データ構造サブフィールドが、DDS 内で指定されているデフォルト値に初期設定できるようになりました。
	INZ(*USER)	いずれの文字フィールドまたはサブフィールドも、現行ユーザー・プロファイルの名前に初期設定できます。
	OVERLAY(名前:*NEXT)	特殊値 *NEXT は、サブフィールドをオーバーレイされたフィールド内の次に使用可能な位置に配置することを指示します。
	OPTIONS(*NOPASS *OMIT *VARSIZE *STRING *RIGHTADJ)	関数プロトタイプ内の値または固定情報パラメーターに指定された新しい OPTIONS(*RIGHTADJ) は、パラメーターとして渡された文字、図形、または UCS-2 値は、プロシージャー呼び出しで渡される前に右寄せすることを示します。
定義仕様書 33 ～ 39 桁目 (終了位置 / 長さ)	データ・タイプ I および U について許されている 3 桁および 20 桁	1 バイトおよび 8 バイトの整数と符号なしデータをサポートするために、内部データ・タイプに使用できる値のリストに追加されました。

表 7. V4R2 以降変更された言語要素 (続き)

言語単位	要素	説明
内部データ・タイプ	C (UCS-2 固定長形式または可変長形式)	定義仕様書で使用可能な内部データ・タイプのリストに追加されました。UCS-2 (Unicode) 文字セットは、多くの作成言語について文字をエンコードすることができます。このフィールドは、文字の長さが 2 バイトの文字フィールドです。
データ形式	C (UCS-2 固定長形式または可変長形式)	UCS-2 形式が、プログラム記述ファイルの入力仕様書および出力仕様書で使用可能なデータ形式のリストに追加されました。
コマンド・パラメーター	OPTION	CRTBNDRPG および CRTRPGMOD コマンドの OPTION パラメーターに *NOSRCSTMT、*SRCSTMT、*NODEBUGIO、および *DEBUGIO が追加されました。

表 8. V4R2 以降の新規言語要素

言語単位	要素	説明
制御仕様書キーワード	CCSID(*GRAPH: *IGNORE | *SRC | 番号)	モジュールについてデフォルトのグラフィック CCSID を設定します。この設定値は、リテラル、コンパイル時刻データ、およびプログラムで記述された入出力フィールドならびに定義に使用されます。デフォルトは *IGNORE です。
	CCSID(*UCS2: 番号)	モジュールについてデフォルトの UCS-2 CCSID を設定します。この設定値は、リテラル、コンパイル時刻データ、およびプログラムで記述された入出力フィールドならびに定義に使用されます。デフォルトは 13488 です。
	INTPREC(10 | 20)	式の 2 進算術演算に整数の符号なし中間値の 10 進精度を指定します。デフォルトである INTPREC(10) は、10 桁の精度が使用されることを示します。
	OPENOPT{(*NOINZOFL | *INZOFL)}	ファイルがオープンされたときにオーバーフロー標識を *OFF にリセットするかどうかを指示します。
	THREAD(*SERIALIZE)	モジュールがマルチスレッド化環境で実行できるようにすることを指示します。モジュール内のプロシージャーへのアクセスは、順番に行われます。
定義仕様書キーワード	CCSID(番号 | *DFT)	定義についてグラフィックおよび UCS-2 CCSID を設定します。

表 8. V4R2 以降の新規言語要素 (続き)

言語単位	要素	説明
組み込み関数	%DIV(n:m)	2 つのオペランド n と m で整数の割り算を実行します。その結果は、n/m の整数部分になります。これらのオペランドは、小数点以下の桁数がない (ゼロ) 数値でなければなりません。
	%GRAPH(文字式 | 図形式 | UCS2 式 {:ccsid})	単一バイト文字、グラフィック、または UCS-2 データからグラフィック・データに変換します。
	%REM(n:m)	2 つのオペランド n と m に対して整数剰余命令を実行しますが、この結果は n/m の剰余です。オペランドは、小数点以下の桁数がゼロの数値でなければなりません。
	%UCS2(文字式 | 図形式 | UCS2 式 {:ccsid})	単一バイト文字、グラフィック、または UCS-2 データから UCS-2 データに変換します。
	%XFOOT(配列式)	指定された数値配列式内の全要素の合計を求めます。
命令コード	EVALR	形式 result=expression の割り当てステートメントを評価します。結果は、右そろえされます。
	FOR	1 つの命令グループを開始し、そのグループが処理される回数を指示します。初期値、増分値、および限界値は、自由形式の式でかまいません。
	ENDFOR	ENDFOR は、FOR 命令で開始された命令グループを終了します。
	LEAVESR	これを使用すると、サブルーチン内のどこからでも終了することができます。

V4R2 での変更点

V3R7 以降の RPG IV の主な拡張機能は、可変長フィールドのサポート、標識に関係するいくつかの拡張機能、および制御仕様書にコンパイル・オプションを指定する機能です。これらの拡張機能によって、OS/400 オペレーティング・システムおよび ILE 言語間通信との統合について、RPG 製品がさらに改善されました。

以下に、これらの強化機能について説明します。

- 可変長フィールドのサポート

  この拡張機能によって、可変長の文字フィールドと図形フィールドが完全にサポートされます。可変長フィールドを使用すれば、多くのストリング処理タスクを単純化することができます。
- INDARA 標識にユーザー固有のデータ構造を使用するための機能

値をデータ管理機能に伝えるための *IN 配列を使用しなくても、ユーザーは論理データ域にアクセスして、INDARA を使用する各 WORKSTN および PRINTER ファイルに標識データ構造を関連付けることができるようになりました。

- 結果の標識の代わりに組み込み関数を使用する機能

  入出力操作の結果を照会するための組み込み関数
  %EOF、%EQUAL、%FOUND、および %OPEN が追加されました。エラー処理のための組み込み関数 %ERROR と %STATUS、および命令コード拡張 'E' が追加されました。
- 制御仕様書でのコンパイル・オプション

  コンパイル・オプションは、CRTBNDRPG および CRTRPGMOD コマンドで指定していましたが、制御仕様書キーワードで指定できるようになりました。これらのコンパイル・オプションは、プログラムのコンパイルごとに使用されます。

さらに、以下の新しい機能が追加されました。

- 大文字と小文字が混合した名前を持つプロシージャーおよび変数のインポートおよびエクスポートのサポート
- DECEDIT 値を実行時に動的に設定する機能
- ストリング処理を容易にするための組み込み関数 %CHAR および %REPLACE
- 外部定義 *CMDY、*CDMY、および *LONGJUL 日付データ形式の新しいサポート
- 世紀日付形式の範囲の拡張
- 標識変数を定義する機能
- OVERLAY キーワードのパラメーターとして現行データ構造名を指定する機能
- 可変長のフィールド・エラーを示すための新しい状況コード 115
- アプリケーション・プロファイル作成のサポート
- FIXNBR(*INPUTPACKED) を使用して、ファイルからの検索時に無効なパック 10 進数を処理する機能
- CRTRPGMOD コマンドに BNDDIR コマンド・パラメーターを指定する機能

以下の表に、影響を受ける言語の部分別に、変更のあった言語要素と新しい言語要素を要約します。

表 9. V3R7 以降に変更された言語要素

言語単位	要素	説明
制御仕様書キーワード	DECEDIT(*JOBRUN | '値')	10 進数の編集値を、ジョブ値またはシステム値から、実行時に動的に決定できるようになりました。
定義仕様書キーワード	DTAARA {(データ域名)}	ユーザーが論理データ域にアクセスできるようになりました。
	EXPORT {(外部名)}	エクスポートする変数の外部名を、このキーワードのパラメーターとして指定できるようになりました。
	IMPORT {(外部名)}	インポートする変数の外部名を、このキーワードのパラメーターとして指定できるようになりました。

表 9. V3R7 以降に変更された言語要素 (続き)

言語単位	要素	説明
	OVERLAY(名前{:位置})	名前パラメーターとして現行データ構造の名前を指定できるようになりました。
世紀の拡張形式	*CYMD (cyy/mm/dd)	世紀の文字 'c' の有効値は次のようになりました。 'c' Years 0 1900-1999 1 2000-2099 . . . . . . 9 2800-2899
内部データ・タイプ	N (標識形式)	定義仕様書で使用可能な内部データ・タイプのリストに追加されました。標識形式に文字データを定義します。
データ形式	N (標識形式)	標識形式が、プログラム記述ファイルの入力仕様書および出力仕様書で使用可能なデータ形式のリストに追加されました。
データ属性	*VAR	プログラム記述ファイルの入力仕様書および出力仕様書で使用可能なデータ属性のリストに追加されました。この属性は、可変長フィールドを指定するために、使用します。
コマンド・パラメーター	FIXNBR	*INPUTPACKED パラメーターが、無効なパック 10 進数データを処理するために追加されました。

表 10. V3R7 以降の新しい言語要素

言語単位	新規	説明
制御仕様書キーワード	ACTGRP(*NEW | *CALLER | '活動化グループ名')	ACTGRP キーワードによって、プログラムが呼び出された時に関連付ける活動化グループを指定することができます。
	ALWNULL(*NO | *INPUTONLY | *USRCTL)	ALWNULL キーワードは、外部記述データベース・ファイルからの、ヌル値可能フィールドを含んでいるレコードを使用する方法を指定します。
	AUT(*LIBRCRTAUT | *ALL | *CHANGE | *USE | *EXCLUDE | '権限認可リスト名')	AUT キーワードは、オブジェクトに対する特定の権限を持っていないユーザー、権限認可リストに載っていないユーザー、およびそのユーザー・グループがオブジェクトに対する特定の権限を持っていないユーザーに与える権限を指定します。

表 10. V3R7 以降の新しい言語要素 (続き)

言語単位	新規	説明
	BNDDIR('バインディング・ディレクトリー名'{:'バインディング・ディレクトリー名'...})	BNDDIR キーワードは、記号の解決で使用するバインディング・ディレクトリーのリストを指定します。
	CVTOPT(*{NO}DATETIME *{NO}GRAPHIC *{NO}VARCHAR *{NO}VARGRAPHIC)	CVTOPT キーワードは、ILE RPG コンパイラーが、外部記述データベース・ファイルから取り出した日付、時刻、タイム・スタンプ、図形データ・タイプ、および可変長データ・タイプを処理する方法を指定するために使用します。
	DFTACTGRP(*YES | *NO)	DFTACTGRP キーワードは、作成済みのプログラムが、呼び出されたときに実行する場所である活動化グループを指定します。
	ENBPFRCOL(*PEP | *ENTRYEXIT | *FULL)	ENBPFRCOL キーワードはパフォーマンス・コレクションを使用可能にするかどうかを指定します。
	FIXNBR(*{NO}ZONED *{NO}INPUTPACKED)	FIXNBR キーワードは、無効な 10 進数データをコンパイラーが修正するかどうかを指定します。
	GENLVL(番号)	GENLVL キーワードはオブジェクトの作成を制御します。
	INDENT(*NONE | '文字値')	INDENT キーワードは、構造化命令を読みやすくするために、ソース・リスト内で字下げするかどうかを指定します。
	LANGID(*JOBRUN | *JOB | '言語識別子')	LANGID キーワードは、分類順序が *LANGIDUNQ または *LANGIDSHR である場合に使用する言語識別子を指定します。
	OPTIMIZE(*NONE | *BASIC | *FULL)	OPTIMIZE キーワードは、もしあれば、オブジェクトの最適化のレベルを指定します。
	OPTION(*{NO}XREF *{NO}GEN *{NO}SECLVL *{NO}SHOWCPY *{NO}EXPDDS *{NO}EXT *{NO}SHOWSKP)	OPTION キーワードは、ソース・メンバーのコンパイル時に使用するオプションを指定します。
	PRFDTA(*NOCOL | *COL)	PRFDTA キーワードは、プロファイル作成データのコレクションを使用可能にするかどうかを指定します。
	SRTSEQ(*HEX | *JOB | *JOBRUN | *LANGIDUNQ | *LANGIDSHR | '分類テーブル名')	SRTSEQ キーワードは、ILE RPG ソース・プログラムで使用する分類順序テーブルを指定します。

表 10. V3R7 以降の新しい言語要素 (続き)

言語単位	新規	説明
	TEXT(*SRCMBRTXT | *BLANK | '記述')	TEXT キーワードによって、オブジェクトおよびその機能について簡単に記述するテキストを入力することができます。
	TRUNCNBR(*YES | *NO)	TRUNCNBR キーワードは、オブジェクトの実行時に数値オーバーフローが発生した場合に、切り捨てた値を結果フィールドに転送するか、あるいは、エラーを生成するかどうかを指定します。
	USRPRF (*USER | *OWNER)	USRPRF キーワードは、作成済みのプログラム・オブジェクトを実行するユーザー・プロファイルを指定します。
ファイル仕様書のキーワード	INDDS(データ構造名)	INDDS キーワードによって、データ構造名をワークステーションまたはプリンター・ファイルの INDARA 標識に関連付けることができます。
定義仕様書キーワード	VARYING	文字データまたは図形データに指定された場合に、可変長フィールドを定義します。
組み込み関数	%CHAR(図形、日付、時刻またはタイム・スタンプの式)	文字データ・タイプの値を戻します。
	%EOF{ファイル名}	最後に実行されたファイル入力操作またはサブファイルへの書き出し (指定された場合は特定のファイルに対する) が、ファイルの終わり条件またはファイルの先頭条件で終了した場合に '1' を戻します。他の場合には '0' を戻します。
	%EQUAL{ファイル名}	最後に実行された SETLL (指定された場合は特定のファイルに対する) または LOOKUP 命令が等しい項目を見付けた場合に '1' を戻します。他の場合には '0' を戻します。
	%ERROR	最後に実行された、拡張 'E' が指定された命令の結果がエラーである場合に '1' を戻します。他の場合には '0' を戻します。
	%FOUND{ファイル名}	最後に実行された関係のある命令 (指定された場合は特定のファイルに対する) がレコード (CHAIN、DELETE、SETGT、SETLL)、要素 (LOOKUP)、または等しい項目 (CHECK、CHECKR、および SCAN) を見付けた場合に '1' を戻します。他の場合には '0' を戻します。

表 10. V3R7 以降の新しい言語要素 (続き)

言語単位	新規	説明
	%OPEN(ファイル名)	指定されたファイルがオープンされている場合に '1' を戻します。指定されたファイルがクローズされている場合には '0' を戻します。
	%REPLACE(置換ストリング: ソース・ストリング {:開始位置 {:置換するソースの長さ }})	**開始位置**から開始し、指定された文字数を置換して、**置換ストリング**を**ソース・ストリング**に挿入することによって生成されるストリングを戻します。
	%STATUS{ファイル名}	最後に実行された、拡張 'E' が指定された命令コード以降、プログラム・エラーまたはファイル・エラーが発生していない場合に '0' を戻します。エラーが発生した場合には、プログラム状況またはファイル状況について最後に設定された値を戻します。ファイルが指定されている場合、戻される値は、そのファイルに関する最新の状況になります。
命令コード拡張	E	CALLP 命令で %ERROR および %STATUS 組み込み関数を使用したエラー処理を可能にし、すべての命令でエラー標識を使用できるようにします。
新しい世紀形式	*CMDY (cmm/dd/yy)	MOVE、MOVEL、および TEST 命令で使用できるようになりました。
	*CDMY(cdd/mm/yy)	MOVE、MOVEL、および TEST 命令で使用できるようになりました。
新しい 4 桁の年形式	*LONGJUL (yyyy/ddd)	MOVE、MOVEL、および TEST 命令で使用できるようになりました。
コマンド・パラメーター	PRFDTA	PRFDTA パラメーターは、プロファイル作成データのコレクションを使用可能にするかどうかを指定します。
	BNDDIR	BNDDIR パラメーターは CRTBNDRPG コマンドでしか使用できず、CRTRPGMOD コマンドでは使用できませんでしたが、両方のコマンドで使用できるようになりました。

V3R7 での変更点

V3R6 以降の RPG IV の主な拡張機能は、データベース・ヌル値フィールドに対する新規サポートと、式の中間結果の精度をさらに厳密に制御する機能です。それ以外の拡張機能としては、浮動小数点データ・タイプの追加や、null 文字で終了するストリングのサポートなどが含まれます。これらの拡張機能によって、OS/400 オペ

レーティング・システムおよび ILE 言語間通信との統合について、RPG 製品がさらに改善されました。これは、アプリケーションにおける柔軟性が大きく向上するということです。

以下は、これらの拡張機能 (いくつかの新しい組み込み関数と使用可能度の強化も含む) をリストしたものです。

- データベース・ヌル値フィールドのサポート

  この拡張機能によって、ヌル値可能フィールドにヌル値があるかどうかをテストし、そのフィールドをヌル値に設定できるようにすることにより、ユーザーが、ヌル値可能フィールドを含むデータベース・ファイルを処理できるようになります。

- 式中間結果の精度

  フリー・フォームで表現される仕様書における新しい制御仕様書キーワードおよび新しい命令コード・拡張によって、ユーザーが、中間結果の精度をより厳密に制御することができるようになります。

- 新しい浮動小数点データ・タイプ

  新しい浮動小数点データ・タイプには、他のデータ・タイプより広い範囲の値があります。このデータ・タイプが追加されたことで、OS/400 データベースの統合が進み、ILE 環境内における言語間通信、特に C および C++ 言語との通信が向上されます。

- NULL 文字で終了するストリングのサポート

  NULL 文字で終了するストリングを新しくサポートすることにより、言語間通信が向上しました。このサポートにより、ユーザーは、null 文字で終了するストリングを定義、処理し、さらに null 文字で終了するストリングを待っているプロシージャーにパラメーターとして文字データを簡単に渡すことができるようになり、ひいては null 文字で終了するストリングを完全に制御できるようになりました。

- ポインターの加算および減算

  フリー・フォームの式の機能が強化され、ポインターにオフセットを加えたり、ポインターからオフセットを引いたり、また、2 つのポインター間の差を求めることができるようになりました。

- 長名のサポート

  10 文字よりも長い名前が、RPG 言語に追加されました。定義仕様書またはプロシージャー仕様書で定義されたものには長名を付けることができ、これらの名前は、記入項目の境界内に収まる位置であればどこでも使用することができます。さらに、フリー・フォームの仕様書で参照される名前は、複数の行に連続する場合があります。

- 新しい組み込み関数

  この言語に新しい組み込み関数がいくつか追加され、それにより、以下の言語機能が向上しました。

  – 編集 (%EDITW、%EDITC、%EDITFLT)
  – 走査ストリング (%SCAN)
  – タイプ変換 (%INT、%FLOAT、%DEC、%UNS)
  – 四捨五入によるタイプ変換 (%INTH、%DECH、%UNSH)

  - 10 進数の式の場合の中間結果の精度 (%DEC)
  - 変数および式の 10 進数の長さ (%LEN, %DECPOS)
  - 絶対値 (%ABS)
  - ヌル値可能フィールドの設定およびテスト (%NULLIND)
  - NULL 文字で終了するストリングの取り扱い (%STR)
- 条件付きコンパイル

  RPG IV は、条件付きコンパイルをサポートするように拡張されています。このサポートには以下の内容が含まれます。
  - 条件の定義 (/DEFINE、/UNDEFINE)
  - 条件のテスト (/IF、/ELSEIF、/ELSE、/ENDIF)
  - 現行ソース・ファイルの読み取りの停止 (/EOF)
  - CRTBNDRPG および CRTRPGMOD コマンドで最高 32 までの条件を定義するための新しいコマンド・オプション (DEFINE)
- データ拡張

  データ処理命令を向上するために、いくつかの拡張が行われました。TIME 命令コードは、結果フィールド内の日付、時刻またはタイム・スタンプの各フィールドをサポートするように拡張されています。文字フィールドとの間で日付または時刻を転送する場合には、区切り記号が必要です。UDATE および *DATE フィールドの転送には、形式コードを指定する必要はなくなりました。日付フィールドは、定義仕様書のシステム (*SYS) またはジョブ (*JOB) の日付に初期設定することができます。
- 代替照合順序による文字比較

  特定の文字変数を定義して、比較において代替照合順序が使用されないようにすることができます。
- ネストされた /COPY メンバー

  /COPY 指示のネストが可能になりました。すなわち、/COPY メンバーに 1 つ (または複数) の /COPY 指示を含めることができます。その指示には、さらに別の /COPY 指示などを含めることができます。
- 記憶域管理

  新しい記憶管理命令コードを使用して、記憶域の割り振り、再割り振り、および割り振り解除が動的に行えるようになりました。
- 記憶管理および浮動アンダーフローのエラーの状況コード

  記憶管理エラーを示すための状況コードが 2 つ (425 および 426) 追加されました。また、中間浮動結果が小さ過ぎることを示すための状況コード 104 も追加されています。

以下の表に、影響を受ける言語の部分別に、変更のあった言語要素と新しい言語要素を要約します。

表 11. V3R6 以降に変更された言語要素

言語単位	要素	説明
定義仕様書キーワード	ALIGN	以前にサポートされている整数および符号なしの位置合わせに加え、浮動サブフィールドの位置合わせをするために、ALIGN が使用できるようになりました。
	OPTIONS(*NOPASS *OMIT *VARSIZE *STRING)	*STRING オプションによって、NULL 文字で終了するストリングとして文字値を渡すことができます。
レコード・アドレス・タイプ	F (浮動形式)	ファイル仕様書で使用可能なレコード・アドレス・タイプのリストに追加されました。プログラム記述ファイルについての浮動処理をシグナルします。
内部データ・タイプ	F (浮動形式)	定義仕様書で使用可能な内部データ・タイプのリストに追加されました。浮動小数点独立フィールド、パラメーター、またはデータ構造サブフィールドを定義します。
データ形式	F (浮動形式)	プログラム記述ファイルの入力仕様書および出力仕様書で使用可能なデータ形式のリストに追加されました。

表 12. V3R6 以降の新しい言語要素

言語単位	新規	説明
制御仕様書キーワード	COPYNEST('1 〜 2048')	/COPY 指示のネストのための最大の深さを指定します。
	EXPROPTS(*MAXDIGITS | *RESDECPOS)	精度のタイプの式オプション (デフォルト値または「結果の小数点以下の桁数」精度規則)
	FLTDIV{(*NO | *YES)}	式の中のすべての除算命令が浮動小数点で計算されることを示します。
定義仕様書キーワード	ALTSEQ(*NONE)	代替照合順序が指定されている場合でも、文字比較に通常の照合順序を使用することを強制します。
組み込み関数	%ABS	パラメーターとして指定されている数値式の絶対値を戻します。
	%DEC & %DECH	数値式の値を、パラメーターとして指定されている桁数および小数点以下の桁数を持つ 10 進 (パック) 形式に変換します。%DECH は %DEC と同じですが、四捨五入が適用されます。
	%DECPOS	数値変数または数値式の小数点以下の桁数を戻します。戻される値は定数で、定数が予期されているところに使用されます。

表 12. V3R6 以降の新しい言語要素 (続き)

言語単位	新規	説明
	%EDITC	この関数は、編集コードに従って編集された数値を表す文字結果を戻します。
	%EDITFLT	数値式の値を、浮動の文字外部表示表現に変換します。
	%EDITW	この関数は、編集ワードに従って編集された数値を表す文字結果を戻します。
	%FLOAT	数値式の値を、浮動形式に変換します。
	%INT & %INTH	数値式の値を、整数に変換します。10 進数はすべて %INT によって切り捨てられ、%INTH によって丸められます。
	%LEN	変数式の数字または文字の数を戻します。
	%NULLIND	ヌル値可能フィールドのヌル標識を照会または設定するために使用されます。
	%SCAN	ソース・ストリングの中の検索引き数の 1 桁目、またはそれが見付からない場合には 0 を戻します。
	%STR	NULL 文字で終了するストリングを作成または使用するために使用します。このストリングは、C および C++ アプリケーションで非常に一般的に使用されています。
	%UNS & %UNSH	数値式の値を符号なし形式に変換します。10 進数はすべて %UNS によって切り捨てられ、%UNSH によって丸められます。
命令コード拡張	N	DEALLOC が正常に行われた後、ポイントを *NULL に設定します。
	M	デフォルトの精度規則
	R	小数点以下の桁数が、結果の小数点以下の桁数より少ない中間値はなくなります (″結果の小数点以下の桁数″ 精度の規則)。
命令コード	ALLOC	記憶域を動的に割り振るために使用します。
	DEALLOC	記憶域を動的に割り振り解除するために使用します。
	REALLOC	記憶域を動的に再割り振りするために使用します。

V3R6/V3R2 での変更点

V3R1 以降の RPG IV の主な拡張機能は、1 つのモジュールを複数のプロシージャーを使用してコーディングする機能です。これは何を意味するでしょう? ナット・シェルでは、1 つのモジュールを 1 つ以上のプロトタイプ・プロシージャーを使用してコーディングできることを意味します。この場合、プロシージャーは RPG サイクルを使用しないで戻り値を持ち、実行することができます。

1 つのモジュールを複数のプロシージャーを使用して作成すると、作成するアプリケーションの機能を高めることができます。すべてのアプリケーションは、特定のタスクを実行すると考えられる一連の論理単位から構成されます。柔軟性を保持してアプリケーションを開発するには、各論理単位をできるだけ独立させることが重要です。各単位を独立させることによって、以下のことが可能になります。

- 特定のタスクを実行する視点から各単位を作成しやすくなります。
- 変更できるように設計したデータ・オブジェクト以外のデータ・オブジェクトを変更する可能性が少なくなります。
- 論理およびデータ項目をより個別化できるので、デバッグが容易になります。
- 変更が必要なアプリケーションの部分を容易に分離できるので、保守が容易になります。

複数のプロシージャーを使用して 1 つのモジュールをコーディングする主な利点は、モジュラー・アプリケーションのコーディングの制御が容易になり、効率が上がることです。この利点はいくつかの方法で実現することができます。以下を行うことができます。

- 同じ呼び出し命令および構文を使用してプロシージャーおよびプログラムを呼び出す。
- 呼び出しインターフェースのコンパイル時に検査を行うプロトタイプを定義する。
- 値によって、または参照によってパラメーターを渡す。
- 値を戻すプロシージャーを定義し、式の中でそのプロシージャーを呼び出す。
- 変数のローカル定義を実行することによって、データ項目へのアクセスを制限する。
- サイクルを使用しないモジュールをコーディングする。
- プロシージャーを回帰的に呼び出す。

モジュール内のメイン・プロシージャーの実行時の動作は、V3R1 プロシージャーの場合と同じです。それ以降のプロシージャーの実行時の動作は、V3R1 プログラムとはいくらか異なります。プロシージャーの最後および例外処理の領域で特に異なります。これらの相違点は、これらのプロシージャーについてはサイクル・コードが生成されないために生じます。

他の拡張は、このリリースについても行われています。次のような機能強化がなされました。

- 2 つの新しい整数データ・タイプ、符号付き整数 (I) と符号なし整数 (U) のサポート

整数データ・タイプによって、2 進データ・タイプより広い範囲の値を使用することができます。整数データ・タイプは、整数計算のパフォーマンスを向上することもできます。

- MOVE、MOVEL、および TEST 命令の *CYMD サポート

  既にこのデータ形式であるシステム値を処理するために、特定の命令の中で *CYMD 日付形式を使用できるようになりました。
- 制御仕様書の COPYRIGHT キーワードを使用して、プログラムおよびモジュールの版権を示す機能

  このキーワードを使用して指定した版権情報は、DSPMOD、DSPPGM、または DSPSRVPGM 情報の一部になります。
- キーワード BLOCK を使用するレコードのブロック化のユーザー制御

  DISK または SEQ ファイルに SETLL、SETGT、または CHAIN 命令が使用されている場合でも、ファイルのレコードのブロック化を要求することができます。ブロック化を実行しないように要求することもできます。このような場合でのブロック化の使用によって、実行時のパフォーマンスがかなり向上する可能性があります。
- PREFIX 機能の改善

  ファイル記述および定義仕様書に関する PREFIX キーワードへの変更によって、既存のフィールド名内の文字を接頭部ストリングで置換できるようになりました。
- トリガー・プログラム・エラーに関する状況コード

  トリガー・プログラム・エラーを示すために 2 つの状況コード 1223 および 1224 が追加されました。

以下の表に、影響を受ける言語の部分別に、変更のあった言語要素と新しい言語要素を要約します。

表 13. V3R1 以降に変更された言語要素

言語単位	要素	説明
ファイル仕様書のキーワード	PREFIX(接頭部ストリング {: 置換する文字数})	フィールド名に対するストリングの接頭部を付けること、またはフィールド名の部分的な変更を可能にします。
定義仕様書キーワード	CONST{(定数)}	名前付き定数の値を指定するか、または参照によって渡されるプロトタイプ・パラメーターが定数値を持っていることを示します。
	PREFIX(接頭部ストリング {: 置換する文字数})	フィールド名に対するストリングの接頭部を付けること、またはフィールド名の部分的な変更を可能にします。
命令コード	RETURN	呼び出し元に制御を戻し、指定されている場合は、値を戻します。

表 14. V3R1 以降の新しい言語要素

言語単位	新規	説明
制御仕様書キーワード	COPYRIGHT('版権ストリング')	版権情報をモジュールおよびプログラムに関連付けることができるようにします。
	EXTBININT{(*NO | *YES)}	プログラム処理中に、外部記述ファイル内の 2 進数フィールドに整数形式を割り当てていることを指定します。
	NOMAIN	モジュールにサブプロシージャーのみがあることを示します。
ファイル仕様書のキーワード	BLOCK(*YES |*NO)	レコードのブロック化の発生を制御できるようにします (他の条件が満たされた場合)
定義仕様書キーワード	ALIGN	整数フィールドまたは符号なしフィールドの位置合わせを行うかどうかを指定します。
	EXTPGM(名前)	プロトタイプ・プログラムの外部名を示します。
	EXTPROC(名前)	プロトタイプ・プロシージャーの外部名を示します。
	OPDESC	プロトタイプ・バインド呼び出しで、操作記述子を渡すかどうかを指定します。
	OPTIONS(*NOPASS *OMIT *VARSIZE)	プロトタイプ・パラメーターに関する各種のオプションを指定します。
	STATIC	ローカル変数が静的記憶域を使用するように指定します。
	VALUE	プロトタイプ・パラメーターを値によって渡すように指定します。
組み込み関数	%PARMS	呼び出しで渡すパラメーターの個数を戻します。
命令コード	CALLP	プロトタイプ・プログラムまたはプロシージャーを呼び出します。
仕様書タイプ	プロシージャー仕様書	サブプロシージャー定義の先頭と終端を示します。
定義のタイプ	PR	プロトタイプ定義の先頭を示します。
	PI	プロシージャー・インターフェース定義の先頭を示します。
	24〜25 桁目のブランク	プロトタイプ・パラメーターを定義します。

第 1 部 ILE RPG の概要

ILE RPG を使用してプログラムを作成する場合、前もってその使用環境について知っておく必要があります。第 1 部では、知っておくべき下記のトピックについて言及します。

- RPG IV 言語の概要
- RPG プログラミングにおける各 ILE 構成要素の役割
- ILE プログラムの作成方針
- 複数のプロシージャーおよびプロトタイプ呼び出しのあるモジュールのコーディングの概要

第 1 章 RPG IV プログラミング言語の概要

この章では、他のプログラミング言語と異なる RPG IV プログラミング言語の機能に関する概要を説明します。RPG IV 言語でプログラミングするためには、前もってこれらすべての機能についてなれ親しんでおく必要があります。ここで説明する機能は次のとおりです。

- コーディング仕様書
- プログラム・サイクル
- 標識
- 命令コード

RPG IV の詳細については、「*WebSphere Development Studio: ILE RPG 解説書*」を参照してください。

RPG IV 仕様書

RPG のコーディングは、それぞれが特定の機能セットをもつ、各種の仕様書上で行われます。仕様書タイプを構成する項目の多くは、位置に依存しています。各項目は、項目のタイプおよび仕様書タイプによって、特定の桁で開始しなければなりません。

RPG IV 仕様書には 7 つのタイプがあります。各仕様書タイプはオプションです。各仕様書は、以下に示される順番でソース・プログラムに入力されなければなりません。

メイン・ソース・セクション

1. **制御仕様書**は、プログラムの生成および実行についての情報をコンパイラーに提供します。その中には、プログラム名、日付の形式、および代替照合順序またはファイル変換の使用などが含まれます。
2. **ファイル仕様書**には、プログラムで使用するすべてのファイルを記述します。
3. **定義仕様書**には、プログラムで使用されるデータを記述します。
4. **入力仕様書**には、プログラムで使用される入力レコードおよびフィールドについて記述します。
5. **演算仕様書**には、データに対して実行される演算および演算の順序を記述します。演算仕様書では、特定の入力および出力操作を制御することもできます。
6. **出力仕様書**には、プログラムで使用される出力レコードおよびフィールドを記述します。

サブプロシージャー・セクション

1. **プロシージャー仕様書**は、サブプロシージャーの始めと終わりにマークを付け、サブプロシージャー名およびエクスポートするかどうかを指示します。
2. **定義仕様書**には、サブプロシージャーで使用される内部データを記述します。
3. **演算仕様書**には、グローバル・データおよび内部データの両方に対して実行される演算および演算の順序を記述します。

サイクル・プログラミング

システムがデータを処理する際、システムは特定の順序で処理を行わなければなりません。この論理順序は次のものによって与えられます。

- ILE RPG コンパイラー
- プログラム・コーディング

コンパイラーによって提供される論理は、**プログラム・サイクル**と呼ばれます。ユーザーが、コンパイラーにプログラムの論理を提供させる時には、**サイクル・プログラミング**と呼ばれます。

プログラム・サイクルとは、ファイル終了状態に達するまで、プログラムが反復する一連のステップのことです。コーディングした仕様書によって、プログラムでサイクル内の各ステップを使用することもあればしないこともあります。

ファイルをサイクルで制御したい場合には、ソース・プログラム中の RPG 仕様書でコーディングした情報は、これらのファイルのレコードが読み取られる時に指定する必要がありません。ソース・プログラムのコンパイル時に、コンパイラーによって、これらの操作および出力操作の論理順序が設定されます。

ファイルをサイクルで制御させたくない場合には、その他の方法でプログラムを終了する必要があります。最終レコード (LR) 標識をオンに設定することによってファイル終了状態を作成するか、戻り (RT) 標識をオンに設定することによって戻り状態を作成するか、または RETURN 命令を使用して直接戻る方法があります。

注: 制御仕様書に NOMAIN が指定されている時には、サブプロシージャー用にサイクル・コードは生成されません。

図 1 は、RPG プログラム・サイクルの一般的流れに特有のステップを示したものです。

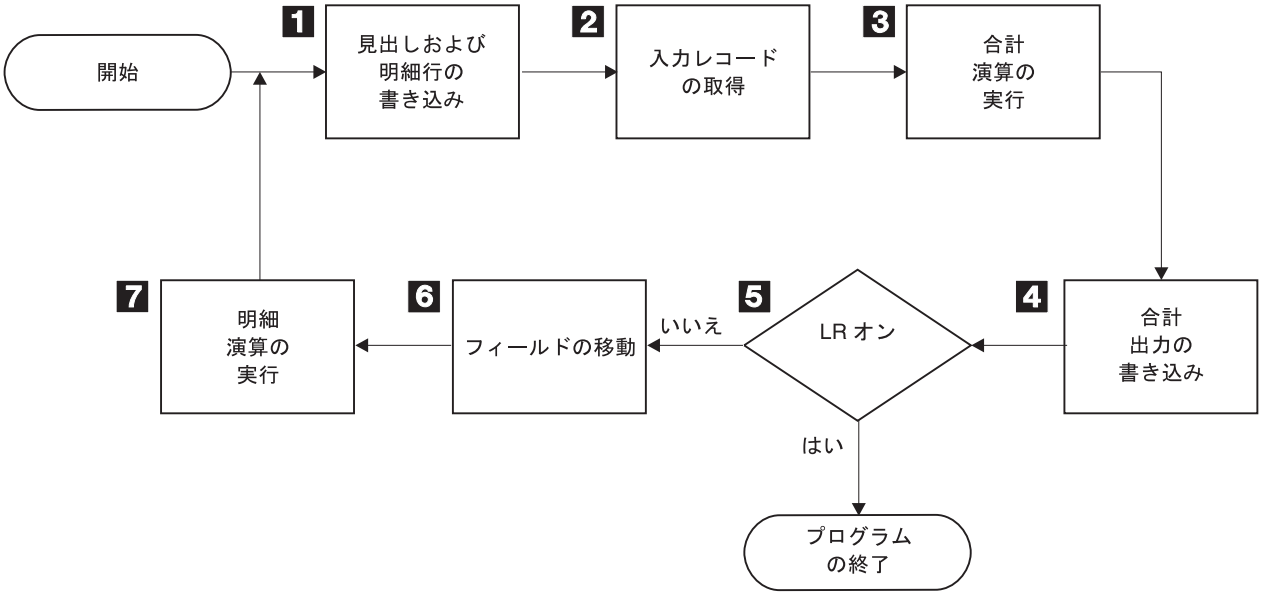


図 1. RPG プログラムの論理サイクル

1 RPG は、すべての見出し行および明細行 (出力仕様書の 17 桁目の H または D) を処理します。

2 RPG は次のレコードを読み取り、レコード識別標識および制御レベル標識をオンに設定します。

3 RPG は合計演算 (制御レベル標識 L1 ～ L9、LR 標識あるいは L0 項目によって条件付けされる) を処理します。

4 RPG はすべての合計出力行 (出力仕様書の 17 桁目の T によって識別) を処理します。

5 RPG は LR 標識がオンかどうかを判定します。オンであれば、プログラムは終了します。

6 選択した入力レコードのフィールドがレコードから処理域に移されます。RPG はフィールド標識をオンに設定します。

7 RPG はすべての明細演算 (演算仕様書の 7 ～ 8 桁目の制御レベル標識では条件付けされていない) を処理します。RPG はサイクルの始めにあるレコードからのデータを使用します。

最初のサイクル

プログラム・サイクルを通過する場合、最初と最後はそれ以外のサイクルと多少異なります。サイクルの最初で最初のレコードを読み取る前に、プログラムは次の 3 つのことを実行します。

- 入力パラメーターの処理、ファイルのオープン、プログラム・データの初期設定
- 1P (1 ページ目) 標識で条件付けられたレコードの書き出し
- すべての見出しおよび明細出力命令の処理

例えば、最初のレコードを読み取る前に印刷される見出し行は、固定情報、ページ見出し情報、または PAGE および *DATE などの特殊なフィールドから構成されるようにすることができます。プログラムはまた、最初のサイクルでの合計演算および合計出力ステップをバイパスします。

最後のサイクル

レコードがなくなるとプログラムは最後のサイクルに進み、LR (最終レコード) 標識および L1 ～ L9 (制御レベル) 標識を**オン** に設定します。プログラムは、合計演算および合計出力を処理し、すべてのファイルをクローズしてから終了します。

サブプロシージャーの論理

サブプロシージャーの一般的なフローは、もっと単純です。サブプロシージャーの演算が 1 回実行されると、サブプロシージャーは戻ります。サブプロシージャー用にはサイクル・コードは作成されません。

標識

標識は、オン ('1') またはオフ ('0') のいずれかが入る 1 バイトの文字フィールドです。これは演算の結果を表示するため、または演算処理を条件付ける (制御する) ために使用されます。標識は、プログラムの論理フローにおけるスイッチに類似しています。標識がどのように設定され使用されるかによって、処理中にプログラムが取るパスが決定されます。

標識は定義仕様書の変数として定義することができます。また RPG IV 標識を使うこともできます。この標識は仕様書上の項目、または RPG IV プログラム自身のいずれかによって定義されます。

各 RPG IV 標識は 2 文字の名前 (例えば、LR、01、H3) を持ちます。これらの標識は 2 文字の名前だけで、任意の仕様書の任意の項目で参照されたり、あるいは特殊な名前 *INxx (xx は 2 文字の名前) により他で参照されます。ユーザーは複数のタイプの標識を使用することができ、各タイプはそれぞれ異なったことを示します。プログラマーが標識を定義する仕様書上の位置により、標識の使用法が決まります。プログラム中で標識を定義すると、演算および出力操作を制限または制御することができます。

標識変数は *INxx 形式の標識が使える任意の場所で使用することができますが、ファイル仕様書の OFLIND および EXTIND キーワードは除きます。

RPG プログラムは、プログラム・サイクル中の特定の時点である標識を設定したり、リセットしたりします。さらに、標識の状況を演算操作の中で明示的に変更することができます。

命令コード

RPG IV プログラミング言語によって、ユーザーのデータにさまざまなタイプの多くの命令を実行することができます。演算仕様書に指定する**命令コード**によって、実行する命令を指示します。例えば、新しいレコードを読み取りたい場合には、READ 命令コードを使用することができます。使用可能な命令のタイプのリストは次のとおりです。

- 算術演算
- 配列命令
- ビット命令
- 分岐命令
- 呼び出し命令
- 比較命令
- 変換命令
- データ域命令
- 日付命令
- 宣言命令
- エラー処理命令
- ファイル命令
- 標識設定命令
- 情報命令
- 初期化命令

- メモリー管理命令
- 移動命令
- ゾーン移動命令
- 結果命令
- サイズ変更命令
- ストリング命令
- 構造化プログラミング命令
- サブルーチン命令
- テスト命令

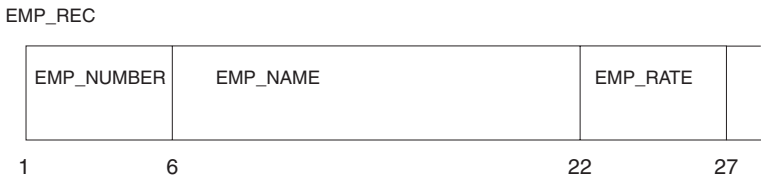
ILE RPG プログラムの例

この節では、給与計算を実行する簡単な ILE RPG プログラムを説明します。

問題文

ある小さい会社の給与計算部門では、その週の社員の給与をリストする印刷出力を作成したいとします。システム上には EMPLOYEE と TRANSACT の 2 つのディスク・ファイルがあるとします。

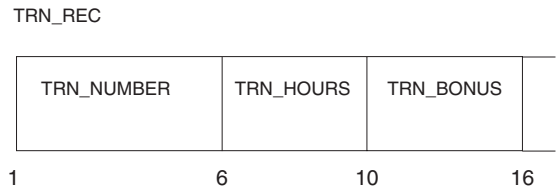
最初のファイル EMPLOYEE には社員のレコードが入っています。下の図は社員レコードの形式を示しています。




```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A..........T.Name++++++RLen++TDpB......Functions++++++++++++++++++++*
A          R EMP_REC
A            EMP_NUMBER     5          TEXT('EMPLOYEE NUMBER')
A            EMP_NAME      16          TEXT('EXPLOYEE NAME')
A            EMP_RATE       5  2       TEXT('EXPLOYEE RATE')
A          K EMP_NUMBER
```

図 2. 社員物理ファイルの DDS

2 番目のファイル TRANSACT には、各社員がその週に働いた時間数とその社員が受け取ったボーナスが記録されています。下の図はトランザクション・レコードの形式を示しています。



```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A.........T.Name++++++RLen++TDpB......Functions++++++++++++++++++++*
A          R TRN_REC
A            TRN_NUMBER     5          TEXT('EMPLOYEE NUMBER')
A            TRN_HOURS      4  1       TEXT('HOURS WORKED')
A            TRN_BONUS      6  2       TEXT('BONUS')
```

図 3. TRANSACT 物理ファイルの DDS

各社員の給与の計算は、「時間数」(TRANSACT ファイルからの) と「社員支給率」 (EMPLOYEE ファイルからの) とを掛けて、それに TRANSACT ファイルからの「ボーナス」を加えることによって行われます。40 時間を超えて働いた場合には、通常の率の 1.5 倍が社員に支払われます。

制御仕様書

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
H DATEDIT(*DMY/)
```

今日の日付は、日、月、年の形式で「/」を区切り記号として印刷されます。

ファイル仕様書

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+...
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++
FTRANSACT  IP   E           K DISK
FEMPLOYEE  IF   E           K DISK
FQSYSPRT   O    F   80        PRINTER
```

ファイル仕様書には次の 3 つのファイルが定義されています。

- TRANSACT ファイルは入力 1 次ファイルとして定義されています。ILE RPG プログラム・サイクルは、このファイルからのレコードの読み取りを制御します。
- EMPLOYEE ファイルは入力全手順ファイルとして定義されています。このファイルからのレコードの読み取りは、演算仕様書の命令によって制御されます。
- QSYSPRT ファイルは出力印刷ファイルとして定義されています。

定義仕様書

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+...
D+Name++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++
D Pay             S               8P 2
D Heading1        C                   'NUMBER  NAME              RATE    H-
D                                     OURS  BONUS    PAY       '
D Heading2        C                   '______  ________________  ______  _-
D                                     ____  _______  __________'
D CalcPay         PR              8P 2
D   Rate                          5P 2 VALUE
D   Hours                        10U 0 VALUE
D   Bonus                         5P 2 VALUE
```

定義仕様書を使用して、社員の週給を入れる ″Pay″ という変数と報告書の見出しの印刷に備えての ″Heading1″ と ″Heading2″ の 2 つの固定情報を宣言します。

演算仕様書

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+...
 /free
    chain trn_number emp_rec;
    if %found(emp_rec);
       pay = CalcPay (emp_rate: trn_hours: trn_bonus);
    endif;
 /end-free
```

演算仕様書のコーディング項目には、以下のものが含まれています。

- CHAIN 命令コードを使用することにより、社員ファイル中の同じ社員番号を持つレコードを見つけるために、トランザクション・ファイルからの TRN_NUMBER フィールドが使用されます。
- CHAIN 命令が正常であれば (すなわち、標識 99 がオフであれば)、その社員の給与が計算されます。結果は「四捨五入」されて、Pay という変数に保管されます。

出力仕様書

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+...
OFilename++DF..N01N02N03Excnam++++B++A++Sb+Sa+.........................
O.............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat
OQSYSPRT   H    1P                    2  3
O                                          35 'PAYROLL REGISTER'
O                        *DATE         Y     60
O          H    1P                     2
O                                          60 Heading1
O          H    1P                     2
O                                          60 Heading2
O          D   N1PN99                   2
O                        TRN_NUMBER           5
O                        EMP_NAME            24
O                        EMP_RATE      L     33
O                        TRN_HOURS     L     40
O                        TRN_BONUS     L     49
O                        Pay                 60 '$     0.  '
O          D   N1P 99                   2
O                        TRN_NUMBER           5
O                                          35 '** NOT ON EMPLOYEE FILE **'
O          T    LR
O                                          33 'END OF LISTING'
```

出力仕様書では、どのフィールドを QSYSPRT 出力に書き出すかを記述します。

- 明細情報用の見出しと同様に固定情報ストリング「PAYROLL REGISTER (給与計算表)」を含んでいる見出し行は、標識 1P がオンの場合に印刷されます。標識 1P は、最初のサイクル時に ILE RPG プログラム・サイクルによってオンにされます。
- 明細行は標識 1P および 99 によって条件付けられます。明細行は 1P 時には印刷されません。N99 は、標識 99 がオフである (対応する社員レコードが見付かったことを示す) 場合にだけ、明細行を印刷することができます。標識 99 がオンの場合には、明細行の代わりに社員番号と固定情報ストリング「** NOT ON EMPLOYEE FILE ** (** 社員ファイル上にない **)」が印刷されます。
- 合計行には、「END OF LISTING (リストの終わり)」という固定情報ストリングが含まれます。これは最後のプログラム・サイクル時に印刷されます。

サブプロシージャー

サブプロシージャーは、渡されたパラメーターを使用して社員の給与を計算します。結果の値は、RETURN ステートメントを使用して呼び出し元に戻されます。

プロシージャー仕様書はプロシージャーの始めと終わりを指示します。定義仕様書は、プロシージャーの戻りタイプ、プロシージャーへのパラメーター、およびローカル変数である Overtime を定義します。

```
P CalcPay         B
D CalcPay         PI             8P 2
D   Rate                         5P 2 VALUE
D   Hours                       10U 0 VALUE
D   Bonus                        5P 2 VALUE
D Overtime        S              5P 2 INZ(0)

  /free
    // 支払われる超過勤務時間を決定します。
    if Hours > 40;
      Overtime = (Hours - 40) * Rate * 1.5;
      Hours = 40;
    endif;
    // 給与合計を計算し、呼び出し元に戻します。
    return  Rate * Hours + Bonus + Overtime;
 /end-free
P CalcPay         E
```

ソース・プログラム全体

次の図は、このプログラムで使用されるすべての仕様書を組み合わせます。この図は、このプログラムのソース・ファイルに入力すべきものを示しています。

```
 *-------------------------------------------------------------------------*
 * 説明: このプログラムは、社員の週給の印刷出力を                          *
 *       作成します。                                                      *
 *-------------------------------------------------------------------------*
H DATEDIT(*DMY/)
 *-------------------------------------------------------------------------*
 * ファイル定義                                                            *
 *-------------------------------------------------------------------------*
FTRANSACT  IP   E           K DISK
FEMPLOYEE  IF   E           K DISK
FQSYSPRT   O    F   80        PRINTER
 *-------------------------------------------------------------------------*
 * 変数の宣言                                                              *
 *-------------------------------------------------------------------------*
D Pay             S              8P 2
```

図 4. 給与計算サンプル・プログラム (1/3)

```
 *----------------------------------------------------------------------*
 * 定数の宣言                                                           *
 *----------------------------------------------------------------------*
D Heading1        C                   'NUMBER  NAME              RATE    H-
D                                     OURS  BONUS    PAY       '
D Heading2        C                   '______  ________________  ______  _-
D                                     ____  _______  __________'
 *----------------------------------------------------------------------*
 * サブプロシージャー CalcPay のためのプロトタイプの定義                *
 *----------------------------------------------------------------------*
D CalcPay         PR             8P 2
D   Rate                          5P 2 VALUE
D   Hours                        10U 0 VALUE
D   Bonus                         5P 2 VALUE
 *----------------------------------------------------------------------*
 * トランザクション・ファイル (TRANSACT) 内の各レコードごとに、         *
 * 該当する社員が見つかった場合は給与を計算し明細を印刷します。         *
 *----------------------------------------------------------------------*
 /free
    chain trn_number emp_rec;
    if %found(emp_rec);
       pay = CalcPay (emp_rate: trn_hours: trn_bonus);
    endif;
 /end-free
 *----------------------------------------------------------------------*
 * 報告書レイアウト                                                     *
 *  -- 1P がオンであれば見出し行を印刷する                              *
 *  -- レコードが見つかった場合 (標識 99 がオフ) は                     *
 *     給与明細を印刷し、それ以外の場合は例外レコードを印刷する         *
 *  -- LR がオンのときは 'リストの終わり' と印刷する                    *
 *----------------------------------------------------------------------*
OQSYSPRT   H    1P                   2  3
O                                         35 'PAYROLL REGISTER'
O                        *DATE        Y     60
O          H    1P                   2
O                                         60 Heading1
O          H    1P                   2
O                                         60 Heading2
O          D   N1PN99                2
O                        TRN_NUMBER         5
O                        EMP_NAME          24
O                        EMP_RATE     L    33
O                        TRN_HOURS    L    40
O                        TRN_BONUS    L    49
O                        Pay               60 '$     0.  '
O          D   N1P 99                2
O                        TRN_NUMBER         5
O                                         35 '** NOT ON EMPLOYEE FILE **'
O          T    LR
O                                         33 'END OF LISTING'
```

図 4. 給与計算サンプル・プログラム (2/3)

```
 *---------------------------------------------------------------------*
 * サブプロシージャー  -- 超過時間給与を計算します。                    *
 *---------------------------------------------------------------------*
P CalcPay         B
D CalcPay         PI             8P 2
D   Rate                         5P 2 VALUE
D   Hours                       10U 0 VALUE
D   Bonus                        5P 2 VALUE
D Overtime        S              5P 2 INZ(0)

  /free
    // 支払われる超過勤務時間を決定します。
    if Hours > 40;
      Overtime = (Hours - 40) * Rate * 1.5;
      Hours = 40;
    endif;
    // 給与合計を計算し、呼び出し元に戻します。
    return  Rate * Hours + Bonus + Overtime;
 /end-free
P CalcPay         E
```

図 4. 給与計算サンプル・プログラム (3/3)

OS/400 システムの使用

iSeries システムとの対話のすべてを制御するオペレーティング・システムを OS/400 システムと呼びます。ワークステーションから OS/400 システムを使用して、以下のことができます。

- サインオンおよびサインオフ
- 画面との対話
- オンライン・ヘルプ情報の使用
- 制御コマンドおよびプロシージャーの入力
- メッセージへの応答
- ファイルの管理
- ユーティリティーおよびプログラムの実行

インターネットにアクセスできる場合は、以下の URL で OS/400 システムを説明する資料の全リストの入手が可能です。

`http://publib.boulder.ibm.com/`

「*V4R3 Library Poster*, G325-6334-02」を発注することもできます。

システムとの対話

コマンド言語 (CL) を使って OS/400 システムを操作することができます。CL コマンドを入力または選択することによって、システムと対話することができます。システムは、画面の状況に対応する一連の CL コマンド、またはコマンド・パラメーターを表示することがよくあります。表示後に所要のコマンドまたはパラメーターを選択します。

よく使用される制御言語コマンド

次の表は、最もよく使用される CL コマンドとその機能、および使用される理由を一覧表にしたものです。

表 15. よく使用される CL コマンド

処置	CL コマンド	結果
システム・メニューの使用	**GO MAIN**	メイン・メニューを表示
	GO INFO	ヘルプ・メニューを表示
	GO CMDRPG	RPG 用のコマンドをリスト
	GO CMDCRT	作成用のコマンドをリスト
	GO CMDxxx	xxx 用のコマンドをリスト
呼び出し	**CALL** *プログラム名*	プログラムを実行
コンパイル	**CRTxxxMOD**	xxx モジュールを作成
	CRTBNDxxx	バインド xxx プログラムを作成
バインド	**CRTPGM**	ILE モジュールからプログラムを作成
	CRTSRVPGM	サービス・プログラムを作成
	UPDPGM	バインド・プログラム・オブジェクトを更新
デバッグ	**STRDBG**	ILE ソース・デバッガーを開始
	ENDDBG	ILE ソース・デバッガーを終了
ファイルの作成	**CRTPRTF**	印刷ファイルを作成
	CRTPF	物理ファイルを作成
	CRTSRCPF	ソース物理ファイルを作成
	CRTLF	論理ファイルを作成

WebSphere Development Studio for iSeries

WebSphere Development Studio は、iSeries サーバー用の e-business アプリケーションの数を、迅速かつ低コストで増加させるためのアプリケーション開発パッケージです。このパッケージにより、ホストとワークステーション双方の主要な iSeries 開発ツールすべてが単一の iSeries オファリングに統合されます。

ホスト開発ツールには大幅な改善が行われました。最新の AIX コンパイラーを全面的に更新した、既存のコンパイラーに置き換わる C および C++ コンパイラーを出荷しています。これにより、カスタマーおよびソリューション・プロバイダーによる他のプラットフォームからの e-business ソリューションの移植が支援されます。ILE RPG にも大幅な強化が行われました。 Java の相互運用性の向上とフリー・フォーム C 仕様が最大の機能拡張点です。 COBOL には、COBOL/Java 相互運用性機能の導入と共に、z/OS マイグレーション機能が追加されました。

WebSphere Development Studio for iSeries には、以下の構成要素が含まれます。

ホスト構成要素:

- ILE RPG

- ILE COBOL
- ILE C/C++
- アプリケーション開発ツールセット (ADTS)

ワークステーション構成要素:

- IBM WebFacing Tool
- iSeries 開発ツール: Remote System Explorer および iSeries プロジェクト
- Java 開発ツール (iSeries 拡張)
- Web 開発ツール (iSeries 拡張)
- Struts 環境サポート
- データベース開発ツール
- Web サービス開発ツール
- サーバー開発ツール
- XML 開発ツール
- CODE
- VisualAge RPG
- 統合 iSeries デバッガー

WebSphere Development Studio Client for iSeries

WebSphere Development Studio Client for iSeries (Development Studio Client) はワークステーション・ツールのアプリケーション開発パッケージであり、iSeries サーバー用の e-business アプリケーション数を迅速かつ低コストで増加するためのものです。

このパッケージにより、iSeries のワークステーション・ベースの主要な開発ツールすべてが単一の iSeries オファリングに統合されます。また、WebSphere Development Studio for iSeries の購入者用にも同梱されます。

WebSphere Development Studio Client for iSeries の機能リスト:

ワークベンチ・ベースの統合された開発環境
IBM WebSphere Development Studio Client for iSeries は WebSphere Studio Workbench (WSWB) バージョン 2.1 を使用します。

IBM WebFacing Tool
IBM WebFacing Tool を使用すると、DDS 表示ソース・ファイルをブラウザー内で実行可能なアプリケーションに変換できます。

Remote System Explorer および iSeries Development Tools
iSeries 開発ツールの一部として含まれる Remote System Explorer は、フレームワーク、ユーザー・インターフェース、編集、およびファイル、コマンド、iSeries 機能のジョブ操作を含みます。

iSeries Java 開発ツール
Java 開発ツールおよび iSeries Java 開発ツールを使用すると、 Java アプリケーションの開発および Java アプリケーション開発のための Java プログラミング言語で記述されたプログラムの作成、コンパイル、テスト、デバッグ、および編集ができます。

iSeries Web 開発ツール

Web 開発ツールを使用すると、iSeries ホスト上に存在する、ILE 言語および非 ILE 言語で作成されたプログラム内のビジネス・ロジックと Web ベースのフロントエンドを使用して通信する、新規の e-business アプリケーションを作成できます。

Struts 環境サポート

Development Studio Client は Struts および Web Diagram エディターに対するサポートを提供します。

データベース開発ツール

データベース開発ツールは、Java Database Connectivity (JDBC) ドライバーを持つローカルまたはリモートのあらゆるデータベースをサポートします。

Web サービス開発ツール

Web サービス開発ツールを使用すると、開発者は WWW 上で呼び出すことができるモジュラー・アプリケーションを作成できます。

サーバー開発ツール

サーバー開発ツールは、ローカルまたはリモートにインストールされたランタイム環境内のアプリケーションのテストに使用します。

XML 開発ツール

XML 開発ツールは、あらゆる XML ベースの開発をサポートします。

CODE (CoOperative Development Environment)

CODE は iSeries 開発のための Windows ツールの標準セットです。 CODE にはソースと DDS ファイルの作成、およびプロジェクト管理のためのユーティリティー・スイートが含まれます。

VisualAge RPG

VisualAge RPG は、ワークステーション上でのクライアント/サーバー・アプリケーションの作成および保守を可能にするビジュアル開発環境です。

統合 iSeries デバッガー

統合 iSeries デバッガーを使用すると、ワークステーション上のグラフィカル・ユーザー・インターフェースを使用して、 iSeries システムまたは Windows システム上で動作するコードのデバッグができます。

WebSphere Development Studio Client for iSeries について詳しく知りたい場合は、WWW の **ibm.com**/software/awdtools/iseries/ にある最新情報を参照してください。

第 2 章 ILE における RPG プログラミング

ILE RPG により ILE で RPG IV プログラミング言語が使用できるようになります。 ILE RPG は、iSeries システムで使用可能な ILE コンパイラー・ファミリーの 1 つです。

ILE は、iSeries システムでのプログラミングに対する新しいアプローチであり、iSeries マシン・アーキテクチャーおよび OS/400 オペレーティング・システムに対する大幅な機能拡張の結果として生まれたものです。ILE ファミリーのコンパイラーには、ILE RPG、ILE C、ILE COBOL、ILE CL、および VisualAge for C++ があります。表 16 は、OS/400 オペレーティング・システムがサポートするプログラム言語のリストです。ILE 言語のサポートの他に、オリジナル・プログラム・モデル (OPM) および拡張プログラム・モデル (EPM) 言語のサポートも引き続き存在します。

表 16. iSeries でサポートされているプログラミング言語

ILE (ILE)	オリジナル・プログラム・モデル (OPM)	拡張プログラム・モデル (EPM)
C++	BASIC (PRPQ)	FORTRAN
C	CL	PASCAL (PRPQ)
CL	COBOL	
COBOL	PL/I (PRPQ)	
RPG	RPG	

OPM と比較して ILE は、アプリケーション・プログラム開発における下記の領域で RPG ユーザーに改善や強化をもたらします。

- プログラムの作成
- プログラムの管理
- プログラムの呼び出し
- ソース・プログラムのデバッグ
- バインド可能なアプリケーション・プログラミング・インターフェース (API)

上記の領域のおのおのについて下記の段落で簡単に説明し、さらに後続の章で詳述します。

プログラムの作成

ILE におけるプログラム作成は次の部分からなります。

1. ソース・コードのモジュールへのコンパイル
2. 1 つ以上のモジュールの 1 つのプログラム・オブジェクトへのバインド (結合)

OPM フレームワークで行うのと同様に、バインド RPG プログラムの作成 (CRTBNDRPG) コマンドを使用してワン・ステップ処理でプログラム・オブジェクトを作成することができます。このコマンドは、一時モジュールを作成し、その後

で一時モジュールをプログラム・オブジェクトにバインドします。また、このコマンドとバインディング・ディレクトリーを使用してその他のオブジェクトをバインドすることもできます。

代わりに、コンパイル用とバインド用の別のコマンドを使用してプログラムを作成することができます。このツー・ステップ処理で、モジュールを再利用するか、またはプログラム内のその他のモジュールをコンパイルし直さないで 1 つのモジュールを更新することができます。さらに、ILE 言語からモジュールをバインドすることができるので、混合言語プログラムを作成および保守することができます。

ツー・ステップ処理で、RPG モジュール作成 (CRTRPGMOD) コマンドを使用してモジュール・オブジェクトを作成します。このコマンドは、ソース・ステートメントをコンパイルしてモジュール・オブジェクトに入れます。モジュールとは実行不能なオブジェクトのことであり、実行するプログラム・オブジェクトにバインドしなければなりません。1 つ以上のモジュールを一緒にバインドするためには、プログラム作成 (CRTPGM) コマンドを使用します。

サービス・プログラムは、1 つ以上のモジュールのプロシージャーを、別個にバインドされたオブジェクトにパッケージ化する手段です。その他の ILE プログラムはサービス・プログラム内のプロシージャーにアクセスすることができますが、システム上にはサービス・プログラムの 1 つのコピーしかありません。サービス・プログラムを使用すると、モジュール性および保守容易性が高まります。他のソフトウェア会社が開発したサービス・プログラムを使用したり、あるいは、逆に他のソフトウェア会社の使用に備えて、ユーザーのサービス・プログラムをパッケージ化することができます。サービス・プログラムは、サービス・プログラム作成 (CRTSRVPGM) コマンドを使用して作成されます。

プログラムまたはサービス・プログラムに必要なモジュールおよびサービス・プログラムの名前の入ったバインディング・ディレクトリーを作成することができます。CRTBNDRPG、CRTSRVPGM、および CRTPGM コマンドを使ってプログラムを作成する時は、バインディング・ディレクトリーのリストを指定することができます。これは CRTRPGMOD コマンドでも指定することができますが、この場合、バインディング・ディレクトリーの検索は CRTPGM または CRTSRVPGM 時にモジュールがバインドされる時に行われます。バインディング・ディレクトリーにリストされたモジュールまたはサービス・プログラムは必要な場合にだけ使用されるので、バインディング・ディレクトリーによりプログラム・サイズを減少させることができます。

19 ページの図 5 はプログラム作成の 2 つの方法を示したものです。

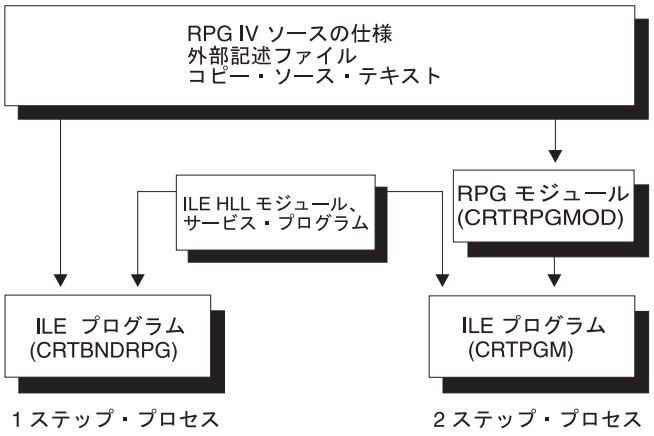


図 5. ILE におけるプログラム作成

一度プログラムを作成しておくと、プログラム更新 (UPDPGM) コマンドまたはサービス・プログラム更新 (UPDSRVPGM) コマンドを使用して、そのプログラムを更新することができます。これは、プログラムを更新するのに、新規または変更されたモジュール・オブジェクトを使用可能にするだけであり、便利です。

ワン・ステップ処理の詳細については、65 ページの『第 6 章 CRTBNDRPG コマンドによるプログラムの作成』を参照してください。ツー・ステップ処理の詳細については、83 ページの『第 7 章 CRTRPGMOD および CRTPGM コマンドによるプログラムの作成』を参照してください。サービス・プログラムについて詳しくは、101 ページの『第 8 章 サービス・プログラムの作成』を参照してください。

プログラムの管理

ILE は、次のものについて共通の基準を提供します。

- プログラム・フローの管理
- 資源の共用
- アプリケーション・プログラミング・インターフェース (API) の使用
- プログラム実行時間中の例外処理

これにより RPG ユーザーは以前よりも効果的に資源の制御を行えます。

ILE プログラムおよびサービス・プログラムは、活動化され、プログラム作成時に指定された活動化グループに入れられます。プログラムまたはサービス・プログラムを実行可能にする処理は、活動化として知られています。活動化は、その空間で 1 つ以上のプログラムが実行できるように、ジョブ内で資源を割り振ります。プログラムが呼び出された時に、プログラムの指定された活動化グループが存在していない場合には、プログラムの活動化を保留するために、ジョブ内で作成されます。

活動化グループは ILE アプリケーションの資源および働きを管理するキー要素です。例えば、コミットメント制御操作の有効範囲を活動化グループ・レベルに設定することができます。ユーザーはまた、ファイル一時変更および共用オープン・データ・パスを実行中のアプリケーション・プログラムの活動化グループに有効範囲を設定することができます。最終的に、終了時のプログラムの働きもまた、プログラムが実行する活動化グループによって影響を受けます。

活動化グループの詳細については、 120 ページの『活動化グループの管理』を参照してください。

ユーザーは、すべての ILE プログラミング言語用に提供されているバインド可能 API を使用して、実行時配列用の記憶域を動的に割り振ることができます。これらの API によって、単一言語および混合言語のアプリケーション・プログラムで記憶域管理機能の中央セットにアクセスすることができ、現在記憶域モデルを用意していない言語に記憶域モデルを提供します。RPG は、命令コードを使用して一部の記憶域管理機能を提供します。記憶域管理の詳細については、 124 ページの『動的に割り振られた記憶域の管理』を参照してください。

プログラムの呼び出し

ILE では、ユーザーは ILE RPG プログラムおよび OPM RPG/400 プログラムが従来の動的プログラム呼び出しを介して相互関係を維持する、アプリケーション・プログラムを作成することができます。 このような呼び出しを使用する時には、呼び出し側プログラムは CALL ステートメントで呼び出されるプログラムの名前を指定します。呼び出されるプログラム名は、呼び出し側プログラムが呼び出されたプログラムに制御を渡す直前に実行時のアドレスに分析解決されます。

ユーザーはまた、より速い静的呼び出しで相互の関係付けができる ILE アプリケーション・プログラムを書くこともできます。 静的呼び出しにはプロシージャー間の呼び出しが含まれます。プロシージャーは、タスクを実行してから呼び出し元に戻る、コーディングの自己完結型セットです。ILE RPG モジュールは、ゼロあるいは 1 つ以上のサブプロシージャーが後に続く任意指定のメイン・プロシージャーから構成されます。プロシージャー名はバインド時 (すなわち、プログラムを作成する時) に分析解決されるので、静的呼び出しは動的呼び出しよりも速くなります。

また静的呼び出しでは次のことも可能です。

- 操作記述子
- 省略パラメーター
- 値によってパラメーターを渡す
- 戻り値の使用
- 渡すことが可能な多数のパラメーター

操作記述子および省略パラメーターは、他の ILE 言語で書かれたバインド可能 API またはプロシージャーを呼び出す時に有用となります。

プログラムの実行の詳細については、 113 ページの『第 9 章 プログラムの実行』を参照してください。プログラム/プロシージャー呼び出しの詳細については、 139 ページの『第 10 章 プログラムおよびプロシージャーの呼び出し』を参照してください。

ソースのデバッグ

WebSphere Development Studio Client for iSeries を使用します。これは推奨される方法であり、プログラムのデバッグに関する説明は、製品のオンライン・ヘルプに記載されています。統合 iSeries デバッガーを使用すると、サーバー上で実行されるプログラムをワークステーション上のグラフィカル・ユーザー・インターフェース

からデバッグできます。また、デバッガーを実行する前にソース内にブレークポイントを直接設定することもできます。統合 iSeries デバッガー・クライアント・ユーザー・インターフェースでは、プログラム実行を制御することもできます。例えば、プログラムの実行、行の設定、ウォッチができ、またエントリー・ポイント・ブレークポイントの設定、プログラム命令のステップスルー、変数の値の表示、およびコール・スタック内容の表示ができます。また、異なる言語で記述された複数のアプリケーションであっても、単一のデバッガー・ウィンドウでデバッグできます。デバッグする各セッションは、「デバッグ」ビュー内に個別に表示されます。

ILE では、単一または混合言語の ILE アプリケーション・プログラムで、ソース・レベルのデバッグを実行することができます。また ILE ソース・デバッガーは、OPM プログラムもサポートします。プログラムの実行中に、デバッグ・コマンドを使用してプログラムのフローを制御することができます。プログラムの実行前に、条件付きまたは無条件のジョブ、またはスレッド停止点を設定することができます。プログラムを呼び出した後に、指定された数のステートメントを実行し、変数を表示または変更することができます。停止点、ステップ・コマンド、または実行時エラーのためにプログラムが停止した時には、プログラムが停止した個所に関連するモジュールが画面に表示されます。この時点で追加のデバッグ・コマンドを入力することができます。

デバッグ・プログラムについては、 215 ページの『第 12 章 プログラムのデバッグ』を参照してください。

バインド可能な API

ILE は、ILE RPG によって現在提供されている機能を補足するために使用できる、多くのバインド可能 API を提供しています。バインド可能 API は、プログラム呼び出しおよび活動化機能、条件および記憶域管理、数学関数、および動的画面管理を提供しています。

ILE RPG のアプリケーション・プログラム中に使用できる API として以下のものがあります。

- CEETREC - 終了直前の状態を知らせる
- CEE4ABN - 異常終了
- CEECRHP - ユーザー独自ヒープの作成
- CEEDSHP - ユーザー独自ヒープの廃棄
- CEEFRST - ユーザー独自ヒープ内の記憶域の解放
- CEEGTST - ユーザー独自ヒープ内のヒープ記憶域の取得
- CEECZST - ユーザー独自ヒープ内の記憶域の再割り振り
- CEEDOD - 操作記述子の分解

注: DFTACTGRP(*YES) で作成されたプログラム内からは、これら、またはその他の ILE バインド可能 API を使用することはできません。これは、バインド呼び出しがこのタイプのプログラムでは許可されないためです。

これらの ILE バインド可能 API について詳しくは、 113 ページの『第 9 章 プログラムの実行』を参照してください。

マルチスレッド・アプリケーション

iSeries システムでマルチスレッド化をサポートできるようになりました。ILE RPG がプログラム・スレッドの初期化や管理を直接にサポートすることはありませんが、ILE RPG プロシージャーは、マルチスレッド環境でスレッドとして実行できます。ILE RPG プログラムをマルチスレッド・アプリケーションで呼び出したい場合は必ず、ILE RPG プロシージャーがスレッド・セーフであるようにする必要があります。また、プロシージャーがアクセスするシステム機能はどれもスレッド・セーフであるようにする必要もあります。

THREAD(*SERIALIZE) 制御仕様書キーワードを指定すると、ILE RPG モジュールについてスレッド・セーフティーを実現する上で役立ちます。
THREAD(*SERIALIZE) を指定すると、ほとんどの変数とすべての内部制御構造が複数のスレッドによって不正にアクセスされないよう保護されます。スレッド・セーフ・モジュールは、モジュール内のプロシージャーに入るとロックされ、モジュール内のプロシージャーがまだ実行していないときはアンロックされます。このようにアクセスを逐次化すると、活動化グループ内のいずれのモジュール内でも、活動状態になれるスレッドは一度に 1 つだけになります。ただし、複数のモジュール間で共用される記憶域のスレッド・セーフティーの取り扱いは、まだ、プログラマーに任されています。これは、アプリケーション内にロジックを追加して記憶域へのアクセスを同期化することによって行います。

詳細については、 172 ページの『マルチスレッド化に関する考慮事項』を参照してください。

第 3 章 プログラムの作成方針

ILE 言語を使用するプログラムの作成には、多くのアプローチがあります。この節では、ILE RPG または他の ILE 言語を使用して ILE プログラムを作成するための 3 つの共通方針について説明します。

1. OPM 互換を最大にするために CRTBNDRPG を使用してプログラムを作成する。
2. CRTBNDRPG を使用して ILE プログラムを作成する。
3. CRTRPGMOD および CRTPGM を使用して ILE プログラムを作成する。

最初の方針は一時的なものとして好ましいものです。この方針は、OPM のアプリケーション・プログラムをもっているユーザーで、時間不足のためにそれらのアプリケーション・プログラムをすべて一度に ILE に変換できないユーザーのためのものです。2 番目の方針も一時的なものとすることができます。この方針の場合、ILE について学習する時間がありますが、その機能の一部をすぐに使用することもできます。3 番目の方針は、これら 2 つよりも複雑ですが、柔軟性は最も高いものです。

最初と 2 番目の方針は両方とも、ワン・ステップ・プログラム作成処理、すなわち CRTBNDRPG を使用します。3 番目の方針は、ツー・ステップ・プログラム作成処理、すなわち CRTRPGMOD に続けて CRTPGM を使用します。

方針 1: OPM 互換アプリケーション・プログラム

方針 1 の結果として ILE プログラムは OPM プログラムと高い互換性をもつことになります。これによって、RPG IV 拡張機能を利用できますが、すべてを利用できるわけではありません。ILE への移行を実行している間に、このようなプログラムが一時的に必要となる場合があります。

方法

このようなプログラムを作成するには、以下の一般的なアプローチを使用します。

1. CVTRPGSRC コマンドを使用して、ソースを RPG IV に変換します。
   変換するソースで使用されているすべての /COPY のメンバーを変換するようにします。
2. DFTACTGRP(*YES) を指定した CRTBNDRPG コマンドを使用してプログラム・オブジェクトを作成します。

DFTACTGRP(*YES) を指定することは、プログラム・オブジェクトがデフォルトの活動化グループでのみ実行されることを意味します (デフォルトの活動化グループとは、すべての OPM プログラムが実行される活動化グループのことです)。結果としてプログラム・オブジェクトは、一時変更の有効範囲設定、オープンの有効範囲設定、および RCLRSC の領域で、OPM プログラムと高い互換性をもつことになります。

このアプローチを使用する時には、ILE 静的バインドを使用することはできません。つまり、ユーザーのソースにバインド・プロシージャー呼び出しをコーディングすることも、このプログラムを作成する時に、CRTBNDRPG コマンドで BNDDIR または ACTGRP パラメーターを使用することもできません。

OPM 互換プログラムの例

図 6 は、OPM 互換のプログラムを必要とするサンプル・アプリケーションの実行時のビューです。 OPM アプリケーション・プログラムは、1 つの CL プログラムおよび 2 つの RPG プログラムから成っています。この例では、RPG プログラムの 1 つが ILE に移動されていて、残りのプログラムは変更されていません。

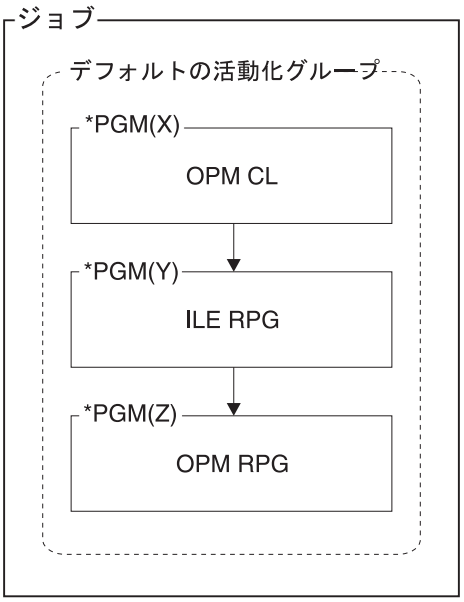


図 6. OPM 互換アプリケーション・プログラム

ILE の影響

以下に、ユーザーのアプリケーション・プログラムが取り扱う ILE の影響について説明します。

プログラム呼び出し

OPM プログラムは以前と同様に働きます。システムは、ユーザーがジョブを開始する時に OPM のデフォルトの活動化グループを自動的に作成し、すべての OPM アプリケーション・プログラムがその中で実行されます。1 つのプログラムが活動化グループ内の別のプログラムを動的呼び出しを使用して呼び出すことができます。

データ

プログラムが活動化される時に静的データの記憶域が作成され、プログラムが非活動化されるまで存在します。プログラムが終了すると (正常または異常終了)、プログラムの記憶域が削除されます。終了せずに戻されたプログラムの記憶域を消去するためには、資源再利用 (RCLRSC) コマンドを使用します。

ファイル ファイル処理は前のリリースと同じです。ファイルは、プログラムの正常終了または異常終了時にクローズされます。

エラー 前のリリースと同様に、コンパイラーは各プログラム内で別個にエラーを処理します。プログラム内で発生したエラーは以前と同じです。しかし、エラーは現在、ILE 条件管理プログラムによってプログラム間で連絡されるので、ユーザーはプログラム間で異なるメッセージを見ることがあります。メッセージは新しいメッセージ ID をもつことができるので、CL プログラムが特定のメッセージ ID を監視する場合には、ユーザーはその ID を変更する必要がある場合があります。

関連情報

RPG IV への変換 458 ページの『ソースの変換』

ワン・ステップ作成処理 65 ページの『第 6 章 CRTBNDRPG コマンドによるプログラムの作成』

ILE 静的バインド 139 ページの『第 10 章 プログラムおよびプロシージャーの呼び出し』、さらに「*ILE 概念*」

例外処理の相違 278 ページの『OPM と ILE RPG 例外処理との違い』

方針 2: CRTBNDRPG を使う ILE プログラム

方針 2 は結果として、ILE 静的バインドの利点を利用する ILE プログラムになります。バインディング・ディレクトリーを使用してモジュールをその他モジュールまたはサービス・プログラムとバインドすることができるので、ユーザーのソースには、静的プロシージャー呼び出しを含めることができます。また、プログラムが実行される活動化グループを指定することもできます。

方法

このようなプログラムを作成するには、以下の一般的なアプローチを使用します。

1. RPG III ソースで始める場合は、CVTRPGSRC コマンドを使ってソースを RPG IV に変換する。

   変換する場合には必ず、すべての /COPY メンバーおよび変換しようとするソースによって呼び出されるすべてのプログラムを変換するようにしてください。また、プログラムの呼び出しに CL を使用する場合には、OPM CL の代わりに ILE CL を使用していることを確認してください。
2. プログラムが実行する活動化グループを決定する。

   この例のように、アプリケーション・プログラム名の後に名前を付けることができます。
3. 使用するバインディング・ディレクトリーがある場合は、それらの名前を識別する。

   このアプローチでは、バインディング・ディレクトリーを使用している場合、該当ディレクトリーは既に作成済みのものと見なしています。例えば自分のソース

にバインドしたい、サード・パーティーのサービス・プログラムがある場合もあります。したがって、バインディング・ディレクトリーの名前だけは知っておく必要があります。

4. CRTBNDRPG を使用して ILE プログラムを作成します。このとき、DFTACTGRP(*NO) を指定し、ACTGRP パラメーターに活動化グループ、さらにバインディング・ディレクトリーがあればそれを BNDDIR パラメーターに指定します。

ACTGRP(*CALLER) が指定されていて、このプログラムがデフォルトの活動化グループ内で実行中のプログラムによって呼び出される場合には、このプログラムは一時変更の有効範囲設定、オープンの有効範囲設定、および RCLRSC の面で、ILE の意味構造論にしたがって働くということに注意してください。

この方針の主な欠点は、ILE プログラムを作成する際、後で再利用できる永続モジュール・オブジェクトがないため、他のモジュールにバインドできないことです。そのうえ、プロシージャー呼び出しはバインディング・ディレクトリーで識別されたモジュールまたはサービス・プログラムに対するものでなければなりません。プログラムを作る時に、バインディング・ディレクトリーを使わずに 2 つ以上のモジュールをバインドしようとすると、3 番目の方針が必要になります。

CRTBNDRPG を使った ILE プログラムの例

図 7 は、提供されるサービス・プログラムにバインドされる ILE RPG プログラムを ILE CL プログラムが呼び出すアプリケーションの実行時のビューです。このアプリケーション・プログラムは XYZ という名前の活動化グループ内で実行されます。

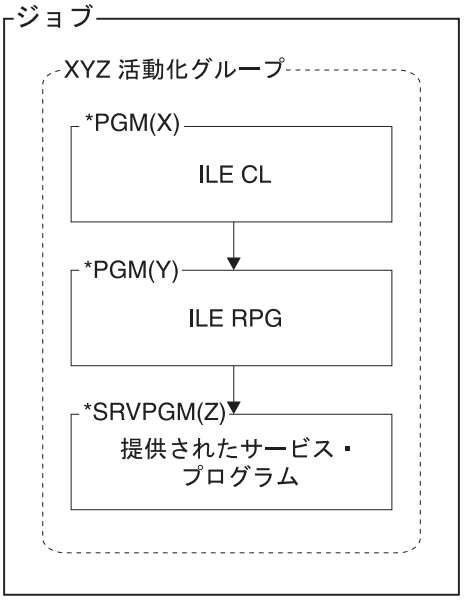


図 7. CRTBNDRPG を使う ILE プログラム

ILE の影響

以下に、ユーザーのプログラムが取り扱う ILE の影響について説明します。

プログラム呼び出し

システムは、アプリケーション・プログラムを開始するときに活動化グループがまだなければ、それを自動的に作成します。

アプリケーション・プログラムには、動的プログラム呼び出しまたは静的プロシージャー呼び出しを含めることができます。バインドされたプログラム内のプロシージャーは、静的呼び出しを使用してお互いに呼び出します。プロシージャーは、動的呼び出しを使用して ILE および OPM プログラムを呼び出します。

データ

プログラム記憶域の存続時間は活動化グループの存続時間と同じです。記憶域は活動化グループが削除されるまで活動状態のままです。

ILE RPG は、プログラム終了およびデータ再初期化の意味が OPM RPG の場合と同じになるように、実行時にデータを管理します。ただし、実際の記憶域は、OPM RPG プログラムが終了した時に削除されたようには、削除されません。プロシージャーへの前の呼び出しが LR オンで終了するか、あるいは異常終了した場合には、データが再び初期設定されます。

エクスポートまたはインポート (それぞれ EXPORT または IMPORT キーワードを使用) として識別されるプログラム・データは、個々のモジュールにとっては外部データです。このデータは、各モジュール間では 1 つのプログラムにバインドされるものとして認識されます。

ファイル

デフォルトでは、システムによるファイル処理 (オープン、共用、一時変更、およびコミットメント制御を含む) は、活動化グループ・レベルに有効範囲指定されます。データ管理レベルでは、別の活動化グループ内のプログラムとファイルを共用することができません。活動化グループをまたがってファイルを共用したい場合には、一時変更コマンドで SHARE(*YES) を指定してジョブ・レベルでファイルをオープンするか、あるいは SHARE(*YES) でファイルを作成しなければなりません。

エラー

同一の活動化グループで ILE RPG プログラムまたはプロシージャーを呼び出す場合、以前に照会メッセージを表示させるような例外があった場合には、ユーザーの呼び出し側プログラムは最初にその例外を見ることになります。

ユーザーの呼び出し側プログラムにエラー標識または *PSSR があると、例外を起こしたプログラムまたはプロシージャーは照会メッセージを表示せずに異常終了することになります。ユーザーの呼び出し側プログラムは同じ動きをします (エラー標識がオンに設定されるか *PSSR が呼び出されます)。

OPM プログラムまたは別の活動化グループ内のプログラムまたはメイン・プロシージャーを呼び出す時には、例外処理は、OPM RPG における場合と同様に、各プログラムが自身の例外を処理することになります。表示するメッセージは新しいメッセージ ID をも

つことができるので、特定のメッセージ ID を監視する場合には、ユーザーはその ID を変更する必要がある場合があります。

各言語はそれ自身のエラーを処理し、別の ILE 言語で書かれたモジュール内で起こるエラーを処理することができます。例えば、エラー標識がコーディングされている場合には、RPG で C 言語のエラーを処理します。C で RPG エラーを処理することができます。

関連情報

RPG IV への変換 458 ページの『ソースの変換』

ワン・ステップ作成処理 65 ページの『第 6 章 CRTBNDRPG コマンドによるプログラムの作成』

活動化グループ 120 ページの『活動化グループの管理』

RCLRSC 123 ページの『資源再利用コマンド』

ILE 静的バインド 139 ページの『第 10 章 プログラムおよびプロシージャーの呼び出し』、さらに「*ILE 概念*」

例外処理の相違 278 ページの『OPM と ILE RPG 例外処理との違い』

一時変更およびオープンの範囲設定 335 ページの『ファイル入出力の一時変更および指定変更』および 339 ページの『オープン・データ・パスの共用』、さらに「*ILE 概念*」

方針 3: CRTRPGMOD を使う ILE アプリケーション

この方針によってユーザーは、ILE が提供する概念を全面的に利用することができます。しかし、最も柔軟性のあるアプローチではありますが、さらに考慮すべき点もあります。この項では、作成するための次の 3 つのシナリオについて説明します。

- 単一言語のアプリケーション・プログラム
- 混合言語のアプリケーション・プログラム
- 拡張アプリケーション・プログラム

ILE の影響は、 27 ページの『ILE の影響』で説明したのと同じです。

この方法を使う前に「*ILE 概念*」の基本的な ILE 概念を読んでおくことをお勧めします。

方法

この方法は最も柔軟性に富んでいるため、ILE アプリケーション・プログラムを作成する多くの方法を含んでいます。以下のリストでは、ユーザーが行う必要がある主要なステップについて説明します。

1. 適切なコマンド (RPG ソースには CRTRPGMOD、CL ソースには CRTCLMOD など) を使用して、各ソース・メンバーからモジュールを作成する。
2. アプリケーション・プログラムの、例えば次のような ILE 特性を決定する。

- アプリケーション・プログラムの開始点となるプロシージャーをモジュールに含めるかどうかを決定する。 入り口モジュールとして選択するモジュールは、最初に制御を取得するモジュールです。 OPM アプリケーション・プログラムでは、これはコマンド処理プログラムまたはメニュー項目の選択によって呼び出されたプログラムです。
- アプリケーション・プログラムが実行される活動化グループを決めます (多くの場合ユーザーはアプリケーション・プログラムの名前に基づく名前を付けた活動化グループ内で実行したいと思われます)。
- 使用すべきエクスポートおよびインポートを決定する。

3. サービス・プログラムを作成するのになんらかのモジュールを一緒にバインドするかどうかを決定する。バインドする場合は、CRTSRVPGM を使用してサービス・プログラムを作成する。
4. 使用するバインディング・ディレクトリーがある場合は、それらの名前を識別する。

   このアプローチでは、バインディング・ディレクトリーを使用している場合、該当ディレクトリーは既に作成済みのものと見なしています。例えば自分のソースにバインドしたい、サード・パーティーのサービス・プログラムがある場合もあります。したがって、バインディング・ディレクトリーの名前だけは知っておく必要があります。
5. CRTPGM を使い、ステップ 2 (28 ページ) で決定された特性に基づいてパラメーターに値を指定して、モジュールとサービス・プログラムを一緒にバインドする。

この方法を使って作成されたアプリケーション・プログラムは完全に保護されて、すなわちそれ自身の活動化グループ内で実行することができます。さらに、UPDPGM または UPDSRVPGM コマンドの使用によって容易に更新することができます。これらのコマンドによってユーザーは、プログラム・オブジェクトの再作成の必要なく、1 つまたは複数のモジュールを追加または置き換えることができます。

単一言語の ILE アプリケーション・シナリオ

このシナリオでは、複数のソース・ファイルをコンパイルしてモジュールにし、それらを ILE RPG プログラムが呼び出す 1 つのプログラムにバインドします。 30 ページの図 8 は、このアプリケーションの実行時のビューです。

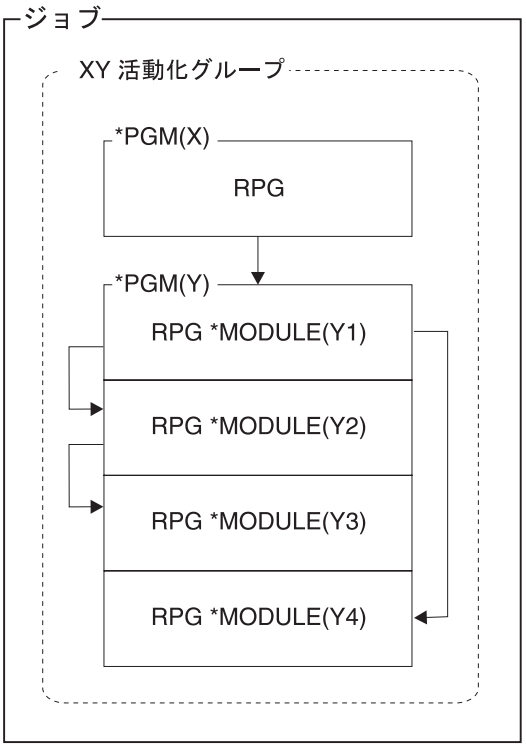


図 8. CRTRPGMOD および CRTPGM を使用する単一言語のアプリケーション・プログラム

プログラム X からプログラム Y への呼び出しは動的呼び出しです。プログラム Y 内でのモジュール間の呼び出しは静的呼び出しです。

アプリケーションが、呼び出し、データ、ファイル、およびエラーを扱う方法に対して ILE が与える影響について詳しくは、27 ページの『ILE の影響』を参照してください。

混合言語の ILE アプリケーション・シナリオ

このシナリオでは、ユーザーは統合された混合言語のアプリケーション・プログラムを作成します。ある ILE 言語で書かれたメイン・モジュールは、別の ILE 言語で書かれたプロシージャーを呼び出します。メイン・モジュールは、他のモジュールが共用するファイルをオープンします。異なった言語を使用しているために、一貫性のある動きを期待できない場合があります。しかし、ILE はこれが可能です。

31 ページの図 9 は、あるモジュールがバインド不可能な API、QUSCRTUS (ユーザー空間作成) を呼び出す、混合言語の ILE プログラムを含むアプリケーションの実行時のビューです。

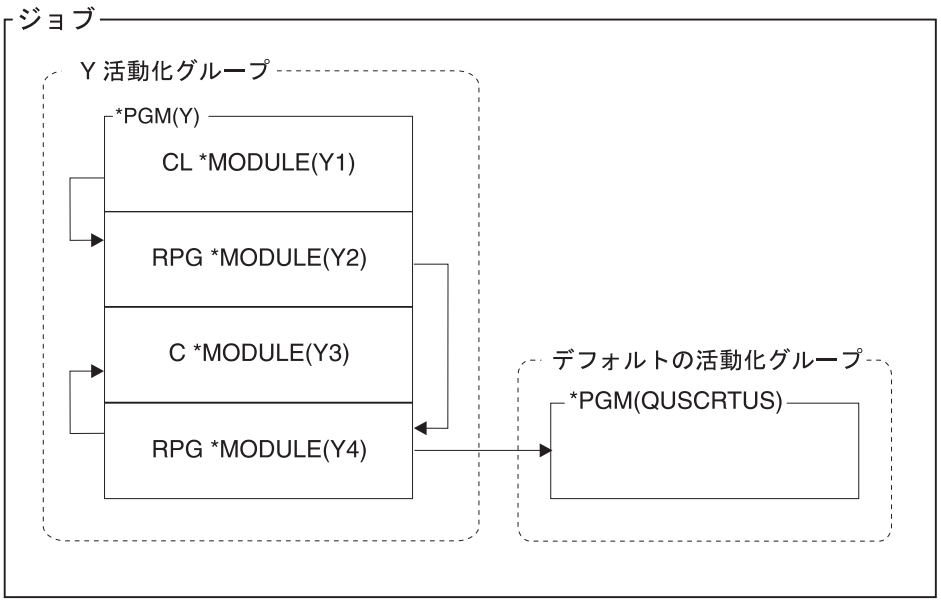


図 9. 混合言語のアプリケーション・プログラム

プログラム Y から OPM API への呼び出しは動的呼び出しです。プログラム Y 内でのモジュール間の呼び出しは静的呼び出しです。

アプリケーションが、呼び出し、データ、ファイル、およびエラーを扱う方法に対して ILE が与える影響について詳しくは、27 ページの『ILE の影響』を参照してください。

拡張アプリケーション・プログラム・シナリオ

このシナリオではユーザーは、サービス・プログラムを含む ILE 機能の利点を全面的に利用することができます。モジュールおよびサービス・プログラム内のプロシージャーに使用されるバインド呼び出しの使用により、特にサービス・プログラムが呼び出し元と同じ活動化グループ内で実行される場合にパフォーマンスが向上します。

32 ページの図 10 は、ILE プログラムが 2 つのサービス・プログラムにバインドされる例を示します。

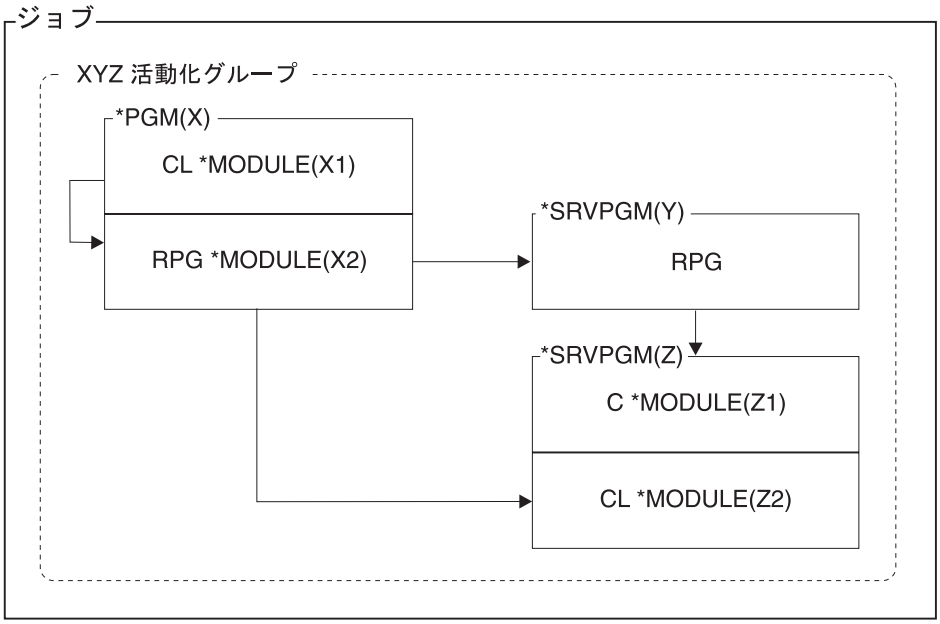


図 10. 拡張アプリケーション・プログラム

プログラム X からプログラム Y および Z への呼び出しは静的呼び出しです。

アプリケーションが、呼び出し、データ、ファイル、およびエラーを扱う方法に対して ILE が与える影響について詳しくは、27 ページの『ILE の影響』を参照してください。

関連情報

ツー・ステップ作成処理	83 ページの『第 7 章 CRTRPGMOD および CRTPGM コマンドによるプログラムの作成』
活動化グループ	120 ページの『活動化グループの管理』
ILE 静的バインド	139 ページの『第 10 章 プログラムおよびプロシージャーの呼び出し』、さらに「*ILE 概念*」
例外処理	273 ページの『第 13 章 例外の処理』、さらに「*ILE 概念*」
サービス・プログラム	101 ページの『第 8 章 サービス・プログラムの作成』、さらに「*ILE 概念*」
プログラムの更新	97 ページの『UPDPGM コマンドの使用』

避けるべき方針

ILE は、プログラムおよびアプリケーションの作成に多くの代替方法を提供します。しかし、そのすべてが同じように望ましい訳ではありません。一般的に、OPM と ILE プログラムからなるアプリケーションが、OPM のデフォルトの活動化グループと指定の活動化グループとに分割されるような状況は避けるべきです。言い換えれば、33 ページの図 11 に示すようなシナリオは避けてください。

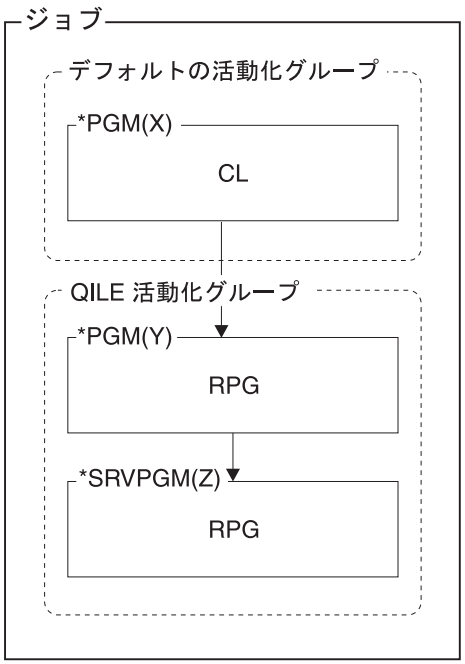


図 11. 避けるべきシナリオ: 1 つのアプリケーション・プログラムに、 OPM のデフォルトの活動化グループ内の CL プログラムと、指定の活動化グループ内の ILE プログラムが入っています。

1 つのアプリケーションが、デフォルトの活動化グループと指定の活動化グループとの間で分割されている場合には、 OPM の働きと ILE の働きとが混用されています。 例えば、デフォルトの活動化グループのプログラムはプログラムの終了時に、ILE プログラムがその資源を解放するのを期待している場合があります。しかし、これは活動化グループが終了するまで起こりません。

同様に、アプリケーション・プログラムがデフォルトの活動化グループと指定の活動化グループとの間で分割されている時には、一時変更および共用 ODP の有効範囲を管理するのがより困難になります。デフォルトでは、指定のグループの有効範囲は活動化グループ・レベルとなりますが、デフォルトの活動化グループに対しては、活動化グループ・レベルではなく、呼び出しレベルまたはジョブ・レベルのどちらかとなります。

注: コマンド行から、または単に呼び出しを行う OPM プログラムから ILE プログラムを呼び出すことは問題ではありません。指定変更やコミットメント制御などの共用リソースを使用する OPM プログラムおよび ILE プログラムで発生した問題、および指定の活動化グループで実行されているプログラムには影響しない RCLRSC などの OPM コマンドを使用しようとしている OPM プログラムで発生した問題は、すべて解決することができます。

第 4 章 複数プロシージャーを使用するアプリケーション・プログラムの作成

ILE RPG モジュールに複数のプロシージャーをコーディングする機能は、モジュラー・アプリケーション・プログラムをコーディングする機能を大きく拡張しています。この章では、アプリケーション・プログラム内のモジュールをこのように使用する理由および方法について説明しています。具体的に、本章では次のことを説明します。

- キー概念の概要
- 複数のプロシージャーのあるモジュールの例
- コーディング上の考慮事項

複数プロシージャーのあるモジュールのコーディング上の詳細説明については、この項の終わりを参照してください。

複数プロシージャー・モジュール — 概要

ILE プログラムは 1 つ以上のモジュールから成っており、モジュールは 1 つ以上のプロシージャーから構成されています。**プロシージャー**とは、バインドされた呼び出しで呼び出すことができるコーディングの一部です。ILE RPG には、メイン・プロシージャーとサブプロシージャーの 2 種類のプロシージャーがあります。サブプロシージャーを呼び出す方法はプロトタイプ呼び出しによります。

注: RPG の資料では、用語「プロシージャー」は、メイン・プロシージャーとサブプロシージャーの両方を示しています。

メイン・プロシージャーおよびサブプロシージャー

ILE RPG モジュールは、メイン・プロシージャーおよびゼロまたは 1 つ以上のサブプロシージャーから構成されています (サブプロシージャーがある場合には、メイン・プロシージャーは任意指定です)。**メイン・プロシージャー**とは、プログラム入力プロシージャーとして指定できる (したがって ILE プログラムが最初に呼び出された時に制御を受け取る) プロシージャーです。メイン・プロシージャーは、モジュールを始める H、F、D、I、C、および O 仕様書のセットである、**メイン・ソース・セクション**で定義されます。V3R1 では、すべての ILE RPG モジュールはメイン・プロシージャーをもち、他のプロシージャーはもっていません。

サブプロシージャーは、メイン・ソース・セクションの後に指定されるプロシージャーです。サブプロシージャーは次の点で基本的にメイン・プロシージャーと異なります。

- サブプロシージャー内で定義された名前は、サブプロシージャーの外ではアクセスすることができない。
- サブプロシージャー用にはサイクル・コードは生成されない。
- 呼び出しインターフェースはプロトタイプでなければならない。

- サブプロシージャーへの呼び出しはバインドされたプロシージャー呼び出しでなければならない。
- P、D、および C 仕様書しか使用することができない。

サブプロシージャーは、データ項目がローカルであるために、他のプロシージャーからの独立性を提供することができます。内部データ項目は通常、自動記憶域に保管されます。これは、ローカル変数の値がプロシージャーへの呼び出し間で保存されないことを意味します。

サブプロシージャーは別の機能を提供します。値によってサブプロシージャーにパラメーターを渡したり、値を戻す式の中でサブプロシージャーを呼び出すことができます。図 12 は、複数プロシージャーのあるモジュールがどのように見えるかを示しています。

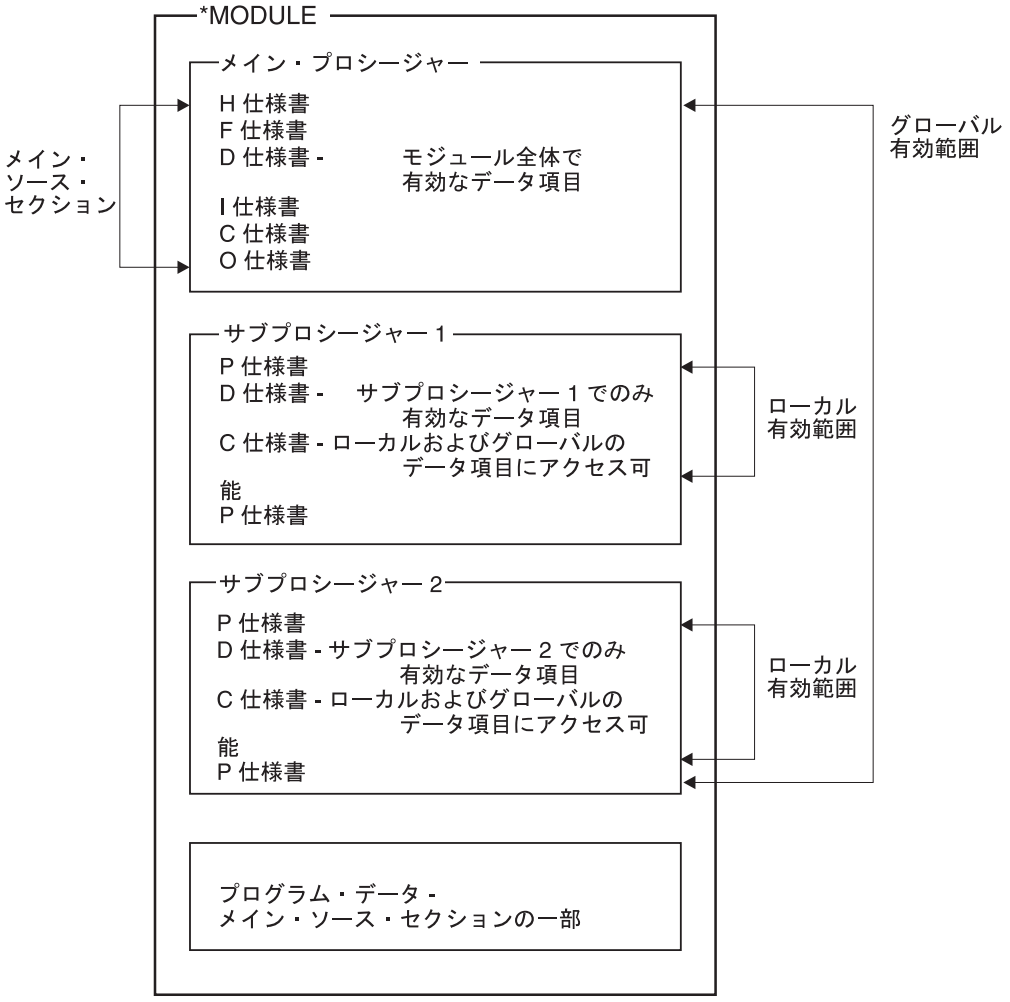


図 12. 複数プロシージャーのある ILE RPG モジュール

図から分かるとおり、特定のタスクを処理するサブプロシージャーをコーディングすることができます。これらのタスクは、メイン・プロシージャーによって、またはアプリケーション・プログラム内の他のモジュールによって必要となる場合があ

ります。さらに、サブプロシージャー内で一時データ項目を宣言することができますが、モジュール内のどこかで宣言されている場合には、気にする必要はありません。

プロトタイプ呼び出し

サブプロシージャーを呼び出す場合には、プロトタイプ呼び出しを使用しなければなりません。また、この方法で任意の言語で書かれたプログラムまたはプロシージャーを呼び出すこともできます。**プロトタイプ呼び出し**とは、プロトタイプを使用して、コンパイル時に呼び出しインターフェースを検査することによって行う呼び出しです。**プロトタイプ**は、呼び出しインターフェースの定義です。これには、次の情報が含まれます。

- 呼び出しがバインド (プロシージャー) または動的 (プログラム) であるかどうか
- プログラムまたはプロシージャー (外部名) の検索方法
- パラメーターの数および特質
- パラメーターが渡す必要があるものかどうか、任意に渡されるものかどうか
- 操作記述子が渡されたかどうか (プロシージャーの場合)
- 戻り値 (ある場合) のデータ・タイプ (プロシージャーの場合)

プロトタイプは、プログラムまたはプロシージャーを正しく呼び出すため、さらに呼び出し元が正しいパラメーターを渡したことを確認するために、コンパイラーによって使用されます。図 13 は、レコードの各種フィールドを読み取り可能な形式に様式設定するプロシージャー FmtCust のプロトタイプを示しています。これには 2 つの出力パラメーターがあります。

```
     // プロシージャー FmtCust のプロトタイプ (定義仕様書の PR に
     // に注意してください)。2 つの出力パラメーターがあります。
    D FmtCust         PR
    D  Name                         100A
    D  Address                      100A
```

図 13. FmtCust プロシージャーのプロトタイプ

アドレスを様式設定するため、アプリケーションはプロシージャー FmtAddr を呼び出します。 FmtAddr には複数の入力パラメーターがあり、可変文字フィールドを戻します。 図 14 は FmtAddr のプロトタイプを示しています。

```
     //-------------------------------------------------------
     // FmtAddr - アドレスを様式設定するためのプロシージャー
     //-------------------------------------------------------
   D FmtAddr         PR           100A     VARYING
   D   streetNum                   10I 0   CONST
   D   streetName                  50A     CONST
   D   city                        20A     CONST
   D   state                       15A     CONST
   D   zip                          5P 0   CONST
```

図 14. FmtAddr プロシージャーのプロトタイプ

プログラムまたはプロシージャーがプロトタイプの場合には、CALLP で、あるいは戻り値で使用したい場合は式で、これを呼び出します。プロトタイプの名前に続くリストにパラメーターを渡します。例えば、*名前 (パラメーター 1 : パラメーター 2 : ...)* です。

図 15 は FmtCust への呼び出しを示します。 37 ページの図 13 に、上で示された OUTPUT パラメーターの名前は CALL ステートメントのものと一致しないことに注意してください。プロトタイプのパラメーター名は文書化の目的だけのものです。プロトタイプは、呼び出しインターフェースの属性を*記述*する のに役立ちます。呼び出しパラメーターの実際の定義は、プロシージャー自身の内部で実行されます。

```
     C                   CALLP     FmtCust(RPTNAME : RPTADDR)
```

図 15. FmtCust プロシージャーの呼び出し

プロトタイプ呼び出しを使用して (同じ構文で) 次のものを呼び出すことができます。

- 実行時にシステム上にあるプログラム
- 同じプログラムまたはサービス・プログラムにバインドされた、他のモジュールまたはサービス・プログラム内のエクスポートされたプロシージャー
- 同じモジュール内のサブプロシージャー

FmtCust は FmtAddr を呼び出してアドレスを様式設定します。 FmtCust は戻り値を使用する必要があるために、 FmtAddr への呼び出しは式の中で行われます。 図 16 はこの呼び出しを示しています。

```
     //-------------------------------------------------------------
     //   FmtAddr プロシージャーを呼び出してアドレスを処理します。
     //-------------------------------------------------------------
     Address = FmtAddress (STREETNUM : STREETNAME :
                           CITY : STATE : ZIP);
```

図 16. FmtAddr プロシージャーの呼び出し

上の図のように、値を戻すためのプロシージャーの使用によって、必要なユーザー定義の機能を書くことができます。さらに、プロトタイプ呼び出しインターフェースを使用すれば、パラメーターを渡す方法の選択肢が広がります。

- プロトタイプ・パラメーターは、次の複数の方法で渡すことができます。参照によって、値によって (プロシージャーの場合のみ)、または読み取り専用の参照によって、渡すことができます。RPG のデフォルトの方法は参照によって渡すことです。しかし、値によってまたは読み取り専用参照によって渡すことは、渡すパラメーターに多くのオプションを与えます。
- プロトタイプが、指定のパラメーターに使用可能であることを指示している場合には、次の 1 つ以上を実行できる場合があります。
  - *OMIT を渡す
  - パラメーター全体をそのままにしておく

– 指定されているよりも短いパラメーターを渡す (文字と図形パラメーターの場合、および配列パラメーターの場合)

複数プロシージャーのあるモジュール例

ここで、複数プロシージャー・モジュールの例を見てみます。この「ミニ・アプリケーション・プログラム」では、勘定が遅れているすべての得意先の報告書を作成するために、プログラム ARRSRPT を作成しています。モジュールとして基本報告書を作成するので、必要な場合には他のモジュールとバインドすることができます。このモジュールに必要な次の 2 つのメインタスクがあります。

1. 得意先ファイルからの勘定のレコードが滞納しているかどうかを判別する。
2. データを報告書に適した様式に様式設定する。

各タスクをサブプロシージャーとしてコーディングすることを決定します。概念的には、モジュールは 図 17 に示されたものと類似したものになります。

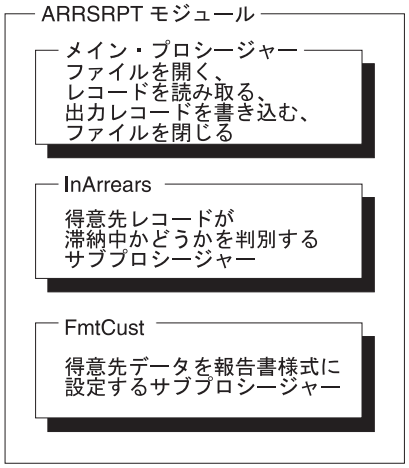


図 17. ARRSRPT モジュールの構成要素

ここで、 40 ページの図 18 に示されている最初のサブプロシージャー InArrears を考えてみます。InArrears は、現行レコードが滞納中であるかどうかを調べるためにメイン・プロシージャーによって呼び出されます。

ヒント

グローバル・フィールドを使用するサブプロシージャーをコーディングする時には、グローバルである項目を表示する命名規則を確立する必要がある場合があります。この例では、大文字のフィールド名は DDS フィールドを指示しています。別のオプションは、グローバルの有効範囲を指示する、接頭部 'g_' または何か他のストリングとなります。

レコードが滞納中である場合には、サブプロシージャーはメイン・プロシージャーに '1' を戻します。

```
 //-------------------------------------------------------------
 // InArrears
 //
 // パラメーター: (なし)
 // グローバル:   DUEDATE, AMOUNT, CurDate
 //
 // 戻り:         得意先が滞納中であれば '1'
 //-------------------------------------------------------------
P InArrears       B                   1
D InArrears       PI             1A   2
 // ローカルの宣言
D DaysLate        S             10I 0  3
D DateDue         S               D    3
 // プロシージャー本体
 /free
     DateDue = %date (DUEDATE: *ISO);
     DaysLate = %diff (CurDate: DateDue: *d);
     // 入力ファイルの中のデータは別のタイプの
     // コンピューターからのデータで、AMOUNTC フィールドは、
     // 数値を含む文字ストリングです。
     // このストリングは、印刷の場合は数値の AMOUNT フィールドに
     // 変換しなければなりません。
     AMOUNT = %dec(AMOUNTC : 31 : 9);
     if DaysLate > 60 AND AMOUNT > 100.00;
        return '1';                  4
     endif;
     return '0';                     4  5
 /end-free
P InArrears       E                   1
```

図 18. サブプロシージャー InArrears のソース

図 18 はすべてのサブプロシージャーに共通な主要な要素を示しています。

1 すべてサブプロシージャーはプロシージャー仕様書で始まり、終わります。

2 開始プロシージャー仕様書 (プロシージャー仕様書の 24 桁目に B) の後に、プロシージャー・インターフェース定義をコーディングします。戻り値 (ある場合) は PI 仕様書で定義します。PI 仕様書の後にパラメーターがリストされます。

3 サブプロシージャーで使用される変数またはプロトタイプは、プロシージャー・インターフェース定義の後に定義します。

4 戻り値 (指定した場合) は、RETURN 命令のある呼び出し元に戻されます。

5 そのレコードが滞納中でない場合は、サブプロシージャーは '0' をメイン・プロシージャーに戻します。

すべてのサブプロシージャーの場合、およびプロトタイプ入り口パラメーターのあるメイン・プロシージャーの場合も、プロシージャー・インターフェースを定義する必要があります。**プロシージャー・インターフェース定義**は、プロシージャーの定義内のプロトタイプ情報の反復です。これは、プロシージャーの入り口パラメーターを定義するために使用されます。プロシージャー・インターフェース定義は、プロシージャーの内部定義が外部定義 (プロトタイプ) と一貫性があることを確認するためにも使用されます。InArrears の場合には、入り口パラメーターはありません。

41 ページの図 19 に示す次のサブプロシージャー FmtCust について考えてみます。FmtCust は、レコードの関連フィールドを最終報告書の出力レコードに様式設定す

るために、 ARRSRPT によって呼び出されます。(レコードは滞納している勘定を表しています。) FmtCust は、グローバル・データを使用しているので、入力パラメーターをもっていません。これはデータを 2 つの出力フィールドに様式設定します。名前用に 1 つとアドレス用に 1 つです。

```
 //-------------------------------------------------------------
 // FmtCust は、CUSTNAME、CUSTNUM、STREETNAME などを
 // 読み取り可能な様式に様式設定します。
 //
 // パラメーター: Name     (出力)
 //               Address  (出力)
 // グローバル:   CUSTNAME、CUSTNUM、STREETNUM、STREETNAME、CITY
 //               STATE、ZIP
 //-------------------------------------------------------------

P FmtCust         B
D FmtCust         PI
D  Name                        100A
D  Address                     100A

 /free
     //-------------------------------------------------------------
     // CUSTNAME と CUSTNUM は次のように様式設定されます。
     // A&P Electronics     (得意先番号 157)
     //-------------------------------------------------------------
     Name = CUSTNAME + ' ' + '(得意先番号 '
                      + %char(CUSTNUM) + ')';
     //-------------------------------------------------------------
     //   FmtAddr プロシージャーを呼び出してアドレスを処理します。
     //-------------------------------------------------------------

 Address = FmtAddress (STREETNUM : STREETNAME :
                               CITY : STATE : ZIP);
 /end-free
P FmtCust         E
```

図 19. サブプロシージャー FmtCust のソース

最後に、このアプリケーション・プログラムの最後のサブプロシージャー FmtAddr を考えてみます。 39 ページの図 17 に示すように、FmtAddr は ARRSRPT モジュールにはないことに注意してください。 FmtAddr は FMTPROCS という別のモジュールに入れることにしました。 FMTPROCS は、他のモジュールで必要となる変換プロシージャーを入れるユーティリティー・モジュールです。

42 ページの図 20 に、モジュール FMTPROCS のソースを示します。これはプロトタイプ・プロシージャーであるため、使用可能なプロトタイプが必要です。プロトタイプは共用できるので、プロトタイプを /COPY ファイルに入れました。

```
     //=============================================================
     // モジュール FMTPROCS のソース。このモジュールは NOMAIN という
     // キーワードが示しているとおり、メイン・プロシージャーを持ちません。
     //=============================================================
    H NOMAIN
     //-------------------------------------------------------------
     // プロトタイプ・プロシージャーを持つモジュールのそれぞれについて
     // プロトタイプが使用可能である必要があります。
     // /COPY を指定すると、FmtAddr 用のプロトタイプを呼び込みます。
     //-------------------------------------------------------------
    D/COPY QRPGLESRC,FMTPROC_P
    P FmtAddr         B                   EXPORT
    D FmtAddr         PI           100A   VARYING
    D   streetNum                  10I 0  CONST
    D   streetName                 50A    CONST
    D   city                       20A    CONST
    D   state                      15A    CONST
    D   zip                         5P 0  CONST
     /free
         //-----------------------------------------------------------
         // STREETNUM、STREETNAME、CITY、STATE、および ZIP は次のように
         // 形式設定されます
         //   27 Garbanzo Avenue, Smallville IN 51423
         //-----------------------------------------------------------
         return  %char(streetNum) + ' ' + %trimr(streetName)
                      + ', ' + %trim(city) + ' ' + %trim(state)
                     + ' ' + %editc(zip : 'X');
    P FmtAddr         E
```

図 20. サブプロシージャー FmtAddr を含む、モジュール FMTPROCS のソース

FMTPROCS は **NOMAIN モジュール**です。つまり、サブプロシージャーだけから構成されていて、メイン・プロシージャーはありません。 NOMAIN モジュールは、モジュールのために作成されるサイクル・コードがないために、高速でコンパイルされ、少ない記憶域しか必要としません。制御仕様書で NOMAIN キーワードをコーディングすることによって NOMAIN モジュールを指定します。NOMAIN モジュールの詳細については、48 ページの『プログラムの作成』を参照してください。

ARRSRPT プログラム全体

ARRSRPT プログラムは、ARRSRPT と FMTPROCS の 2 つのモジュールから構成されます。 43 ページの図 21 はミニ・アプリケーションの異なる部分を示します。

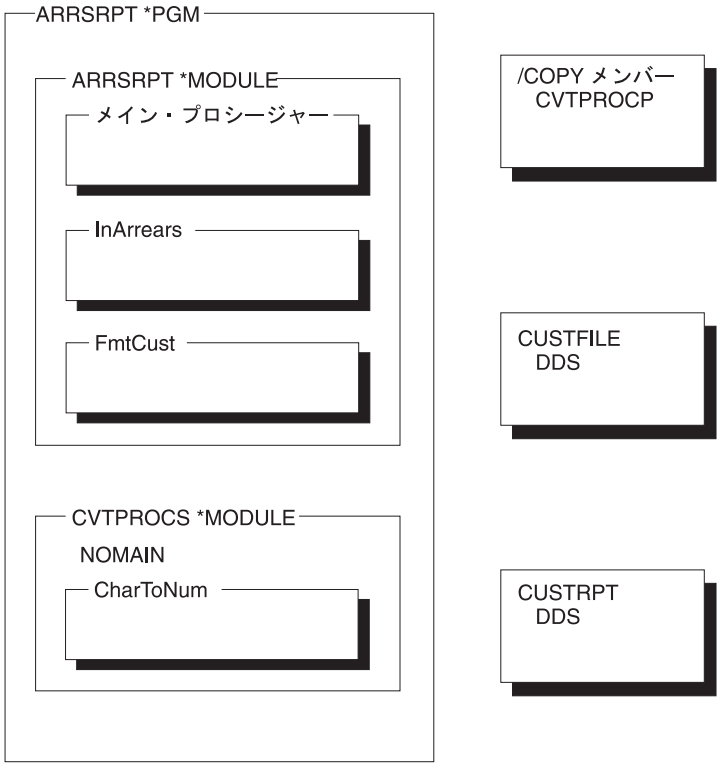


図 21. ARRSRPT アプリケーション・プログラム

44 ページの図 22 は ARRSRPT モジュールのソース全体を示したものです。

```
     //=================================================================
     // モジュール ARRSRPT のソース。メイン・プロシージャーと 2 つの
     // サブプロシージャー InArrears および FmtCust が含まれています。
     //
     // 関連モジュール:  CVTPROCS  (InArrears によって呼び出された CharToNum)
     //=================================================================
     //-----------------------------------------------------------------
     // ファイル
     //
     // CUSTFILE - 得意先情報が入っている
     // CUSTRPT  - プリンター・ファイル (様式 ARREARS を使用)
     //-----------------------------------------------------------------
    FCUSTFILE  IP   E             DISK
    FCUSTRPT   O    E             PRINTER
     *-----------------------------------------------------------------*
     * プロトタイプ
     *-----------------------------------------------------------------*
#    /COPY QRPGLE,FMTPROC_P
     *-----------------------------------------------------------------*
     * 得意先が滞納中の場合に、InArrears は '1' を戻します
     *-----------------------------------------------------------------*
    D InArrears       PR             1A
     *-----------------------------------------------------------------*
     * FmtCust は CUSTNAME、CUSTNUM、STREETNAME などを
     * 読み取り可能な様式に様式設定します。
     *-----------------------------------------------------------------*
    D FmtCust         PR
    D  Name                         100A
    D  Address                      100A
```

図 22. ILE RPG ARRSRPT モジュールの全ソース (1/3)

```
     *------------------------------------------------------------*
     * グローバル定義
     *------------------------------------------------------------*
    D CurDate         S               D
    ICUSTREC       01
     *------------------------------------------------------------*
     * メイン・プロシージャー
     *------------------------------------------------------------*
    C                   IF        InArrears = '1'
    C                   CALLP     FmtCust(RPTNAME : RPTADDR)
    C                   EVAL      RPTNUM = CUSTNUM
    C                   WRITE     ARREARS
    C                   ENDIF
    C     *INZSR        BEGSR
    C                   MOVEL     UDATE         CurDate
    C                   ENDSR

      *------------------------------------------------------------*
      * サブプロシージャー
      *------------------------------------------------------------*
      //--------------------------------------------------------------
      // InArrears
      //
      // パラメーター: (なし)
      // グローバル:   DUEDATE, AMOUNT, CurDate
      //
      // 戻り:         得意先が滞納中であれば '1'
      //--------------------------------------------------------------
    P InArrears       B
    D InArrears       PI             1A
      // ローカルの宣言
    D DaysLate        S             10I 0
    D DateDue         S               D
      // プロシージャー本体
     /free
         DateDue = %date (DUEDATE: *ISO);
         DaysLate = %diff (CurDate: DateDue: *d);
          // 入力ファイルの中のデータは別のタイプの
           // コンピューターからのデータで、AMOUNTC フィールドは、
           // 数値を含む文字ストリングです。
           // このストリングは、印刷の場合は数値の AMOUNT フィールドに
           // 変換しなければなりません。
          AMOUNT = %dec(AMOUNTC : 31 : 9);
         if DaysLate > 60 AND AMOUNT > 100.00;
            return '1';
         endif;
         return '0';
     /end-free
    P InArrears       E
```

図 22. ILE RPG ARRSRPT モジュールの全ソース (2/3)

```
 //-------------------------------------------------------------
 // FmtCust は、CUSTNAME、CUSTNUM、STREETNAME などを
 // 読み取り可能な様式に様式設定します。
 //
 // パラメーター:   Name     (出力)
 //                 Address  (出力)
 // グローバル:     CUSTNAME、CUSTNUM、STREETNUM、STREETNAME、CITY
 //                 STATE、ZIP
 //-------------------------------------------------------------

P FmtCust         B
D FmtCust         PI
D  Name                        100A
D  Address                     100A

 /free
     //-----------------------------------------------------------
     // CUSTNAME と CUSTNUM は次のように様式設定されます。
     // A&P Electronics     (得意先番号 157)
     //-----------------------------------------------------------
     Name = CUSTNAME + ' ' + '(得意先番号 '
                       + %char(CUSTNUM) + ')';
     //-----------------------------------------------------------
     //   FmtAddr プロシージャーを呼び出してアドレスを処理します。
     //-----------------------------------------------------------

 Address = FmtAddress (STREETNUM : STREETNAME :
                               CITY : STATE : ZIP);
 /end-free
P FmtCust         E
```

図 22. ILE RPG ARRSRPT モジュールの全ソース (3/3)

ARRSRPT について次のことに注意してください。

- 定義仕様書はプロトタイプ呼び出しのプロトタイプから始まります。/COPY ファイルは、呼び出されたプロシージャー FmtAddr のプロトタイプを提供するために使用されます。

  プロトタイプは最初にある必要はありませんが、いろいろなタイプの定義の一貫性のある順序を確立する必要があります。
- 日付フィールド CurDate はグローバル・フィールドであり、モジュール内のすべてのプロシージャーがそれをアクセスできることを意味します。
- メイン・プロシージャーは単純に続けます。これには、2 つのメインタスクである入出力および初期設定ルーチン用の演算仕様書が含まれています。
- メイン・プロシージャーに続く各サブプロシージャーには、タスクの 1 つの詳細が含まれています。

プログラム ARRSRPT のサンプル出力を、47 ページの図 23 で示します。

```
得意先番号: 00001
    ABC Electronics      (得意先番号 1)
    15 Arboreal Way, Treetop MN 12345
    未処理の金額:          $1234.56   期日: 1995-05-01

得意先番号: 00152
    A&P Electronics      (得意先番号 152)
    27 Garbanzo Avenue, Smallville MN 51423
    未処理の金額:         $26544.50   期日: 1995-02-11
```

図 23. ARRSRPT の出力

図 24 および図 25 はそれぞれ、ファイル CUSTFILE および CUSTRPT の DDS ソースを示します。

```
A*=================================================================*
A* ファイル名     : CUSTFILE
A* 関連プログラム : ARRSRPT
A* 説明           : これは物理ファイル CUSTFILE です。これには
A*                   CUSTREC と呼ばれる 1 レコード様式があります。
A*=================================================================*
A* 得意先マスター・ファイル -- CUSTFILE
A          R CUSTREC
A            CUSTNUM        5  0       TEXT('CUSTOMER NUMBER')
A            CUSTNAME      20          TEXT('CUSTOMER NAME')
A            STREETNUM      5  0       TEXT('CUSTOMER ADDRESS')
A            STREETNAME    20          TEXT('CUSTOMER ADDRESS')
A            CITY          20          TEXT('CUSTOMER CITY')
A            STATE          2          TEXT('CUSTOMER STATE')
A            ZIP            5  0       TEXT('CUSTOMER ZIP CODE')
A            AMOUNTC       15          TEXT('AMOUNT OUTSTANDING')
A            DUEDATE       10          TEXT('DATE DUE')
```

図 24. CUSTFILE の DDS

```
A*=================================================================*
A* ファイル名     : CUSTRPT
A* 関連プログラム : ARRSRPT
A* 説明           : これはプリンター・ファイル CUSTRPT です。これには
A*                   ARREARS と呼ばれる 1 レコード様式があります。
A*=================================================================*
A          R ARREARS
A                                  2  6
A                                     '得意先番号:'
A            RPTNUM         5  0   2 23
A                                     TEXT('CUSTOMER NUMBER')
A            RPTNAME      100A     3 10
A                                     TEXT('CUSTOMER NAME')
A            RPTADDR      100A     4 10
A                                     TEXT('CUSTOMER ADDRESS')
A                                  5 10'未処理の金額:'
A            AMOUNT        10  2   5 35EDTWRD('       $0.  ')
A                                     TEXT('AMOUNT OUTSTANDING')
A                                  5 50'期日:'
A            DUEDATE       10      5 60
A                                     TEXT('DATE DUE')
```

図 25. CUSTRPT の DDS

コーディング上の考慮事項

この項には、複数プロシージャー・モジュールのあるアプリケーションの設計を始める前に注意すべきいくつかの考慮事項があります。項目は次のカテゴリーにグループ化されています。

- その他
- プログラムの作成
- メイン・プロシージャー
- サブプロシージャー

一般的な考慮事項

- 複数プロシージャーのあるモジュールをコーディングする時には、ユーザーのアプリケーション・プログラムに必要となる可能性があるプロトタイプを基本的に含めておくために、 /COPY ファイルを利用します。サービス・プログラムを作成している場合には、サービス・プログラムとプロトタイプ (ある場合) の両方を提供する必要があります。
- アプリケーション・プログラムの保守は、各構成要素が最新のレベルにあること、および変更が別の部分に影響しないよう保証することを意味します。ユーザーのアプリケーションの保守のために、アプリケーション開発管理のようなツールの使用を考えることができます。

  例えば、別のプログラマーがプロトタイプを含む /COPY ファイルに対して変更したと仮定します。ユーザーのアプリケーション・プログラムの再作成を要求した時に、/COPY ファイルを使用しているモジュールまたはプログラムが自動的にコンパイルし直されます。/COPY ファイルに対する変更がアプリケーション・プログラムの呼び出しまたはプロシージャー・インターフェースに影響する場合には、高速で検索されます。コンパイル・エラーがある場合には、これらのエラーを避けるために、プロトタイプに対する変更を受け入れるかどうか、または呼び出しインターフェースに対する変更を受け入れるかどうかを決定することができます。

プログラムの作成

- モジュールがメイン・プロシージャーをもたないよう指定した場合には、プログラムの作成に CRTBNDRPG コマンドを使用できません。(制御仕様書に NOMAIN キーワードが指定された場合には、モジュールはメイン・プロシージャーをもちません。) これは、CRTBNDRPG コマンドがモジュールにプログラム入力プロシージャーを含めることを要求し、メイン・プロシージャーしかプログラム入力プロシージャーとすることができないためです。
- 同様に、プログラムを作成するために CRTPGM を使用している時には、プログラム入力プロシージャーをもっていないので NOMAIN モジュールを入り口モジュールとすることができないことに注意してください。
- デフォルトの OPM 活動化グループで実行するために作成されたプログラム (CRTBNDRPG コマンドで DFTACTGRP(*YES) を指定して) は、バインドされたプロシージャー呼び出しを含めることはできません。

メイン・プロシージャーの考慮事項

- メイン・プロシージャーは使用可能な仕様書の完全セット (P 仕様書を除く) を用いる唯一のプロシージャーであるために、モジュール内のすべてのプロシージャーの環境をセットアップするために、使用する必要があります。
- メイン・プロシージャーは常にエクスポートされ、これは、プログラム内の他のプロシージャーがバインド呼び出しの使用によってメイン・プロシージャーを呼び出すことができることを意味します。
- メイン・プロシージャーの呼び出しインターフェースは次の 2 つの方法の 1 つで定義することができます。

  1. プロトタイプおよびプロシージャー・インターフェースの使用
  2. プロトタイプなしで *ENTRY PLIST の使用

- *ENTRY PLIST の機能は、プロトタイプ呼び出しインターフェースに類似しています。しかし、プロトタイプ呼び出しインターフェースは、コンパイル時にパラメーター検査を行うのでさらに強力です。メイン・プロシージャーをプロトタイプする場合には、プロトタイプ定義で EXTPROC または EXTPGM キーワードのいずれかを指定して呼び出す方法を指定します。EXTPGM を指定した場合には、外部プログラム呼び出しが使用されます。EXTPROC を指定した場合、またはどちらのキーワードも指定しない場合には、プロシージャー呼び出しを使用して呼び出されます。
- メイン・プロシージャーの戻り値を定義することはできず、そのパラメーターが値によって渡されるよう指定することもできません。

サブプロシージャーの考慮事項

- 演算命令のどれもサブプロシージャーでコーディングすることができます。しかし、すべてのファイルはグローバルに定義しなければならないので、すべての入出力仕様書はメイン・ソース・セクションで定義しなければなりません。同様に、サブプロシージャーで使用することができますが、すべてのデータ域はメイン・プロシージャーで定義しなければなりません。
- 制御仕様書は、モジュール全体を制御するので、メイン・ソース・セクションでのみコーディングすることができます。
- サブプロシージャーは再帰的に呼び出すことができます。各再帰呼び出しによって、プロシージャーの新しい呼び出しが呼び出しスタックに入れられることになります。新しい呼び出しには自動記憶域のすべてのデータ項目に対し新しい記憶域があり、その記憶域は局所的なために他の呼び出しでは使用できません。(定義に STATIC キーワードを指定しないかぎり、サブプロシージャーで定義されるデータ項目は自動記憶域を使用します。)

  以前の呼び出しと関連した自動記憶域は後からの呼び出しによって影響は受けません。すべての呼び出しは同じ静的記憶域を共用しているので、後からの呼び出しは静的記憶域の変数によって保留されている値に影響を与えることはあります。

  再帰は、適切に理解した時に強力なプログラミング手法となることができます。
- サブプロシージャーの実行時の動きは、サブプロシージャーのためのサイクル・コードがないために、メイン・プロシージャーの動きとは一部異なります。
  - サブプロシージャーが終了した時には、呼び出し元に単に戻るだけです。ファイルのクローズなどの通常の終了活動のいずれも、サブプロシージャー自身と

関連したメイン・プロシージャーが終了するまでは起こりません。アプリケーション・プログラム終了時のプログラム入力プロシージャーによって、およびプログラム入力プロシージャー用に使用可能な取り消し処理プログラムによって、両方とも呼び出される『終結処置』サブプロシージャーをコーディングする必要があります。

代替処理は、暗黙のファイル・オープンまたはデータ域ロックがないように、また任意のサブプロシージャー内でオープンがクローズと対応し、IN が OUT と、CRT<temp obj> が DLT<temp obj> と対応するように、NOMAIN モジュールをコーディングすることです。この代替処理は、メイン・プロシージャーが活動状態になっていない時に、サブプロシージャーを活動状態にすることができるモジュールに適用されます。

– サブプロシージャー内の例外処理は、サブプロシージャーのデフォルトの例外処理プログラムがないために、基本的にメイン・プロシージャーと異なっています。結果として、メイン・プロシージャー用にデフォルトの処理プログラムが呼び出される状況は、サブプロシージャーの異常終了と対応しています。

詳細情報

ここで説明されているトピックの詳細を検索するためには、次のリストを参照してください。

メイン・プロシージャー

トピック	参照先
例外処理	276 ページの『メイン・プロシージャー内の例外処理プログラム』
メイン・プロシージャーの終了	166 ページの『メイン・プロシージャーからの戻り』

サブプロシージャー

トピック	参照先
定義	「*WebSphere Development Studio: ILE RPG 解説書*」のサブプロシージャーの章
NOMAIN モジュール	86 ページの『NOMAIN モジュールの作成』
例外処理	277 ページの『サブプロシージャー内の例外処理プログラム』
プロシージャー仕様書	「*WebSphere Development Studio: ILE RPG 解説書*」のプロシージャー仕様書の章
プロシージャー・インターフェース	「*WebSphere Development Studio: ILE RPG 解説書*」の、データおよびプロトタイプの定義の章
サブプロシージャーの終了	169 ページの『サブプロシージャーからの戻り』

プロトタイプ呼び出し

トピック	参照先
自由形式の呼び出し	145 ページの『プロトタイプ呼び出しの使用』
一般情報	「*WebSphere Development Studio: ILE RPG 解説書*」の第 24 章
パラメーターの受け渡し	147 ページの『プロトタイプ・パラメーターの受け渡し』
プロトタイプ	「*WebSphere Development Studio: ILE RPG 解説書*」の、データおよびプロトタイプの定義の章

第 2 部 ILE RPG アプリケーション・プログラムの作成と実行

この部では、ILE RPG プログラムの作成と実行に必要な情報を提供します。以下の方法を説明します。

- ソース・ステートメントの入力
- モジュールの作成
- コンパイラー・リストの解釈
- プログラムの作成
- サービス・プログラムの作成
- プログラムの実行
- パラメーターの引き渡し
- 実行時の管理
- 他のプログラムまたはプロシージャーの呼び出し

WebSphere Development Studio Client for iSeries を使用します。これは推奨される方法であり、ILE RPG アプリケーションの作成および実行についての説明は、製品のオンライン・ヘルプに記載されています。

以降に示すページで、多くの ILE 用語および概念が簡単に説明されています。これらの用語や概念については、「*ILE 概念*」で詳しく説明しています。

第 5 章 ソース・ファイルの使用

この章では、RPG ソース・ステートメントの入力に必要な事項について説明します。また、このステップを完結するのに必要なツールについても簡単に説明します。

RPG ソース・ステートメントをシステムに入力するには、次のいずれかの方法を使用します。

- SEU を使用した対話式入力
- Remote Systems LPEX エディターを使用した対話式入力

WebSphere Development Studio Client for iSeries を使用します。これは推奨される方法であり、ソースの編集に関する説明は、製品のオンライン・ヘルプに記載されています。 Remote Systems LPEX エディターを使用すると、プログラムの編集タスクが単純化されます。このエディターは、ワークステーション上または iSeries サーバー上のソース・ファイルにアクセスできます。コンパイルでエラーが発生した場合には、コンパイラーのメッセージからソースを含むエディターに直接ジャンプできます。問題のソース・ステートメントにカーソルが置かれた状態でエディターが開き、訂正を行うことができます。

最初に、ソース・ステートメントを QRPGLESRC と呼ばれるファイルに入れる必要があります。ファイル QRPGLESRC の新しいメンバーは、自動的に RPGLE のデフォルトのタイプを受け取ります。さらに、モジュールを作成しプログラム・オブジェクトにそれをバインドする ILE RPG コマンドのデフォルトのソース・ファイルは QRPGLESRC です。IBM® は、ライブラリー QGPL のソース・ファイル QRPGLESRC を提供します。このファイルのレコード長は 112 桁です。

注: ソース・ステートメントの入力時には、大文字と小文字を混ぜて使用することができます。しかし、ILE RPG コンパイラーは、ほとんどのソース・ステートメントをコンパイルの時に大文字に変換します。リテラル、配列データ、あるいは表データは変換されません。

ソース・ファイルには次の 2 つの形式があります。

1. ソース物理ファイル

2. IFS (Integrated File System) ファイル

ソース物理ファイルの使用

ライブラリーおよびソース物理ファイルの作成

ソース・ステートメントは、ソース物理ファイルのメンバー中に入れられます。プログラムの入力を可能にするには、ライブラリーおよびソース物理ファイルが必要です。

ライブラリーを作成するためには、CRTLIB コマンドを使用してください。ソース物理ファイルを作成するためには、ソース物理ファイル作成 (CRTSRCPF) コマンド

を使用してください。ファイルの推奨レコード長は 112 桁です。このレコード長は、図 26 に示すように新しい ILE RPG 構造を考慮に入れています。

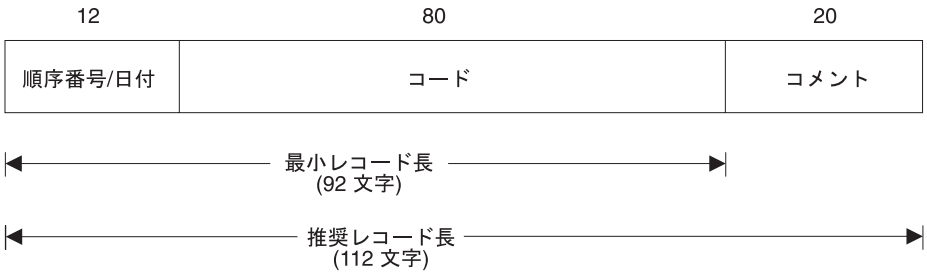


図 26. ILE RPG レコード長の説明

ソース物理ファイルのシステムのデフォルトの値は 92 桁なので、明示的に最小レコード長 112 桁を指定する必要があります。92 桁より小さい値を指定した場合には、ソース・コードを切り捨てることになるのでプログラムはコンパイルされません。

ライブラリーおよびソース物理ファイルの作成情報は、「*AS/400 適用業務開発ツールセット AS/400 用 原始ステートメント入力ユーティリティー 使用者の手引きと参照*」マニュアルと「*AS/400 プログラム開発管理機能 (PDM)*」マニュアルを参照してください。

原始ステートメント入力ユーティリティー (SEU) の使用

ソース・ステートメントを入力するために、原始ステートメント入力ユーティリティー (SEU) を使用することができます。また、SEU は構文検査だけでなく異なる仕様書のテンプレートのプロンプトを提供します。SEU の開始には、STRSEU (原始ステートメント入力ユーティリティー開始) コマンドを使用します。これ以外の SEU の開始方法および使用方法については、「*AS/400 適用業務開発ツールセット AS/400 用 原始ステートメント入力ユーティリティー 使用者の手引きと参照*」マニュアルを参照してください。

ソース・ファイルを QRPGLESRC と名前を付けた場合には、SEU は新しいメンバーの編集セッションを開始する時にソース・タイプを RPGLE に自動的に設定します。そうでない場合は、メンバーを作成する時に RPGLE を指定する必要があります。

STRSEU の入力の後にプロンプトの必要がある場合には、F4 キーを押してください。STRSEU 画面が表示され、パラメーターをリストし、デフォルトの値を提示します。プロンプトを要求する前にパラメーター値を指定した場合には、その値の入った画面が表示されます。

次の例では、マスター・ファイルから社員情報を印刷するプログラムのためのソース・ステートメントを入力します。この例では、以下のことを行う方法を示しています。

- ライブラリーの作成
- ソース物理ファイルの作成
- SEU 編集セッションの開始

- ソース・ステートメントの入力

1. MYLIB というライブラリーを作成には、次の入力をします。

```
CRTLIB LIB(MYLIB)
```

   CRTLIB コマンド が MYLIB というライブラリーを作成します。

2. QRPGLESRC というソース物理ファイルを作成するには、次の入力をします。

```
CRTSRCPF FILE(MYLIB/QRPGLESRC) RCDLEN(112)
TEXT('Source physical file for ILE RPG programs')
```

   CRTSRCPF コマンドは、ライブラリー MYLIB にソース物理ファイル QRPGLESRC を作成します。

3. 編集セッションを開始し、ソース・メンバー EMPRPT を作成するには、次のように入力してください。

```
STRSEU SRCFILE(MYLIB/QRPGLESRC)
SRCMBR(EMPRPT)
TYPE(RPGLE) OPTION(2)
```

   OPTION(2) の入力は、新しいメンバーのセッションを開始したいことを指示します。STRSEU コマンドは新しいメンバー EMPRPT をライブラリー MYLIB のファイル QRPGLESRC に作成し、編集セッションを開始します。

   SEU 編集画面が図 27 に示すように表示されます。 6 桁目 (仕様書タイプ) が左端になるように、画面が自動的にシフトされることに注意してください。

```
桁 . . . :   6  76           編集                        MYLIB/QRPGLESRC
SEU==>  ____________________________________________________________      EMPRPT
FMT H  HKEYWORDS++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        ********** データの始め ************************************
'''''''
'''''''
'''''''
'''''''
'''''''
'''''''
'''''''
'''''''
'''''''
'''''''
'''''''
'''''''
'''''''
'''''''
'''''''
        ********** データの終わり ************************************
 F3=終了    F4=プロンプト    F5=最新表示    F9=コマンドの複写    F10=カーソル
 F16=検索の反復        F17=変更の反復          F24=キーの続き
メンバー EMPRPT が MYLIB/QRPGLESRC に追加された。                          +
```

図 27. 新しいメンバーの編集画面

4. プロンプトを出すためには、次の SEU 接頭部コマンドを使って、SEU 編集画面に以下のソース・ステートメントを入力してください。
   - IPF — ファイル仕様書の場合
   - IPD — 定義仕様書の場合
   - IPI — 入力仕様書の場合

- IPC — 演算仕様書の場合
- IPCX — 拡張演算項目 2 のある演算仕様書の場合
- IPO — 出力仕様書の場合
- IPP — 出力仕様書継続の場合
- IPPR — プロシージャー仕様書の場合

```
     *===============================================================*
     * モジュール名:    EMPRPT
     * 関連ファイル:    EMPMST   (物理ファイル)
     *                  QSYSPRT  (PRINTER ファイル)
     * 説明:            このプログラムはファイル EMPMST から
     *                  社員情報を印刷します。
     *===============================================================*
    FQSYSPRT   O    F   80        PRINTER
    FEMPMST    IP   E           K DISK
    D TYPE             S              8A
    D EMPTYPE          PR             8A
    D   CODE                          1A
    IEMPREC        01
    C                    EVAL      TYPE = EMPTYPE(ETYPE)
    OPRINT     H    1P                     2  6
    O                                          50 'EMPLOYEE INFORMATION'
    O          H    1P
    O                                          12 'NAME'
    O                                          34 'SERIAL #'
    O                                          45 'DEPT'
    O                                          56 'TYPE'
    O          D    01
    O                         ENAME           20
    O                         ENUM            32
    O                         EDEPT           45
    O                         TYPE            60
     * プロシージャー EMPTYPE は、パラメーター CODE によって
     * 示された社員タイプを表すストリングを戻します。
    P EMPTYPE          B
    D EMPTYPE          PI             8A
    D   CODE                          1A
    C                    SELECT
    C                    WHEN      CODE = 'M'
    C                    RETURN    'Manager'
    C                    WHEN      CODE = 'R'
    C                    RETURN    'Regular'
    C                    OTHER
    C                    RETURN    'Unknown'
    C                    ENDSL
    P EMPTYPE          E
```

図 28. EMPRPT メンバーのソース・ステートメント

5. F3 (終了) を押して終了画面に行きます。EMPRPT を保管するためには、Y (はい) をタイプしてください。

   メンバー EMPRPT が保管されます。

59 ページの図 29 は、EMPRPT ソースによって参照される DDS を示したものです。

```
A*********************************************************************
A* 説明:  これは物理ファイル EMPMST の DDS です。                    *
A*        これには 1 つのレコード様式 EMPREC が入っています。        *
A*        このファイルには、会社の各社員ごとに 1 レコードが          *
A*        入っています。                                             *
A*********************************************************************
A*
A          R EMPREC
A            ENUM           5  0       TEXT('EMPLOYEE NUMBER')
A            ENAME         20          TEXT('EMPLOYEE NAME')
A            ETYPE          1          TEXT('EMPLOYEE TYPE')
A            EDEPT          3  0       TEXT('EMPLOYEE DEPARTMENT')
A            ENHRS          3  1       TEXT('EMPLOYEE NORMAL WEEK HOURS')
A          K ENUM
```

図 29. EMPRPT の DDS

このソース・ステートメントからプログラムを作成するには、 DFTACTGRP(*NO) を指定して CRTBNDRPG コマンドを使用してください。

SQL ステートメントの使用

DB2 UDB for iSeries® データベースは、SQL ステートメントをユーザーのプログラム・ソースに組み込むことによって、ILE RPG プログラムからアクセスすることができます。SQL ステートメントを入れるには、次の規則を使用してください。

- SQL ステートメントは演算仕様書に入れます。
- SQL ステートメントは、7 ～ 15 桁目に /EXEC SQL (7 桁目に /) の区切り文字を使用して開始します。
- SQL ステートメントの入力は、開始区切り文字と同一の行から始めることができます。
- ステートメントを後続の行に続けるためには、継続記入行区切り文字 (7 桁目に +) を使用します。
- SQL ステートメントの終わりを示すには、7 ～ 15 桁目に /END-EXEC (7 桁目に /) の終了区切り文字を使用します。

注: SQL ステートメントは、プログラム内で 80 桁を超えることはできません。

60 ページの図 30 は、組み込み SQL ステートメントの例を示したものです。

```
...+....1....+....2....+....3....+....4....+....5....+....6....+....7..
     C
     C              (ILE RPG 演算命令)
     C
     C/EXEC SQL     (開始区切り文字)
     C+
     C+        (SQL ステートメントを含んでいる継続記入行)
     C+
     .
     .
     .
     C/END-EXEC     (終了区切り文字)
     C
     C              (ILE RPG 演算命令)
     C
```

図 30. ILE RPG プログラムの SQL ステートメント

SQL ステートメントを処理するには、別のコマンドを入力する必要があります。詳細については、Web サイト http://www.ibm.com/eserver/iseries/infocenter で **iSeries 400 Information Center** の中の『データベースおよびファイル・システム』カテゴリーの『*DB2 UDB for iSeries*』の節を参照してください。

SEU が SQL ステートメントの構文検査をどのように扱うかについての情報は、「*AS/400 適用業務開発ツールセット AS/400 用 原始ステートメント入力ユーティリティー 使用者の手引きと参照*」を参照してください。

IFS ソース・ファイルの使用

CRTBNDRPG コマンドおよび CRTRPGMOD コマンドには、ソース・ファイルの場所を QSYS ファイル・システムまたは IFS ファイル・システム内のどちらにでも指定できるパラメーターがあります。有効なパラメーターは次のとおりです。

SRCSTMF
SRCSTMF は、SRCFILE および SRCMBR の代わりに、ストリーム・ファイルがメイン・ソース・ファイルであることを示す場合に使用します。

INCDIR
INCDIR は、コピー・ファイルが入っているディレクトリーをリストする場合に使用します。

SRCSTMF で指定するストリーム・ファイルは、ファイルの絶対パス (スラッシュから始まる) でも、現行ディレクトリーを基準とする相対パスでも構いません。

組み込みファイル

/COPY 指示および /INCLUDE 指示では、 QSYS ファイル・システムまたは IFS ファイル・システムいずれかのファイルを指定することができます。コンパイラーが、指示が参照するファイル・システムを認識できない場合は、 /COPY 指示が入っているファイルのファイル・システムで検索が開始されます。

コンパイラーが /COPY ステートメントを検出すると、そのステートメントは IFS または QSYS ファイル・システム内のファイルを参照できるようになります。名前の先頭がスラッシュの場合、または名前が単一引用符で囲まれて指定された場合、

その名前は IFS のファイルだけを指すことになります。 IFS の名前の指定には、二重引用符も使用できます。次のように、名前の一部分しか二重引用符で囲まれていない場合、

```
/copy "SOME-LIB"/QRPGLESRC,MBR
```

その名前は QSYS ファイル・システム名のみを指すことになります。

名前が QSYS ファイル・システムの名前か、IFS の名前か分からない場合は、最初に、/COPY ステートメントが入っているファイルのファイル・システムが検索されます。大文字は、QSYS ファイル・システムでは使用できますが (ただし、″A/B″ のように二重引用符を使って指定する拡張名は除きます)、 IFS では使用できないことに注意してください (IFS では大文字小文字の区別がありません)。

表 17. QSYS および IFS の /Copy ファイル名解釈

/Copy ステートメント	**QSYS での解釈**	**IFS での解釈 (″.suffix″ の意味については、下記を参照)**
/COPY MYMBR	FILE(*LIBL/QRPGLESRC) MBR(MYMBR)	組み込みパス内のディレクトリーの 1 つにある MYMBR または MYMBR.suffix
/COPY mymbr	FILE(*LIBL/QRPGLESRC) MBR(MYMBR)	組み込みパス内のディレクトリーの 1 つにある mymbr または mymbr.suffix
/COPY myfile,mymbr	FILE(*LIBL/MYFILE) MBR(MYMBR)	myfile,mymbr または myfile,mymbr.suffix (MYFILE,MYMBR は IFS ファイル・システムの有効な名前であることに注意)
/COPY mylib/myfile,mymbr	FILE(MYLIB/MYFILE) MBR(MYMBR)	mylib/myfile,mymbr (ディレクトリー mylib およびファイル myfile,mymbr)
/COPY ″A/b″,mymbr	FILE(*LIBL/″A/b″) MBR(MYMBR)	N/A (名前の一部分しか二重引用符で囲まれていない)
/COPY ″A/B″	FILE(*LIBL/QRPGLESRC) MBR(″A/B″)	A/B
/COPY a b	FILE(*LIBL/QRPGLESRC) MBR(A) (ブランクの後ろはすべてコメントと見なされる)	a または a.suffix (ブランクの後ろはすべてコメントと見なされる)
/COPY ’a b’	N/A (名前が単一引用符で囲まれている)	a b または a b.suffix
/COPY /home/mydir/myfile.rpg	N/A (名前がスラッシュで始まる)	/home/mydir/myfile.rpg
/COPY /QSYS.LIB/ L.LIB/F.FILE/M.MBR	N/A (名前がスラッシュで始まる)	/QSYS.LIB/L.LIB/F.FILE/ M.MBR (実際には QSYS ファイル・システム内のファイルだが、RPG には IFS ファイルと見なされる)

注: IFS でファイルを検索している場合、ファイル名にドットが入っていないと、RPG コンパイラーは次の拡張子を持つファイルを (この順序で) 検索します。

1. 拡張子なし (abc)
2. .rpgleinc (abc.rpgleinc)
3. .rpgle (abc.rpgle)

IFS 内での検索パス

IFS 内で /COPY ファイルおよび /INCLUDE ファイルを探す場所を示す方法は 2 つあります。

1. INCDIR パラメーター。ディレクトリーを検索順にリストします。
2. RPGINCDIR 環境変数。ディレクトリーを検索順にコロンで区切って指定したリストが入っています。この環境変数を設定するには、ADDENVVAR コマンドまたは CHGENVVAR コマンドを使用します。

   例:
   ```
   ADDENVVAR ENVVAR(RPGINCDIR)
   VALUE('/home/mydir:/project/prototypes')ADDENVVAR
   ```

IFS で**相対**ファイル (パスの先頭が / ではないファイル) を検索する場合は、ファイルは次の場所および次の順序で検索されます。

1. 現行ディレクトリー
2. INCDIR コマンド・パラメーターで指定されたパス
3. RPGINCDIR 環境変数内のディレクトリー
4. ソース・ディレクトリー (ソースが IFS ファイルの場合)

次に例を示します。

- 現行ディレクトリーが /home/auser。
- INCDIR パラメーターが /driver/v5r2/inc:/driver/v5r1/inc。
- RPGINCDIR 環境変数が /home/auser/temp。
- ソースがディレクトリー /home/auser/src 内。

ディレクトリー検索パスは、デフォルト拡張子の順序に優先します。拡張子のないファイルを複数の異なるディレクトリーで検索する場合は、次のディレクトリーに変換する前に、各ディレクトリーですべての拡張子を付けた検索が行われます。

表 18. /Copy ファイルの検索順序

/Copy ステートメント	**検索するファイル**
IFS では、/COPY が入っている ソース・ファイルは /driver/src/main.rpg であると見なす /COPY file.rpg	IFS の場合: /home/auser/file.rpg /driver/v5r2/inc/file.rpg /driver/v5r1/inc/file.rpg /home/auser/temp/file.rpg /home/auser/src/file.rpg QSYS の場合: FILE(*LIBL/QRPGLESRC) MBR(FILE.RPG)

表 18. /Copy ファイルの検索順序 (続き)

/Copy ステートメント	検索するファイル
QSYS ファイル・システムでは、 /COPY が入っているソース・ファイルは MYLIB/QRPGLESRC MYMBR であると 見なす /COPY file	QSYS の場合: FILE(*LIBL/QRPGLESRC) MBR(FILE) IFS の場合: /home/auser/file /home/auser/file.rpgleinc /home/auser/file.rpgle /driver/v5r2/inc/file /driver/v5r2/inc/file.rpgleinc /driver/v5r2/inc/file.rpgle /driver/v5r1/inc/file /driver/v5r1/inc/file.rpgleinc /driver/v5r1/inc/file.rpgle /home/auser/temp/file /home/auser/temp/file.rpgleinc /home/auser/temp/file.rpgle /home/auser/src/file /home/auser/src/file.rpgleinc /home/auser/src/file.rpgle

第 6 章 CRTBNDRPG コマンドによるプログラムの作成

この章では、バインド RPG プログラム作成 (CRTBNDRPG) コマンドにより、RPG IV ソース・プログラムを使用して ILE プログラムを作成する方法を示します。このコマンドによってユーザーは以下の 2 つのタイプの ILE プログラムのどちらかを作成することができます。

1. 静的バインドを行わない OPM 互換プログラム
2. 静的バインドを行う単一モジュールの ILE プログラム

最初のタイプのプログラム、2 番目のタイプのプログラム、どちらになるかは、CRTBNDRPG の DFTACTGRP パラメーターが、それぞれ *YES に設定されているか *NO に設定されているかによります。

最初のタイプのプログラムの作成で オープンの範囲指定、一時変更の範囲指定、および RCLRSC の面で OPM プログラムのように機能するプログラムが作成されます。この高度の互換性は、1 つには OPM プログラムと同一の活動化グループ、すなわちデフォルトの活動化グループ内で実行されることによります。

しかし、この高い互換性により静的バインド機能がなくなります。静的バインドとは、プロシージャーを呼び出し (他のモジュールまたはサービス・プログラム)、プロシージャー・ポインターを使用する機能のことです。静的バインド機能がないと、以下のことができなくなります。

- ソース・ステートメントで CALLB 命令を使う
- プロトタイプ・プロシージャーを呼び出す
- プログラム作成時に他のモジュールにバインドする

2 番目のタイプのプログラムの作成では、静的バインドなどの ILE の特性をもつプログラムが作成されます。プログラム作成時に、プログラムを実行する活動化グループおよび静的バインドのためのモジュールを指定することができます。さらに、ソース・ステートメントからプロシージャーを呼び出すことができます。

WebSphere Development Studio Client for iSeries を使用します。これは推奨される方法であり、製品のオンライン・ヘルプの ILE RPG プログラム作成の項に記載されています。

CRTBNDRPG コマンドの使用

バインド RPG 作成 (CRTBNDRPG) コマンドがワン・ステップで RPG IV ソースからプログラム・オブジェクトを作成します。またこのコマンドにより、バインディング・ディレクトリーを使っている他のモジュールあるいはサービス・プログラムをバインドすることができます。

コマンドは ILE RPG コンパイラーを開始し一時モジュール・オブジェクトをライブラリー QTEMP に作成します。その後で、このコマンドはそれをタイプ *PGM

のプログラム・オブジェクトにバインドします。プログラム・オブジェクトがいったん作成されると、プログラムの作成に使用された一時モジュールが削除されます。

CRTBNDRPG コマンドは、作成のステップとバインドのステップを組み合わせているため、独立型のソース・コード (バインドすべきモジュールを必要としないコード) からプログラム・オブジェクトを作成したいときに有用です。さらに、このコマンドで OPM 互換プログラムを作成することができます。

注: モジュール・オブジェクトを他のモジュールとともにプログラム・オブジェクトにバインドするために保存したい場合には、CRTRPGMOD コマンドを使用してモジュールを作成する必要があります。詳細については、83 ページの『第 7 章 CRTRPGMOD および CRTPGM コマンドによるプログラムの作成』を参照してください。

CRTBNDRPG コマンドは、対話式に使用することも、バッチで使用することも、あるいはコマンド言語 (CL) プログラムから使用することもできます。コマンドを対話式に使用していて、プロンプトが必要な場合には、CRTBNDRPG と入力して、F4 (プロンプト) を押してください。ヘルプが必要な場合には、CRTBNDRPG と入力して、F1 (ヘルプ) を押してください。

表 19 は、CRTBNDRPG コマンドのパラメーターを要約するとともに、それらのデフォルト値を示しています。

表 19. CRTBNDRPG パラメーターおよび機能ごとにグループ化されたデフォルト値

プログラム識別	
PGM(*CURLIB/*CTLSPEC)	作成されるプログラムの名前およびライブラリーを決める。
SRCFILE(*LIBL/QRPGLESRC)	指定されている場合は、ソース・ファイルおよびライブラリーを示す。
SRCMBR(*PGM)	指定されている場合は、ソースの仕様が入っているファイル・メンバーを示す。
SRCSTMF(path)	指定されている場合は、IFS のソース・ファイルへのパスを示す。
INCDIR('path to directory 1:path to directory 2')	/copy ファイルおよび /include ファイルを検索するディレクトリーのリストを示す。
TEXT(*SRCMBRTXT)	プログラムの簡単な説明を用意する。
プログラム作成	
GENLVL(10)	プログラム作成をエラーの重大度 (0 ～ 20) に条件付ける。
OPTION(*DEBUGIO)	*DEBUGIO/*NODEBUGIO は、入出力仕様書について停止点を生成するかどうかを決定する。
OPTION(*GEN)	*GEN/*NOGEN は、プログラムを作成するかどうかを決める。
OPTION(*NOSRCSTMT)	コンパイラーがデバッグ用にステートメント番号を生成する方法を指定する。
DBGVIEW(*STMT)	プログラムに含めるべきデバッグ・ビューがあれば、そのタイプを指定する。
OPTIMIZE(*NONE)	最適化をする場合、そのレベルを決める。
REPLACE(*YES)	プログラムを既存のプログラムと置き換えるべきかどうかを決める。
BNDDIR(*NONE)	記号の解決に使用するバインディング・ディレクトリーを指定する。

表 19. CRTBNDRPG パラメーターおよび機能ごとにグループ化されたデフォルト値 (続き)

USRPRF(*USER)	プログラムを実行するユーザー・プロファイルを指定する。
AUT(*LIBCRTAUT)	作成されるプログラムの権限のタイプを指定する。
TGTRLS(*CURRENT)	オブジェクトを実行するリリース・レベルを指定する。
ENBPFRCOL(*PEP)	パフォーマンス収集を使用可能にするかどうかを指定する。
DEFINE(*NONE)	コンパイルの開始前に定義する条件名を指定する。
PRFDTA(*NOCOL)	プログラムのプロファイル作成データ属性を指定する。
コンパイラー・リスト	
OUTPUT(*PRINT)	コンパイラー・リストを作成するかどうかを決める。
INDENT(*NONE)	リスト出力中に字下げを示すかどうかを決め、また字下げのマーク付けのための文字を識別する。
OPTION(*XREF *NOSECLVL *SHOWCPY *EXPDDS *EXT *NOSHOWSKP *NOSRCSTMT)	コンパイラー・リストの内容を指定する。
データ変換オプション	
CVTOPT(*NONE)	外部記述ファイルからの各種のデータ・タイプを取り扱う方法を指定する。
ALWNULL(*NO)	null 可能フィールドからの値をプログラムが受け入れるかどうかを決める。
FIXNBR(*NONE)	無効な 10 進数データのどれをコンパイラーで修正するかを決める。
実行時の考慮事項	
DFTACTGRP(*YES)	このプログラムを常に OPM のデフォルトの活動化グループ内で実行するかどうかを識別する。
OPTION(*DEBUGIO)	*DEBUGIO/*NODEBUGIO は、入出力仕様書について停止点を生成するかどうかを決定する。
ACTGRP(QILE)	プログラムを実行すべき活動化グループを識別する。
SRTSEQ(*HEX)	使用すべき分類順序テーブルを指定する。
LANGID(*JOBRUN)	分類順序の言語識別コードを指定するために SRTSEQ と一緒に使用される。
TRUNCNBR(*YES)	固定長形式操作でパック 10 進数、ゾーン 10 進数、および 2 進数フィールドの数値オーバーフローが起きた時に取る処置を指定する。
INFOSTMF(path)	PGMINFO と一緒に使用し、PCML を受け取る IFS のストリーム・ファイルを指定する。
PGMINFO(*NONE)	*PCML は、プログラムに対し PCML (プログラム呼び出しマークアップ言語) を生成することを示す。
LICOPT(オプション)	ライセンス内部コード・オプションを指定します。

CRTBNDRPG の構文図およびパラメーターの説明については、483 ページの『付録 C. 作成コマンド』を参照してください。

ソース・デバッグ用のプログラムの作成

この例では、ソース・デバッガーを使ってデバッグできるよう、プログラム EMPRPT を作成します。CRTBNDRPG または CRTRPGMOD のどちらかの

DBGVIEW パラメーターがコンパイルの間にどのタイプのデバッグ・データを作成するかを決定します。このパラメーターには必要なビューを選択できる 6 個のオプションがあります。

- *STMT — コンパイラー・リストを使用して、変数を表示し、ステートメント位置への停止点の設定を可能にする。このビューでは、ソース・ステートメントは表示されません。
- *SOURCE — 入力ソース・ステートメントと同一のビューを作成する。
- *COPY — ソース・ビューおよび任意の /COPY メンバーのソースを含むビューを作成する。
- *LIST — コンパイラー・リストと同様のビューを作成する。
- *ALL — 上記のすべてのビューを作成する。
- *NONE — デバッグ・データを作成しない。

EMPRPT のソース・ステートメントは 58 ページの図 28 に示されています。

1. オブジェクトを作成するには、以下のとおり入力してください。

```
CRTBNDRPG PGM(MYLIB/EMPRPT) DBGVIEW(*SOURCE) DFTACTGRP(*NO)
```

   プログラムはそれが基礎になっているソース・メンバーと同じ名前、すなわち EMPRPT のライブラリー MYLIB に作成されます。デフォルトにより、QILE という名前のデフォルトの活動化グループで実行するということに注意してください。このプログラム・オブジェクトはソース・ビューを使ってデバッグすることができます。

2. プログラムをデバッグするには、以下のとおり入力してください。

```
STRDBG EMPRPT
```

   図 31 は、上記のコマンドを入力したあとで表示される画面です。

```
                          モジュール・ソースの表示
PROGRAM:   EMPRPT         LIBRARY:   MYLIB          MODULE:   EMPRPT
     1        *============================================================*
     2        * MODULE NAME:    EMPRPT
     3        * RELATED FILES:  EMPMST   (PHYSICAL FILE)
     4        *                 QSYSPRT  (PRINTER FILE)
     5        * DESCRIPTION:    This program prints employee information
     6        *                 from the file EMPMST.
     7        *============================================================*
     8       FQSYSPRT   O    F   80        PRINTER
     9       FEMPMST    IP   E           K DISK
    10
    11       D TYPE            S              8A
    12       D EMPTYPE         PR             8A
    13       D   CODE                         1A
    14
    15       IEMPREC        01
                                                                    続く...
デバッグ . . .  ____________________________________________________________
F3=終了プログラム  F6=停止点の追加/消去  F10=ステップ  F11=変数の表示
F12=再開       F17=ウォッチ変数   F18=ウォッチの処理   F23=表示出力
```

図 31. EMPRPT 用のモジュール・ソースの表示画面

ユーザーは、この画面 (モジュール・ソースの表示画面) からデバッグ・コマンドを入力して、フィールド値の表示および変更を行ったり、デバッグ中のプログラムの流れを制御するために停止点を設定することができます。

デバッグの詳細については、 215 ページの『第 12 章 プログラムのデバッグ』を参照してください。

静的バインドによるプログラムの作成

この例では CRTBNDRPG を使用して COMPUTE というプログラムを作成します。このプログラムにはプログラム作成時にサービス・プログラムをバインドします。

高度の計算をするために購入したサービスにプログラム COMPUTE をバインドしたいとします。ソースをバインドしなければならないバインディング・ディレクトリーは MATH と呼ばれています。このディレクトリーには、サービスを構成する各種のプロシージャーを含むサービス・プログラムの名前が入っています。

オブジェクトを作成するには、以下のとおり入力してください。

```
CRTBNDRPG PGM(MYLIB/COMPUTE)
          DFTACTGRP(*NO) ACTGRP(GRP1) BNDDIR(MATH)
```

ソースはプログラム作成時にバインディング・ディレクトリー MATH で指定されたサービス・プログラムにバインドされます。このことはサービス・プログラムでのプロシージャーに対する呼び出しが、動的呼び出しより時間がかからないことを意味します。

プログラムが呼び出された場合、指定の活動化グループ GRP1 で実行します。CRTBNDRPG 上のデフォルト値の ACTGRP パラメーターは QILE です。しかし、関連した資源が完全に保護されるよう、アプリケーションを固有のグループとして実行するようお奨めします。

注: ACTGRP および BNDDIR パラメーターに値を入力できるよう、DFTACTGRP は *NO に設定しなければなりません。

サービス・プログラムについて詳しくは、 101 ページの『第 8 章 サービス・プログラムの作成』を参照してください。

OPM 互換プログラム・オブジェクトの作成

この例では、 71 ページの図 32 に示す CRTBNDRPG コマンドを使用して給与計算プログラムのソースから OPM 互換プログラム・オブジェクトを作成します。

1. オブジェクトを作成するには、以下のとおり入力してください。

   ```
   CRTBNDRPG PGM(MYLIB/PAYROLL)
             SRCFILE(MYLIB/QRPGLESRC)
             TEXT('ILE RPG program')  DFTACTGRP(*YES)
   ```

   CRTBNDRPG コマンドはプログラム PAYROLL を MYLIB の中に作成します。このプログラムはデフォルトの活動化グループで実行されることになります。デフォルトでは、コンパイラー・リストが作成されます。

   注: DFTACTGRP(*YES) のセットは OPM の互換性を提供するものです。またこのセットは、ACTGRP および BNDDIR パラメーターの値を入力できないようにします。さらに、ソースにバインド・プロシージャー呼び出しが入っている場合には、エラーが出され、コンパイルは終了します。
2. 作成されたリスト出力を見るには、以下の CL コマンドのいずれか 1 つを入力してください。

- DSPJOB で、オプション 4 (*スプール・ファイルの表示*) を選択します。
- WRKJOB
- WRKOUTQ *待ち行列名*
- WRKSPLF

```
 *------------------------------------------------------------------------*
 * 説明: このプログラムは、社員の週給の印刷出力を                         *
 *       作成します。                                                     *
 *------------------------------------------------------------------------*
H DATEDIT(*DMY/)
 *------------------------------------------------------------------------*
 * ファイル定義                                                           *
 *------------------------------------------------------------------------*
FTRANSACT  IP   E           K DISK
FEMPLOYEE  IF   E           K DISK
FQSYSPRT   O    F   80        PRINTER
 *------------------------------------------------------------------------*
 * 変数の宣言                                                             *
 *------------------------------------------------------------------------*
D Pay             S              8P 2
 *------------------------------------------------------------------------*
 * 定数の宣言                                                             *
 *------------------------------------------------------------------------*
D Heading1        C                   'NUMBER  NAME              RATE    H-
D                                     OURS  BONUS    PAY       '
D Heading2        C                   '______  ________________  ______  _-
D                                     ____  _______  _________'
 *------------------------------------------------------------------------*
 * トランザクション・ファイル (TRANSACT) 内の各レコードごとに、           *
 * 社員が見つかればその給与を計算して明細を印刷します。                   *
 *------------------------------------------------------------------------*
C     TRN_NUMBER    CHAIN     EMP_REC                          99
C                   IF        NOT *IN99
C                   EVAL (H)  Pay = EMP_RATE * TRN_HOURS + TRN_BONUS
     C                    ENDIF

 *------------------------------------------------------------------------*
 * 報告書レイアウト                                                       *
 *  -- 1P がオンであれば見出し行を印刷する                                *
 *  -- レコードが見つかった場合 (標識 99 がオフ) は                       *
 *     給与明細を印刷し、それ以外の場合は例外レコードを印刷する           *
 *  -- LR がオンのときは 'リストの終わり' と印刷する                      *
 *------------------------------------------------------------------------*
OQSYSPRT   H    1P                   2  3
O                                         35 '給与計算表'
O                         *DATE       Y   60
O          H    1P                    2
O                                         60 Heading1
O          H    1P                    2
O                                         60 Heading2
O          D   N1PN99                 2
O                         TRN_NUMBER       5
O                         EMP_NAME        24
O                         EMP_RATE    L   33
O                         TRN_HOURS   L   40
O                         TRN_BONUS   L   49
O                         Pay             60 '$     0.  '
O          D   N1P 99                 2
O                         TRN_NUMBER       5
O                                         35 '** 社員ファイル上にない **'
O          T    LR
O                                         33 'END OF LISTING'
```

図 32. 給与計算サンプル・プログラム

コンパイラー・リストの使用

この項では、リスト出力を取る方法および以下のことに役立てるためのその使用方法について説明します。

- コンパイル・エラーの修正
- 実行時エラーの修正
- 保守の目的のための文書化の準備

リストの各部分の詳細および全体のサンプル・リストについては 507 ページの『付録 D. コンパイラー・リスト』を参照してください。

コンパイラー・リストの作成

コンパイラー・リストを作成するには、CRTBNDRPG コマンドまたは CRTRPGMOD コマンドのどちらかで OUTPUT(*PRINT) を指定します。(これはデフォルトの設定です。) OUTPUT(*NONE) と指定すると、リスト出力を止めます。

OUTPUT(*PRINT) を指定すると次のセクションからなる、*最小限* のコンパイラー・リストを作り出します。

- プロローグ (コマンド・オプションの要約)
- 以下のものを含むソース・リスト
  - インライン診断メッセージ
  - 突き合わせフィールド・テーブル (突き合わせフィールドの RPG サイクルを使用している場合)
- 追加の診断メッセージ
- 出力バッファー中のフィールド位置
- /COPY メンバー・テーブル
- 以下のものを含むコンパイル時データ
  - 代替照合順序レコードおよびテーブル、あるいは NLSS 情報およびテーブル
  - ファイル変換レコード
  - 配列レコード
  - テーブル・レコード
- メッセージの要約
- 最終の要約
- コード生成報告書 (エラーがある場合にのみ現れる)
- バインド報告書 (CRTBNDRPG に対してだけ適用され、エラーがある場合にのみ現れる)

以下の追加情報は、いずれかの作成コマンドの OPTION パラメーターに適切な値が指定された場合に、コンパイラー・リスト中に含められます。

***EXPDDS**
外部記述ファイルの仕様書 (リスト出力のソース・セクションに現れる)

***SHOWCPY**
/COPY メンバーのソース・レコード (リスト出力のソース・セクションに現れる)

***SHOWSKP**
条件付きコンパイル指示によって除外されるソース行 (リスト出力のソース・セクションに現れる)

***EXPDDS**
キー・フィールド情報 (別のセクション)

***XREF**
相互参照リスト (別のセクション)

***EXT** 外部参照のリスト (別のセクション)

***SECLVL**
第 2 レベル・メッセージ・テキスト (メッセージの要約セクションに現れる)

注: *SECLVL と *SHOWSKP を除き、上記のすべての値が両方の作成コマンドの OPTION パラメーターのデフォルトの設定に影響します。特定のリスト出力セクション、あるいは含まれる第 2 レベルのテキストが必要でなければ、OPTION パラメーターを変更する必要はありません。

コンパイラー・リストに入っている情報は、OPTION パラメーターに *SRCSTMT または *NOSRCSTMT が指定されるかどうかによって異なります。この情報がどのように変化するかについては、 514 ページの『*NOSRCSTMT ソース見出し』および 514 ページの『*SRCSTMT ソース見出し』を参照してください。

コンパイル・オプション・キーワードが制御仕様書に指定されると、有効なコンパイラー・オプションがリストのソース・セクションに現れます。

コンパイラー・リストのカスタマイズ

以下のいずれかまたはすべての方法によってコンパイラー・リストをカスタマイズすることができます。

- ページ見出しのカスタマイズ
- スペーシングのカスタマイズ
- 構造化命令の字下げ

ページ見出しのカスタマイズ

ページ見出し情報にはプロダクト情報行および /TITLE 指示によって指定されたタイトルが含まれます。プロダクト情報行には ILE RPG コンパイラーとライブラリー著作権表示、ソースのメンバー、およびライブラリー、モジュールが作成された日時、それにリストのページ数が入っています。

/TITLE コンパイラー指示の使用によってコンパイラー・リストの見出し情報を指定することができます。この指示により、コンパイラー・リストの各ページの最上部に表示されるテキストを指定することができます。この通知は通常のページ見出し情報に先行して出てきます。指示がソース・メンバーの最初のレコードである場合には、この情報もプロローグ・セクションに表示されます。

また、ページ見出しやその他の情報ボックスで使用される日付区切り記号、日付の形式、および時刻区切り文字をリスト全体について変更することができます。通

常、コンパイラーがジョブ属性を見て、これらを決定します。これらのどれかを変更するには、ジョブ変更 (CHGJOB) コマンドを使用してください。このコマンドの入力後、以下のことが可能です。

- 次の日付区切り記号の 1 つを選択する: *SYSVAL、*BLANK、スラッシュ (/)、ハイフン (-)、ピリオド (.)、あるいはコンマ (,)
- 次の日付の形式の 1 つを選択する: *SYSVAL、*YMD、*MDY、*DMY、または *JUL
- 次の時刻区切り記号の 1 つを選択する: *SYSVAL、*BLANK、コロン (:)、コンマ (,)、あるいはピリオド (.)

リスト出力のどこに現れても、日付または時刻フィールドにはこれらの値が使用されます。

スペーシングのカスタマイズ

リスト出力の各セクションは通常、改ページで開始します。すなわち、ソース・メンバーに /TITLE 指示が入っていなければリスト出力の各ページはプロダクト情報で始まります。この場合、プロダクト情報は 2 行目に表示され、タイトルは最初の行に表示されます。

/EJECT および /SPACE コンパイル指示を使用することによって、コンパイラー・リストのスペーシングおよびページ編集を制御することができます。/EJECT 指示はページ替えを行います。/SPACE 指示はリスト出力内の行スペーシングを制御します。これらの指示の詳しい情報については、「*WebSphere Development Studio: ILE RPG 解説書*」を参照してください。

構造化命令の字下げ

注: 演算を字下げすることができるのは、伝統的な構文にのっとって書かれている演算の場合だけです。RPG コンパイラーは、リスト内の自由形式の演算書 (/FREE と /END-FREE の間) については、字下げを変更しません。自由形式の演算書では、ユーザーのソース・ファイル内で直接字下げを行なうことができます。

ソースの仕様に構造化命令が含まれている場合には (DO-END または IF-ELSE-END など)、ソース・リストで字下げされたものが必要なことがあります。INDENT パラメーターによって、字下げの表示をするかどうかを指定し、字下げの印付けをする文字を指定することができます。字下げをしたくない場合には、INDENT(*NONE) を指定してください。これがデフォルト値です。字下げをしたい場合には、字下げに印付けをするために最大 2 桁を指定してください。

例えば、構造化命令を字下げして縦線 (|) とそれに続くスペースで印付けしたいことを指示するには、INDENT('| ') と指定します。

字下げを要求する場合には、情報ソース・リストに現れる通知のいくつかが字下げを認めるために除去されます。次の欄はリスト出力に表示され**ません**。

- DO 番号
- 最終更新日
- PAGE/LINE

字下げを指定し、さらにリスト・デバッグ・ビューも指定した場合には、デバッグ・ビューには字下げは現れません。

図 33 は、字下げして作成されたソース・リストの一部を示したものです。字下げの印は '| ' です。

```
行   <---------------------------ソースの仕様 ---------------------------------------------------><------ 注記 -----> SRC   SEQ
番号 ....1....+....2....+<-------- 26 - 35 -------->....4....+....5....+....6....+....7....+....8....+....9....+...10 ID    番号
    33 C*******************************************************************                                           002000
    34 C*    メインライン                                                         *                                   002100
    35 C*******************************************************************                                           002200
    36 C                   WRITE                     FOOT1                                                            002300
    37 C                   WRITE                     HEAD                                                             002400
    38 C                   EXFMT                     PROMPT                                                           002500
    39 C*                                                                                                              002600
    40 C                   DOW                       NOT *IN03                                                        002700
    41 C     CSTKEY        | SETLL                   CMLREC2                            ----20                        002800
    42 C                   | IF                      *IN20                                                            002900
    43 C                   | | MOVE                  '1'           *IN61                                              003000
    44 C                   | ELSE                                                                                     003100
    45 C                   | | EXSR                  SFLPRC                                                           003200
    46 C                   | END                                                                                      003300
    47 C                   | IF                      NOT *IN03                                                        003400
    48 C                   | | IF                    *IN04                                                            003500
    49 C                   | | | IF                  *IN61                                                            003600
    50 C                   | | | | WRITE             FOOT1                                                            003700
    51 C                   | | | | WRITE             HEAD                                                             003800
    52 C                   | | | ENDIF                                                                                003900
    53 C                   | | | EXFMT               PROMPT                                                           004000
    54 C                   | | ENDIF                                                                                  004100
    55 C                   | ENDIF                                                                                    004200
    56 C                   ENDDO                                                                                      004300
    57 C*                                                                                                              004500
    58 C                   SETON                                                        LR----                        004600
```

図 33. 字下げ付きのサンプル・ソース・リストの一部

コンパイル・エラーの訂正

コンパイル・エラーの修正に有用なコンパイラー・リストの主要なセクションは以下のとおりです。

- ソース・セクション
- 追加のメッセージ・セクション
- /COPY テーブル・セクション
- 各種の要約セクション

ソース・セクションにあるインライン診断メッセージはコンパイラーがすぐにフラグを付けることができるエラーを示します。その他のエラーはコンパイル時に追加の情報が受け取られてからフラグが付けられます。これらのエラーにフラグを付けるメッセージは、ソース・セクションおよび追加のメッセージ・セクションにあります。

コンパイル・エラーの訂正を援助するために、また特にユーザーが RPG の初心者の場合、リストに第 2 レベルのメッセージを組み込みたいがあります。これを実行するためには、どちらかの作成コマンドで OPTION(*SECLVL) を指定します。これにより、第 2 レベルのテキストがメッセージ要約にリストされたメッセージに追加されます。

最後に、コンパイラー・リストがプログラムのレコードであることを忘れないでください。したがって、プログラムのテスト中にエラーを見付けた場合には、ソースが思ったとおりにコード化されていることを検査するために、このリスト出力を使用することができます。ソースの他に確認したいリスト出力の一部に以下のものが含まれています。

- 突き合わせフィールド・テーブル

  突き合わせフィールドで RPG サイクルを使用している場合には、これをすべての突き合わせフィールドの長さが正しいか、正しい位置にあるかを確認するのに使用することができます。

- 出力バッファー位置

  リテラル・テキストまたはフィールド名と一緒に開始位置および終了位置をリストします。これを使用して、出力仕様書のエラーを確認してください。

- コンパイル時データ

  ALTSEQ と FTRANS レコードおよびテーブルがリストされます。NLSS 通知およびテーブルがリストされます。テーブルおよび配列が明示的に識別されます。これを使用して正しい順序でコンパイル時データを指定したこと、およびコンパイラーに対し SRTSEQ および LANGID パラメーターの正しい値を指定したことを確認してください。

インライン診断メッセージの使用

2 つのタイプのインライン診断メッセージがあります。フィンガーとノンフィンガーです。フィンガー・メッセージは、どこでエラーが起きたかを正確に示します。図 34 は、フィンガー・インライン診断メッセージの例を示したものです。

```
行   <---------------------------ソースの仕様 ------------------------------------><------ 注記 -----> DO  PAGE変更   SRC SEQ
番号 ....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+....9....+...10 NUM 行  日付   ID  番号
   63  C                   SETOFF                                    _12___                                                  003100
======>                                                              AAHH
======>                                                              CCCCCC
*RNF5051 20 A      003100  結果の標識の項目が正しくありません。省略時の値としてブランクが使用されます。
*RNF5051 20 B      003100  結果の標識の項目が正しくありません。省略時の値としてブランクが使用されます。
*RNF5053 30 C      003100  結果の標識の項目が指定された操作に対してブランクです。
```

図 34. サンプル・フィンガー・インライン診断メッセージ

この例では、標識が 71 ～ 72 桁目または 73 ～ 74 桁目ではなく間違って 72 ～ 73 桁目に入れられています。3 つのフィンガー 'aa'、'bb'、および 'cccccc' がエラーのある行の一部を識別します。実際の欄はメッセージによってさらに説明される変数によって、強調表示されます。この場合には、メッセージ RNF5051 は、'aa' および 'bb' で印が付けられたフィールドには正しい標識が入っていないことを示しています。正しい標識がないために、コンパイラーはフィールドがブランクであると見なします。しかし、SETOFF 命令が標識を必要とするので、 'cccccc' およびメッセージ RNF5053 によって指摘されたように別のエラーが起こります。

エラーは見付かった順序でリストされます。一般的に、しばしば他のエラーの原因となるため最初のいくつかの重大度 30 および 40 のエラーの訂正に重点を置くべきです。

ノンフィンガー・インライン診断メッセージもまたエラーを示します。しかし、これらはただちにエラーの行に続いて出されるわけではありません。図 35 は、ノンフィンガー・インライン診断メッセージの例を示したものです。

```
行   <---------------------------ソースの仕様 ------------------------------------><------ 注記 -----> DO  PAGE  変更   SRC SEQ
番号 ....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+....9....+...10 NUM 行    日付   ID  番号
     1 D FLD1           S             +5    LIKE(FLD2)                                                          000100
     2 D FLD2           S               D                                                                       000200
*RNF3479 20      1 000100  指定されたデータ・タイプのフィールドでは長さ調整はできません。
```

図 35. ノンフィンガー・インライン診断メッセージの例

この例では、FLD1 は FLD2 と同じように 5 バイト以上の長さで定義されます。後から FLD2 は日付として定義されますが、これは FLD1 の定義において長さの調整を無効にします。メッセージ RNF3479 は、リストの 1 行目を指して出されます。SEU 順序番号 (000100) も与えられますが、これは、エラーのソース行をより早く見つけるのを助けるためであることに注意してください (また、SEU 順序番号はリスト出力行 1 にも入っています)。

追加の診断メッセージの使用

追加の診断メッセージのセクションは複数のコード行の検討で見付かるエラーを指定します。これらのメッセージは、問題のあるコード内には置かれません。一般的には、ソースのその部分の検査時にコンパイラーは、何か問題があることは分かりません。しかし、可能な場合には、リスト出力行番号および SEU 順序番号または (メッセージに関連するソース行の) ステートメント番号がメッセージ行に入れられます。

SEU を使用したコンパイラー・リストのブラウズ

SEU 分割 / ブラウズ・セッション (F15) によって、出力待ち行列のコンパイラー・リストをブラウズすることができます。ソース・コードに必要な変更を加えながら、前のコンパイルの結果を表示することができます。

コンパイラー・リストのブラウズ中に、エラーを走査し、エラーのあるこれらのソース・ステートメントを訂正することができます。エラーを走査するには、ブラウズ・セッションの SEU コマンド行に F *ERR を入力します。最初の (または次の) エラーがある行は強調表示され、同じメッセージの第 1 レベル・テキストが画面の最下部に現れます。最下部のメッセージにカーソルを置き F1 (ヘルプ) キーを押すことによって、第 2 レベル・テキストを見ることができます。

可能の時は、リスト中のエラー・メッセージは、エラーのある行の SEU 順序番号を指定します。順序番号は、メッセージ・テキストの直前にあります。

コンパイラー・リストのブラウズについての完全な情報は、「*AS/400 適用業務開発ツールセット AS/400 用 原始ステートメント入力ユーティリティー 使用者の手引きと参照*」を参照してください。

実行時エラーの訂正

リストのソース・セクションは、実行時エラーの訂正にも有用です。多くの実行時エラー・メッセージでは問題になっているエラーが起こったステートメント番号を指定します。

OPTION(*NOSRCSTMT) が指定された場合、コンパイラー・リストの左 側の行番号が実行時エラー・メッセージのステートメント番号に対応します。コンパイラー・リストの右 側のソース ID 番号および SEU 順序番号がソース・メンバーおよびレコードを指定します。この 2 つを一緒に使用し、特に SEU を使ってソースを編集している場合には、どの行を検査する必要があるか判別することができます。

OPTION(*SRCSTMT) が指定された場合には、コンパイラー・リストの右 側のステートメント番号が実行時エラー・メッセージのステートメント番号に対応します。ステートメントがメイン・ソース・メンバーからのものであれば、これは、コンパイラー・リストの左 側のステートメントと同じものであり、SEU 順序番号とも同じです。

/COPY メンバーがある場合には、リストの終わりの /COPY メンバー・テーブルの実際のファイルのソース ID 番号を見つけることができます。/COPY メンバー・テーブルの例については、516 ページの『/COPY メンバー・テーブル』を参照してください。

デバッグ・ビュー・オプションによるリスト出力オプションの調整

多くの場合、実行時エラーの訂正にはプログラムのデバッグが必要です。プログラムのデバッグを行う際、次の考慮事項が役立ちます。

- プログラムをデバッグするためにソース・デバッガーを使用する場合、次のデバッグ・ビュー (*STMT、*SOURCE、*LIST、*COPY、*ALL) を選択します。
- デバッグ中に補助としてコンパイラー・リストを使用する場合には、OUTPUT(*PRINT) を指定して、リストを得ることができます。停止点を設定するためのステートメント番号はソース・リストに示されるので、ステートメント (*STMT) ビューを使用してデバッグする場合には、リスト出力は重要です。ステートメント番号は、OPTION(*NOSRCSTMT) が指定された場合には「行番号」というラベルの列にリストされ、OPTION(*SRCSTMT) が指定された場合には「ステートメント番号」という列にリストされます。
- デバッグがかなり大変であることが分っている場合には、DBGVIEW(*ALL)、OUTPUT(*PRINT) および OPTION(*SHOWCPY) を指定してソースをコンパイルすることができます。これによってソースまたはリスト出力ビューのいずれかを使用することができ、 /COPY メンバーを組み込むことができます。
- DBGVIEW(*LIST) を指定した場合には、デバッグで使用できる情報は、OPTION パラメーターの指定によって変わります。OPTION(*SHOWCPY *EXPDDS) を指定した場合にだけ、ビューに /COPY メンバーおよび外部記述ファイルが組み込まれます (デフォルトでこのようになっています)。

保守のためのコンパイラー・リストの使用

エラーのないプログラムのコンパイラー・リストは、以下のための文書として使用することができます。

- 新人プログラマーへのプログラム研修
- 後日のプログラム更新

いずれの場合にも完全なリスト出力、すなわち OUTPUT(*PRINT) および OPTION(*XREF *SHOWCPY *EXPDDS *EXT *SHOWSKP) を指定して作成されたリストが望まれます。

注: *SHOWSKP を除いて、どちらの作成コマンドにおいても、これらのパラメーターのおのおのに対して、これがデフォルトの設定値です。

プログラム保守のための特別の値が、リストのプロローグ・セクションにあります。このセクションでは、以下のことをユーザーに知らせます。

- モジュール/プログラムをだれがコンパイルしたか
- モジュール/プログラムを作成するためにどのソースが使用されたか
- モジュール/プログラムをコンパイルするときどのオプションが使用されたか

プログラムを後から変更する際に、コマンド・オプション (例えば、選択されたデバッグ・ビュー、あるいは使用されたバインディング・ディレクトリー) について知ることが必要な場合があります。

OPTION パラメーターに対する以下の指定により、説明されている追加情報が提供されます。

- *SHOWCPY および *EXPDDS により、/COPY メンバーからのすべての仕様書、および外部記述ファイルから生成された仕様書を含む、プログラムの完全な記述が提供されます。
- *SHOWSKP により、/IF、/ELSEIF、/ELSE、または /EOF 指示の結果としてコンパイラーによって無視されるステートメントを表示することができます。
- *XREF によりユーザーは、モジュール/プログラム内での、ファイル、フィールド、および標識の使用をチェックすることができます。
- *EXT によって、どのプロシージャーおよびフィールドがモジュール/プログラムでインポートまたはエクスポートされるかを知ることができます。これはまた、外部記述ファイルおよびデータ構造の記述の生成に使用された実際のファイルを指定します。

RETURNCODE データ域のアクセス

CRTBNDRPG および CRTRPGMOD (84 ページの『CRTRPGMOD コマンドの使用』を参照) コマンドはともに、最後のコンパイル時の状況でデータ域を作成し更新します。このデータ域の名前は RETURNCODE で、400 桁の長さでライブラリー QTEMP に入れられています。

RETURNCODE データ域をアクセスするには、*DTAARA DEFINE ステートメントの演算項目 2 に RETURNCODE を指定します。

データ域 RETURNCODE の形式は以下のとおりです。

バイト	内容および意味
1	CRTRPGMOD では、文字 '1' はモジュールが指定したライブラリーに作成されたことを意味します。CRTBNDRPG では、文字 '1' はプログラム名と同じ名前のモジュールが QTEMP に作成されたことを意味します。

2	文字 '1' はコンパイラー・エラーのためにコンパイルが正常に実行されなかったことを意味します。
3	文字 '1' は、ソース・エラーのために、コンパイルが正常に実行されなかったことを意味します。
4	未設定。常に '0' です。
5	文字 '1' は OPTION(*NOGEN) が CRTRPGMOD または CRTBNDRPG コマンドで指定されたか、あるいは変換プログラムが呼び出される前にコンパイルが正常に実行されなかったために変換プログラムが呼び出されていないことを意味します。
6 ～ 10	ソース・ステートメントの数
11 ～ 12	コマンドからの重大度レベル
13 ～ 14	診断メッセージの最高の重大度
15 ～ 20	モジュール (CRTRPGMOD) またはプログラム (CRTBNDRPG) で見つかったエラーの数
21 ～ 26	コンパイル日付
27 ～ 32	コンパイル時刻
33 ～ 100	未設定。常にブランク
101 ～ 110	モジュール (CRTRPGMOD) の名前またはプログラム (CRTBNDRPG) の名前
111 ～ 120	モジュール (CRTRPGMOD) のライブラリー名またはプログラム (CRTBNDRPG) のライブラリー名
121 ～ 130	ソース・ファイル名
131 ～ 140	ソース・ファイル・ライブラリー名
141 ～ 150	ソース・ファイル・メンバー名
151 ～ 160	コンパイラー・リスト・ファイル名
161 ～ 170	コンパイラー・リスト・ライブラリー名
171 ～ 180	コンパイラー・リスト・メンバー名
181 ～ 329	未設定。常にブランク
330 ～ 334	10 分の 1 秒単位での合計経過コンパイル時刻 (あるいは、この時間の計算中にエラーが発生した場合には -1)
335	未設定。常にブランク
336 ～ 340	10 分の 1 秒単位での経過コンパイル時刻 (あるいは、この時間の計算中にエラーが発生した場合には -1)
341 ～ 345	10 分の 1 秒単位での変換プログラム経過時間 (あるいは、この時間の計算中にエラーが発生した場合には -1)
346 ～ 379	未設定。常にブランク
380 ～ 384	10 分の 1 秒単位での合計コンパイル CPU 時間
385	未設定。常にブランク

386 ～ 390 10 分の 1 秒単位での、コンパイラーが使用する CPU 時間

391 ～ 395 10 分の 1 秒単位での、変換プログラムが使用する CPU 時間

396 ～ 400 未設定。常にブランク

第 7 章 CRTRPGMOD および CRTPGM コマンドによるプログラムの作成

プログラム作成のツー・ステップ処理は CRTRPGMOD を使ってソースをモジュールにコンパイルすることと、1 つ以上のモジュール・オブジェクトを CRTPGM を使ってプログラム中にバインドするということから構成されています。この処理によって、永続モジュールを作成することができます。これにより全体のアプリケーション・プログラムをコンパイルし直さなくてもアプリケーション・プログラムをモジュール化することができます。また、同じモジュールを別のアプリケーション・プログラムで再利用することもできます。

この章では以下のことを行う方法について示します。

- RPG IV ソースからモジュール・オブジェクトを作成する
- CRTPGM を使用して各モジュールを 1 つのプログラムにバインドする
- バインド・プログラムのリストの読み取り
- モジュールまたはプログラムの変更

WebSphere Development Studio Client for iSeries を使用します。これは推奨される方法であり、 ILE RPG プログラム作成の作成に関する説明は、製品のオンライン・ヘルプに記載されています。

モジュール・オブジェクトの作成

モジュールとは、ILE コンパイラーの出力である実行不能なオブジェクト (タイプ *MODULE) のことです。これは ILE プログラムの基本的な構成単位のことです。

ILE RPG モジュールは 1 つ以上のプロシージャー、およびモジュールのすべてのプロシージャーによって使用されるファイル制御ブロックと静的記憶域から構成されます。ILE RPG モジュールを構成するプロシージャーは次のとおりです。

- ソースを開始する H、F、D、I、C、および O 仕様書のセットで構成される任意指定**メイン・プロシージャー**。メイン・プロシージャーには、独自の LR 認識と論理サイクルがあります。すなわち、どちらもプログラム中の他の ILE RPG モジュールのプロシージャーの影響は受けません。
- P、D、および C 仕様書でコーディングされたゼロまたは 1 つ以上の**サブプロシージャー**。サブプロシージャーは RPG サイクルを使用しません。サブプロシージャーそれ自体にのみには使用可能なローカル記憶域があります。

メイン・プロシージャー (コーディングされている場合) は常にプログラム中の他のモジュールによって呼び出すことができます。サブプロシージャーは、モジュールに対してローカルとすることもエクスポートとすることもできます。ローカルの場合には、モジュールの別のプロシージャーだけがこれを呼び出すことができます。モジュールからエクスポートされた場合には、プログラムの任意のプロシージャーがこれを呼び出すことができます。

モジュールの作成は、ソース・メンバーのコンパイルと、それが正常に実行された場合の *MODULE オブジェクトの作成から構成されます。*MODULE オブジェクトにはモジュール内で参照されるインポートおよびエクスポートのリストが含まれます。またこれには、コンパイル時に要求した場合には、デバッグ・データも含まれます。

モジュールはそれ自身で実行することはできません。実行可能なプログラム・オブジェクト (タイプ *PGM) を作成するには 1 つ以上のモジュールを互いにバインドしなければなりません。また、サービス・プログラム・オブジェクト (タイプ *SRVPGM) を作成するにも 1 つ以上のモジュールを互いにバインドしなければなりません。その後で、ユーザーは静的プロシージャー呼び出しによってバインド・モジュール内のプロシージャーをアクセスします。

各モジュールを結合するこの能力により、以下のことが可能になります。

- 一般的には、より小さいプログラムとなるコードの一部を再利用する。より小さいプログラムは、パフォーマンスも高く、デバッグが容易です。
- 全体のプログラムのその他の部分にエラーを持ち込む機会を最小にして、共用のコードを保守する。
- 大きいプログラムをより効果的に管理する。モジュールによって、旧プログラムを別々に管理できるパーツに分けることができます。プログラムを拡張する必要がある場合には、変更されたモジュールをコンパイルし直すだけですみます。
- 混合言語のプログラムを作成する。この場合には、必要な仕事に最適な言語で書かれたモジュールを互いにバインドします。

モジュールの概念についての詳しい情報は、「*ILE 概念*」を参照してください。

CRTRPGMOD コマンドの使用

RPG モジュール作成 (CRTRPGMOD) コマンドを使用してモジュールを作成します。コマンドは、対話式に使用することも、バッチ入力ストリームの一部として使用することも、あるいはコマンド言語 (CL) プログラムから使用することもできます。

コマンドを対話式に使用していて、プロンプトが必要な場合には、CRTRPGMOD を入力して、F4 (プロンプト) を押してください。ヘルプが必要な場合には、CRTRPGMOD を入力して、F1 (ヘルプ) を押してください。

表 20 は CRTRPGMOD コマンドのパラメーター、およびシステム提供のそれらのデフォルトの値です。コマンドの構文図およびパラメーターの説明は 483 ページの『付録 C. 作成コマンド』に入っています。

表 20. CRTRPGMOD パラメーターおよび機能ごとにグループ化されたデフォルト値

	モジュールの指定	
#	MODULE(*CURLIB/*CTLSPEC)	作成されるモジュールの名前およびライブラリーを決める。
#	SRCFILE(*LIBL/QRPGLESRC)	指定する場合は、ソース・ファイルおよびライブラリーを示す。
# #	SRCMBR(*MODULE)	指定する場合は、ソースの仕様が入っているファイル・メンバーを示す。
	SRCSTMF(path)	指定する場合は、IFS のソース・ファイルへのパスを示す。

表 20. CRTRPGMOD パラメーターおよび機能ごとにグループ化されたデフォルト値 (続き)

INCDIR('path to directory 1:path to directory 2')	/copy ファイルおよび /include ファイルを検索するモジュールのリストを示す。
TEXT(*SRCMBRTXT)	モジュールの簡単な説明を用意する。
モジュールの作成	
GENLVL(10)	モジュール作成をエラーの重大度 (0 ～ 20) に条件付ける。
OPTION(*DEBUGIO)	*DEBUGIO/*NODEBUGIO は、入出力仕様書について停止点を生成するかどうかを決定する。
OPTION(*GEN)	*GEN/*NOGEN は、モジュールを作成するかどうかを決める。
OPTION(*NOSRCSTMT)	コンパイラーがデバッグ用にステートメント番号を生成する方法を指定する。
DBGVIEW(*STMT)	モジュールに含めるデバッグ・ビューがあれば、そのタイプを指定する。
OPTIMIZE(*NONE)	最適化をする場合、そのレベルを決める。
REPLACE(*YES)	既存のモジュールをモジュールに置き換えるかどうかを決める。
AUT(*LIBCRTAUT)	作成されるモジュールの権限のタイプを指定する。
TGTRLS(*CURRENT)	オブジェクトを実行するリリース・レベルを指定する。
BNDDIR(*NONE)	記号の解決に使用するバインディング・ディレクトリーを指定する。
ENBPFRCOL(*PEP)	パフォーマンス収集を使用可能にするかどうかを指定する。
DEFINE(*NONE)	コンパイルの開始前に定義する条件名を指定する。
PRFDTA(*NOCOL)	プログラムのプロファイル作成データ属性を指定する。
コンパイラー・リストの作成	
OUTPUT(*PRINT)	コンパイラー・リストを作成するかどうかを決める。
INDENT(*NONE)	リスト出力中に字下げを示すかどうかを決め、また字下げの印付けのための文字を識別する
OPTION(*XREF *NOSECLVL *SHOWCPY *EXPDDS *EXT *NOSHOWSKP *NOSRCSTMT)	コンパイラー・リストの内容を指定する。
データ変換オプション	
CVTOPT(*NONE)	外部記述ファイルからの各種のデータ・タイプを取り扱う方法を指定する。
ALWNULL(*NO)	null 可能フィールドからの値をモジュールが受け入れるかどうかを決める。
FIXNBR(*NONE)	無効な 10 進数データのどれをコンパイラーで修正するかを決める。
実行時の考慮事項	
SRTSEQ(*HEX)	使用すべき分類順序テーブルを指定する。
OPTION(*DEBUGIO)	*DEBUGIO/*NODEBUGIO は、入出力仕様書について停止点を生成するかどうかを決定する。
LANGID(*JOBRUN)	分類順序の言語識別コードを指定するために SRTSEQ と一緒に使用される。
INFOSTMF(path)	PGMINFO と一緒に使用し、PCML を受け取る IFS のストリーム・ファイルを指定する。
PGMINFO(*NONE)	*PCML は、モジュールに対し PCML (プログラム呼び出しマークアップ言語) を生成することを示す。

表 20. CRTRPGMOD パラメーターおよび機能ごとにグループ化されたデフォルト値 (続き)

TRUNCNBR(*YES)	パック 10 進数、ゾーン 10 進数、および固定形式の 2 進数フィールド命令の数値オーバーフローが起こった時に取るべき処置を指定する。
LICOPT(オプション)	ライセンス内部コード・オプションを指定します。

要求した場合には、CRTRPGMOD コマンドは、CRTBNDRPG コマンドで作成したリストとほとんどの部分が同様のコンパイラー・リストを作成します (CRTRPGMOD で作成されたリストに、バインド・セクションが入ることはありません)。

コンパイラー・リストの使用については、72 ページの『コンパイラー・リストの使用』を参照してください。サンプル・コンパイラー・リストは 507 ページの『付録 D. コンパイラー・リスト』に入っています。

NOMAIN モジュールの作成

この例では、CRTRPGMOD コマンドおよびそのデフォルトの設定を使用して NOMAIN モジュール・オブジェクト TRANSSVC を作成します。TRANSSVC には、他のモジュールのプロシージャーのトランザクション・サービスを実行するプロトタイプ・プロシージャーが入っています。TRANSSVC のソースは 87 ページの図 36 に示されています。TRANSSVC におけるプロシージャーのプロトタイプは、88 ページの図 37 に示すように /COPY メンバーに保管されます。

1. モジュール・オブジェクトを作成するには、以下のとおり入力してください。

   ```
   CRTRPGMOD MODULE(MYLIB/TRANSSVC)  SRCFILE(MYLIB/QRPGLESRC)
   ```

   モジュールは、コマンドに指定された名前、TRANSSVC でライブラリー MYLIB に作成されます。モジュールのソースは、ライブラリー MYLIB のファイル QRPGLESRC のソース・メンバー TRANSSVC です。

   次のコマンドの 1 つを使用して NOMAIN の入ったモジュールを別のモジュールとバインドします。

   a. CRTPGM コマンド
   b. CRTSRVPGM コマンド
   c. CRTBNDRPG コマンドここでは NOMAIN モジュールがバインディング・ディレクトリーに含まれます。
2. いったんバインドされると、このモジュール・オブジェクトはステートメント・ビューを使用してデバッグすることができます。このモジュールのコンパイラー・リストも生成されます。
3. コンパイラー・リストを見るには、以下の CL コマンドのいずれか 1 つを入力してください。
   - DSPJOB で、オプション 4 (*スプール・ファイルの表示*) を選択します。
   - WRKJOB
   - WRKOUTQ *待ち行列名*
   - WRKSPLF

```
 *===================================================================*
 * モジュール名:    TRANSSVC (トランザクション・サービス)
 * 関連ファイル:  なし
 * 関連ソース: TRANSRPT
 * エクスポート・プロシージャー:  Trans_Inc-- パラメーター・リスト
 *        にあるフィールドのデータを使用して
 *        取引の収入を計算します。すべての計算が行なわれた後にそれを
 *        呼び出し元に戻します。
 *
 *     Prod_Name --  製品番号が入っている入力パラメーターに基づいて
 *                   製品名を検索します。
 *===================================================================*
 * このモジュールはサブプロシージャーのみを含んでいます。
 * つまりこれは NOMAIN モジュールです。
H NOMAIN
 *-------------------------------------------------------------------
 * /COPY メンバーからプロトタイプを取り込みます。
 *-------------------------------------------------------------------
 /COPY TRANSP

 *-------------------------------------------------------------------
 * サブプロシージャー Trans_Inc
 *-------------------------------------------------------------------
P Trans_Inc       B                   EXPORT
D  Trans_Inc      PI            11P 2
D    ProdNum                    10P 0    VALUE
D    Quantity                    5P 0    VALUE
D    Discount                    2P 2    VALUE
D  Factor         S              5P 0
 *
C                   SELECT
C                   WHEN      ProdNum = 1
C                   EVAL      Factor = 1500
C                   WHEN      ProdNum = 2
C                   EVAL      Factor = 3500
C                   WHEN      ProdNum = 5
C                   EVAL      Factor = 20000
C                   WHEN      ProdNum = 8
C                   EVAL      Factor = 32000
C                   WHEN      ProdNum = 12
C                   EVAL      Factor = 64000
C                   OTHER
C                   EVAL      Factor = 0
C                   ENDSL
C                   RETURN    Factor * Quantity * (1 - Discount)
P  Trans_Inc      E
```

図 36. TRANSSVC メンバーのソース・ステートメント (1/2)

```
     *---------------------------------------------------------------
     * サブプロシージャー Prod_Name
     *---------------------------------------------------------------
    P Prod_Name       B                   EXPORT
    D  Prod_Name      PI            40A
    D    ProdNum                    10P 0    VALUE
     *
    C                   SELECT
    C                   WHEN      ProdNum = 1
    C                   RETURN    'Large'
    C                   WHEN      ProdNum = 2
    C                   RETURN    'Super'
    C                   WHEN      ProdNum = 5
    C                   RETURN    'Super Large'
    C                   WHEN      ProdNum = 8
    C                   RETURN    'Super Jumbo'
    C                   WHEN      ProdNum = 12
    C                   RETURN    'Incredibly Large Super Jumbo'
    C                   OTHER
    C                   RETURN    '***Unknown***'
    C                   ENDSL
    P  Prod_Name      E
```

図 36. TRANSSVC メンバーのソース・ステートメント (2/2)

```
     * Trans_Inc のプロトタイプ
    D  Trans_Inc      PR            11P 2
    D    Prod                       10P 0    VALUE
    D    Quantity                    5P 0    VALUE
    D    Discount                    2P 2    VALUE

     * Prod_Name のプロトタイプ
    D  Prod_Name      PR            40A
    D    Prod                       10P 0    VALUE
```

図 37. TRANSP /COPY メンバーのソース・ステートメント

ソース・デバッグのためのモジュールの作成

この例では、ソース・デバッガーを使ってデバッグすることのできる ILE RPG モジュール・オブジェクトを作成します。モジュール TRANSRPT には、報告書処理を実行するメイン・プロシージャーが含まれています。これは必要なタスクを行うために TRANSSVC のプロシージャーを呼び出します。このモジュールのソースが 89 ページの図 38 に示されています。

モジュール・オブジェクトを作成するには、以下のとおり入力してください。

```
CRTRPGMOD MODULE(MYLIB/TRANSRPT)  SRCFILE(MYLIB/QRPGLESRC)
DBGVIEW(*SOURCE)
```

モジュールはそれが基礎となっているソース・ファイルと同じ名前、すなわち TRANSRPT でライブラリー MYLIB に作成されます。このモジュール・オブジェクトはソース・ビューを使用してデバッグすることができます。その他の使用可能なビューについては、 219 ページの『デバッグのためのプログラムの準備』を参照してください。

TRANSRPT モジュールのコンパイラー・リストが作成されます。

```
     *=================================================================*
     * モジュール名:  TRANSRPT
     * 関連ファイル:  TRNSDTA  (PF)
     * 関連ソース:    TRANSSVC (トランザクション・サービス)
     * エクスポート・プロシージャー:  TRANSRPT
     *       プロシージャー TRANSRPT は、物理ファイル TRNSDTA に保管され
     *       ているすべてのトランザクション・レコードを読み取ります。
     *       これは、計算を実行して値を戻すサブプロシージャー
     *       Trans_Inc を呼び出します。次に Prod_Name を呼び出して、
     *       製品名を判別します。その後で TRANSRPT は
     *       トランザクション・レコードを印刷します。
     *=================================================================*
FTRNSDTA   IP   E             DISK
FQSYSPRT   O    F   80        PRINTER       OFLIND(*INOF)
 /COPY QRPGLE,TRANSP
 * プロシージャー 'Prod_Name' の戻り値と類似した製品名の
 * 読み取り可能バージョンを定義します。
D    ProdName      S              30A
D    Income        S              10P 2
D    Total         S              +5      LIKE(Income)
 *
ITRNSREC       01
 *  サブプロシージャー Trans_Inc を使用して収入を計算します。
C                   EVAL      Income = Trans_Inc(PROD : QTY : DISC)
C                   EVAL      Total = Total + Income
 *  製品名を検出します。
C                   EVAL      ProdName = Prod_Name(PROD)
OQSYSPRT   H    1P                      1
O         OR    OF
O                                            12 'Product name'
O                                            40 'Quantity'
O                                            54 'Income'
OQSYSPRT   H    1P                      1
O         OR    OF
O                                            30 '----------+
O                                               ----------+
O                                               ----------'
O                                            40 '--------'
O                                            60 '------------'
OQSYSPRT   D    01                      1
O                       ProdName             30
O                       QTY           1      40
O                       Income        1      60
OQSYSPRT   T    LR                       1
O                                               'Total: '
O                       Total         1
```

図 38. TRANSRPT モジュールのソース・ステートメント

ファイル TRNSDTA の DDS を 90 ページの図 39 に示します。/COPY メンバーは 88 ページの図 37 に示されています。

```
A*****************************************************************
A* 関連ファイル:  TRNSRPT                                        *
A* 説明:          これは物理ファイル TRNSDTA です。これには      *
A*                TRNSREC というレコード様式が 1 つあります。    *
A*****************************************************************
A* 部品トランザクション・ファイル -- TRNSDTA
A          R TRNSREC
A            PROD          10S 0       TEXT('Product')
A            QTY            5S 0       TEXT('Quantity')
A            DISCOUNT       2S 2       TEXT('Discount')
```

図 39. TRNSDTA の DDS

追加の例

モジュール作成の追加の例については以下を参照してください。

- サービス・プログラム用のモジュールの作成例については、 104 ページの『サンプル・サービス・プログラム』。
- サービス・プログラムで使用するモジュールの作成例については、 108 ページの『プログラムへのバインド』。
- 実行時配列用の記憶域の動的割り振りのためのモジュールの作成例については、 131 ページの『ILE バインド可能 API を使用したユーザー独自のヒープ管理』。
- サンプル・デバッグ・プログラムに使用する RPG および C モジュールの作成例については、 267 ページの『デバッグ用サンプル・ソースの例』。

バインドされた ILE RPG モジュールの動作

ILE RPG では、*メイン・プロシージャー* は LR 意味構造の有効範囲と RPG サイクルの境界になります。*モジュール* がオープン・ファイルの有効範囲です。

任意の ILE プログラムに、複数の活動中 RPG サイクルがある場合があります。すなわち、メイン・プロシージャーをもつ各 RPG モジュールごとに 1 つの RPG サイクルがあるということです。サイクルは独立しています。1 つのメイン・プロシージャーの LR での設定は別のプロシージャーのサイクルには影響を与えません。

関連する CL コマンド

モジュールについては、以下の CL コマンドを使用することができます。

- モジュールの表示 (DSPMOD)
- モジュールの変更 (CHGMOD)
- モジュールの削除 (DLTMOD)
- モジュールの処理 (WRKMOD)

これらのコマンドについての詳細は、Web サイト http://www.ibm.com/eserver/iseries/infocenter で **iSeries 400 Information Center** の中の『プログラミング』カテゴリーの『*CL および API*』の節を参照してください。

プログラムへのモジュールのバインド

バインドとは、1 つ以上のモジュールと任意のサービス・プログラムを結合し、その間で渡される記号を解決することによって実行可能な ILE プログラムを作成するプロセスです。この結合および解決を行うシステム・コードは、iSeries システムでは**バインダー**と呼ばれています。

この結合プロセスの一部として、プロシージャーは始動プロシージャー、すなわち**プログラム入力プロシージャー**として識別される必要があります。プログラムが呼び出されるとプログラム入力プロシージャー はコマンド行からパラメーターを受け取り、そのプログラムの初期制御権が与えられます。プログラム入力プロシージャーに関連するユーザーのコードが、ユーザー入り口プロシージャーです。

ILE RPG モジュールにメイン・プロシージャーがあると、これには暗黙的にプログラム入力プロシージャーも含まれています。したがって、どの ILE RPG モジュールも NOMAIN モジュールでない限り、入り口モジュールとして指定することができます。

図 40 はプログラム・オブジェクトの内部構造についての考え方を示しています。ここでは、2 つのモジュール TRANSRPT と TRANSSVC をバインドして作成されたプログラム・オブジェクト TRPT が示されています。TRANSRPT は入り口モジュールです。

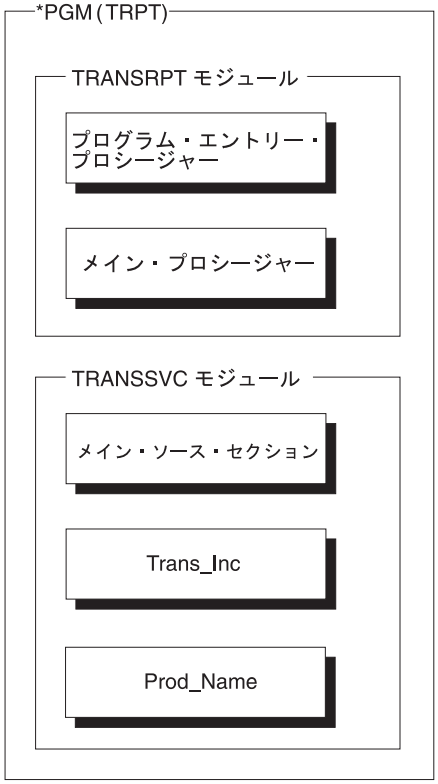


図 40. プログラム TRPT の構造

バインドされたオブジェクト内では、プロシージャーは静的プロシージャー呼び出しを使用して相互に関係づけることができます。これらのバインド呼び出しは外部

呼び出しより高速です。したがって、多くのバインド呼び出しのある単一のバインド済みプログラムから成るアプリケーションは、多くの外部呼び出しがある各プログラムから構成される同様のアプリケーションより高速で実行されるはずです。

モジュールの相互のバインドに加え、これをサービス・プログラムにバインドすることもできます (*SRVPGM と入力)。サービス・プログラムによって、プログラム・モジュールから独立してモジュールをコーディングおよび保守することができます。共通のルーチンをサービス・プログラムとして作成しておいて、ルーチンが変更された場合には、サービス・プログラムを再度バインドすることによってこの変更を組み込むことができます。これらの共通ルーチンを使用するプログラムは再作成する必要がありません。サービス・プログラムの作成についての情報は、101 ページの『第 8 章 サービス・プログラムの作成』を参照してください。

バインド・プロセスおよびバインダーについての情報は、「*ILE 概念*」を参照してください。

CRTPGM コマンドの使用

プログラムの作成 (CRTPGM) コマンドは、前に作成した 1 つ以上のモジュール、および必要な場合には 1 つ以上のサービス・プログラムからプログラム・オブジェクトを作成します。ILE モジュール作成コマンド、CRTRPGMOD、CRTCMOD、CRTCBLMOD、または CRTCLMOD のいずれかによって作成したモジュールをバインドすることができます。

注: 必要なモジュールまたはサービス・プログラム (あるいはその両方) は、CRTPGM コマンドを使用する前に作成しておかなければなりません。

CRTPGM コマンドを使ってプログラム・オブジェクトを作成する前に、以下のことを実行しなければなりません。

1. プログラム名を決める。
2. 1 つ以上のモジュールと、必要であれば、プログラム・オブジェクトにバインドしたいサービス・プログラムを識別する。
3. 入り口モジュールを識別する。

   どのモジュールにプログラム入力プロシージャーが入っているかを、CRTPGM の ENTMOD パラメーターによって指示します。デフォルト値は ENTMOD(*FIRST) であり、MODULE パラメーターのリストで見つかった最初のプログラム入力プロシージャーを含むモジュールが入り口モジュールであることを意味しています。

   メイン・プロシージャーのモジュールが 1 つだけあるとします。すなわち、1 つを除いてすべてのモジュールに NOMAIN が指定されていると仮定し、ユーザーはデフォルト値 (*FIRST) を受け入れることができるとします。二者択一的にユーザーは (*ONLY) を指定することができます。この場合には、実際に 1 つだけのモジュールがメイン・プロシージャーを持っていることがチェックされます。例えば、次の両方の状況において ENTMOD(*ONLY) を指定することができます。

   - main() 関数のない C モジュールを RPG モジュールにバインドする。
   - 2 つの RPG モジュールをバインドするが、1 つは制御仕様書に NOMAIN がある。

注: メイン・プロシージャーに複数の ILE RPG モジュールをバインドしている場合には、プログラムが呼び出された時に制御を受け取りたいモジュールの名前を指定する必要があります。また、メイン・プロシージャーのモジュールが MODULE パラメーターに指定されたリストのメイン・プロシージャーのその他のモジュールに先行する場合には、*FIRST を指定することができます。

4. プログラムを使用する活動化グループを識別する。

   ユーザーのプログラムに特殊な要件がないか、あるいは使用するグループが確定していない場合には、活動化グループ QILE を指定してください。一般に、アプリケーションをそれ自身の活動化グループで実行するのが望ましいことです。したがって、アプリケーション・プログラムの名前を付けた後に活動化グループの名前を付けることができます。

   CRTPGM のデフォルトの活動化グループが *NEW であることに注意してください。これはプログラムがその自身の活動化グループで実行され、活動化グループはプログラムが終了する時に終了することを意味します。LR を設定してもしなくても、プログラムは次に呼び出された時にデータの新しいコピーをもちます。活動化グループの詳細については、120 ページの『活動化グループの指定』を参照してください。

CRTPGM コマンドを使ってプログラム・オブジェクトを作成するには、次のステップを行ってください。

1. CRTPGM コマンドを入力する。
2. コマンド・パラメーターに適切な値を入力する。

表 21 は CRTPGM コマンドのパラメーターとそれらのデフォルト値をリストしたものです。CRTPGM コマンドおよびそのパラメーターについての全般的な説明は、Web サイト http://www.ibm.com/eserver/iseries/infocenter で **iSeries 400 Information Center** の 中の『*プログラミング*』カテゴリーの『*CL* および *API*』の節を参照してください。

表 21. CRTPGM コマンドのパラメーターとそのデフォルト値

パラメーター・グループ	**パラメーター (デフォルト値)**
識別	PGM(*ライブラリー名/プログラム名*) MODULE(*PGM)
プログラム・アクセス	ENTMOD(*FIRST)
バインド	BNDSRVPGM(*NONE) BNDDIR(*NONE)
実行時	ACTGRP(*NEW)
その他	OPTION(*GEN *NODUPPROC *NODUPVAR *WARN *RSLVREF) DETAIL(*NONE) ALWUPD(*YES) ALWRINZ(*NO) REPLACE(*YES) AUT(*LIBCRTAUT) TEXT(*ENTMODTXT) TGTRLS(*CURRENT) USRPRF(*USER)

CRTPGM コマンドを入力すると、システムは以下の処置を行います。

1. リストされたモジュールをプログラム・オブジェクトになるものにコピーし、サービス・プログラムをこのプログラム・オブジェクトに連係する。
2. プログラム入力プロシージャーが入っているモジュールを指定し、このモジュールの最初のインポートを見つける。
3. リストされている順序にモジュールをチェックし、最初のインポートをモジュール・エクスポートと突き合わせる。
4. 最初のモジュールに戻り、次のインポートを見つける。
5. 最初のモジュール内のすべてのインポートを分析解決する。
6. 次のモジュールへ続け、すべてのインポートを分析解決する。
7. すべてのインポートの分析解決が終わるまで、後続の各モジュール内のすべてのインポートを分析解決する。
8. いずれかのインポートがエクスポートによって解決できない場合は、プログラム・オブジェクトを作成せずにバインド処理は打ち切られる。
9. すべてのインポートが解決されるとバインド処理は完了し、プログラム・オブジェクトが作成される。

注: 変数またはプロシージャーをエクスポートするよう (EXPORT キーワードを使用して) 指定した場合には、この変数またはプロシージャーの名前を、バインド済みプログラム・オブジェクト内の別のプロシージャーの変数またはプロシージャーと同じにすることが可能です。この場合に予期しない結果となることがあります。この状況の処理方法についての情報は、「*ILE 概念*」を参照してください。

複数モジュールのバインド

この例では CRTPGM コマンドを使って 2 つの ILE RPG モジュールをプログラム TRPT にバインドする方法を示します。このプログラムでは、以下のことを行います。

- モジュール TRANSRPT がファイル TRNSDTA から各トランザクション・レコードを読み取る。
- 次に式中のバインド呼び出しを使ってモジュール TRANSSVC のプロシージャー Trans_Inc および Proc_Name を呼び出す。
- Trans_Inc が各トランザクションに関係する収入を計算し、その値を呼び出し元に戻す。
- Proc_Name がプロダクト名を決定しそれを戻す。
- 次に TRANSRPT がトランザクション・レコードを印刷する。

TRANSRPT、TRANSSVC、および TRNSDTA のソース・ステートメントはそれぞれ、89 ページの図 38、 87 ページの図 36、および 90 ページの図 39 に示されています。

1. 最初にモジュール TRANSRPT を作成する。以下のとおり入力してください。

   ```
   CRTRPGMOD MODULE(MYLIB/TRANSRPT)
   ```

2. 次に、以下のとおり入力して、モジュール TRANSSVC を作成する。

```
CRTRPGMOD MODULE(MYLIB/TRANSSVC)
```

3. プログラム・オブジェクトを作成するために、以下のとおり入力する。

```
CRTPGM PGM(MYLIB/TRPT) MODULE(TRANSRPT TRANSSVC)
       ENTMOD(*FIRST) ACTGRP(TRPT)
```

CRTPGM コマンドは、ライブラリー MYLIB にプログラム・オブジェクト TRPT を作成します。

TRANSRPT が MODULE パラメーターに最初にリストされることに注意してください。ENTMOD(*FIRST) では、プログラム入力プロシージャーのある最初のモジュールを検索します。2 つのモジュールのうちプログラム入力プロシージャーを持っているのは 1 つだけですから、2 つのモジュールをいずれを先に入力しても構いません。

プログラム TRPT は指定の活動化グループ TRPT 内で実行します。プログラムは、他のプログラムがその資源に影響を与えないことを確実にするために指定のグループ内で実行します。

図 41 は、TRPT が実行される時に作成させる出力ファイルを示したものです。

```
Product name                    Quantity        Income
------------------------------  --------        ------------
Large                                245          330,750.00
Super                                 15           52,500.00
Super Large                            0                .00
Super Jumbo                          123        2,952,000.00
Incredibly Large Super Jumbo          15          912,000.00
***Unknown***                         12                .00
Total:         4,247,250.00
```

図 41. TRPT のファイル QSYSPRT

追加の例

プログラム作成の追加の例については以下を参照してください。

- モジュールおよびサービス・プログラムのバインドの例については、108 ページの『プログラムへのバインド』。
- RPG と C モジュールから構成されるプログラムの作成例については、267 ページの『デバッグ用サンプル・ソースの例』。

関連する CL コマンド

プログラムでは、以下の CL コマンドを使用することができます。

- プログラム変更 (CHGPGM)
- プログラム削除 (DLTPGM)
- プログラム表示 (DSPPGM)
- プログラム参照表示 (DSPPGMREF)
- プログラムの更新 (UPDPGM)

- プログラムの処理 (WRKPGM)

これらのコマンドについての詳細は、Web サイト
http://www.ibm.com/eserver/iseries/infocenter で **iSeries 400 Information Center** の中の『プログラミング』カテゴリーの『*CL および API*』の節を参照してください。

バインダー・リストの使用

バインド・プロセスは使用される資源、検出された記号とオブジェクト、およびバインド・プロセスで解決された、あるいはされなかった問題を説明したリストを作成することができます。このリストは CRTPGM コマンドを入力するのに使用するジョブのスプール・ファイルとして作成されます。デフォルトのコマンドではこの情報を作成しませんが、詳細レベルには次の 3 つのレベルがあり、DETAIL パラメーター値を選択することによって生成することができます。

- *BASIC
- *EXTENDED
- *FULL

バインダー・リストには、DETAIL に指定された値によって、以下のセクションが含まれます。

表 22. DETAIL パラメーターに基づくバインダー・リストのセクション

セクション名	*BASIC	*EXTENDED	*FULL
コマンド・オプションの要約	X	X	X
簡易要約表	X	X	X
拡張要約表		X	X
バインダー情報リスト		X	X
相互参照表			X
バインド統計			X

このリストの情報は、バインドが正常に行われなかった場合に問題を診断するのを容易にしたり、あるいはバインダーの処理で検出されたものについてのフィードバックを与えることができます。モジュールまたはプログラムのモジュール・ソースを保管するファイルに、ILE プログラムのリストを保管したい場合があります。このリストをデータベース・ファイルへコピーするために、スプール・ファイル・コピー (CPYSPLF) コマンドを使用することができます。

注: CRTBNDRPG コマンドはバインダー・リストを作成しません。しかし、バインドの段階でバインド・エラーが起こった場合には、エラーはジョブ・ログに通知され、コンパイラー・リストにこの影響に対するメッセージが入れられます。

基本バインダー・リストの例については、 110 ページの『サンプル・バインダー・リスト』を参照してください。

バインダー・リストの詳しい情報は、「*ILE 概念*」を参照してください。

モジュールまたはプログラムの変更

機能拡張または保守の目的で、ILE オブジェクトの変更が必要となることがあります。変更必要個所は、デバッグ情報または CRTPGM コマンドからのバインダー・リストを使用して、判断することができます。この情報から、どのモジュールに変更が必要か、そしてしばしばどのプロシージャーまたはフィールドに変更が必要かを判別することができます。

さらに、モジュールまたはプログラムの最適化レベルあるいはプログラム識別情報を変更したい場合があります。これは、プログラムまたはモジュールをデバッグしたい時、あるいはプログラムを実行用に入れる準備ができている時によく起こります。このような変更は、該当するオブジェクトを再作成するより迅速に実行でき、少ないシステム資源ですみます。

最後に、アプリケーション・プログラムを完成させた後でプログラム・サイズを小さくする必要がある場合があります。ILE プログラムには、プログラムに追加される付加データがあるため、同様の OPM プログラムよりも大きくなることがあります。

上記のものには、変更のためにそれぞれ異なるデータが必要です。ILE プログラムの場合、必要な資源を使用できないことがあります。

以下の項で、次のことについて説明します。

- プログラムの更新
- 最適化レベルの変更
- プログラム識別情報の変更
- オブジェクト・サイズの縮小

注: この項のこれ以降、「オブジェクト」という用語は ILE モジュールまたは、ILE プログラムのいずれかを指すときに使用します。

UPDPGM コマンドの使用

一般に、プログラムは必要に応じてモジュールを置き換えることによって更新することができます。例えば、新しいプロシージャーをモジュールに追加した場合、モジュール・オブジェクトをコンパイルし直してからプログラムを更新します。プログラムを再作成する必要はありません。これは、アプリケーションを他の場所に提供する場合に有用です。ユーザーは変更モジュールを送信するだけでよく、受け取った場所は、UPDPGM または UPDSRVPGM コマンドを使用してアプリケーションを更新することができます。

UPDPGM コマンドはプログラムおよびモジュール・オブジェクトの両方を処理します。このコマンドのパラメーターは CRTPGM コマンドのパラメーターに非常によく類似しています。例えば、プログラムのモジュールを置き換えるには、MODULE パラメーターにモジュール名を入力しライブラリー名を入力します。UPDPGM コマンドでは、置き換えられるモジュールをプログラムの作成時と同じライブラリーに入れることが必要です。すべてのモジュールの置き換え、または一部のサブセットを指定することができます。

UPDPGM コマンドでは、モジュール・オブジェクトが存在していなければなりません。このように、独立したコンパイルおよびバインド・ステップを使用してプログラムを作成していれば、コマンドの使用が一層簡単になります。モジュール・オブジェクトが既に存在しているので、コマンドを出す時に名前とライブラリーを指定するだけですみます。

CRTBNDRPG コマンドによって作成されたプログラムを更新するには、変更モジュールがライブラリー QTEMP にあることを確認する必要があります。これは、CRTBNDRPG コマンドが出された時に使用される一時モジュールが QTEMP に作成されるためです。モジュールが QTEMP にあると、そのモジュールを置き換えるために UPDPGM コマンドを出すことができます。

詳細については、「*ILE 概念*」を参照してください。

最適化レベルの変更

オブジェクトの**最適化**とは、コンパイルされたコードを見て、実行時パフォーマンスをできるだけ高速にするには何ができるかを判断し、必要な変更を行うことを意味します。一般に、最適化の要求が高くなるにしたがって、オブジェクトの作成にかかる時間が長くなります。実行時には、高度に最適化されたプログラムまたはサービス・プログラムは、対応する最適化されていないプログラムあるいはサービス・プログラムより高速で実行されることになります。

しかし、高水準の最適化では、デバッグ・セッションに表示された時、あるいは例外からの回復後にフィールドの値が正確でないことがあります。さらに、最適化処理によって一部のステートメントが再配置されたり削除されることがあるため、最適化されたコードでは、ソース・デバッグ・プログラムによって使用される停止点およびステップ・ロケーションが変更されていることがあります。

フィールドの内容が (特に例外回復後に) 最新の値を示していることを保証するために、対応する定義仕様書に NOOPT キーワードを使用することができます。詳細については、282 ページの『最適化に関する考慮事項』を参照してください。

デバッグ時のこの問題を回避するには、プログラムをデバッグする時にはフィールドを正しく表示するためにモジュールの最適化レベルを低くし、その後プログラムを実行用に使用可能にする時に、プログラムの効率を高めるためにレベルを再び上げることができます。

プログラム・オブジェクトの現行の最適化レベルを判別するためには、DSPPGM コマンドを使用してください。このコマンドの表示 3 は現行レベルを示しています。プログラムの最適化レベルを変更するには、CHGPGM コマンドを使用してください。プログラムの最適化パラメーターで、*FULL、*BASIC、*NONE のいずれかを指定することができます。これらは、それぞれの作成コマンドの OPTIMIZE パラメーターで指定できるのと同じ値です。コマンドの実行時に、プログラムが自動的に再作成されます。

同様に、現行のモジュール の最適化レベルを決定するために、 DSPMOD コマンドを使用してください。このコマンドの表示 1、ページ 2 は現行レベルを示しています。最適化レベルを変更するには、CHGMOD コマンドを使用してください。次に UPDPGM または CRTPGM を使用してプログラムを再作成する必要があります。

プログラム識別情報の除去

プログラム識別情報には、オブジェクトと一緒に保管でき、ソースの再コンパイルなしでオブジェクトを変更できる種類のデータが含まれています。このデータを追加するとオブジェクトのサイズが増します。したがって、オブジェクトのサイズを減らすためにデータを除去したくなるかもしれません。しかし、データを除去してしまうとプログラム識別情報も除去されます。データを置き換えるためには、ソースをコンパイルし直し、プログラムを再作成しなければなりません。データのタイプは以下のとおりです。

作成データ
*CRTDTA 値で表されます。このデータはコードを機械語命令に変換するのに必要です。最適化レベルを変更する前に、オブジェクトにこのデータがなければいけません。

デバッグ・データ
*DBGDTA 値で表されます。このデータはオブジェクトをデバッグするのに必要です。

プロファイル作成データ
*BLKORD および *PRCORD 値で表されます。このデータは、システムがブロック・オーダーおよびプロシージャー・オーダーのプロファイル作成データを再度適用できるようにするために必要です。

CHGPGM コマンド、または CHGMOD コマンドはそれぞれ、プログラムまたはモジュールからデータの一部、または全部を除去するために使います。すべてのプログラム識別情報の除去で、オブジェクトは最小サイズ (圧縮なしで) になります。この場合、オブジェクトを再作成しないかぎり、オブジェクトを変更することはできません。したがって、プログラムの作成に必要なすべてのソースをもっているか、あるいは CRTDATA と同等のプログラム・オブジェクトをもっていることを確認してください。これを再作成するには、ソース・コードへのアクセスの認可をもっていなければなりません。

オブジェクト・サイズの縮小

ILE プログラムまたはモジュールに関連した作成データ (*CRTDTA) は、オブジェクト・サイズの半分以上を占める場合があります。このデータを除去するかまたは圧縮することにより、プログラムのための 2 次的記憶域の必要量を大きく減らすことになります。

データを除去する場合には、プログラムの作成に必要なすべてのソースをもっているか、あるいは CRTDATA による同等のプログラム・オブジェクトをもっていることを確認してください。そうでない場合には、オブジェクトを変更することはできません。

代わりに、オブジェクトの圧縮 (CPROBJ) コマンドを使用して、オブジェクトを圧縮する方法もあります。圧縮されたオブジェクトは圧縮されていないオブジェクトより小さなシステム記憶域を占めます。圧縮プログラムが呼び出された場合には、実行可能コードが入っているオブジェクトの一部が自動的に圧縮解除されます。また、オブジェクトの圧縮解除 (DCPOBJ) コマンドを使用することによって、圧縮オブジェクトを圧縮解除することができます。

これらの CL コマンドについての詳細は、Web サイト http://www.ibm.com/eserver/iseries/infocenter で **iSeries 400 Information Center** の中の『プログラミング』カテゴリーの『*CL および API*』の節を参照してください。

第 8 章 サービス・プログラムの作成

この章では以下のことについて説明します。

- サービス・プログラム概念の概要
- サービス・プログラム作成の方針
- CRTSRVPGM コマンドの簡単な説明
- サービス・プログラムの例

WebSphere Development Studio Client for iSeries を使用します。これは推奨される方法であり、サービス・プログラムの作成に関する説明は、製品のオンライン・ヘルプに記載されています。

サービス・プログラムの概要

サービス・プログラムは、他のバインドされたプログラムのプロシージャーによって呼び出すことができる、一組のプロシージャーのセットから成るバインドされたプログラム (タイプ *SRVPGM) です。

サービス・プログラムは通常、アプリケーション内およびアプリケーション間で頻繁に呼び出される共通機能として使用されます。例えば、ILE コンパイラーは数学的機能および入出力ルーチンのような実行時サービスを提供するサービス・プログラムを使用します。サービス・プログラムによって、再利用、保守の単純化、記憶域所要量の減少が可能になります。

サービス・プログラムは、以下の 2 つの点でプログラムとは異なっています。

- これにはプログラム入力プロシージャーが含まれていない。このことは、CALL 命令を用いてサービス・プログラムを呼び出すことはできないことを意味します。
- サービス・プログラムは、参照によるバインドを用いてプログラム、または他のサービス・プログラムにバインドされます。

サービス・プログラムをプログラム中にバインドする場合には、サービス・プログラムの内容はバインド・プログラムにはコピーされません。その代わり、サービス・プログラムの連係情報がプログラム中にバインドされます。これは、モジュールをプログラムにバインドするのに使用される静的バインド・プロセスと対比して、「参照によるバインド」と呼ばれます。

サービス・プログラムが参照によってプログラムにバインドされるため、バインド・プロシージャー呼び出しを使って、サービス・プログラムのエクスポート・プロシージャーを呼び出すことができます。サービス・プログラムが呼び出されるまでバインドが完了しないので、初期呼び出しにはある程度のオーバーヘッドがかかります。しかし、そのプロシージャーに対する後続の呼び出しはプログラム呼び出しより高速です。

サービス・プログラムに含まれるエクスポートのセットは、これによって指定されるサービスに対するインターフェースです。 サービス・プログラムの表示

(DSPSRVPGM) コマンドまたはサービス・プログラム・リストを使って、プロシージャーの呼び出しに使用可能な変数およびプロシージャー名を調べることができます。サービス・プログラム PAYROLL に関連したエクスポートを参照するには、以下を入力してください。

```
DSPSRVPGM PAYROLL DETAIL(*PROCEXP *DATAEXP)
```

サービス・プログラム作成の方針

サービス・プログラム作成の際、以下のことを覚えておいてください。

1. 後日プログラムを更新する意図があるかどうか
2. 更新の中にインターフェース (すなわち、使用されているインポートおよびエクスポート) に対する変更が含まれるかどうか。

サービス・プログラムに対するインターフェースが変更された場合には、元のサービス・プログラムにバインドされているすべての プログラムを再バインドしなければなりません。しかし、必要な変更が上向きの互換性がある場合、バインダー言語を使用してサービス・プログラムが作成されていれば、再バインドの量を減らすことができます。この場合には、新しいエクスポートを識別するためのバインダー言語ソースの更新後に、これらを使用するプログラムだけを再バインドする必要があります。

ヒント

サブプロシージャーだけのモジュール (すなわち、制御仕様書でキーワード NOMAIN を指定したモジュール) を計画している場合には、これをサービス・プログラムとして作成することができます。システム上にはサービス・プログラムのコピーが 1 つしか必要でないのでモジュールに必要な記憶域がより小さくなります。

また、制御仕様書の COPYRIGHT キーワードを使用してサービス・プログラムを版権表示することができます。

バインダー言語によって、サービス・プログラムのエクスポートを制御することができます。以下のことを行いたい場合には、この制御が非常に有用です。

- サービス・プログラム・ユーザーからの特定のサービス・プログラム・プロシージャーのマスクをする
- 問題を修正する
- 機能を拡張する
- アプリケーションのユーザーに対して変更の影響を軽減する

サービス・プログラムを作成するためのバインダー言語の使用例については、 104 ページの『サンプル・サービス・プログラム』を参照してください。

バインダー言語、エクスポートのマスク、およびその他のサービス・プログラムの概念については、「*ILE 概念*」を参照してください。

CRTSRVPGM を使用したサービス・プログラムの作成

サービス・プログラムの作成 (CRTSRVPGM) コマンドを使用してサービス・プログラムを作成します。いかなる ILE モジュールも、サービス・プログラムにバインドすることができます。このモジュールは、このモジュール使用のサービス・プログラムを作成する前に存在していなければなりません。

表 23 は CRTSRVPGM のパラメーターおよびそれらのデフォルト値をリストしたものです。 CRTSRVPGM コマンドおよびそのパラメーターについての完全な説明は、Web サイト http://www.ibm.com/eserver/iseries/infocenter で **iSeries 400 Information Center** の中の『プログラミング』カテゴリーの『*CL および API*』の節を参照してください。

表 23. CRTSRVPGM コマンドのパラメーターとそのデフォルト値

パラメーター・グループ	**パラメーター (デフォルト値)**
識別	SRVPGM(*ライブラリー名/サービス・プログラム名*) MODULE(*SRVPGM)
プログラム・アクセス	EXPORT(*SRCFILE) SRCFILE(*LIBL/QSRVSRC) SRCMBR(*SRVPGM)
バインド	BNDSRVPGM(*NONE) BNDDIR(*NONE)
実行時	ACTGRP(*CALLER)
その他	OPTION(*GEN *NODUPPROC *NODUPVAR *WARN *RSLVREF) DETAIL(*NONE) ALWUPD(*YES) ALWRINZ(*NO) REPLACE(*YES) AUT(*LIBCRTAUT) TEXT(*ENTMODTXT) TGTRLS(*CURRENT) USRPRF(*USER)

CRTSRVPGM コマンドの使用例については、 107 ページの『サービス・プログラムの作成』を参照してください。

サービス・プログラムの変更

ユーザーは、プログラム・オブジェクトと同じ方法でサービス・プログラムを更新または変更することができます。すなわち、以下のことが行えます。

- サービス・プログラムの更新 (UPDSRVPGM を使用)
- 最適化レベルの変更 (CHGSRVPGM を使用)
- プログラム識別情報の除去 (CHGSRVPGM を使用)
- サイズの減少 (CPROBJ を使用)

上記の事項について詳しくは、 97 ページの『モジュールまたはプログラムの変更』を参照してください。

関連する CL コマンド

以下の CL コマンドも、サービス・プログラムについて使用されます。

- サービス・プログラム変更 (CHGSRVPGM)
- サービス・プログラム表示 (DSPSRVPGM)
- サービス・プログラム削除 (DLTSRVPGM)
- サービス・プログラム更新 (UPDSRVPGM)
- サービス・プログラム処理 (WRKSRVPGM)

サンプル・サービス・プログラム

以下の例は、文字ストリングを 16 進数に変換するサービス・プログラム CVTTOHEX の作成方法を示しています。次の 2 つのパラメーターがサービス・プログラムに渡されます。

1. 変換すべき文字フィールド (InString)
2. 2 バイトの 16 進数が入る文字フィールド (HexString)

フィールド HexString は変換の結果を入れ、また変換するストリングの長さを指示するために用いられます。例えば、文字ストリング 30 桁を渡すが最初の 10 桁だけを変換したい場合には、20 バイト (2 × 10) の 2 番目のパラメーターを渡すことになります。渡されたフィールドの長さを基に、サービス・プログラムは処理の長さを決定します。

105 ページの図 42 はサービス・プログラムのソースを示したものです。107 ページの図 43 は CvtToHex のプロトタイプを含む /COPY メンバーを示したものです。

サービス・プログラム中に含まれるプロシージャーの基本ロジックは、以下に示すとおりです。

1. 操作記述子は渡されるパラメーターの長さを決めるために使用される。
2. 変換すべき長さが決められる。文字ストリングの長さよりも小さいか、あるいは 16 進ストリング・フィールドの 2 分の 1 の長さになります。
3. ストリングの各文字はサブルーチン GetHex を使用して 2 バイトの 16 進値に変換されます。

   GetHex が、実行時のパフォーマンスを向上させるため、サブプロシージャーではなくサブルーチンとしてコーディングされていることに注意してください。EXSR 命令はバインド呼び出しよりずっと高速で実行され、この例では、GetHex が頻繁に呼び出されます。
4. プロシージャーが呼び出し元に戻る。

サービス・プログラムは操作記述子を利用しますが、これは渡されたパラメーターの正確な性質 (この場合には長さ) が前もって分らない場合に、使用される ILE 構造です。操作記述子は、CALLB 命令の命令拡張子 (D) を指定した時またはプロトタイプに OPDESC を指定した時にプロシージャーに対する呼び出し時に作成されます。

操作記述子を使用するためには、サービス・プログラムで ILE バインド可能 API の CEEDOD (操作記述子の検索) を呼び出さなければなりません。この API には CALLB 命令に定義しなければならないある種のパラメーターが必要です。しかし、

これだけで必要な情報、すなわち長さを得ることができます。操作記述子の詳細については、 151 ページの『操作記述子の使用』を参照してください。

```
 *====================================================================*
 * CvtToHex - 入力ストリングを 16 進出力ストリングに変換
 *====================================================================*
H COPYRIGHT('(C) Copyright MyCompany 1995')
D/COPY RPGGUIDE/QRPGLE,CVTHEXPR
 *--------------------------------------------------------------------*
 * メイン入力パラメーター
 * 1. 入力:  ストリング                文字(n)
 * 2. 出力:  16 進ストリング           文字(2 * n)
 *--------------------------------------------------------------------*
D CvtToHex        PI                    OPDESC
D   InString                 16383      CONST OPTIONS(*VARSIZE)
D   HexString                32766      OPTIONS(*VARSIZE)

 *--------------------------------------------------------------------*
 * CEEDOD (操作記述子用の検索) 用のプロトタイプ
 *--------------------------------------------------------------------*
D CEEDOD          PR
D ParmNum                       10I 0 CONST
D                               10I 0
D                               10I 0
D                               10I 0
D                               10I 0
D                               10I 0
D                               12A   OPTIONS(*OMIT)

 * CEEDOD に渡されるパラメーター
D DescType        S              10I 0
D DataType        S              10I 0
D DescInfo1       S              10I 0
D DescInfo2       S              10I 0
D InLen           S              10I 0
D HexLen          S              10I 0

 *--------------------------------------------------------------------*
 * プログラムで使用されるその他のフィールド                           *
 *--------------------------------------------------------------------*
D HexDigits       C                     CONST('0123456789ABCDEF')
D IntDs           DS
D   IntNum                       5I 0 INZ(0)
D   IntChar                      1    OVERLAY(IntNum:2)
D HexDs           DS
D   HexC1                        1
D   HexC2                        1
D InChar          S              1
D Pos             S              5P 0
D HexPos          S              5P 0
```

図 42. サービス・プログラム CvtToHex のソース (1/2)

```
 *----------------------------------------------------------------*
 * 渡されたパラメーターの長さを判別するために、操作記述子を       *
 * 使用します。                                                   *
 *----------------------------------------------------------------*
C                   CALLP     CEEDOD(1         : DescType : DataType :
C                                    DescInfo1 : DescInfo2: Inlen    :
C                                    *OMIT)
C                   CALLP     CEEDOD(2         : DescType : DataType :
C                                    DescInfo1 : DescInfo2: HexLen   :
C                                    *OMIT)
 *----------------------------------------------------------------*
 * 取り扱う長さの判別 (入力の長さの最小、および                   *
 * 16 進の長さの半分)                                             *
 *----------------------------------------------------------------*
C                   IF        InLen > HexLen / 2
C                   EVAL      InLen = HexLen / 2
C                   ENDIF

 *----------------------------------------------------------------*
 * 入力ストリングの文字ごとに、2 バイトの 16 進表記に変換         *
 * (例えば、'5' --> 'F5')                                         *
 *----------------------------------------------------------------*
C                   EVAL      HexPos = 1
C                   DO        InLen         Pos
C                   EVAL      InChar = %SUBST(InString : Pos :1)
C                   EXSR      GetHex
C                   EVAL      %SUBST(HexString : HexPos : 2) = HexDs
C                   EVAL      HexPos = HexPos + 2
C                   ENDDO

 *----------------------------------------------------------------*
 * 実行後; 呼び出し元へ戻ります。                                 *
 *----------------------------------------------------------------*
C                   RETURN

 *================================================================*
 * GetHex - 'InChar' を 'HexDs' へ変換するサブルーチン            *
 *                                                                *
 * 2 つの 16 進の桁を区切るために、16 による除算を使用します。    *
 * その商を最初の桁に、剰余を 2 番目の桁とします。                *
 *================================================================*
C     GetHex        BEGSR
C                   EVAL      IntChar = InChar
C     IntNum        DIV       16            X1                5 0
C                   MVR                     X2                5 0
 *----------------------------------------------------------------*
 * 16 進文字のリスト '012...CDEF' のサブストリングを得るために、  *
 * 16 進の桁 (プラス 1) を使用します。                            *
 *----------------------------------------------------------------*
C                   EVAL      HexC1 = %SUBST(HexDigits:X1+1:1)
C                   EVAL      HexC2 = %SUBST(HexDigits:X2+1:1)
C                   ENDSR
```

図 42. サービス・プログラム CvtToHex のソース (2/2)

```
     *===================================================================*
     * CvtToHex - 入力ストリングを 16 進出力ストリングに変換
     *
     * パラメーター
     * 1. 入力:  ストリング                  文字(n)
     * 2. 出力:  16 進ストリング             文字(2 * n)
     *===================================================================*
    D CvtToHex        PR                    OPDESC
    D   InString                 16383      CONST OPTIONS(*VARSIZE)
    D   HexString                32766      OPTIONS(*VARSIZE)
```

図 43. CvtToHex のプロトタイプでの /COPY メンバーのソース

このサービス・プログラムを設計する時には、後日プログラムをより容易に更新できるようインターフェースの判別にバインダー言語を利用するよう決定されました。 図 44 は、サービス・プログラム CVTTOHEX のエクスポートを定義するのに必要なバインダー言語を示します。このソースは CRTSRVPGM コマンドの EXPORT、SRCFILE、および SRCMBR パラメーターで使われます。

```
STRPGMEXP SIGNATURE('CVTHEX')
   EXPORT SYMBOL('CVTTOHEX')
ENDPGMEXP
```

図 44. CvtToHex のためのバインダー言語のソース

STRPGMEXP のパラメーター SIGNATURE は、サービス・プログラムが提供するインターフェースを指定します。この場合には、バインダー言語で識別されるエクスポートがインターフェースです。CVTTOHEX へバインドされるプログラムがこのインターフェース識別値を使用します。

バインダー言語 EXPORT ステートメントがサービス・プログラムのエクスポートを指定します。呼び出し元に対して使用したいエクスポートをもつ各プロシージャーに 1 つ必要です。この場合、サービス・プログラムには 1 つのプロシージャーを含む 1 つのモジュールが入っています。そのため、EXPORT ステートメントが 1 つだけ必要になります。

バインダー言語およびインターフェース識別値について詳しくは、「*ILE 概念*」を参照してください。

サービス・プログラムの作成

サービス・プログラム CVTTOHEX を作成するには、次のステップに従ってください。

1. 次のとおり入力して、 105 ページの図 42 のソースからモジュール CVTTOHEX を作成する。

   ```
   CRTRPGMOD MODULE(MYLIB/CVTTOHEX) SRCFILE(MYLIB/QRPGLESRC)
   ```

2. モジュール CVTTOHEX および図 44 に示してあるバインダー言語を使用して、サービス・プログラムを作成する。

   ```
   CRTSRVPGM SRVPGM(MYLIB/CVTTOHEX)  MODULE(*SRVPGM)
             EXPORT(*SRCFILE)  SRCFILE(MYLIB/QSRVSRC)
             SRCMBR(*SRVPGM)
   ```

上記のコマンドの最後の 3 つのパラメーターが、サービス・プログラムで使用可能なエクスポートを指定します。この場合には、これはライブラリー MYLIB のファイル QSRVSRC のメンバー CVTTOHEX で見付かったソースが基礎になっています。

サービス・プログラムの作成に必要なすべてのモジュールが MODULE パラメーターで指定されているので、ここではバインディング・ディレクトリーは不要であることに注意してください。

サービス・プログラム CVTTOHEX はライブラリー MYLIB に作成されます。これはステートメント・ビューを使用してデバッグされます。このことは CRTRPGMOD コマンドのデフォルトの値 DBGVIEW によって決まります。バインダー・リストは作成されません。

プログラムへのバインド

この例を完成させるために、サービス・プログラムへバインドされるプログラム CVTHEXPGM から構成される「アプリケーション」を作成します。これは CVTTOHEX に 7 桁のストリングを 2 回渡しますが、 1 回目には 16 進数ストリングの値が 10 (すなわち 5 桁を変換) で、 2 回目には値が 14、すなわち実際の長さで渡されます。

サービス・プログラム CVTTOHEX の使用を説明するためにプログラム CVTHEXPGM が使われていることに注意してください。実際のアプリケーションでは、CVTTOHEX の呼び出し元は CVTTOHEX のテスト以外に別の主要な目的をもっています。さらに、サービス・プログラムは通常他の多くのプログラムで使用されるか、あるいはいくつかのプログラムで頻繁に使用されます。そうでなくては、初期の呼び出しのオーバーヘッドから考えてこれをサービス・プログラムにしたことが正当化されません。

アプリケーション・プログラムを作成するには、次のステップに従ってください。

1. 次のとおり入力して、 109 ページの図 45 のソースからモジュールを作る。

   ```
   CRTRPGMOD MODULE(MYLIB/CVTHEXPGM) SRCFILE(MYLIB/QRPGLESRC)
   ```

2. 次のとおり入力してプログラムを作成する。

   ```
   CRTPGM PGM(MYLIB/CVTHEXPGM)
              BNDSRVPGM(MYLIB/CVTTOHEX)
              DETAIL(*BASIC)
   ```

   CVTHEXPGM が作成される際、サービス・プログラムとの対話に使用するインターフェースに関する情報が含められます。これは CVTTOHEX 用のバインダー言語に反映されるものと同じです。

3. 次のとおり入力して、プログラムを呼び出す。

   ```
   CALL CVTHEXPGM
   ```

   CVTHEXPGM を実行可能にする処理の段階で、システムは以下のことを検査します。

   - サービス・プログラム CVTTOHEX はライブラリー MYLIB 内にある。
   - 作成時に CVTHEXPGM で使用された共通インターフェースが、実行時にも有効である。

上のいずれかが真でなければ、エラー・メッセージが出されます。

CVTHEXPGM の出力を下に示します。(入力ストリングは 'ABC123*' です。)

```
Result14++++++
Result10++
C1C2C3F1F2         10 character output
C1C2C3F1F2F35C     14 character output
```

```
     *-------------------------------------------------------------*
     * サービス・プログラム CVTTOHEX をテストするプログラム       *
     *                                                             *
     * 1. 7 文字の入力ストリングを使用                             *
     * 2. 10 文字の 16 進ストリングに変換 (結果が入力ストリング全体に *
     *    とっては小さ過ぎるため、                                 *
     *    入力文字の先頭 5 文字だけが使用されることになります)     *
     * 3. 14 文字の 16 進ストリングに変換 (結果に十分な長さがある   *
     *    ため、入力文字 7 文字すべてが使用されます)               *
     *-------------------------------------------------------------*
    FQSYSPRT   O    F   80        PRINTER
     * CvtToHex のプロトタイプ
    D/COPY RPGGUIDE/QRPGLE,CVTHEXPR
    D ResultDS        DS
    D   Result14              1     14
    D   Result10              1     10
    D InString        S              7
    D Comment         S             25
    C                   EVAL      InString = 'ABC123*'

     *-------------------------------------------------------------*
     * プロトタイプ呼び出しを使用して、文字ストリングと 10 文字の  *
     * 結果フィールドを渡します。呼び出されたプロシージャー        *
     * CvtToHex の必要に応じて、操作記述子が渡されます。           *
     *-------------------------------------------------------------*
    C                   EVAL      Comment = '10 character output'
    C                   CLEAR                   ResultDS
    C                   CALLP     CvtToHex(Instring : Result10)
    C                   EXCEPT

     *-------------------------------------------------------------*
     * CALLB(D) を使用して、文字ストリングと 14 文字の結果の       *
     * フィールドを渡します。命令拡張 (D) は、渡されたパラメーター *
     * 用の操作記述子を作成します。CALLB は、上記の CALLP との比較の *
     * ためにここで使用されます。                                  *
     *-------------------------------------------------------------*
    C                   EVAL      Comment = '14 character output'
    C                   CLEAR                   ResultDS
    C                   CALLB(D)  'CVTTOHEX'
    C                   PARM                    InString
    C                   PARM                    Result14
    C                   EXCEPT
    C                   EVAL      *INLR = *ON

    OQSYSPRT   H    1P
    O                                       'Result14++++++'
    OQSYSPRT   H    1P
    O                                       'Result10++'
    OQSYSPRT   E
    O                     ResultDS
    O                     Comment           +5
```

図 45. テスト・プログラム CVTHEXPGM のソース

サービス・プログラムの更新

バインダー言語の利用で、サービス・プログラムを更新することができ、しかもプログラム CVTHEXPGM はコンパイルし直さなくても済みます。例えば、新しいプロシージャーが既存のモジュールに入れるか、あるいは新しいモジュールに入れるかによって、CVTTOHEX に新しいプロシージャーを追加する 2 つの方法があります。

新しいプロシージャーを既存のモジュールに加えるには、 以下のことを実行します。

1. 新しいプロシージャーを既存のモジュールに追加する。
2. 変更モジュールをコンパイルし直す。
3. 新しいプロシージャーと関連したインターフェースを処理するためにバインダー言語のソースを変更する。これには、既存のステートメントに続く 新しいエクスポート・ステートメントの追加が関係しています。
4. CRTSRVPGM を使用してサービス・プログラムを再作成する。

新しいモジュールを使って新しいプロシージャーを加えるには、次のようにします。

1. 新しいプロシージャーのモジュール・オブジェクトを作成する。
2. 上で述べたように、新しいプロシージャーと関連したインターフェースを処理するためにバインダー言語のソースを変更する。
3. サービス・プログラム CVTTOHEX を作成し直して、そのサービス・プログラムに新しいモジュールをバインドする。

どちらの方法でも、新しいプログラムは新しい機能をアクセスできます。旧エクスポートが同じ順序なので、既存のプログラムはまだこれを使用することができます。既存のプログラムの更新も必要となるまで、これらをコンパイルし直す必要はありません。

サービス・プログラムの更新について詳しくは、「*ILE 概念*」を参照してください。

サンプル・バインダー・リスト

111 ページの図 46 は CVTHEXPGM のバインダー・リストの例です。このリストは基本リストの例です。バインダー・リストについて詳しくは、96 ページの『バインダー・リストの使用』および「*ILE 概念*」を参照してください。

```
                                                  プログラムの作成                                                              ページ  1
#  5722SS1 V5R2M0  020719                                                                 MYLIB/CVTHEXPGM    ISERIES1  02/08/15  23:24:00
    プログラム . . . . . . . . . . . . . . . . . . . . . . :   CVTHEXPGM
      ライブラリー  . . . . . . . . . . . . . . . . . . .:     MYLIB
    プログラム入力プロシージャー・モジュール . . . . . . :   *FIRST
      ライブラリー  . . . . . . . . . . . . . . . . . . :
    活動化グループ . . . . . . . . . . . . . . . . . . . :   *NEW
    作成オプション . . . . . . . . . . . . . . . . . . . :   *GEN         *NODUPPROC  *NODUPVAR   *WARN       *RSLVREF
    明細のリスト . . . . . . . . . . . . . . . . . . . . :   *BASIC
    更新可能   . . . . . . . . . . . . . . . . . . . . . :   *YES
    ユーザー・プロファイル . . . . . . . . . . . . . . . :   *USER
    既存のプログラムの置き換え . . . . . . . . . . . . . :   *YES
    権限       . . . . . . . . . . . . . . . . . . . . . :   *LIBCRTAUT
    ターゲット・リリース . . . . . . . . . . . . . . . . :   *CURRENT
    再初期設定可能. . . .  . . . . . . . . . . . . . . . :   *NO
    テキスト . . . . . . . . . . . . . . . . . . . . . . :   *ENTMODTXT
    モジュール      ライブラリー        モジュール      ライブラリー      モジュール      ライブラリー      モジュール      ライブラリー
    CVTHEXPGM   MYLIB
    サービス・                          サービス・                        サービス・                        サービス・
    プログラム      ライブラリー        プログラム      ライブラリー      プログラム      ライブラリー      プログラム      ライブラリー
    CVTTOHEX    MYLIB
    バインド・                          バインド・                         バインド・                       バインド・
    ディレクトリー  ライブラリー        ディレクトリー   ライブラリー      ディレクトリー  ライブラリー     ディレクトリー  ライブラリー
    *NONE
                                                  プログラムの作成                                                             ページ  2
#  5722SS1 V5R2M0  020719                                                                 MYLIB/CVTHEXPGM    ISERIES1  02/08/15   23:24:00
                                                            簡略要約表
    プログラム入力プロシージャー . . . . . . . . . . . :   1
      記号    タイプ      ライブラリー    オブジェクト      識別コード
                *MODULE   MYLIB       CVTHEXPGM   _QRNP_PEP_CVTHEXPGM
    複数強定義. . . . . . . . . . . . . . . . . . . :   0
    未解決の参照 . . . . . . . . . . . . . . . . . . :   0
                                  * * * * *   簡 略 要 約 表 の 終 わ り   * * * * *
                                                  プログラムの作成                                                              ページ  3
#  5722SS1 V5R2M0  020719                                                                 MYLIB/CVTHEXPGM    ISERIES1  02/08/15  23:24:00
                                                             バインド統計
    記号収集 CPU 時間 . . . . . . . . . . . . . . . . . . . . . . . . :               .016
    記号分析解決 CPU 時間 . . . . . . . . . . . . . . . . . . . . . . :               .004
    バインディング・ディレクトリー分析解決 CPU 時間 . . . . . . . . . :               .175
    BIND プログラム言語コンパイル CPU 時間  . . . . . . . . . . . . . :               .000
    リスト作成 CPU 時間   . . . . . . . . . . . . . . . . . . . . . . :               .068
    プログラム / サービス・プログラム作成 CPU 時間  . . . . . . . . . :               .234
    合計 CPU 時間 . . . . . . . . . . . . . . . . . . . . . . . . . . :               .995
    合計経過時間  . . . . . . . . . . . . . . . . . . . . . . . . . . :              3.531
                                   * * * * *   バ イ ン ド 統 計 の 終 わ り   * * * * *
   *CPC5D07 - プログラム CVTHEXPGM がライブラリー MYLIB に作成された。
                              * * * * *   プ ロ グ ラ ム 作 成 リ ス ト の 終 わ り  * * * * *
```

図 46. CVTHEXPGM の基本バインダー・リスト

第 9 章 プログラムの実行

この章では、以下のことを行う方法について説明します。

- CL CALL コマンドを使用してプログラムを実行し、パラメーターを渡す
- メニュー方式アプリケーションからプログラムを実行する
- ユーザー作成コマンドを使用してプログラムを実行する
- 活動化グループを管理する
- 実行時記憶域を管理する

さらに、次を使用してプログラムを実行することができます。

- プログラマー・メニュー。「*CL プログラミング*, SD88-5038-05」には、このメニューの情報があります。
- PDM 開始 (STRPDM) コマンド。「*AS/400 プログラム開発管理機能 (PDM)*」には、このコマンドの情報があります。
- QCMDEXC プログラム。「*CL プログラミング*」には、このプログラムの情報があります。
- 高水準言語。139 ページの『第 10 章 プログラムおよびプロシージャーの呼び出し』には、他の HLL からのプログラムの実行やサービス・プログラムおよびプロシージャーの呼び出しについての情報があります。

注: WebSphere Development Studio Client for iSeries を使用します。これは推奨される方法であり、プログラム実行に関する説明は、製品のオンライン・ヘルプに記載されています。

CL CALL コマンドを使用したプログラムの実行

CL CALL コマンドを使用して、プログラム (タイプ *PGM) を実行することができます。コマンドは、対話式に使用したり、バッチ・ジョブの一部として使用したり、あるいは CL プログラムにこれを入れることができます。プロンプトが必要な場合には、CALL を入力し F4 (プロンプト) キーを押してください。ヘルプが必要な場合には、CALL を入力し F1 (ヘルプ) を押してください。

例えば、コマンド入力行からプログラム EMPRPT を呼び出すためには、次を入力してください。

```
CALL EMPRPT
```

指定したプログラム・オブジェクトは、ライブラリー中に存在していなければならず、またこのライブラリーはライブラリー・リスト *LIBL に入っていなければなりません。また、次のように CL CALL コマンドでライブラリーを明示的に指定することができます。

```
CALL MYLIB/EMPRPT
```

CL CALL コマンドの使用についての詳細は、Web サイト http://www.ibm.com/eserver/iseries/infocenter で **iSeries 400 Information Center** の中の『プログラミング』カテゴリーの『*CL および API*』の節を参照してください。

ユーザーが自分のプログラムを呼び出すと、OS/400 システムはそのプログラムにある命令を実行します。

CL CALL コマンドを使用したパラメーターの受け渡し

ILE プログラムを実行する時にそのプログラムにパラメーターを渡すには、CL CALL コマンドの PARM オプションを使用します。

```
CALL PGM(program-name)
     PARM(parameter-1 parameter-2 ... parameter-n)
```

パラメーターは、次のようにキーワードを指定せずに入力することもできます。

```
CALL library/program-name (parameter-1 parameter-2 ... parameter-n)
```

各パラメーター値は、CL プログラム変数として、あるいは次の 1 つとして指定することができます。

- 文字ストリング定数
- 数値定数
- 論理定数

ILE RPG プロシージャーがプログラム入力プロシージャーであるようなプログラムに対してパラメーターを渡す場合には、そのプログラムには *ENTRY PLIST がただ 1 つだけ指定されていなければなりません。(PARM ステートメント中で) 後に続くパラメーターは、CALL コマンドを介して渡されるものに 1 対 1 の関係で対応していなければなりません。

パラメーターが処理される方法についての完全な説明は、「*CL プログラミング*」の『プログラム間のパラメーターの受け渡し』のセクションの CALL コマンドを参照してください。

例えば、最初の開始時にプログラム EMPRPT2 には、正しいパスワードが渡されなければなりません。そうでない場合には、そのプログラムは実行しません。 115 ページの図 47 はそのソースを示したものです。

1. プログラムを作成するためには、次を入力してください。

   ```
   CRTBNDRPG PGM(MYLIB/EMPRPT2)
   ```

2. プログラムを実行するためには、次を入力してください。

   ```
   CALL MYLIB/EMPRPT2 (HELLO)
   ```

   CALL コマンドが出されると、そのコマンドによって渡されたパラメーターの内容が保管され、プログラム・パラメーター PSWORD がその位置を示します。次にプログラムは PSWORD の内容がプログラムに保管された値 ('HELLO') と一致するかどうかを調べます。この場合には、2 つの値が同じなので、プログラムは実行を続行します。

```
     *=============================================================*
     * プログラム名:   EMPRPT2                                     *
     * 関連ファイル:   EMPMST   (物理ファイル)                    *
     *                 PRINT    (PRINTER ファイル)                 *
     * 説明:           このプログラムは、入力されたパスワードが    *
     *                 正しい場合に、ファイル EMPMST に保管されている*
     *                 社員情報を印刷します。コマンド行に          *
     *                 "CALL ライブラリー名/EMPRPT2 (パスワード)" を *
     *                 入力してプログラムを実行します。            *
     *                 ここで、このプログラムのパスワードは        *
     *                 'HELLO' です。                              *
     *=============================================================*
    FPRINT     O    F   80        PRINTER
    FEMPMST    IP   E           K DISK
    IEMPREC         01
```

図 47. 実行時にパラメーターを要求する ILE RPG プログラム (1/2)

```
     *-------------------------------------------------------------*
     * 入力パラメーター・リストはこのプログラムの中で指定されます。  *
     * PSWORD というパラメーターが 1 つあり、これは                *
     * 長さ 5 桁の文字フィールドです。                             *
     *-------------------------------------------------------------*
    C     *ENTRY        PLIST
    C                   PARM                    PSWORD          5
     *-------------------------------------------------------------*
     * このプログラムのパスワードは 'HELLO' です。フィールド PSWORD  *
     * に 'HELLO' が入っているかチェックします。                   *
     * 入っていない場合には、最終レコード標識 (LR) と *IN99 がオンに *
     * 設定されます。*IN99 はメッセージの印刷を制御します。        *
     *-------------------------------------------------------------*
    C     PSWORD        IFNE      'HELLO'
    C                   SETON                                        LR99
    C                   ENDIF
    OPRINT     H    1P                   2  6
    O                                         50 'EMPLOYEE INFORMATION'
    O          H    1P
    O                                         12 'NAME'
    O                                         34 'SERIAL #'
    O                                         45 'DEPT'
    O                                         56 'TYPE'
    O          D    01N99
    O                         ENAME           20
    O                         ENUM            32
    O                         EDEPT           45
    O                         ETYPE           55
    O          D    99
    O                                         16 '***'
    O                                         40 'Invalid Password Entered'
    O                                         43 '***'
```

図 47. 実行時にパラメーターを要求する ILE RPG プログラム (2/2)

116 ページの図 48 は、EMPRPT2 ソースによって参照される DDS を示したものです。

```
A*****************************************************************
A* 説明:  これは物理ファイル EMPMST の DDS です。              *
A*        これには 1 つのレコード様式 EMPREC が入っています。    *
A*        このファイルには、会社の各社員ごとに 1 レコードが      *
A*        入っています。                                       *
A*****************************************************************
A*
A          R EMPREC
A            ENUM           5  0      TEXT('EMPLOYEE NUMBER')
A            ENAME         20         TEXT('EMPLOYEE NAME')
A            ETYPE          1         TEXT('EMPLOYEE TYPE')
A            EDEPT          3  0      TEXT('EMPLOYEE DEPARTMENT')
A            ENHRS          3  1      TEXT('EMPLOYEE NORMAL WEEK HOURS')
A          K ENUM
```

図 48. EMPRPT2 の DDS

メニュー方式アプリケーションからのプログラムの実行

ILE プログラムを実行するもう 1 つの方法はメニュー方式アプリケーションからです。ワークステーション・ユーザーはメニューからオプションを選択し、それが特定のプログラムを呼び出します。図 49 は、アプリケーション・メニューの例です。

```
                                                    PAYROLL DEPARTMENT MENU
次の中から 1 つを選んでください:
     1.  Inquire into employee master
     2.  Change employee master
     3.  Add new employee









選択項目またはコマンド
===> __________________________________________________________________
____________________________________________________________________________
F3=終了   F4=プロンプト   F9=コマンドの複写   F12=取消し
F13=情報援助    F16=AS/400 メイン・メニュー
```

図 49. アプリケーション・メニューの例

図 49 に示す例は各オプションが別の ILE プログラムを呼び出す、メニュー・プログラムによって表示されます。STRSDA を使用し、オプション 2 を選択して、メニューを作成することができます (「メニューの設計」)。

117 ページの図 50 は、上の PAYROLL DEPARTMENT MENU の表示装置ファイルの DDS を示します。ソース・メンバーは PAYROL といい、そのソース・タイプは MNUDDS です。ファイルは SDA を使用して作成されています。

```
     A* 自由形式メニュー: PAYROL
    A*
    A                                     DSPSIZ(24 80 *DS3                -
    A                                           27 132 *DS4)
    A                                     CHGINPDFT
    A                                     INDARA
    A                                     PRINT(*LIBL/QSYSPRT)
    A          R PAYROL
    A                                     DSPMOD(*DS3)
    A                                     LOCK
    A                                     SLNO(01)
    A                                     CLRL(*ALL)
    A                                     ALWROL
    A                                     CF03
    A                                     HELP
    A                                     HOME
    A                                     HLPRTN
    A                                   1 34'PAYROLL DEPARTMENT MENU'
    A                                     DSPATR(HI)
    A                                   3  2'Select one of the following:'
    A                                     COLOR(BLU)
    A                                   5  7'1.'
    A                                   6  7'2.'
    A                                   7  7'3.'
    A* CMDPROMPT  この DDS 仕様書を削除しないでください。
    A                                 019  2'Selection or command             -
    A                                          '
    A                                   5 11'Inquire'
    A                                   5 19'into'
    A                                   5 24'employee'
    A                                   5 33'master'
    A                                   6 11'Change'
    A                                   6 18'employee'
    A                                   6 27'master'
    A                                   7 11'Add'
    A                                   7 15'new'
    A                                   7 19'employee'
```

図 50. アプリケーション・メニューのデータ記述仕様書

図 51 は、 116 ページの図 49 で示したアプリケーション・メニューのソースです。ソース・メンバーは PAYROLQQ といい、そのソース・タイプは MNUCMD です。これもまた SDA を使用して作成されています。

```
PAYROLQQ,1
0001 call RPGINQ
0002 call RPGCHG
0003 call RPGADD
```

図 51. メニュー・プログラムのソース

メニューは、次を入力して実行します。

```
GO library name/PAYROL
```

ユーザーがアプリケーション・メニューから 1、2、または 3 を入力すると、図 51 のソースはそれぞれプログラム RPGINQ、RPGCHG、または RPGADD を呼び出します。

ユーザー作成コマンドを使用したプログラムの実行

コマンドを作成し、コマンド定義を使用してプログラムを実行することができます。**コマンド定義**とは、コマンドの定義 (コマンド名、パラメーターの説明、および妥当性検査情報を含む) が入っていて、そのコマンドによって要求される機能を実行するプログラムを識別するオブジェクト (タイプ *CMD) です。

例えば、プログラム PAYROLL を呼び出すコマンド PAY を作成することができますが、ここで PAYROLL は実行したい RPG プログラムの名前です。コマンドを対話式に、またはバッチ・ジョブで入力することができます。コマンド定義の使い方について詳しくは、「*CL プログラミング*」を参照してください。

実行時照会メッセージに対する応答

ILE RPG プロシージャーでプログラムを実行した場合には、実行時照会メッセージが生成されることがあります。これはメイン・プロシージャー内で機能チェックのためにデフォルト・エラー処理プログラムが呼び出された場合に起こります。 276 ページの『メイン・プロシージャー内の例外処理プログラム』を参照してください。プログラムを続いて実行したい場合には、照会メッセージに応答が必要です。

注: サブプロシージャーの機能チェックのデフォルト・エラー処理ではサブプロシージャーが取り消され、サブプロシージャーの呼び出し元に対して例外がパーコレートされるため、照会メッセージがサブプロシージャーに対して発行されることはありません。『サブプロシージャー内の例外処理プログラム』を参照してください。

サブプロシージャーの呼び出し元が RPG プロシージャーの場合には、実際の例外に関連する状況コードには依存せずに、呼び出しは状況 00202 で失敗します。呼び出しの失敗により RPG メイン・プロシージャーがデフォルト・ハンドラーを呼び出した場合には、照会メッセージ RNQ0202 が発行されます。

メッセージへ自動応答するために、システム応答リストに照会メッセージを追加することができます。これらのメッセージに対する応答は個別に、あるいは全般的に指定することができます。照会メッセージに対するこれらの応答の方法は、バッチ・プログラムに特に適しています。そうでない場合には、応答を出すオペレーターが必要になります。

システム応答リストに、次の ILE RPG 照会メッセージを追加することができます。

表 24. ILE RPG 照会メッセージ

RNQ0100	RNQ0231	RNQ0421	RNQ1023	RNQ1235
RNQ0101	RNQ0232	RNQ0425	RNQ1024	RNQ1241
RNQ0102	RNQ0299	RNQ0426	RNQ1031	RNQ1251
RNQ0103	RNQ0301	RNQ0431	RNQ1041	RNQ1255
RNQ0104	RNQ0302	RNQ0432	RNQ1042	RNQ1261
RNQ0112	RNQ0303	RNQ0450	RNQ1051	RNQ1271
RNQ0113	RNQ0304	RNQ0501	RNQ1071	RNQ1281
RNQ0114	RNQ0305	RNQ0502	RNQ1201	RNQ1282
RNQ0115	RNQ0306	RNQ0802	RNQ1211	RNQ1284
RNQ0120	RNQ0333	RNQ0803	RNQ1215	RNQ1285
RNQ0121	RNQ0401	RNQ0804	RNQ1216	RNQ1286
RNQ0122	RNQ0402	RNQ0805	RNQ1217	RNQ1287
RNQ0123	RNQ0411	RNQ0907	RNQ1218	RNQ1299
RNQ0202	RNQ0412	RNQ1011	RNQ1221	RNQ1331
RNQ0211	RNQ0413	RNQ1021	RNQ1222	RNQ9998
RNQ0221	RNQ0414	RNQ1022	RNQ1231	RNQ9999
RNQ0222	RNQ0415			

注: ILE RPG 照会メッセージは RNQ というメッセージ ID 接頭部を持っています。

応答リスト項目追加コマンドを使用してシステム応答リストに照会メッセージを追加するためには、次を入力してください。

```
ADDRPYLE sequence-no message-id
```

ここで、*sequence-no* は 1 〜 9999 の番号で、これはリストの中のどこで項目が加えられているかを示し、*message-id* は、追加したいメッセージ番号です。追加したい各メッセージにこのコマンドを繰り返してください。

ジョブ変更 (CHGJOB) コマンド (または他の CL ジョブ・コマンド) を使用して、ジョブで照会メッセージの応答リストを使用することを指示します。これを行うためには、QUERY メッセージ応答 (INQMSGRPY) 属性に *SYSRPYL を指定する必要があります。

QUERY メッセージ応答 (INQMSGRPY) 属性が INQMSGRPY(*SYSRPYL) として指定されているジョブによって照会メッセージが送られた場合にのみ応答リストが使用されます。INQMSGRPY パラメーターは次の CL コマンドにあります。

- ジョブ変更 (CHGJOB)
- ジョブ記述変更 (CHGJOBD)
- ジョブ記述作成 (CRTJOBD)
- ジョブ投入 (SBMJOB)

またシステム応答リスト項目処理 (WRKRPYLE) コマンドを使用して、システム応答リストの項目を変更または除去することができます。ADDRPYLE コマンドおよび WRKRPYLE コマンドについての詳細は、Web サイト http://www.ibm.com/eserver/iseries/infocenter で **iSeries 400 Information Center** の中の『プログラミング』カテゴリーの『*CL および API*』の節を参照してください。

ILE プログラムの終了

ILE プログラムが正常に終了した時は、システムは制御を呼び出し元に戻します。呼び出し元はワークステーション・ユーザーでも、(メニュー処理プログラムのような) 別のプログラムでもかまいません。

ILE プログラムが異常終了し、そのプログラムが呼び出し元と異なる活動化グループで実行中であったときは、次のようなエスケープ・メッセージ CEE9901

```
Error message-id caused program to end.
```

が出され、制御は呼び出し元に戻されます。

CL プログラムは、メッセージ監視 (MONMSG) コマンドを使用してこの例外をモニターすることができます。他の ILE 言語における例外もモニターすることができます。

ILE プログラムがその呼び出し元と同じ活動化グループで実行中に異常終了した場合には、出されるメッセージはプログラムが終了した理由によって異なります。機能チェックで終了した場合には、CPF9999 が出されます。RPG プロシージャーによって例外が出された場合には、RNX のメッセージ接頭部をもちます。

例外メッセージの詳細については、 274 ページの『例外処理の概要』を参照してください。

活動化グループの管理

活動化グループとはジョブのサブ構造であり、1 つ以上の ILE または OPM プログラムを実行するために割り当てられるシステム資源 (例えば記憶域、コミットメント定義、オープン・ファイルなど) から構成されます。活動化グループによって同じジョブ内で実行中の ILE プログラムは、互いに干渉されることなく (例えば、コミットメント制御や一時変更に影響されることなく) 独立して実行できるようになります。基本的な考え方は、1 つの活動化グループ内で活動状態になるすべてのプログラムが 1 つの連携アプリケーションとして開発されるということです。

プログラムの作成時に、ILE プログラムが実行される活動化グループを指定します。プログラム・オブジェクトの作成時に、ACTGRP パラメーターで指定された値によって活動化グループが決められます (OPM プログラムは常にデフォルトの値の活動化グループで実行します。これらの活動化グループの仕様書を変更することはできません)。いったん ILE プログラム (オブジェクト・タイプ *PGM) が活動状態になると、活動化グループが削除されるまで活動状態のままです。

この節の残りの部分では活動化グループの指定方法と削除方法について説明します。活動化グループについて詳しくは、「*ILE 概念*」を参照してください。

活動化グループの指定

ILE プログラムが稼働する活動化グループは、ユーザー・プログラムの作成時 (CRTPGM または CRTBNDRPG を使用) またはサービス・プログラムの作成時 (CRTSRVPGM を使用) に、ACTGRP パラメーターに値を指定することによって制御します。

注: CRTBNDRPG コマンドを使用する場合には、DFTACTGRP の値が *NO の場合にのみ ACTGRP に値を指定することができます。

次の値の中から 1 つを選択することができます。

- 名前付き活動化グループ

  名前付き活動化グループによって、ILE プログラムおよびサービス・プログラムの集まりを、1 つのアプリケーションとして管理することができます。活動化グループは、作成時に活動化グループ名を指定した最初のプログラムが呼び出された時に作成されます。それからは同じ活動化グループ名を指定するすべてのプログラムおよびサービス・プログラムによって使用されます。

  名前付き活動化グループは、CL コマンド RCLACTGRP を使用して削除された時に終了します。このコマンドは活動化グループがもう使用されない場合にのみ使用することができます。これが終了した時には、名前付き活動化グループのプログラムおよびサービス・プログラムと関連したすべての 資源がシステムに戻されます。

  名前付き活動化グループ QILE が、CRTBNDRPG コマンドの ACTGRP パラメーターのデフォルト値です。しかし、活動化グループはアプリケーションに対応するように意図されているので、このパラメーターには異なる値を指定されるようお奨めします。例えば、アプリケーション名の後に活動化グループの名前を指定することができます。

- *NEW

  *NEW を指定すると、プログラムが呼び出されるたびに新しい活動化グループが作成されます。システムが活動化グループの名前を作成します。この名前はジョブ内で固有のものです。

  *NEW で作成された活動化グループは常に、それと関連したプログラムの終了時に終了します。このため、プログラムを活動状態にしておくために LR OFF でプログラムから戻るつもりの場合には、ACTGRP パラメーターに *NEW を指定すべきではありません。

  注: この値はサービス・プログラムの場合正しくありません。サービス・プログラムは、名前付き活動化グループまたは呼び出し元の活動化グループでのみ実行することができます。

  *NEW は、CRTPGM コマンドの ACTGRP パラメーターのデフォルト値です。

  ACTGRP(*NEW) で ILE RPG プログラムを作成する場合には、以前の呼び出しから戻らずに必要な回数だけプログラムを呼び出すことができます。呼び出しごとに、プログラムが新しくコピーされます。それぞれの新しいコピーは独自のデータをもち、ファイルをオープンしたりします。しかし、「それ自身」への呼び出しを終了するための何らかの方法がなければなりません。そうでない場合には、単に新しい活動化グループを作成し続けるだけで、プログラムは決して戻らないことになります。

- *CALLER

  プログラムまたはサービス・プログラムは、呼び出し側プログラムの活動化グループ内に活動化されます。ACTGRP(*CALLER) で作成された ILE プログラムが OPM プログラムによって呼び出された場合には、プログラムは OPM デフォルトの活動化グループ (*DFTACTGRP) 内で活動化されます。

OPM デフォルト活動化グループでの実行

OS/400 ジョブの開始時に、システムは OPM プログラムによって使用される活動化グループを作成します。この活動化グループを表すのに使用される記号は、*DFTACTGRP です。OPM デフォルトの活動化グループを削除することはできません。これはジョブの終了時にシステムにより削除されます。

OPM プログラムは自動的に OPM デフォルトの活動化グループで実行されます。次のいずれかが起こると、ILE プログラムも OPM デフォルトの活動化グループで実行されます。

- プログラムが CRTBNDRPG コマンドの DFTACTGRP(*YES) で作成された。
- プログラムがプログラム作成時に ACTGRP(*CALLER) で作成され、プログラムの呼び出し元がデフォルトの活動化グループで実行する。DFTACTGRP(*NO) も指定されている場合には、CRTBNDRPG コマンドの ACTGRP(*CALLER) しか指定できないことに注意してください。

注: *CALLER を介して OPM デフォルトの活動化グループで実行しているプログラムに関連する資源は、ジョブが終了するまで削除されません。

OPM RPG/400 と ILE RPG プログラムの互換性の維持

いくつかの RPG プログラムからなる OPM アプリケーションがある場合、ILE アプリケーションを次のように作成すると、移行されたアプリケーションが OPM アプリケーションであるかのように機能させることができます。

1. CVTRPGSRC コマンドを使用して各 OPM ソース・メンバーを変換し、/COPY メンバーを確実に変換できるようにする。

   詳細については、 458 ページの『ソースの変換』を参照してください。

2. CRTBNDRPG コマンドを使い、DFTACTGRP(*YES) を指定して、変換された各ソース・メンバーを別々にコンパイルして 1 つのプログラム・オブジェクトにバインドする。

OPM 互換性プログラムの詳細については、 23 ページの『方針 1: OPM 互換アプリケーション・プログラム』を参照してください。

活動化グループの削除

活動化グループが削除されると、その資源が再利用されます。この資源には静的記憶域およびオープン・ファイルが含まれます。*NEW 活動化グループは、関連したプログラムが呼び出し元に戻る時に削除されます。

名前の付いた活動化グループ (QILE など) は、明示的に削除されるか、ジョブが終了しない限り削除されないので、*永続* 活動化グループです。名前の付いた活動化グループで実行中のプログラムと関連した記憶域は、これらの活動化グループが削除されるまで解放されません。

DFTACTGRP(*YES) によって作成された ILE RPG プログラムは、LR オンで終了するか、異常終了した時にその記憶域が解放されます。

注: *CALLER を経由してデフォルトの活動化グループで実行中の ILE プログラムに関連した記憶域は、サインオフするか (対話式ジョブの場合)、ジョブが終了する (バッチ・ジョブの場合) まで解放されません。

多くの ILE RPG プログラムが活動状態になっている (つまり少なくとも 1 回呼び出されている) 場合には、システム記憶域が不足する可能性があります。したがって、ジョブが終了するまで記憶域は再利用されないので、OPM デフォルト活動化グループでは多量の静的記憶域を使用する ILE プログラムを実行しないようにしてください。

サービス・プログラムと関連した記憶域が再利用されるのは、関連した活動化グループが終了した時だけです。サービス・プログラムがデフォルトの活動化グループ内に呼び出された場合には、ジョブの終了時にその資源が再利用されます。

RCLACTGRP コマンドを使用すると、活動化グループの名前を指定して削除することができます。このコマンドを使用して、使用中でない非デフォルト活動化グループを削除してください。このコマンドには該当するすべての活動化グループを削除するか、あるいはある名前の活動化グループ 1 つを削除するかのオプションがあります。

RCLACTGRP コマンドについての詳細は、Web サイト http://www.ibm.com/eserver/iseries/infocenter で **iSeries 400 Information Center** の中の『*プログラミング*』カテゴリーの『*CL および API*』の節を参照してください。RCLACTGRP および活動化グループについて詳しくは、「*ILE 概念*」を参照してください。

資源再利用コマンド

資源再利用 (RCLRSC) コマンドは、もはや活動状態ではないプログラムの資源を解放するように設計されています。プログラムの作成方法によって、このコマンドは異なる処理を行います。プログラムが OPM プログラムであるか、あるいは DFTACTGRP(*YES) で作成されている場合には、RCLRSC コマンドはオープンされているファイルをクローズし、静的記憶域を解放します。

*CALLER で作成されたため OPM デフォルトの活動化グループ内で活動化された ILE プログラム、またはサービス・プログラムの場合、RCLRSC コマンドが出されるとファイルはクローズされます。プログラムの場合、記憶域は再初期化されますが、解放はされません。サービス・プログラムの場合は、記憶域が再初期化されることも解放されることもありません。

注: これは、デフォルトの活動化グループで実行しファイルをオープンにしたまま (LR をオフにして戻る) のサービス・プログラムがあり、そのサービス・プログラムを再度呼び出した時に RCLRSC を出すと、そのファイルはまだオープンした状態となり、何か入出力命令を出すとエラーが起こるという意味です。

名前の付いた活動化グループと関連を持つ ILE プログラムの場合、RCLRSC コマンドは影響力を持ちません。名前の付いた活動化グループの資源を解放するためには、RCLACTGRP コマンドを使用しなければなりません。

RCLRSC コマンドについての詳細は、Web サイト http://www.ibm.com/eserver/iseries/infocenter で **iSeries 400 Information Center** の中の『*プログラミング*』カテゴリーの『*CL および API*』の節を参照してください。RCLRSC と活動化グループについて詳しくは、「*ILE 概念*」を参照してください。

動的に割り振られた記憶域の管理

ILE では、ヒープを管理することによってプログラムから実行時記憶域を直接管理することができます。**ヒープ**は動的記憶域の割り振りに使用される記憶域のことです。 アプリケーションに必要な動的記憶域の量は、ヒープを使用するプログラムおよびプロシージャーによって処理されるデータによって異なります。

ヒープを管理するために、以下を使用できます。
- 命令コード ALLOC、REALLOC、および DEALLOC
- 組み込み関数 %ALLOC および %REALLOC
- ILE バインド可能 API

実行時記憶域を明示的に管理する必要はありません。しかし、動的に割り振られた実行時記憶域を使用したい場合には、明示的に管理することができます。例えば、配列または複数回繰り返しデータ構造の大きさをどれぐらいにすべきか正確には分からない場合に、明示的に管理することができます。配列またはデータ構造を BASED と定義し、プログラムがこれらの大きさを決定した時に、配列またはデータ構造の実際の記憶域を獲得することができます。

システムでは、デフォルトヒープとユーザー作成ヒープの 2 種類のヒープを使用することができます。RPG 記憶域管理命令は、デフォルトヒープを使用します。これ以降の節では、デフォルトヒープで RPG 記憶域管理命令を使用する方法、および記憶域管理 API を使用して独自のヒープを作成して使用する方法について説明します。ユーザー作成のヒープ、およびその他の ILE 記憶域管理の概念について詳しくは、「*ILE 概念*」を参照してください。

RPG 命令を使用したデフォルトヒープの管理

活動化グループ内で動的記憶域の最初の要求があると、**デフォルトヒープ**が作成され、ここから記憶域割り振りが行われます。動的記憶域の追加の要求は、デフォルトヒープからの割り振りによって満たされます。動的記憶域の現行要求を満たすための記憶域がヒープに不足している場合には、ヒープが拡張され、追加の記憶域が割り振られます。

割り振られた動的記憶域は、明示的に解放するか、あるいはヒープが破棄されるまで割り振られたままです。デフォルトヒープが破棄されるのは、所有する活動化グループが終了したときだけです。

同一の活動化グループのプログラムはすべて、同じデフォルトヒープを使用します。あるプログラムが割り振られた以上の記憶域をアクセスした場合には、別のプログラムに問題を起こすことがあります。例えば、2 つのプログラム、PGM A と PGM B が同じ活動化グループで実行されているものとします。10 バイトが PGM A に割り振られても 11 バイトが PGM A で変更されています。もし PGM B にエクストラ・バイトが実際に割り振られていた場合は、PGM B に問題が起こります。

デフォルトヒープでは次の RPG 命令を使用することができます。
- ALLOC 命令コードおよび %ALLOC 組み込み関数は、デフォルトヒープ内に記憶域を割り振ります。
- DEALLOC 命令コードは、ヒープ記憶域の 1 つ前の割り振りを解放します。

- REALLOC 命令コードおよび %REALLOC 組み込み関数は、あらかじめヒープから割り振られている記憶域のサイズを変更します。

注: ALLOC および %ALLOC が処理するのはデフォルトヒープだけですが、DEALLOC、REALLOC、および %REALLOC はデフォルトヒープとユーザー作成ヒープの両方を処理します。

図 52 は、名前のリンク・リストを作るために記憶域管理命令コードがどのように使われるかを示します。

```
 *---------------------------------------------------------------*
 * このモジュール内のサブプロシージャーのプロトタイプ           *
 *---------------------------------------------------------------*
D AddName         PR
D   name_parm                  40A
D Display         PR
D Free            PR
 *---------------------------------------------------------------*
 * リスト内の各要素には、名前へのポインターと次の要素への       *
 * ポインターが含まれています。                                  *
 *---------------------------------------------------------------*
D elem            DS                  BASED(elem@)
D   name@                      *
D   next@                      *
D   name_len                  5U 0
D nameVal         S           40A    BASED(name@)
D elemSize        C                   %SIZE(elem)
 *---------------------------------------------------------------*
 * リストの最初の要素は静的記憶域にあります。                    *
 * この要素の名前のフィールドには値を設定しません。              *
 *---------------------------------------------------------------*
D first           DS
D                                *   INZ(*NULL)
D                                *   INZ(*NULL)
D                               5U 0 INZ(0)
 *---------------------------------------------------------------*
 * これは現在の要素へのポインターです。                          *
 * elem@ を <first> のアドレスに設定すると、                     *
 * リストは空になります。                                        *
 *---------------------------------------------------------------*
D elem@           S               *   INZ(%ADDR(first))
 *---------------------------------------------------------------*
 * リスト内の 5 つの要素を書き込みます。                         *
 *---------------------------------------------------------------*
C                   DO        5
C     'Name?'       DSPLY                   name             40
C                   CALLP     AddName(name)
C                   ENDDO

 *---------------------------------------------------------------*
 * リストを表示した後で、このリストを解放します。                *
 *---------------------------------------------------------------*
C                   CALLP     Display
C                   CALLP     Free
C                   EVAL      *INLR = '1'
```

図 52. メモリー管理 - 名前のリンク・リストの作成 (1/5)

```
 *------------------------------------------------------------------*
 * サブプロシージャー                                               *
 *------------------------------------------------------------------*
 *------------------------------------------------------------------*
 * AddName - リストの末尾に名前を追加します                         *
 *------------------------------------------------------------------*
P AddName          B
D AddName          pi
D   name                         40A
 *------------------------------------------------------------------*
 * リストの現在の末尾の「次の」ポインターが指す新しい要素を         *
 * 配列に割り振ります。                                             *
 *                                                                  *
 * 割り振り前:                                                      *
 *                                                                  *
 *  .-------------.                                                 *
 *  |             |                                                 *
 *  | name     *--->abc                                            *
 *  | name_len 3  |                                                 *
 *  | next     *-------|||                                         *
 *  |             |                                                 *
 *  '-------------'                                                 *
 *                                                                  *
 *------------------------------------------------------------------*
C                   ALLOC     elemSize      next@
 *------------------------------------------------------------------*
 *                                                                  *
 * 割り振り後: elem@ がまだ元の要素を指しているため、まだ           *
 *  その元の要素が現在の要素のままであることに注意してください     *
 *                                                                  *
 *  .-------------.               .--------------.                  *
 *  |             |       .------>|              |                  *
 *  | name     *--->abc   |       |              |                  *
 *  | name_len 3  |       |       |              |                  *
 *  | next     *----------'       |              |                  *
 *  |             |               |              |                  *
 *  '-------------'               '--------------'                  *
 *                                                                  *
 * ここで、elem@ が新しい要素を指すように設定します                 *
 *------------------------------------------------------------------*
C                   EVAL      elem@ = next@
```

図 52. メモリー管理 - 名前のリンク・リストの作成 (2/5)

```
 *------------------------------------------------------------------*
 *                                                                  *
 * 設定後: これで、name@、name_len および next@ という名前が       *
 *        新しい要素にある記憶域を指すようになります。              *
 *                                                                  *
 *  .-------------.              .--------------.                   *
 *  |             |       .------>|              |                   *
 *  |          *--->abc   |      | name      *  |                   *
 *  |          3  |       |      | name_len     |                   *
 *  |          *----------'      | next      *  |                   *
 *  |             |              |              |                   *
 *  '-------------'              '--------------'                   *
 *                                                                  *
 * ここで新しい要素の値を設定します。                               *
 * 次のポインターを *NULL に設定して、リストの末尾であることを     *
 * 指定します。                                                     *
 *------------------------------------------------------------------*
C                   EVAL      next@ = *NULL
 *------------------------------------------------------------------*
 * 名前の長さを保管します (末尾のブランクはカウントしません)
 *------------------------------------------------------------------*
C                   EVAL      name_len = %len(%trimr(name))
 *------------------------------------------------------------------*
 * 記憶域を名前に割り振った後に、名前の値に設定します。
 *
 *------------------------------------------------------------------*
C                   ALLOC     name_len      name@
C                   EVAL      %SUBST(nameVal:1&gml.name_len) = name
 *------------------------------------------------------------------*
 *                                                                  *
 * 設定後:                                                          *
 *                                                                  *
 *  .-------------.              .--------------.                   *
 *  |             |       .------>|              |                   *
 *  |          *--->abc   |      | name      *--->newname          *
 *  |          3  |       |      | name_len  nn |                   *
 *  |          *----------'      | next      *--->|||              *
 *  |             |              |              |                   *
 *  '-------------'              '--------------'                   *
 *------------------------------------------------------------------*
P AddName         E
```

図 52. メモリー管理 - 名前のリンク・リストの作成 (3/5)

```
 *---------------------------------------------------------------*
 * 表示 - リストを表示します                                     *
 *---------------------------------------------------------------*
P Display         B
D saveElem@       S               *
D dspName         S             40A
 *---------------------------------------------------------------*
 * 現在の elem ポインターを保管して、表示した後に復元できるように  *
 * します。                                                      *
 * リスト・ポインターをリストの先頭に設定します。                *
 *---------------------------------------------------------------*
C                   EVAL      saveElem@ = elem@
C                   EVAL      elem@ = %ADDR(first)
 *---------------------------------------------------------------*
 * 次のポインターが                                              *
 * *NULL になるまで、リストの要素をループします。                *
 *---------------------------------------------------------------*
C                   DOW       next@ <> *NULL
C                   EVAL      elem@ = next@
C                   EVAL      dspName = %SUBST(nameVal:1:name_len)
C     'Name: '      dsply                   dspName
C                   ENDDO
 *---------------------------------------------------------------*
 * リスト・ポインターを元の場所に復元します。
 *---------------------------------------------------------------*
C                   EVAL      elem@ = saveElem@
P Display         E
```

図 52. メモリー管理 - 名前のリンク・リストの作成 (4/5)

```
 *---------------------------------------------------------------*
 * 解放 - リストが使用した記憶域を解放します                     *
 *---------------------------------------------------------------*
P Free            B
D prv@            S               *
 *---------------------------------------------------------------*
 * リスト内の最初の実要素から開始し、次のポインターが *NULL に   *
 * なるまで、リストの要素をループします。                        *
 *---------------------------------------------------------------*
C                   EVAL      elem@ = %ADDR(first)
C                   EVAL      elem@ = next@
C                   DOW       elem@ <> *NULL
 *---------------------------------------------------------------*
 * 名前の記憶域を解放します                                      *
 *---------------------------------------------------------------*
C                   DEALLOC                 name@
 *---------------------------------------------------------------*
 * 現在の elem@ へのポインターを保管します                       *
 *---------------------------------------------------------------*
C                   EVAL      prv@ = elem@
 *---------------------------------------------------------------*
 * elem@ を次の要素に進めます
 *---------------------------------------------------------------*
C                   EVAL      elem@ = next@

 *---------------------------------------------------------------*
 * 現在の要素の記憶域を解放します
 *---------------------------------------------------------------*
C                   DEALLOC                 prv@
C                   ENDDO

 *---------------------------------------------------------------*
 * 新しいリストの準備をします
 *---------------------------------------------------------------*
C                   EVAL      elem@ = %ADDR(first)
P Free            E
```

図 52. メモリー管理 - 名前のリンク・リストの作成 (5/5)

ヒープ記憶域の問題

130 ページの図 53 は、ヒープ記憶域の誤用に関連して起こる可能性のある問題を示しています。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
 *----------------------------------------------------------------*
 * ヒープ記憶域の誤用                                             *
 *----------------------------------------------------------------*
D Fld1            S             25A    BASED(Ptr1)
D Ptr1            S               *

 /FREE
    Ptr1 = %ALLOC(25);
    DEALLOC Ptr1;

  // この時点以降は、基底ポインター Ptr1 は割り振られた記憶域を
  // 指さなくなるので、Fld1 にアクセスすることはできません。

    SomePgm();

  // 'SomePgm' への以前の呼び出し時に、複数の記憶域の割り振りが行なわれた
  // 可能性があります。
  // いずれにしても、記憶域の 25 バイトに 'a' が埋め込まれているため、
  // 以下の割り振りを行なうのは非常に危険です。
  // この記憶域が現在何に使用されているのかを知ることはできません。

    Fld1 = *ALL'a';
 /END-FREE
```

図 53. ヒープ記憶域の誤用

同様に、以下のような場合にもエラーが発生する可能性があります。

- 再割り振りまたは割り振り解除が行われる前にポインターがコピーされている場合には、同様のエラーが生じる可能性があります。ポインターを割り振り済み記憶域にコピーする際には十分に注意して、記憶域の割り振り解除あるいは再割り振りが行われた後でポインターを使用しないようにする必要があります。
- ヒープ記憶域へのポインターがコピーされると、記憶域の割り振り解除または再割り振りにそのコピーが使用できます。この場合、元のポインターに新しい値を設定するまでは、このポインターを使用できません。
- ヒープ記憶域へのポインターがパラメーターとして渡されると、呼び出された側は、記憶域の割り振り解除または再割り振りを行うことができます。呼び出しが戻った後で、ポインターにアクセスしようとすると、問題が生じる場合があります。
- ヒープ記憶域へのポインターが *INZSR に設定されていると、その後でポインターの RESET を行うと、ポインターが、割り振られていない記憶域に設定される可能性があります。
- ヒープ記憶域へのポインターが (消去されたり、例えば ALLOC 命令によって新しいポインターに設定されることによって) 失われた場合には、別の問題が生じる場合があります。ポインターが失われた後では、このポインターが指していた記憶域を解放することはできません。この記憶域がアドレス不可能になったことをシステムが認識していないため、この記憶域を割り振ることはできません。

  この記憶域が解放されるのは、活動化グループが終了した後です。

ILE バインド可能 API を使用したユーザー独自のヒープ管理

1 つ以上のユーザー作成ヒープを作成することによって、活動化グループ内の一部のプログラムおよびプロシージャーで使用される動的記憶域を分離することができます。ユーザー作成ヒープの作り方については、「*ILE 概念*」を参照してください。

次の例は、ILE RPG プロシージャーからユーザー作成ヒープを使用して、実行時配列用の動的記憶域を管理する方法を示したものです。この例では、モジュール DYNARRAY 内のプロシージャーが、実際にバインドされていないパック配列に記憶域を動的に割り振ります。このモジュール内のプロシージャーは、この配列で以下の処置を実行します。

- 配列の初期化
- 配列への要素の追加
- 要素の値の戻し
- 配列用の記憶域の解放

DYNARRAY は、REALLOC 命令コードだけでなく、CEECRHP (ヒープ作成)、CEEGTST (記憶域取得)、および CEEDSHP (ヒープ廃棄) という 3 つの ILE バインド可能記憶域 API を使用してこれらの処置を行います。記憶域管理バインド可能 API に特有の情報は、Web サイト http://www.ibm.com/eserver/iseries/infocenter で **iSeries 400 Information Center** の中の『*プログラミング*』カテゴリーの『*CL および API*』の節を参照してください。

図 54 では、DYNARRAY 内のプロシージャーのプロトタイプを含む、/COPY ファイル DYNARRI を示します。この /COPY ファイルは、DYNARRAY にプロシージャーを呼び出すその他のモジュールだけでなく、DYNARRAY モジュールによって使用されます。

DYNARRAY は、(15,0) パック 10 進数配列で使用されるように定義されています。これは、単に DYNA_TYPE の定義を文字フィールドに変更することによって、文字配列の処理を容易に変換することができます。

```
     *====================================================================
     * DYNARRAY :  (実際には) バインドされていない実行時パック (15,0)
     *             配列の処理。DYNARRAY モジュールには、配列を割り振り、
     *             配列値を戻すか設定し、配列を割り振り解除する
     *             プロシージャーが含まれています。
     *====================================================================
     D DYNA_TYPE       S             15P 0
     D DYNA_INIT       PR
     D DYNA_TERM       PR
     D DYNA_SET        PR
     D   Element                             VALUE LIKE(DYNA_TYPE)
     D   Index                        5I 0   VALUE
     D DYNA_GET        PR                    LIKE(DYNA_TYPE)
     D   Index                        5I 0   VALUE
```

図 54. DYNARRAY モジュールのプロトタイプを含む /COPY ファイル DYNARRI

132 ページの図 55 は、制御仕様書と定義仕様書を含むモジュール DYNARRAY の最初の部分を示します。

```
 *====================================================================
 * DYNARRAY :  (実際には) バインドされていない実行時パック (15,0)
 *             配列の処理。このモジュールには、配列を割り振り、
 *             配列値を戻すか設定し、配列を割り振り解除する
 *             プロシージャーが含まれています。
 *====================================================================
H NOMAIN
 *--------------------------------------------------------------------
 * このモジュール内のプロシージャー用のプロトタイプ
 *--------------------------------------------------------------------
 /COPY DYNARRI
 *--------------------------------------------------------------------
 * CEEGTST API (ヒープ記憶域取得) へのインターフェース
 *  1) HeapId     = ヒープの ID。
 *  2) Size   = 割り振るバイト数
 *  3) RetAddr= 割り振られた記憶域の戻りアドレス
 *  4) *OMIT  = フィードバック・パラメーター。ここに *OMIT を指定
 *                 することは、要求が満たされなかった時に API から
 *                 例外を受け取ることを意味します。
 *                 これを監視しているわけではないので、呼び出し
 *                 プロシージャーが例外を受け取ることになります。
 *--------------------------------------------------------------------
D CEEGTST         PR
D   HeapId                      10I 0       CONST
D   Size                        10I 0       CONST
D   RetAddr                       *
D   Feedback                    12A         OPTIONS(*OMIT)
 *--------------------------------------------------------------------
 * CEECRHP API (ヒープ作成) へのインターフェース
 *  1) HeapId     = ヒープの ID。
 *  2) InitSize   = ヒープの初期サイズ。
 *  3) Incr       = ヒープを拡大する必要がある場合に
 *                  増分するバイト数。
 *  4) AllocStrat = このヒープの場合の割り振り方法。値を
 *                  0 に設定すると、システムが
 *                  最適な方法を選択することができます。
 *  5) *OMIT      = フィードバック・パラメーター。ここに *OMIT を指定
 *                  することは、要求が満たされなかった時に API から
 *                  例外を受け取ることを意味します。
 *                  これを監視しているわけではないので、呼び出し
 *                  プロシージャーが例外を受け取ることになります。
 *--------------------------------------------------------------------
D CEECRHP         PR
D   HeapId                      10I 0
D   InitSize                    10I 0       CONST
D   Incr                        10I 0       CONST
D   AllocStrat                  10I 0       CONST
D   Feedback                    12A         OPTIONS(*OMIT)
```

図 55. DYNARRAY のグローバル変数とローカル・プロトタイプ (1/2)

```
 *---------------------------------------------------------------
 * CEEDSHP API (ヒープ廃棄) へのインターフェース
 *  1) HeapId     = ヒープの ID。
 *  2) *OMIT      = フィードバック・パラメーター。ここに *OMIT を指定
 *                  することは、要求が満たされなかった時に API から
 *                  例外を受け取ることを意味します。
 *                  これを監視しているわけではないので、呼び出し
 *                  プロシージャーが例外を受け取ることになります。
 *---------------------------------------------------------------
D CEEDSHP         PR
D   HeapId                      10I 0
D   Feedback                    12A         OPTIONS(*OMIT)
 *---------------------------------------------------------------
 * グローバル変数
 *---------------------------------------------------------------
D HeapVars        DS
D   HeapId                      10I 0
D   DynArr@                       *
 *---------------------------------------------------------------
 * 動的配列の定義。要素数を使用可能な最大数でコーディングしますが、
 * この定義には実際には記憶域は宣言されないことに注意して
 * ください (これが BASED であるため)。
 *---------------------------------------------------------------
D DynArr          S                   DIM(32767) BASED(DynArr@)
D                                     LIKE(DYNA_TYPE)
 *---------------------------------------------------------------
 * 動的配列の要素の現在数を、
 * 常にグローバルに記録しておきます。
 *---------------------------------------------------------------
D NumElems        S             10I 0 INZ(0)

 *---------------------------------------------------------------
 * 配列に割り振られる要素の初期数および
 * 後続の割り振りで配列に追加される
 * 要素の最小数。
 *---------------------------------------------------------------
D INITALLOC       C                   100
D SUBSALLOC       C                   100
```

図 55. DYNARRAY のグローバル変数とローカル・プロトタイプ (2/2)

134 ページの図 56 は DYNARRAY にあるサブプロシージャーを示したものです。

```
 *====================================================================
 * DYNA_INIT: 配列を初期化します。
 *
 * 機能:       ヒープを作成し、実行時配列に記憶域の初期量を
 *             割り振ります。
 *====================================================================
P DYNA_INIT       B                   EXPORT
 *--------------------------------------------------------------------
 * ローカル変数。
 *--------------------------------------------------------------------
D Size            S             10I 0
 *
 * 事前に定義された要素数で開始します。
 *
C                   Z-ADD     INITALLOC     NumElems
 *
 * 配列に必要なバイト数を決定します。
 *
C                   EVAL      Size =  NumElems * %SIZE(DynArr)
 *
 * ヒープの作成
 *
C                   CALLP     CEECRHP(HeapId : Size : 0 : 0 : *OMIT)

 *
 * 記憶域を割り振り、配列の基底ポインターを、API から
 * 戻されたポインターに設定します。
 *
 * ALLOC 命令コードがデフォルトヒープを使用するため、CEEGTST API を使用して
 * 別のヒープを指定する必要があることに注意してください。
 *
C                   CALLP     CEEGTST(HeapId : Size : DynArr@ : *OMIT)

 *
 * 配列の記憶域の初期化
 *
C     1             DO        NumElems      I                 5 0
C                   CLEAR                   DynArr(I)
C                   ENDDO
P DYNA_INIT       E

 *====================================================================
 * DYNA_TERM: 配列処理を終了させます。
 *
 * 機能:       ヒープを削除します。
 *====================================================================
P DYNA_TERM       B                   EXPORT
C                   CALLP     CEEDSHP(HeapId : *OMIT)
C                   RESET                   HeapVars
P DYNA_TERM       E
```

図 56. DYNARRAY のサブプロシージャー (1/4)

```
 *===============================================================
 * DYNA_SET: 配列要素を設定します。
 *
 * 機能:      この要素に対して配列が十分な大きさであることを確認して、
 *           要素を指定された値に設定します。
 *===============================================================
P DYNA_SET        B                   EXPORT
 *---------------------------------------------------------------
 * このプロシージャーの入力パラメーター。
 *---------------------------------------------------------------
D DYNA_SET        PI
D   Element                             VALUE LIKE(DYNA_TYPE)
D   Index                          5I 0 VALUE
 *---------------------------------------------------------------
 * ローカル変数。
 *---------------------------------------------------------------
D Size            S             10I 0
 *---------------------------------------------------------------
 * 配列への追加を選択した場合には、配列が十分に大きいかどうかが
 * 最初に検査され、十分でない場合にはそのサイズが増やされます。
 * 要素を追加します。
 *---------------------------------------------------------------
C     Index         IFGT      NumElems
C                   EXSR      REALLOC
C                   ENDIF
C                   EVAL      DynArr(Index) = Element
 *===============================================================
 * REALLOC: 記憶域再割り振りサブルーチン
 *
 *         機能:    動的配列のサイズを増やして、
 *                  新しい要素を初期化します。
 *===============================================================
C     REALLOC       BEGSR

 *
 *  元の要素の数を記憶
 *
C                   Z-ADD     NumElems      OldElems          5 0
```

図 56. DYNARRAY のサブプロシージャー (2/4)

```
 *
 *  新しい要素の数を計算します。指標が配列内の現在の要素数に
 *  新しい割り振り数を加えたものより大きい場合には、その
 *  指標まで割り振り、そうでない場合には
 *  配列に新しい割り振り量を加算します。
 *
C                   IF        Index > NumElems + SUBSALLOC
C                   Z-ADD     Index         NumElems
C                   ELSE
C                   ADD       SUBSALLOC     NumElems
C                   ENDIF
 *
 * 配列の新しいサイズを計算します
 *
C                   EVAL      Size =  NumElems * %SIZE(DynArr)
 *
 * 記憶域を再割り振りします。新しい記憶域の値は、元の記憶域と
 * 同じです。
 *
C                   REALLOC   Size          DynArr@
 *
 * 配列の新しい要素を初期化します。
 *
C     1             ADD       OldElems      I
C     I             DO        NumElems      I                 5 0
C                   CLEAR                   DynArr(I)
C                   ENDDO
C                   ENDSR
P DYNA_SET        E
```

図 56. DYNARRAY のサブプロシージャー (3/4)

```
 *==================================================================
 * DYNA_GET: 配列要素を戻します。
 *
 * 機能:     配列要素が配列のサイズ内である場合には、この要素の
 *           現在の値を戻します。そうでない場合には、
 *           デフォルト値を戻します。
 *==================================================================
P DYNA_GET        B                   EXPORT
 *------------------------------------------------------------------
 * このプロシージャーの入力パラメーター。
 *------------------------------------------------------------------
D DYNA_GET        PI                  LIKE(DYNA_TYPE)
D   Index                        5I 0 VALUE
 *------------------------------------------------------------------
 * ローカル変数。
 *------------------------------------------------------------------
D Element         S                   LIKE(DYNA_TYPE) INZ
 *------------------------------------------------------------------
 * 要求した配列が現行サイズ内であれば、この要素の現行値を
 * 戻します。そうでない場合には、デフォルト (初期化) の値を
 * 使用することができます。
 *------------------------------------------------------------------
C     Index       IFLE      NumElems
C                 EVAL      Element = DynArr(Index)
C                 ENDIF
C                 RETURN    Element
P DYNA_GET        E
```

図 56. DYNARRAY のサブプロシージャー (4/4)

このサブプロシージャーのロジックは次のとおりです。

1. DYNA_INIT は、ILE バインド可能 API CEECRHP (ヒープ作成) を使用してヒープを作成し、グローバル変数 HeapId にヒープ ID を保管する。これは、ILE バインド可能 API CEEGTST (ヒープ記憶域取得) を呼び出すことによって、配列の初期値 (この場合は 100) に基づいてヒープ記憶域を割り振ります。
2. DYNA_TERM は、ILE バインド可能 API CEEDSHP (ヒープ廃棄) を使用してヒープを破棄する。
3. DYNA_SET は要素の値を配列に設定する。

   要素を配列に追加する前に、プロシージャーは十分なヒープ記憶域があるかどうかを調べます。十分でない場合には、命令コード REALLOC を使用して追加の記憶域を取得します。
4. DYNA_GET は指定された要素の値を戻す。プロシージャーは、要求された要素かゼロのいずれかを呼び出し元に戻します。ゼロが戻されるのは、要求された要素が実際には配列に保管されていない場合です。

モジュール DYNARRAY を作成するには、以下のとおりタイプしてください。

```
CRTRPGMOD MODULE(MYLIB/DYNARRAY) SRCFILE(MYLIB/QRPGLESRC)
```

このプロシージャーは、この後で CRTPGM または CRTSRVPGM を使用してその他のモジュールとバインドすることができます。

図 57 DYNARRAY 内のモジュールをテストする、別のモジュールを示します。

```
     *====================================================================
     * DYNTEST: DYNARRAY モジュールのテスト・プログラム
     *====================================================================
     /COPY EXAMPLES,DYNARRI
    D X               S                   LIKE(DYNA_TYPE)
     * 配列の初期化
    C                   CALLP     DYNA_INIT
     * 一部要素の設定
    C                   CALLP     DYNA_SET (25 : 3)
    C                   CALLP     DYNA_SET (467252232 : 1)
    C                   CALLP     DYNA_SET (-2311 : 750)
     * 一部要素の検索
    C                   EVAL      X = DYNA_GET (750)
    C     '750'         DSPLY                   X
    C                   EVAL      X = DYNA_GET (8001)
    C     '8001'        DSPLY                   X
    C                   EVAL      X = DYNA_GET (2)
    C     '2'           DSPLY                   X

     * 終結処理
    C                   CALLP     DYNA_TERM
    C                   SETON                                        LR
```

図 57. DYNARRAY のプロシージャーを使用するサンプル・モジュール

第 10 章 プログラムおよびプロシージャーの呼び出し

ILE では、プログラムまたはプロシージャーのどちらかを呼び出すことができます。さらに、ILE RPG ではプロトタイプまたは非プロトタイプのプログラムおよびプロシージャーを呼び出す機能が提供されます(プロトタイプとは、コンパイラーがコンパイル時にインターフェースをチェックできる呼び出しインターフェースの外部定義です)。

プログラムまたはプロシージャーを呼び出すのに望ましい方法は、プロトタイプ呼び出しを使用することです。プロトタイプのプロシージャーまたはプログラムに対するパラメーターの呼び出しおよび受け渡しのための構文は、組み込み関数と一緒に使用される、または式の中で使用される同一の自由形式構文を使用します。このため、プロトタイプ呼び出しは「自由形式」呼び出しとして参照されることがよくあります。

次の場合には、プログラムまたはプロシージャーを呼び出すのに CALL または CALLB 命令を使用してください。

- 非常に単純な呼び出しインターフェースの場合
- 演算項目 1 および演算項目 2 に PARM 命令の機能が必要な場合
- プロトタイプのパラメーター検査で可能である以上の柔軟性が必要な場合

この章では、次の処理を行う方法について説明します。

- プログラムまたはプロシージャーを呼び出す
- プロトタイプ呼び出しを使用する
- プロトタイプのパラメーターを渡す
- 固定形式の呼び出しを使用する
- プログラムまたはプロシージャーから戻る
- ILE バインド可能 API を使用する
- グラフィックス・ルーチンを呼び出す
- 特殊なルーチンを呼び出す

プログラム/プロシージャー呼び出しの概要

ILE 内でのプログラム処理はプロシージャー・レベルで行われます。ILE プログラムは 1 つ以上のモジュールから成り、モジュールは 1 つ以上のプロシージャーから成ります。ILE RPG モジュールには任意指定のメイン・プロシージャーおよびゼロかそれ以上のサブプロシージャーが含まれます。この章では、「プロシージャー」という語はメイン・プロシージャーおよびサブプロシージャーの両方に適用されます。

ILE「プログラム呼び出し」はプロシージャー呼び出しの特殊な形式です。すなわち、これはプログラム入力プロシージャーに対する呼び出しです。プログラム入力プロシージャーは、プログラム作成時に指定され、プログラムが呼び出される時に制御を受け取るプロシージャーです。プログラムの入り口モジュールが ILE RPG

モジュールの場合には、そのモジュールのメイン・プロシージャーが、プログラムの呼び出し直後にプログラム入力プロシージャーによって呼び出されます。

この項には次の一般説明が入っています。

- プロシージャー呼び出しとプログラム呼び出しの比較
- 呼び出しスタック (あるいは一連の呼び出しの対話方法)
- 再帰
- パラメーターの受け渡しについての考慮事項

プログラムの呼び出し

プログラム呼び出しを使用して OPM または ILE プログラムを呼び出すことができます。**プログラム呼び出し**は、プログラム・オブジェクト (*PGM) に対して行われる呼び出しです。呼び出されるプログラムの名前は実行時にアドレスに分析解決されますが、これは呼び出し側プログラムが、制御を初めて呼び出されるプログラムに渡す直前になります。このため、プログラム呼び出しはよく動的呼び出しと言われます。

ILE プログラム、EPM プログラム、または OPM プログラムに対する呼び出しは、すべてプログラム呼び出しの例です。また、バインド不可能 API の呼び出しも、プログラム呼び出しの例です。

プログラム呼び出しを行うためには、CALLP 命令または CALL と PARM の両方の命令を使用します。CALL および PARM 命令を使用する場合には、コンパイラーはパラメーター上のタイプ検査を実行できませんが、これは実行時エラーになります。

ILE プログラムが呼び出される時に、プログラム入力プロシージャーはプログラム・パラメーターを受け取り、プログラムの初期制御権が与えられます。さらに、プログラム内のすべてのプロシージャーがプロシージャー呼び出し用に使用可能となります。

プロシージャーの呼び出し

OPM プログラムとは異なり、ILE プログラムはプログラム呼び出し以外の呼び出しも使用することができます。ILE プログラムでは、他のプロシージャーを呼び出すために、静的プロシージャー呼び出しまたはプロシージャー・ポインター呼び出しも使用することができます。また、プロシージャー呼び出しはバインド呼び出しとも言われています。

静的プロシージャー呼び出しとは、バインド中にプロシージャー呼び出しの名前があるアドレスに帰着する ILE プロシージャーの呼び出しで、静的という用語はここから出ています。結果として、静的プロシージャー呼び出しを使用した実行時パフォーマンスは、プログラム呼び出しを使用した実行時パフォーマンスより高速です。静的呼び出しによって、操作記述子、省略パラメーターが可能になり、渡されるパラメーター数の範囲が拡張します (399 まで)。

プロシージャー・ポインター呼び出しは、プロシージャーを動的に呼び出す手段を提供します。例えば、プロシージャー・ポインターをパラメーターとして別のプロシージャーに渡すことができ、このプロシージャーは次に渡されたパラメーターで

指定されたプロシージャーを実行することになります。また、プロシージャー名の配列または異なるプロシージャーへプロシージャー呼び出しを動的に経路指定するアドレスも取り扱うことができます。呼び出されたプロシージャーが同一の活動化グループにある場合には、プロシージャー・ポインター呼び出しのコストはほとんど静的プロシージャー呼び出しのコストと同じです。

プロシージャー呼び出しのどちらかのタイプを使用して次を呼び出すことができます。

- 同じ ILE プログラム、またはサービス・プログラム内の別々のモジュールのプロシージャー
- 別々の ILE サービス・プログラムのプロシージャー

静的プロシージャー呼び出しを使用して呼び出すことができるプロシージャーは、プロシージャー・ポインターによっても呼び出すことができます。

静的プロシージャー呼び出しの使用例のリストについては、 147 ページの『自由形式呼び出しの例』および 164 ページの『CALL および CALLB の例』を参照してください。プロシージャー・ポインターの使用例については、「*WebSphere Development Studio: ILE RPG 解説書*」のプロシージャー・ポインターのデータ・タイプのセクションを参照してください。

プロシージャー呼び出しを行うためには、CALLP 命令または CALLB と PARM の両方の命令を使用します。プロシージャーが値を戻す場合には、式でプロトタイプ・プロシージャーを呼び出すこともできます。CALLB および PARM 命令を使用する場合には、コンパイラーは、実行時にエラーを引き起こす可能性のあるパラメーター上のタイプ検査を行えません。

呼び出しスタック

呼び出しスタックとは、呼び出しスタック項目のリストであり、後入れ先出し (LIFO) 順序に従います。**呼び出しスタック項目**は、プログラムまたはプロシージャーの呼び出しです。1 つのジョブにつき 1 つの呼び出しスタックがあります。

ILE プログラムが呼び出されると、最初にプログラム入力プロシージャーが呼び出しスタックに追加されます。するとシステムは自動的にプロシージャー呼び出しを実行し、関連したユーザーのプロシージャー (メイン・プロシージャー) が追加されます。プロシージャーが呼び出される時に、ユーザーのプロシージャー (メイン・プロシージャーまたはサブプロシージャー) だけが追加されます。すなわちプログラム入力プロシージャーのオーバーヘッドはありません。

142 ページの図 58 は、ILE プログラムを呼び出す OPM プログラムから構成されるアプリケーションの呼び出しスタックを示します。ILE プログラムの RPG メイン・プロシージャーは RPG サブプロシージャーを呼び出し、次に C プロシージャーを呼び出します。本書の図では、最新項目がスタックの下部にあるということに注意してください。

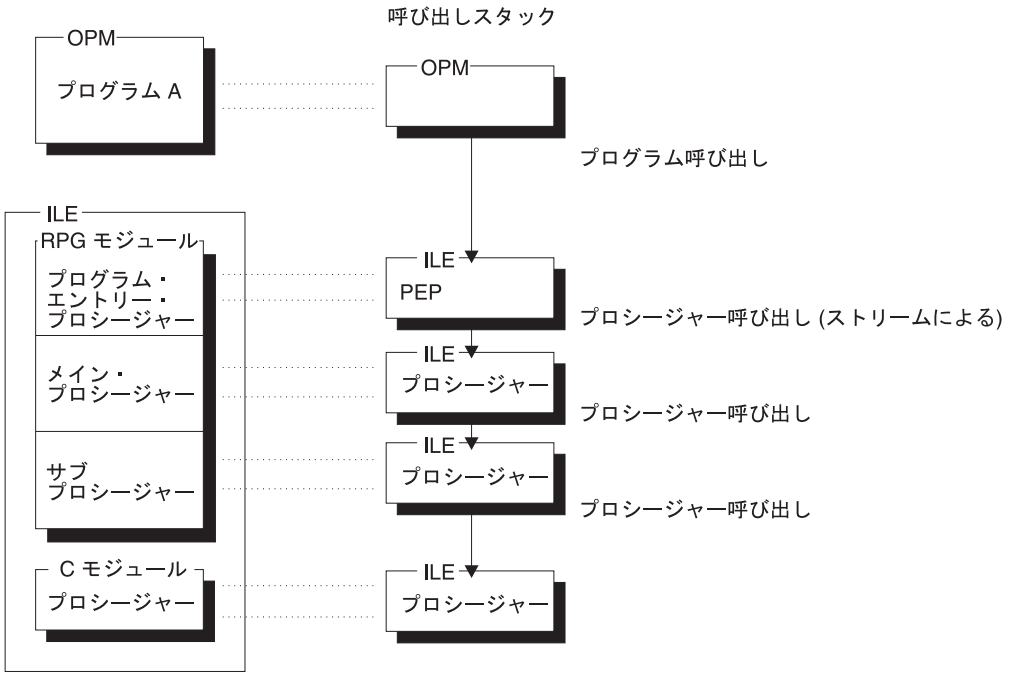


図 58. 呼び出しスタック上のプログラム呼び出しとプロシージャー呼び出し

注: ユーザー入り口プロシージャー (UEP) の呼び出しは自動的に行われるので、プログラム呼び出しでは、プログラム入力プロシージャーとユーザー入り口プロシージャーの呼び出しが同時に行われます。したがってここからは、プログラム呼び出しの 2 つのステップはこの章および残りの章で出てくる呼び出しスタックの図表と合わせて参照してください。

再帰呼び出し

再帰呼び出しはサブプロシージャーの場合にだけ使用することができます。**再帰呼び出し**とは、プロシージャー A がそれ自身を呼び出すか、あるいはプロシージャー B を呼び出し、プロシージャー B がそれから再びプロシージャー A を呼び出す呼び出しです。各再帰呼び出しによって、プロシージャーの新しい呼び出しが呼び出しスタックに入れられることになります。新しい呼び出しには自動記憶域のすべてのデータ項目に対し新しい記憶域があり、その記憶域は局所的なために他の呼び出しでは使用できません (定義に STATIC キーワードを指定しないかぎり、サブプロシージャーで定義されるデータ項目は自動記憶域を使用します)。早い方の呼び出しに関連した自動記憶域は後からの呼び出しに影響されないことにも注意してください。

呼び出しスタックにあるメイン・プロシージャーは、その呼び出し元に戻るまで呼び出すことはできません。したがって、既に活動中のメイン・プロシージャーを呼び出す可能性のあるプロシージャーを呼び出さないように注意してください。

不注意に再帰呼び出しに至る状況を避けてください。例えば、 143 ページの図 59 に示すように、3 つのモジュールがあるとします。

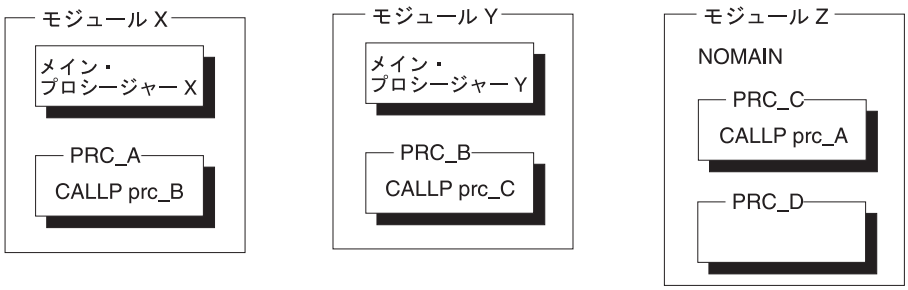


図 59. それぞれがサブプロシージャーをもつ 3 つのモジュール

モジュール X のプロシージャー A はモジュール Y のプロシージャー B を呼び出すプログラムを実行しているとします。プロシージャー B があるフィールドを処理するということ以外、B が何をするか分かりません。一方、プロシージャー B はプロシージャー C を呼び出し、C はプロシージャー A を呼び出します。プロシージャー C がプロシージャー A を呼び出すと、再帰呼び出しになります。呼び出しスタックの順序は図 60 に示すとおりです。最新の呼び出しスタック項目が最下部にあることに注意してください。

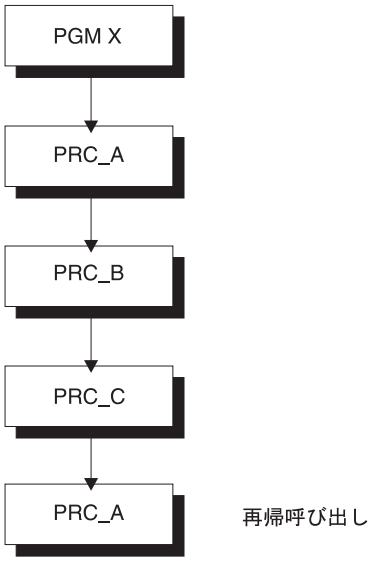


図 60. 避けるべき再帰呼び出しスタック

サブプロシージャーは再帰的に呼び出されることが可能ですので、再帰が起こっていることに気付かない場合には、システム資源を使い果たすことがあります。

注意!
無条件再帰呼び出しを使用すると無限再帰の状態になり、システム資源の過剰使用につながる可能性があります。無限の再帰は適切なプログラミングで避けることができます。一般に、適切な再帰的プロシージャーは、望ましい結果が得られたかどうかを判別するテストで開始されます。必要な結果が得られていれば、再帰プロシージャーは最新の呼び出し元に戻ります。

パラメーターの受け渡しについての考慮事項

呼び出しインターフェースの設計時には、パラメーターの渡し方について多くの決定をしなければなりません。他方、ユーザーが呼び出し元の場合には、大部分の決定が既にユーザーのためになされています。次に呼び出しインターフェースの設計時に忘れてはならないパラメーター受け渡しの考慮事項をいくつかリストします。

- コンパイル時のパラメーターの検査

  プロトタイプ呼び出しの呼び出しインターフェースはコンパイル時に検査されます。この検査によって次のことが確認されます。

  – データ・タイプが正しく使用されていること
  – すべての必要なパラメーターが渡されていること
  – *OMIT が使用可能な場所でだけ渡されていること

- パラメーターの受け渡し方法

  各 HLL は 1 つ以上のパラメーターの受け渡し方法を提供します。これらにはパラメーター値へのポインターの受け渡し、値のコピーの受け渡し、または値自身の受け渡しが含まれます。

- 操作記述子の受け渡し

  時には、ユーザーに渡されるデータの正確な形式が分らないことがあります。この場合には、渡されたパラメーターの形式に関する詳細説明を提供するために操作記述子を渡すよう要求することができます。

- パラメーターの数

  一般に、呼び出されるプログラムまたはプロシージャーに必要な数のパラメーターを渡す必要があります。必要数より少ないパラメーターが渡され、呼び出された側がデータのないパラメーターを参照した場合には、呼び出された側でエラーとなります。

- 少ないデータの受け渡し

  パラメーターを渡したとき渡したデータが小さすぎる場合には、アプリケーションは正しく機能しません。パラメーターを変更する場合には、記憶域を重ね書きすることがあります。パラメーターを使用する場合には、パラメーターを誤って解釈することがあります。コンパイラーは、パラメーターのプロトタイピングによって、パラメーターの長さが適切かどうかを調べます。

  呼び出された側がパラメーターが最大長よりも短いことを指示した場合には (文書またはプロトタイプによって)、安心して短いパラメーターを渡すことができます (しかし、呼び出されたプロシージャーが、必要なものより少ないデータを 処理するように作成されていなければならないことに注意してください)。

- 評価の順序

  プロトタイプの呼び出しのパラメーターの評価順序は保証されていません。パラメーターがパラメーター・リストで複数回使用されている場合には、このことは重要であり、副次的な影響を与えることがあります。

- 言語間呼び出しについての考慮事項

  HLL によって、プログラムとプロシージャー間のデータの受け渡し方法が異なるだけでなく、サポートしているデータ表示方法も異なっています。一般に、呼び出し側と呼び出された側のプログラムまたはプロシージャーに共通のデータ・タイプで、両方でサポートされるデータだけを使用して渡す必要があります。

表 25 は上記の考慮事項と、プロトタイプまたは非プロトタイプの 2 つのパラメーターを関連付けています。

表 25. パラメーター受け渡しオプション

パラメーター・オプション	プロトタイプ	非プロトタイプ	参照ページ
コンパイル時のパラメーターの検査	はい		147
参照による受け渡し	はい	はい	148
値による受け渡し	はい (b)		148
読み取り専用参照による受け渡し	はい		148
操作記述子の受け渡し	はい (b)	はい (b)	151
*OMIT を渡す	はい (b)	はい (b)	152
パラメーター省略の制御	はい	はい	153
渡されたパラメーターの数の取得	はい	はい	154
正しくないパラメーター長の使用不能	はい		159
注: (b) - バインドされたプロシージャーにのみ適用されます			

プロトタイプ呼び出しの使用

プロトタイプ呼び出しは、パラメーター検査に使用可能なプロトタイプのある呼び出しです。これはかなり単純な呼び出しインターフェースをもち、より多くの機能を提供します。例えば、プロトタイプ呼び出しを使用して (同じ構文で) 次を呼び出すことができます。

- 実行時にシステム上にあるプログラム
- 同じプログラムまたはサービス・プログラムにバインドされた、他のモジュールまたはサービス・プログラム内のエクスポートされたプロシージャー
- 同じモジュール内のサブプロシージャー

RPG では、プロトタイプ呼び出しは自由形式呼び出しとしても知られています。**自由形式呼び出し**は、組み込み関数の引き数に非常によく似た、呼び出しの引き数が自由形式構文を使って指定される呼び出し構文のことを言います。これは固定形式呼び出しと対比するもので、この場合には引き数は別個の仕様書に入れられます。使用される戻り値があるかどうかにより、自由形式呼び出しを作成には 2 つの方法があります。戻り値がない場合には、CALLP 命令を使用してください。戻り値があり、戻された値を使用したい場合には、例えば EVAL を使って式の中にプロトタイ

プ・プロシージャーを入れてください。値を戻すプロシージャーに対し CALLP を使用した場合には、戻り値は無視されます。

注: プロトタイプ・プロシージャーだけが値を戻すことができ、プロトタイプのプログラムは戻すことができません。

プロシージャー呼び出しにおいて、オプションで、パラメーターがないことを示す括弧をコーディングすることができます。これにより、プロシージャー呼び出しとスカラー変数の区別がより容易になります。

プロトタイプ・パラメーターの受け渡しの説明については、147 ページの『プロトタイプ・パラメーターの受け渡し』を参照してください。

CALLP 命令の使用

ユーザーは CALLP (プロトタイプ・プロシージャーの呼び出し) 命令を使って、任意の言語で書かれたプロトタイプ・プログラムまたはプロシージャーを呼び出すことができます。CALLP 命令は次の拡張因子 2 構文を使用します。

```
C                   CALLP     NAME{ (PARM1 {:PARM2 ...}) }
```

自由形式演算では、命令拡張がない場合には CALLP を省略することができます。自由形式命令は次の形式のいずれかを使用することができます。

```
/free
    callp name { (parm1 { :parm2 ...}) };
    name({parm1 {:parm2 ... }});
/end-free
```

プロトタイプ・プログラムまたはプロシージャーを呼び出すには、以下の一般ステップに従います。

1. 呼び出されるプログラムまたはプロシージャーのプロトタイプを定義仕様書に組み込んでください。
2. 拡張演算項目 2 フィールドに、プログラムまたはプロシージャーのプロトタイプ名を入力し、パラメーターがあれば括弧に入れて、これに続けます。パラメーターはコロン (:) で区切ります。演算項目 1 はブランクでなければなりません。

次の例は、プロシージャー Switch の呼び出しを示します。このプロシージャーは、渡される標識 (この場合には *IN10) の状態を変更します。

```
C                   CALLP     Switch(*in10)
```

プログラム呼び出しでは、最大 255 パラメーターを使用することができ、プロシージャー呼び出しでは、最大 399 パラメーターを使用することができます。

モジュール内のどこからでも CALLP を使用することができます。プロトタイプにキーワード EXTPGM が指定されている場合には、呼び出しは動的外部呼び出しになります。そうでない場合には、これはバインド・プロシージャー呼び出しです。

CALLP を使用して値を戻すプロシージャーを呼び出した場合には、その値は呼び出し側には使用できないことに注意してください。値が必要な場合は、式の中でプロトタイプ・プロシージャーを呼び出します。

式の中での呼び出し

値を戻すようにプロトタイプ・プロシージャーを定義し、この戻り値を使用したい場合には式の中でプロシージャーを呼び出さなければなりません。指定された戻り値のデータ・タイプと一貫性のあるやり方で、プロシージャー名を使用してください。例えば、プロシージャーが数値を戻すように定義された場合には、式の中でのプロシージャーの呼び出しは、数値の必要な個所でなければなりません。

図 61 は、数値の入力パラメーターを受け取りストリングを戻すプロシージャー CVTCHR のプロトタイプです。図 62 は、式の中でプロシージャーがどのように使用されるかを示しています。

```
     * CVTCHR のプロトタイプ
     * - 数値パラメーターの文字表現を戻す
     * 例:         CVTCHR(5) は '5' を戻す
     *             CVTCHR(15-124) は '-109' を戻す
    D CVTCHR          PR            31A
    D   NUM                         30P 0   VALUE
```

図 61. CVTCHR のプロトタイプ

```
    C                   EVAL      STRING = 'Address: ' +
    C                                      %TRIM(CVTCHR(StreetNum))
    C                                      + ' ' + StreetName
     * STREETNUM = 427 で STREETNAME = 'Mockingbird Lane' の場合、
     * EVAL operation STRING = 'ADDRESS: 427 Mockingbird Lane'
```

図 62. 式の中のプロトタイプ・プロシージャーの呼び出し

自由形式呼び出しの例

CALLP 命令の使用例については、次を参照してください。

- 44 ページの図 22
- 107 ページの図 43
- 268 ページの図 122
- 158 ページの図 70
- 302 ページの図 135

式を使用した呼び出しの例については、次を参照してください。

- 11 ページの図 4
- 41 ページの図 19
- 89 ページの図 38
- 268 ページの図 122

プロトタイプ・パラメーターの受け渡し

プロトタイプ・パラメーターを渡すと、次のことが行われます。

- コンパイラーは、呼び出し側と呼び出される側の両方がプロトタイプを使用してコンパイルされるものとすれば、両方のコンパイル時にパラメーター定義が一致することを確認します。
- PARM 命令が必要ではないので、必要とされる仕様書が少なくなります。

この項では、プロトタイプ・パラメーターの定義時に使用可能な各種のオプション、および呼び出しインターフェースに対するこれらのオプションの影響について説明します。

パラメーターの受け渡しスタイル

システム API 呼び出しを含むプログラム呼び出しでは、パラメーターを参照によって渡すことが必要です。しかし、プロシージャー呼び出しにはそのような必要性はありません。ILE RPG では、次の 3 つの方法でプロトタイプ・パラメーターの受け渡しをできるようにします。

- 参照による
- 値による
- 読み取り専用参照による

プロトタイプでないパラメーターは、参照によってのみ渡すことができます。

参照による受け渡し

ILE RPG のデフォルトのパラメーター受け渡しスタイルは、参照によるものです。したがって、参照によってパラメーターを渡す場合には、パラメーター定義にキーワードをコーディングする必要はありません。呼び出される側が渡されたフィールドを変更する時には、参照によってパラメーターをプロシージャーに渡す必要があります。また、例えば大きな文字フィールドを渡す時などには実行時のパフォーマンスを向上させるため、参照によって渡すことが必要となることもあります。外部プログラム呼び出しで渡されるパラメーターは、参照によってしか渡すことができないことにも注意してください。

値による受け渡し

プロトタイプ・プロシージャーでは参照の代わりに、値によってパラメーターを渡すことができます。パラメーターが値によって渡されると、コンパイラーは実際の値を呼び出されたプロシージャーに渡します。

パラメーターを値によって渡した時には、呼び出されるプログラムまたはプロシージャーはパラメーターの値を変更できますが、呼び出し側は変更された値を見ることはできません。

値によってパラメーターを渡すためには、下の図のようにプロトタイプのパラメーター定義にキーワード VALUE を指定してください。

注: OS/400 プログラム呼び出しでは、そのパラメーターが参照によって渡されることが必要です。したがって、値によってパラメーターをプログラムに渡すことはできません。

読み取り専用参照による受け渡し

パラメーターをプロトタイプ・プロシージャーまたはプログラムに渡すもう 1 つの方法は、読み取り専用の参照によって渡すことです。参照によってパラメーターを

渡す必要があり、そのパラメーター値が呼び出し中に変更されないことが分っている場合には、読み取り専用参照による受け渡しが便利です。例えば、多くのシステム API は、形式または長さを指定する読み取り専用のパラメーターをもっています。

読み取り専用参照によるパラメーターの受け渡しには値による受け渡しと同じ利点があります。特に、この方法によってリテラルおよび式を渡すことができます。しかし、パラメーターが呼び出し中に変更されないことを承知しておくことが重要です。

パラメーターを読み取り専用参照により渡した時には、コンパイラーはパラメーターを一時フィールドにコピーしてその一時フィールドのアドレスを渡します。これを引き起こす条件の一部は、渡されたパラメーターが式であることか、渡されたパラメーターの形式が異なっている場合です。

注: 呼び出されるプログラムまたはプロシージャーが、読み取り専用の参照方式を使用する言語 (プロトタイプを使用する ILE RPG、または C のどちらか) のプロトタイプを使用してコンパイルされる場合には、パラメーターは変更されません。呼び出されるプログラムまたはプロシージャーがプロトタイプを使用しない場合には、コンパイラーはパラメーターが変更されないことを保証できません。この場合には、プロトタイプを定義する人がこのパラメーター受け渡し方法の指定時に注意しなければなりません。

読み取り専用参照でパラメーターを渡すためには、プロトタイプのパラメーター定義の定義仕様書でキーワード CONST を指定してください。 151 ページの図 65 は、ILE CEE API CEETSTA プロトタイプ定義 (省略引き数のテスト) 例です。

値または読み取り専用参照による受け渡しの利点

値または読み取り専用の参照の受け渡しでは、次のことが可能になります。

- リテラルおよび式をパラメーターとして渡すこと
- 必要なタイプおよび長さとは正確には一致しないパラメーターを渡すこと
- 呼び出し側の見通しで、変更されそうもない変数を渡すこと

値または読み取り専用参照による受け渡しを使用する主要な目的の 1 つは、渡されるパラメーターの属性の一致があまり厳格でなくてもよいことです。例えば、定義がパック 10 進数の数字フィールドのタイプで、小数点以下 2 桁の長さ 5 である場合には、数値を渡さなければなりませんが、これを次のようにすることができます。

- 任意の桁数および任意の小数点以下の桁数のパック、ゾーン、または 2 進数の定数または変数
- 数値を戻す組み込み関数
- 数値を戻すプロシージャー
- 次のような複合数字式

```
2 * (Min(Length(First) + Length(Last) + 1): %size(Name))
```

プロトタイプに 4 要素の配列が必要な場合には、渡されるパラメーターは次のようにすることができます。

- 4 要素より少ない要素の配列。この場合には、受け取ったパラメーターの残りの要素にはそのタイプのデフォルト値が入ります。
- 4 要素の配列。この場合には、受け取ったパラメーターの各要素が渡されたパラメーターの要素と対応します。
- 4 要素を超える配列。この場合には、渡される配列の一部の要素は受け取りパラメーターに渡されません。
- 非配列。この場合には、受け取ったパラメーターのそれぞれの要素に、渡されたパラメーター値が含まれます。

パラメーターの受け渡しスタイルの選択

既存のプログラムまたはプロシージャーを呼び出す場合は、参照による方法または値による方法のいずれかの、プロシージャーが予期する方法でパラメーターを渡す必要があります。パラメーターを参照により渡す必要があり、呼び出されたプロシージャー・プログラムまたはプロシージャーによってパラメーターが変更されない場合は、読み取り専用の参照によって渡します (CONST キーワードを使用)。値による受け渡しまたは読み取り専用参照による受け渡しを自由に選択できる場合、大量のパラメーターがある場合は読み取り専用の参照によって渡します。次の一般的なガイドラインに従ってください。

- パラメーターが数値またはポインターであり、**配列ではない場合**は、読み取り専用の参照、または値により渡します。これらのデータ・タイプを値により渡した場合、ごくわずかにパフォーマンスが向上します。
- それが不要の場合は、読み取り専用の参照により渡します。

```
     *-------------------------------------------------------------
     *  プロシージャーは 10 桁の整数値を戻します。
     *  3 つのパラメーターはすべて、VALUE で渡された 5 桁の整数です。
     *-------------------------------------------------------------
    D MyFunc          PR            10I 0 EXTPROC('DO_CALC')
    D                                5I 0 VALUE
    D                                5I 0 VALUE
    D                                5I 0 VALUE
    ....
```

図 63. VALUE パラメーターのプロシージャー DO_CALC のプロトタイプ

```
P DO_CALC         B                   EXPORT
 *-------------------------------------------------------------
 * このプロシージャーは、VALUE パラメーターとして渡された
 * 3 つの数値で機能を実行し、また値を戻します。
 *-------------------------------------------------------------
D DO_CALC         PI            10I 0
D    Term1                       5I 0 VALUE
D    Term2                       5I 0 VALUE
D    Term3                       5I 0 VALUE
D Result          S             10I 0
C                   EVAL      Result = Term1 ** 2 * 17
C                                    + Term2      * 7
C                                    + Term3
C                   RETURN    Result * 45 + 23
P                 E
```

図 64. DO_CALC プロシージャーのプロシージャー・インターフェース定義

```
 *--------------------------------------------------------------------
 * CEETSTA (省略引き数のテスト) -- ILE CEE API
 *     1. 存在フラグ                  出力   2 進 (4)
 *     2. 引き数番号                  入力   2 進 (4)
 *--------------------------------------------------------------------
D CEETSTA         PR                  EXTPROC('CEETSTA')
D   Present                   10I 0
D   ArgNum                    10I 0   CONST
D   Feedback                  12A     OPTIONS(*OMIT)
...
D HaveParm        S             10I 0
...
C                     CALLP     CEETSTA(HaveParm : 3 : *OMIT)
C                     IF        HaveParm = 1
 *        3 番目のパラメーターで何かを行なう
C                     ENDIF
```

図 65. CONST パラメーターの ILE CEE API CEETSTA のプロトタイプ

CEETSTA に渡される 2 番目のパラメーターは数字フィールド、リテラル、組み込み関数、または式とすることができます。

操作記述子の使用

時には、データ・タイプが呼び出されるプロシージャーに正確に分からない場合でも (例えば、異なるタイプのストリング)、パラメーターをプロシージャーに渡すことが必要な場合があります。こうした場合には、パラメーターの形式について呼び出されたプロシージャーに記述情報を提供するために、**操作記述子**を使用することができます。この追加情報によって、プロシージャーはストリングを適切に解釈することができます。呼び出されるプロシージャーに必要な場合にだけ、操作記述子を使用する必要があります。

多くの ILE バインド可能 API には、操作記述子が必要です。任意のパラメーターが「記述子による」と定義されている場合は、操作記述子を API に渡す必要があります。この例は、ILE CEE API CEEDATM (秒数を文字タイム・スタンプに変換) です。2 番目および 3 番目のパラメーターには操作記述子が必要です。

注: 現在、ILE RPG コンパイラーは文字および図形フィールド、それにサブフィールドの操作記述子しかサポートしません。操作記述子は、データ構造、配列、またはテーブルでは使用できません。さらに、操作記述子は数字タイプのデータ、日付、時刻、タイム・スタンプ、基底ポインター、またはプロシージャー・ポインターでは使用できません。

操作記述子は、渡されるパラメーターやパラメーターが渡される方法には影響しません。必要でない操作記述子がプロシージャーに渡された時には、単にその操作記述子が無視されるだけです。

プロトタイプと非プロトタイプの両方のパラメーターについて操作記述子を要求することができます。プロトタイプのパラメーターでは、プロトタイプ定義にキーワード OPDESC を指定します。非プロトタイプのパラメーターでは、CALLB 命令の命令コード拡張子として (D) を指定します。いずれの場合にも、操作記述子は呼び出しプロシージャーによって作成され、呼び出されたプロシージャーに隠れたパラメーターとして渡されます。操作記述子は省略されたパラメーターには作成されません。

ILE バインド可能 API の検索操作記述子 (CEEDOD)、およびストリング引き数に関する記述情報の取得 (CEESGI) を使って、操作記述子から情報を取り出すことができます。

操作記述子はバインド呼び出しでしか使用できないことに注意してください。さらに、非プロトタイプ呼び出しで 'D' 操作コード拡張子が CALL 命令に指定されると、エラー・メッセージがコンパイラーによって出されます。

図 66 は、キーワード OPDESC の例を示します。

```
     *---------------------------------------------------------
     *  Len が 10 桁の整数値を戻します。パラメーターは
     * 読み取り専用参照により渡されたストリングです。
     * Len がパラメーターの長さを知るために、
     * 操作記述子が必要です。
     * パラメーターを 32767 バイトよりも小さくするために
     * OPTIONS(*VARSIZE) が必要です。
     *---------------------------------------------------------
     D Len             PR            10I 0 OPDESC
     D                              32767A   OPTIONS(*VARSIZE) CONST
```

図 66. プロトタイプ・プロシージャーの操作記述子の要求

操作記述子の使用方法の例については、 104 ページの『サンプル・サービス・プログラム』を参照してください。この例は、渡される文字ストリングを 16 進等価値に変換するサービス・プログラムから構成されます。サービス・プログラムは操作記述子を使用して、文字ストリングの長さおよび変換する長さを決定します。

パラメーターの省略

プロシージャーの呼び出し時に、パラメーターを指定しないですませたい場合もあります。これは、パラメーターが呼び出されたプロシージャーと関連がないという場合です。例えば、この状況は ILE バインド可能 API を呼び出す時に起こること

があります。別の理由として、この特殊なパラメーターを処理しない旧プロシージャーを呼び出す場合があります。呼び出しでパラメーターを省略する必要がある場合には、次の 2 つの方法があります。

- OPTIONS(*OMIT) を指定して、*OMIT を渡す
- OPTIONS(*NOPASS) を指定して、パラメーターを渡さない

2 つの方法の主な違いは、パラメーターが省略されたかどうかを検査する方法と関係しています。どちらの場合も、省略されたパラメーターは呼び出されたプロシージャーにより参照することはできません。参照した場合には、結果は予測できません。そのため、呼び出されたプロシージャーが異なる数のパラメーターを処理するように設計されている場合には、渡されたパラメーターの数を検査する必要があります。*OMIT が渡された場合には、これはパラメーターとして「カウント」されます。

*OMIT の受け渡し

呼び出されたプロシージャーが、*OMIT が渡されるかもしれないことを知っている場合には、プロトタイプ・パラメーターに *OMIT を渡すことができます。言い換えると、プロトタイプの対応するパラメーター定義に、キーワード OPTIONS(*OMIT) を指定すると、*OMIT を渡すことができます。*OMIT を指定すると、コンパイラーが必要なコードを生成し、呼び出されたプロシージャーに対し、パラメーターが省略されたことを指示します。

注: *OMIT は参照によって渡されるパラメーターにしか指定できません。

*OMIT が ILE RPG プロシージャーに渡されたかどうかを判別するには、%ADDR 組み込み関数を使って問題のパラメーターのアドレスを検査します。 アドレスが *NULL の場合には、*OMIT が渡されています。また CEETSTA (省略された引き数の検査) バインド可能 API も使用することができます (簡略な例については、151 ページの図 65 を参照してください)。

次は *OMIT の簡単な使用法の例です。この例では、操作記述子を分解するために、プロシージャーが ILE バインド可能 API CEEDOD を呼び出します。CEEDOD API は 7 つのパラメーターを受け取ることを期待していますが、呼び出し元の発呼側プロシージャーには 6 つのパラメーターしか定義されていません。CEEDOD (およびほとんどのバインド可能 API) の最後のパラメーターは API がどのように終了したかを判別するのに使用できるフィードバック・コードです。しかし、呼び出しプロシージャーは、このフィードバック・コードではなく、例外を介して、エラー・メッセージを受け取るように設計されています。したがって、プロシージャーは CEEDOD への呼び出し時に、フィードバック・コードのパラメーターが省略されたことを指示しなければなりません。

*OMIT の使用例については、104 ページの『サンプル・サービス・プログラム』を参照してください。

パラメーターの省略

パラメーターを省略するもう 1 つの方法は、単に呼び出しでこれを省略するというやり方です。呼び出されたプロシージャーはこれを予定している必要があり、このことは、そのプロシージャーがプロトタイプで指示されなければならないことを意味します。プロトタイプのパラメーターを呼び出しで渡す必要がないことを指示す

るために、対応するパラメーター定義にキーワード OPTIONS(*NOPASS) を指定してください。最初の *NOPASS 以降のすべてのパラメーターにも OPTIONS(*NOPASS) を指定しなければならないことに注意してください。

*NOPASS と *OMIT の両方を同じパラメーターに任意の順序で、すなわち OPTIONS(*NOPASS:*OMIT) または OPTIONS(*OMIT:*NOPASS) のどちらでも指定することができます。

OPTIONS(*NOPASS) の例として、任意指定の 3 番目のパラメーターをもつシステム API QCMDEXC (コマンドの実行) を考えてください。このパラメーターが使えるようにするには、図 67 に示すように QCMDEXC のプロトタイプを書くことができます。

```
     *----------------------------------------------------------
     *  QCMDEXC のこのプロトタイプは次の 3 つのパラメーターを定義します。
     *   1- 必要な長さよりも短い
     *      文字フィールド
     *   2- 数値フィールド
     *   3- 任意指定の文字フィールド
     *----------------------------------------------------------
     D qcmdexc         PR                  EXTPGM('QCMDEXC')
     D   cmd                         3000A   OPTIONS(*VARSIZE) CONST
     D   cmdlen                        15P 5 CONST
     D                                  3A   CONST OPTIONS(*NOPASS)
```

図 67. 任意指定パラメーターのシステム API QCMDEXC のパラメーター

渡されるパラメーターの数の検査

呼び出しで渡されるパラメーター数の検査を必要とする場合があります。プロシージャーの作成方法により、この数によって、渡されないパラメーターを参照せずに済むことがあります。例えば、3 つのパラメーターを渡される時もあれば、4 つのパラメーターを渡される時もあるプロシージャーを作成するとします。このことは、新しいパラメーターが必要となった時に起こります。呼び出されるプロシージャーを書いて、組み込み関数 %PARMS によって戻される値に従ってどちらの数も処理することができます。新しい呼び出しでは、このパラメーターが渡されます。元からの呼び出しは変更されないままです。

%PARMS はパラメーターを使用しません。%PARMS によって戻される値には *OMIT が渡されるパラメーターも含まれます。*PARMS フィールドを使用するためには PSDS もコーディングしなければなりませんが、メイン・プロシージャーでは、%PARMS は PSDS の *PARMS フィールドに入っているのと同じ値を戻します。

*PARMS と %PARMS の両方で、渡されたパラメーターの数が分からない場合には、値 -1 が戻されます (渡されたパラメーターの数を判別するには、少なくとも操作記述子を渡さなければなりません。ILE RPG は常に呼び出しでこれを渡しますが、他の ILE 言語では渡さない場合があります)。メイン・プロシージャーが活動状態になっていない場合には、 *PARMS は信頼できません。サブプロシージャーから *PARMS を参照することは望ましくありません。

%PARMS の使用

この例では、会社の社員の住所情報の変更に使用するため、プロシージャー FMTADDR が数回変更されています。FMTADDR は 3 つの異なるプロシージャーによって呼び出されます。プロシージャーは、社員情報を処理するのに使用するパラメーターの数だけが異なります。すなわち、FMTADDR に対する新たな必要性が起こり、それらをサポートするために、新しいパラメーターが追加されました。しかし、FMTADDR を呼び出す旧プロシージャーはまだサポートされており、変更したりコンパイルし直したりする必要はありません。

社員の住所の変更は、次のように要約することができます。

- 最初は、すべての社員が同じ市に住んでいたので、必要なのは町名と番地のみでした。したがって、市と県にはデフォルト値を使用することができました。
- 後になって、会社が発展したので、一部の全社的アプリケーションにおいて市の情報が変わるようになりました。
- さらに発展して、県の情報も変わるようになりました。

プロシージャーは渡されたパラメーター数を基にして情報を処理します。数は 3 から 5 の間で変わることがあります。この数により、プログラムに市または県、あるいはその両方のデフォルト値を提供するかどうかが通知されます。 156 ページの図 68 は、このプロシージャーのソースを示したものです。 157 ページの図 69 は、/COPY メンバーがこのプロトタイプを含んでいる場合のソースです。

FMTADDR の主ロジックは、次のとおりです。

1. %PARMS を使用して、渡されたパラメーターの数を検査する。この組み込み関数は、渡されたパラメーターの数を戻します。
   - この数が 4 より大きい場合には、デフォルトの県が 5 番目のパラメーター P_Province によって提供される実際の県で置き換えられます。
   - この数が 3 より大きい場合には、デフォルトの市が 4 番目のパラメーター P_City によって提供される実際の市で置き換えられます。
2. サブルーチン GetStreet# を使用して、番地を印刷用に訂正する。
3. 完全な住所を連結する。
4. 戻る。

```
 *================================================================*
 * FMTADDR - 住所の形式設定
 *
 * インターフェース・パラメーター
 * 1. 住所          文字 (70)
 * 2. 番地          パック (5,0)
 * 3. 町名          文字 (20)
 * 4. 市区町村      文字 (15)   (呼び出し元によっては渡されない場合もある)
 * 5. 都道府県      文字 (15)   (呼び出し元によっては渡されない場合もある)
 *================================================================*
 * /COPY メンバーからプロトタイプを取り込みます
 /COPY  FMTADDRP
DFmtAddr          PI
D Address                       70
D Street#                        5  0 CONST
D Street                        20    CONST
D P_City                        15    OPTIONS(*NOPASS) CONST
D P_Province                    15    OPTIONS(*NOPASS) CONST
 *----------------------------------------------------------------*
 * 渡されない可能性のあるパラメーターのデフォルト値
 *----------------------------------------------------------------*
D City            S             15    INZ('Toronto')
D Province        S             15    INZ('Ontario')
 *----------------------------------------------------------------*
 * 都道府県 (Province) パラメーターが渡されたかどうかを検査します。
 * 渡された場合には、デフォルト値をパラメーター値で置き換えます。
 *----------------------------------------------------------------*
C                   IF        %PARMS > 4
C                   EVAL      Province = P_Province
C                   ENDIF
 *----------------------------------------------------------------*
 * 市区町村 (City) パラメーターが渡されたかどうかを検査します。
 * 渡された場合には、デフォルト値をパラメーター値で置き換えます。
 *----------------------------------------------------------------*
C                   IF        %PARMS > 3
C                   EVAL      City = P_City
C                   ENDIF
 *----------------------------------------------------------------*
 * 'CStreet#' を 'Street#' の文字形式に設定します。                *
 *----------------------------------------------------------------*
C                   EXSR      GetStreet#
 *----------------------------------------------------------------*
 * 住所を番地、町名、市区町村、都道府県の形式に設定します。        *
 *----------------------------------------------------------------*
C           EVAL     ADDRESS = %TRIMR(CSTREET#) + ' ' +
C                              %TRIMR(CITY) + ' ,' +
C                              %TRIMR(PROVINCE)
C                   RETURN
```

図 68. プロシージャー FMTADDR のソース (1/2)

```
     *====================================================================*
     * サブルーチン: GetStreet#
     * 番地の文字形式を入手し、左寄せして、右にブランクを                 *
     * 埋め込みます。                                                     *
     *====================================================================*
     C     GetStreet#    BEGSR
     C                   MOVEL     Street#       CStreet#         10
     *--------------------------------------------------------------------*
     * 最初の非ゼロを検出します。                                         *
     *--------------------------------------------------------------------*
     C     '0'           CHECK     CStreet#      Non0              5 0
     *--------------------------------------------------------------------*
     * 非ゼロがあった場合には、非ゼロで始まる番地のサブストリングを       *
     * 作成します。                                                       *
     *--------------------------------------------------------------------*
     C                   IF        Non0 > 0
     C                   SUBST(P)  CStreet#:Non0 CStreet#
     *--------------------------------------------------------------------*
     * 非ゼロがない場合には、番地として '0' だけを使用します。            *
     *--------------------------------------------------------------------*
     C                   ELSE
     C                   MOVEL(P)  '0'           CStreet#
     C                   ENDIF
     C                   ENDSR
```

図 68. プロシージャー FMTADDR のソース (2/2)

```
     *====================================================================*
     * FMTADDR のプロトタイプ - 住所の形式設定
     *====================================================================*
     DFmtAddr          PR
     D  addr                         70
     D  strno                         5  0 CONST
     D  st                           20    CONST
     D  cty                          15    OPTIONS(*NOPASS) CONST
     D  prov                         15    OPTIONS(*NOPASS) CONST
```

図 69. プロシージャー FMTADDR のプロトタイプでの /COPY メンバーのソース

158 ページの図 70 にはプロシージャー PRTADDR のソースを示してあります。このプロシージャーは、FMTADDR の使用を説明するために提供されます。便宜上、それぞれに FMTADDR を呼び出す 3 つのプロシージャーがこの 1 つのプロシージャーに結合されています。また、この例のデータはプログラム記述です。

PRTADDR は '3 つのプロシージャーを 1 つにまとめている' ので、異なる 3 つの住所データ構造を定義しなければなりません。同様に、演算仕様書に 3 つの部分があり、それぞれが各段階でプログラムと対応しています。住所の印刷後、プロシージャー PRTADDR は終了します。

```
 *===============================================================*
 * PRTADDR - 住所の印刷
 *           住所を形式設定するために FmtAddr を呼び出す
 *===============================================================*
FQSYSPRT   O    F   80        PRINTER
 *---------------------------------------------------------------*
 * FmtAddr のプロトタイプ
 *---------------------------------------------------------------*
DFmtAddr          PR
D  addr                         70
D  strno                         5  0
D  st                           20
D  cty                          15    OPTIONS(*NOPASS)
D  prov                         15    OPTIONS(*NOPASS)
DAddress          S             70
 *---------------------------------------------------------------*
 * ステージ 1: 元の住所のデータ構造。
 * 町名および番地だけが可変情報。
 *---------------------------------------------------------------*
D Stage1          DS
D   Street#1                     5P 0 DIM(2) CTDATA
D   StreetNam1                  20    DIM(2) ALT(Street#1)
 *---------------------------------------------------------------*
 * ステージ 2: 市区情報を可変とした
 * 改訂住所データ構造。
 *---------------------------------------------------------------*
D Stage2          DS
D   Street#2                     5P 0 DIM(2) CTDATA
D   Addr2                       35    DIM(2) ALT(Street#2)
D     StreetNam2                20    OVERLAY(Addr2:1)
D     City2                     15    OVERLAY(Addr2:21)
 *---------------------------------------------------------------*
 * ステージ 3: 都道府県情報を可変とした
 *  改訂住所データ構造。
 *---------------------------------------------------------------*
D Stage3          DS
D   Street#3                     5P 0 DIM(2) CTDATA
D   Addr3                       50    DIM(2) ALT(Street#3)
D     StreetNam3                20    OVERLAY(Addr3:1)
D     City3                     15    OVERLAY(Addr3:21)
D     Province3                 15    OVERLAY(Addr3:36)
 *---------------------------------------------------------------*
 * 'プログラム 1'- 市区パラメーターの追加前の FMTADDR の使用。
 *---------------------------------------------------------------*
C                   DO        2             X                 5 0
C                   CALLP     FMTADDR (Address:Street#1(X):StreetNam1(X))
C                   EXCEPT
C                   ENDDO
```

図 70. プロシージャー PRTADDR のソース (1/2)

```
     *---------------------------------------------------------------*
     * 'プログラム 2'- 都道府県パラメーター追加前の FMTADDR の使用。   *
     *---------------------------------------------------------------*
    C                   DO        2             X                 5 0
    C                   CALLP     FMTADDR (Address:Street#2(X):
    C                             StreetNam2(X):City2(X))
    C                   EXCEPT
    C                   ENDDO
     *---------------------------------------------------------------*
     * 'プログラム 3' - 都道府県パラメーター追加後の FMTADDR の使用。  *
     *---------------------------------------------------------------*
    C                   DO        2             X                 5 0
    C                   CALLP     FMTADDR (Address:Street#3(X):
    C                             StreetNam3(X):City3(X):Province3(X))
    C                   EXCEPT
    C                   ENDDO
    C                   SETON                                       LR
     *---------------------------------------------------------------*
     * 住所の印刷。                                                  *
     *---------------------------------------------------------------*
    OQSYSPRT   E
    O                       Address
**
00123Bumble Bee Drive
01243Hummingbird Lane
**
00003Cowslip Street     Toronto
01150Eglinton Avenue    North York
**
00012Jasper Avenue      Edmonton      Alberta
00027Avenue Road        Sudbury       Ontario
```

図 70. プロシージャー PRTADDR のソース (2/2)

これらのプログラムを作成するためには、次のステップに従ってください。

1. 156 ページの図 68 のソースを使って FMTADDR を作るには、以下のように入力します。

   ```
   CRTRPGMOD MODULE(MYLIB/FMTADDR)
   ```

2. PRTADDR を作成するために、158 ページの図 70 のソースを使用して、次のように入力してください。

   ```
   CRTRPGMOD MODULE(MYLIB/PRTADDR)
   ```

3. プログラム PRTADDR を作成するために、次のように入力してください。

   ```
   CRTPGM PGM(MYLIB/PRTADDR) MODULE(PRTADDR FMTADDR)
   ```

4. PRTADDR を呼び出します。出力を以下に示します。

   ```
   123 Bumble Bee Drive, Toronto, Ontario
   1243 Hummingbird Lane, Toronto, Ontario
   3 Cowslip Street, Toronto, Ontario
   1150 Eglinton Avenue, North York, Ontario
   12 Jasper Avenue, Edmonton, Alberta
   27 Avenue Road, Sudbury, Ontario
   ```

必要なデータより少ないデータの受け渡し

パラメーターがプロトタイプの場合には、コンパイラーは長さがパラメーターに適切かどうかを調べます。呼び出し先が、パラメーターが最大長よりも短くても構わないことを (文書またはプロトタイプによって) 示した場合には、安全に短いパラメーターを渡すことができます。

図 71 は QCMDEXC のプロトタイプを示しますが、ここで、最初のパラメーターは OPTIONS(*VARSIZE) を使って定義することができ、これは、最初のパラメーターに異なる長さのパラメーターを渡すことができることを意味します。OPTIONS *VARSIZE が文字フィールド、UCS-2 フィールド、図形フィールド、または配列にしか指定できないことに注意してください。

```
     *----------------------------------------------------------
     *  QCMDEXC のこのプロトタイプは 3 つのパラメーターを定義します。
     *  最初のパラメーターは *VARSIZE を指定して定義されるので、
     *  さまざまな長さの文字フィールドを渡すことができます。
     *----------------------------------------------------------
    D qcmdexc         PR                  EXTPGM('QCMDEXC')
    D   cmd                         3000A   OPTIONS(*VARSIZE) CONST
    D   cmdlen                        15P 5 CONST
    D                                  3A   CONST OPTIONS(*NOPASS)
```

*図 71. *VARSIZE パラメーターのシステム API QCMDEXC のプロトタイプ*

評価の順序

プロトタイプの呼び出しのパラメーターの評価順序は保証されていません。結果が期待したものと異なることがあるので、副次作用を起こすパラメーターを使用する時には、このことが重要になります。

副次作用はパラメーターの処理が次のものを変更した場合に起こります。

- 参照パラメーターの値
- グローバル変数の値
- ファイルまたはデータ域などの外部オブジェクト

副次作用が起こり、そのパラメーターがパラメーター・リストのどこか他の場所で使用された場合には、リストのある個所のパラメーターに使用された値が、別の個所で使用された値と同じでないことがあります。例えば、次の呼び出しステートメントを考えてみてください。

```
CALLP         procA (fld : procB(fld) : fld)
```

procA がすべての値のパラメーターをもち、procB が参照パラメーターをもっているとしましょう。また、*fld* は値 3 で開始し、 procB が *fld* を 5 に変更し、10 を戻すとしましょう。パラメーターが評価される順序によって、procA は 3、10、および 5 のいずれか、または可能性として 3、10、および 3 のいずれかを受けます。また可能性として 5、10、および 3 があり、5、10、および 5 という場合さえあります。

簡単に言うと、副次作用が起こる可能性に注意することが大切です。特に、エンド・ユーザーがプロシージャーの一部の詳細を知らないような、サード・パーティー用のプログラムを提供する場合には、渡されたパラメーターの値が予定の値であることを確認することが重要です。

言語間呼び出し

別の言語で作成されたプログラムまたはプロシージャーでデータを受け渡しする時には、その別の言語が ILE RPG と同じパラメーターの受け渡し方法および同じデータ・タイプをサポートしているかどうかを知ることが重要です。表 26 は、ILE RPG が許可する異なるパラメーターの受け渡し方法を示し、また適用できる場合は、ILE 言語以外の言語でコーディングする方法を示します。この表には、比較のために OPM RPG/400® 用コンパイラーも含まれています。

表 26. RPG パラメーター受け渡し方式

参照による受け渡し	

ILE RPG - プロトタイプ	`D  proc      PR`
	`D    parm            1A`
	`C        CALLP  proc(fld)`

ILE C	`void proc(char *parm);` `proc(&fld);`
ILE COBOL	`CALL PROCEDURE "PROC" USING BY REFERENCE PARM`
RPG - 非プロトタイプ	`C        CALL   'PROC'` `C        PARM                   FLD`
ILE CL	`CALL PROC (&FLD)`
値による受け渡し	

ILE RPG - プロトタイプ	`D  proc      PR`
	`D    parm            1A   VALUE`
	`C        CALLP  proc('a')`

ILE C	`void proc(char parm);` `proc('a');`
ILE COBOL	`CALL PROCEDURE "PROC" USING BY VALUE PARM`
RPG - 非プロトタイプ	N/A
ILE CL	N/A
読み取り専用参照による受け渡し	

ILE RPG - プロトタイプ	`D  proc      PR`
	`D    parm            1A     CONST`
	`C        CALLP  proc(fld)`

ILE C	`void proc(const char *parm);` `proc(&fld);`
ILE COBOL	N/A[1]
RPG - 非プロトタイプ	N/A
ILE CL	N/A

注:

1. 読み取り専用参照による受け渡しを COBOL の内容による受け渡しと混同しないでください。RPG では、Fld1 を内容によって受け渡しするためには、次のとおりコーディングします。

   ```
   C                   PARM      Fld1          TEMP
   ```

   Fld1 は変更できないようになっていますが、TEMP は保護されていません。パラメーターが変更されない保証はありません。

異なる HLL によりサポートされるデータ・タイプの情報については、適切な言語解説書を参照してください。

言語間呼び出しに関する考慮事項

1. RPG プロシージャーが ILE CL プロシージャーと正しくやりとりを行なうことを確認するには、ILE CL プロシージャーのプロトタイプにおいて、または ILE CL プロシージャーによって呼び出される RPG プロシージャーにおいて、`EXTPROC(*CL:'procedurename')` とコーディングします。
2. RPG プロシージャーが ILE C プロシージャーと正しくやりとりを行なうことを確認するには、ILE C プロシージャーのプロトタイプにおいて、または ILE C プロシージャーによって呼び出される RPG プロシージャーにおいて、`EXTPROC(*CWIDEN:'procedurename')` または `EXTPROC(*CNOWIDEN:'procedurename')` とコーディングします。ILE C ソースがそのプロシージャーに対して `#pragma argument(procedure-name,nowiden)` を含んでいる場合は *CNOWIDEN を使用し、そうでない場合は *CWIDEN を使用します。
3. どの ILE 言語からも RPG プロシージャーを正常に使用したいならば、EXTPROC キーワードに特殊な値を指定してはなりません。代わりに、値または戻り値によって渡されるパラメーターには以下のタイプは避けます。
   - 長さ 1 の文字 (1A または 1N)
   - 長さ 1 の UCS-2 (1C)
   - 長さ 1 のグラフィック (1G)
   - 4 バイトの浮動 (4F)
   - 1 バイトまたは 2 バイトの整数または符号なし (3I、3U、5、または 5U)
4. ILE C およびその他の言語を使用して、テラバイト・スペース・メモリーへのポインターを宣言することができます。ILE C では、このタイプの記憶域をアドレス指定するのに特別なコンパイル時オプションが必要ですが、ILE RPG は、V4R4M0 またはそれ以降のターゲット・リリースでコンパイルされれば、必ずこの記憶域をアドレス指定できます。テラバイト・スペース内のポインターの詳細については、「*ILE 概念*, SD88-5033-06」を参照してください。

固定形式の呼び出し命令の使用

CALL (プログラム呼び出し) 命令を使ってプログラム呼び出しを行い、CALLB (バインド・プロシージャーの呼び出し) 命令を使って、プロトタイプされていないプログラムまたはプロシージャーに対するプロシージャー呼び出しを行います。この 2 つの呼び出し命令は、構文と使用法がよく似ています。プログラムまたはプロシージャーを呼び出すためには、次の概略ステップに従ってください。

1. 呼び出すオブジェクトを演算項目 2 で識別する。
2. オプションで、エラー標識 (73 ～ 74 桁目) または LR 標識 (75 ～ 76 桁目)、あるいはその両方をコーディングする。

   呼び出されたオブジェクトがエラーで終了した場合には、指定されればエラー標識がオンに設定されます。同様に、呼び出されたオブジェクトが LR で戻った場合には、指定されれば LR 標識がオンに設定されます。

3. 呼び出されたオブジェクトにパラメーターを渡すために、呼び出し命令の結果フィールドに PLIST を指定するか、あるいは呼び出し命令の直後に PARM 命令を置く。

どちらの命令も呼び出し元のオブジェクトから呼び出されたオブジェクトに制御権を転送します。呼び出されたオブジェクトの実行が終了すると制御権は、呼び出し元のプログラムまたはプロシージャー内の、呼び出し命令の後にある最初の実行可能な命令に戻されます。

次の考慮事項は、どちらの呼び出し命令にも適用されます。

- 演算項目 2 は、変数、リテラルまたは名前付き固定情報とすることができます。この項目は大文字小文字の区別があることに注意してください。

  CALL の場合のみ: 演算項目 2 はライブラリー名/プログラム名、例えば、MYLIB/PGM1 になります。ライブラリー名が指定されなかった場合には、プログラムを検索するためにライブラリー・リストが使用されます。呼び出されるプログラムの名前は、実行時に演算項目 2 で文字変数を指定することによって提供することができます。

  CALLB の場合のみ: プロシージャー・ポインター呼び出しを行うためには、呼び出されるプロシージャーのアドレスを含むプロシージャー・ポインターの名前を指定します。
- 1 つのプロシージャーに、同じオブジェクトの呼び出しを複数入れることができます。その際、PLIST は同じものを指定しても異なるものを指定してもかまいません。
- ILE RPG プロシージャー (プログラム入力プロシージャーを含む) が最初に呼び出されると、そのフィールドが初期化され、そのプロシージャーに制御が与えられます。前の呼び出しで終了しなかった場合、同じプロシージャーの後続の呼び出しでは、呼び出されるプロシージャーのすべてのフィールド、標識、およびファイルが、前の呼び出しで戻されたものと同じです。
- システムは RPG プロシージャー内で呼び出されるすべてのプログラムの名前を記録します。RPG プロシージャーがプログラム (*PGM) の中にバインドされる場合には、どのプロシージャーまたはモジュールが呼び出しを行っているのか分からなくても、DSPPGMREF を使用してこれらの名前を照会することができます。

  変数を使用してプログラムを呼び出す場合には、 *VARIABLE という名前 (ライブラリー名ではない) の項目を参照します。

  モジュールでは、DSPMOD DETAIL(*IMPORT) を使用して呼び出されたプロシージャーの名前を参照することができます。このリストの一部のプロシージャーはシステム・プロシージャーです。すなわち、これらの名前は通常、下線で始まるかまたはブランクが入っており、あまりこれらを気にかける必要はありません。
- **CALLB の場合のみ:** コンパイラーは、CALLB 命令で渡されたパラメーターの数を示す操作記述子を作成し、この値を、呼び出されたプロシージャーのプログラム状況データ構造の *PARMS フィールドに入れます。この数には、省略パラメーターとして指定された (PARM 命令の *OMIT) パラメーターも含まれます。

CALLB 命令で (D) 命令拡張が使用された場合には、コンパイラーは、各文字フィールド、図形フィールド、およびサブフィールドについても操作記述子を作成します。

操作記述子の詳細については、 151 ページの『操作記述子の使用』を参照してください。

- CALL または CALLB 命令コードを使用する時に適用される制約事項がさらにあります。これらの制約事項の詳しい説明は、「*WebSphere Development Studio: ILE RPG 解説書*」を参照してください。

CALL および CALLB の例

CALL 命令の使用例については、次の項を参照してください。

- RPG プログラムの呼び出し例については、 267 ページの『デバッグ用サンプル・ソースの例』

CALLB 命令の使用例については、次の項を参照してください。

- サービス・プログラム内のプロシージャーの呼び出し例については、 109 ページの図 45
- バインド可能 API の呼び出し例については、 134 ページの図 56
- 各種の RPG プロシージャーを呼び出すメイン照会プログラムの例は、 414 ページの『CUSMAIN: RPG ソース』

PARM および PLIST を使用したパラメーターの受け渡し

固定形式呼び出しを使用してパラメーターを渡す場合には、 PARM および PLIST 命令を使用してパラメーターを渡さなければなりません。すべてパラメーターは参照によって渡されます。操作記述子の受け渡しを指定することができ、またパラメーターの省略も指示できます。

PARM 命令の使用

PARM 命令はプロシージャーで受け渡しが行われるパラメーターを識別するために使用されます。各パラメーターは別々の PARM 命令で定義されます。結果のフィールドにパラメーターの名前を指定します。すなわち、この名前は呼び出す / 呼び出されるプロシージャーにおける名前と同じである必要はありません。

演算項目 1 および演算項目 2 は任意指定で、変数またはリテラルを指示します。これらの値は、これらの項目が呼び出すプログラム/プロシージャーか、あるいは呼び出されるプログラム/プロシージャーにあるかによって、結果フィールドに渡されたり、あるいは結果フィールドから受け取られます。表 27 は、演算項目 1 と演算項目 2 が使われる方法を示します。

表 27. PARM 命令の演算項目 1 および演算項目 2 の意味

状況	演算項目 1	演算項目 2
呼び出し元 プロシージャー	戻り時に結果フィールドから転送される値。	呼び出し時に結果フィールドに入れられる値。
呼び出される プロシージャー	呼び出し時に結果フィールドから転送される値。	戻り時に結果フィールドに入れられる値。

注: 演算項目 1 または結果フィールドへの転送が行われるのは、呼び出されたプロシージャーが正常に呼び出し元に戻った時だけです。データを転送しようとした時にエラーが起こった場合には、転送は完了しません。

プロシージャーの呼び出し時に十分なパラメーターが指定されていない場合には、未解決のパラメーターが呼び出されたプロシージャーによって使用された時にエラーが起こります。エラーを避けるためには次のどちらかを行うことができます。

- 渡されたパラメーターの数を調べるために %PARMS を検査する。%PARMS の使用例については、154 ページの『渡されるパラメーターの数の検査』を参照してください。
- 渡されないパラメーターの PARM 命令の結果フィールドに *OMIT を指定してください。呼び出されたプロシージャーは、%ADDR (パラメーター) = *NULL を使用して、パラメーターが値 *NULL をもっているかどうかを調べることによってパラメーターが省略されたかどうかを調べることができます。詳細については、152 ページの『パラメーターの省略』を参照してください。

PARM 命令を指定する時には、次の点に留意してください。

- 1 つまたは複数の PARM 命令を PLIST 命令の直後に指定しなければなりません。
- 1 つまたは複数の PARM 命令を CALL または CALLB 命令の直後に指定することができます。
- PARM 命令の結果フィールドに複数回繰り返しデータ構造が指定された時には、そのデータ構造のすべての繰り返しが単一のフィールドとして渡されます。
- PARM 命令の演算項目 1 と結果フィールドには、リテラル、先読みフィールド、名前付き固定情報、またはユーザー日付予約語を入れることはできません。
- 次の規則が非プロトタイプ・パラメーターの *OMIT に適用されます。
  - *OMIT は、CALLB 命令の直後にある PARM 命令か、CALLB で使用される PLIST の中でしか使用することができない。
  - *OMIT が指定された場合には、PARM 命令の演算項目 1 および演算項目 2 はブランクでなければならない。
  - *ENTRY PLIST の一部である PARM 命令に *OMIT を使用することはできない。
- 上記のほかにも PARM 命令コードの使用時に適用される制約事項があります。これらの制約事項の詳しい説明は、「*WebSphere Development Studio: ILE RPG 解説書*」を参照してください。

PARM 命令の例については、次の図を参照してください。

- 115 ページの図 47
- 105 ページの図 42
- 299 ページの図 134

PLIST 命令の使用

PLIST 命令は、次のことを行います。

- 名前を定義する。この名前によって、パラメーターのリストを参照することができます。このパラメーターのリストは、PLIST 命令の直後に置く PARM 命令によって指定されます。

- 入力パラメーター・リスト (*ENTRY PLIST) を定義する。

PLIST 命令の演算項目 1 には PLIST 名が入っていなければなりません。この名前は 1 つ以上の呼び出し命令の結果フィールドで指定することができます。パラメーター・リストが呼び出されたプロシージャーの入り口パラメーター・リストの場合には、演算項目 1 に *ENTRY が入っていなければなりません。

プロシージャーに複数の PLIST を指定することができます。しかし、*ENTRY PLIST は 1 つだけで、メイン・プロシージャーにだけ指定することができます。

PLIST 命令の例については、115 ページの図 47 と 299 ページの図 134 を参照してください。

呼び出されたプログラムまたはプロシージャーからの戻り

プログラムまたはプロシージャーが戻る場合には、その呼び出しスタック項目が呼び出しスタックから除去されます (それがプログラムの場合、プログラム入力プロシージャーも除去されます)。プロシージャーの外側でなにかの理由によってその呼び出しが終了した時には、プロシージャーが異常終了します。例えば、これは ILE RPG プロシージャー X が、X を呼び出すプロシージャーに対して直接、エスケープ・メッセージを出す (CL プロシージャーのような) 別のプロシージャーを呼び出す場合に起こります。これはまた、プロシージャーが呼び出しスタックのさらに上の方にあるプロシージャーの例外処理プログラム (*PSSR またはエラー標識) によって処理される例外を受け取ったときにも起こります。

メイン・プロシージャーと関連したサイクル・コードのために、これらの戻りも特定の終了ルーチンと関連しています。この項では、メイン・プロシージャーとサブプロシージャーを戻すいろいろな方法、およびそれぞれの場合に起こることについて説明します。

メイン・プロシージャーからの戻り

メイン・プロシージャーから戻ると、次のことが起こります。

- LR がオンの場合にはファイルがクローズされ、他の資源は解放される。
- プロシージャーの呼び出しスタック項目が呼び出しスタックから除去される。
- プロシージャーがプログラム入力プロシージャーで呼び出された場合には、そのプログラム入力プロシージャーも呼び出しスタックから除去される。

メイン・プロシージャーは、次の 1 つの方法で呼び出しプロシージャーに制御権を戻します。

- 正常終了
- 異常終了
- 未終了

呼び出されたメイン・プロシージャーから戻る方法について次に説明します。

LR、H1 ～ H9、および RT 標識が RPG プログラム・サイクルのどこでテストされるかについての詳細は、「*WebSphere Development Studio: ILE RPG 解説書*」の RPG プログラム・サイクルのセクションを参照してください。

正常終了

LR 標識がオンで、H1 ～ H9 標識がオンでない時には、メイン・プロシージャーが正常に終了し、制御権が呼び出し元プロシージャーに戻されます。LR 標識は、次のようにしてオンに設定することができます。

- RPG プログラム・サイクルでプライマリーまたはセカンダリー・ファイルからの最終レコードが処理される時は、暗黙的に。
- ユーザーが LR をオンに設定する時は、明示的に。

次の場合にも、メイン・プロシージャーは正常に終了します。

- RETURN 命令が (ブランクの演算項目 2 で) 処理され、 H1 ～ H9 標識がオンではなく、LR 標識がオンである。
- RT 標識がオンで、H1 ～ H9 標識がオンでなく、LR 標識がオンである。

メイン・プロシージャーが正常に終了すると、次のことが起こります。

- *ENTRY PARM 命令の演算項目 2 から結果フィールドへの転送が実行される。
- 定義仕様書に「受け入れファイル名」が指定されているすべての配列およびテーブルと、ロックされているすべてのデータ域データ構造が書き出される。
- このプロシージャーによってロックされているデータ域はすべてアンロックされる。
- オープンされているすべてのファイルがクローズされる。
- このプロシージャーが正常に終了したことを呼び出し元に示すように戻りコードが設定され、その後呼び出し元に制御権が戻される。

メイン・プロシージャーへの次回の呼び出しでは、エクスポートされた変数は別として、処理用に新しいコピーが使用可能になります (エクスポートされた変数は 1 回だけ初期設定されますが、これはプログラムが活動化グループで最初に活動化される時です。たとえ LR が前の呼び出しでオンであっても、エクスポートされた変数は新しい呼び出し時には最後に割り当てられた値のままです。それらを再初期化したい場合は、手操作でリセットしなければなりません)。

ヒント

通常は LR オンで終了して記憶域を解放するところを、指定された (持続する) 活動化グループで実行している場合には、終了しないで戻ることを考慮したいことがあります。理由は次のとおりです。

- 活動化グループが終了するまで記憶域は解放されないので、LR オンで終了しても、記憶域に関して少しも利点がない。
- 呼び出すたびにプログラムを初期設定し直すことを避ければ、呼び出しのパフォーマンスが向上する。

これを行うのは、プログラムをその都度初期設定し直す必要がない場合のみにしてください。

異常終了

次のいずれかが起こると、メイン・プロシージャーが異常終了し、制御権が呼び出し元プロシージャーに戻されます。

- ILE RPG 照会メッセージが出されて取り消しオプションが取られた場合。
- *PSSR または INFSR エラー処理サブルーチンの ENDSR *CANCL 命令が処理された場合 (*PSSR および INFSR エラー処理サブルーチンの *CANCL 戻り点の詳細については、296 ページの『ENDSR 命令での戻り点の指定』を参照してください)。
- RETURN 命令 (ブランクの演算項目 2 で) が処理された時に、H1 ～ H9 標識がオンになっている場合。
- RPG サイクルで最終レコード (LR) 処理が行われた時に、H1 ～ H9 標識がオンである場合。

メイン・プロシージャーが異常終了した時には、次のことが起こります。

- オープンされているすべてのファイルがクローズされる。
- このプロシージャーによってロックされているデータ域はすべてアンロックされる。
- 照会メッセージに対する取り消し応答でメイン・プロシージャーが終了した場合には、異常終了の原因となるのは機能チェックである。この場合には、機能チェックが呼び出し元に渡されます。'*CANCL' で終了するエラー処理サブルーチンのために終了した場合には、呼び出し元にエスケープ・メッセージが直接出されます。そうでない場合には、どのような例外が異常終了を起こしたかを呼び出し元が調べることになります。

このプロシージャーを次回に呼び出す時には、最新コピーを処理に使用することができます (例外処理プログラムについて詳しくは、283 ページの『RPG 特有の処理プログラムの使用』を参照してください)。

未終了の戻り

LR 標識または H1 ～ H9 標識のどれもがオンでない時に、次のいずれかが起こると、メイン・プロシージャーが未終了で、制御権を呼び出しプロシージャーに戻すことができます。

- RETURN 命令 (ブランクの演算項目 2 で) が処理された時。
- RT 標識がオンで制御が RPG サイクルの *GETIN 部分に達した時。この場合には制御権がただちに呼び出し元プロシージャーに戻されます (RT 標識について詳しくは、「*WebSphere Development Studio: ILE RPG 解説書*」を参照してください)。

メイン・プロシージャーを呼び出し、それが未終了で戻った場合には、プロシージャーを再び呼び出した時に、このプロシージャーのすべてのフィールド、標識、およびファイルがプロシージャーを終了した時と同じ値をもっています。しかし、次のような 3 つの例外があります。

- プログラムが *NEW 活動化グループ内で実行している場合には、プログラムが戻った時に活動化グループが削除されるので、これは当てはまらない。その場合には、次回にプログラムを呼び出した時には、LR オンで終了した場合と同じになります。
- ファイルを共用している場合には、ファイルの状態は、プロシージャーを出た時の状態と異なることがある。
- 同じモジュール内の別のプロシージャー間で呼び出された場合には、結果は予測できない。

LR 標識と H1 ～ H9 標識を組み合わせて、RETURN 命令 (ブランクの演算項目 2 で) または RT 標識を使用することができます。その場合に RPG プログラム・サイクル内での、RETURN 命令、RT 標識、および H1 ～ H9 標識に対するテスト順序に注意してください。LR 標識、または停止標識のどれかがオンで、次の条件のどちらかが当てはまる場合には、戻りによって終了します。

- RETURN 命令が実行された
- RT により未終了の戻りが起こった

サブプロシージャーからの戻り

RETURN 命令が正常に実行された時、あるいはプロシージャーの最後のステートメント (RETURN 命令ではない) が処理された時には、**サブプロシージャーは正常に戻ります。**しかし、呼び出しスタックからのサブプロシージャーの除去以外には、プログラムのメイン・プロシージャーが終了するまで、終了処置は実行されません。言い換えると、メイン・プロシージャーの正常終了時に実行される項目としてリストされているすべての処置は、メイン・プロシージャーの場合に のみ 実行されます。

処理できない例外が起こった場合には、**サブプロシージャーが異常終了**し、制御権は呼び出し元プロシージャーに戻されます。ここでも、メイン・プロシージャーが終了するまでこれ以上の処理は起こりません。

メイン・プロシージャーがまったく呼び出されない (したがって、終了できない) 場合には、ファイルや、データ域などはクローズされません。あるサブプロシージャーでこれが起こるかもしれないと考えられる場合には、サブプロシージャーの終了時に呼び出す終了プロシージャーをコーディングする必要があります。このことは、そのサブプロシージャーが制御仕様書に NOMAIN の指定されたモジュール内にある場合に特に言えます。

ILE バインド可能 API を使う戻り

ILE バインド可能 API CEETREC を使ってプロシージャーを正常に終了することができます。 しかし、この API は、同じ活動化グループにあるすべての 呼び出しスタック項目を制御の境界まで終了させます。プロシージャーが CEETREC を使用して終了した場合には、メイン・プロシージャーおよびサブプロシージャー用に上で説明した*正常な* 終了処理に従います。このプロシージャーを次回に呼び出す時には、最新コピーを処理に使用することができます。

同様に、ILE バインド可能 API CEE4ABN を使ってプロシージャーを異常終了させることができます。この場合には、プロシージャーは上述の異常終了処理に従います。

注: DFTACTGRP(*YES) によって作成したプログラム中では、プロシージャー内でプロシージャー呼び出しを使用できないので、これらの API のどちらも使用することはできません。

メイン・プロシージャーが活動状態でない場合、あるいはメイン・プロシージャーがない場合には、何もクローズまたは解放されないことに注意してください。この場合は CEERTX を使って、ILE 取り消し処理プログラムを使用可能にする必要が

あります。取り消し処理プログラムが同一のモジュールにある場合には、このプログラムはファイルをクローズし、データ域をアンロックし、その他の終了処置を実行することができます。

CEETREC および CEE4ABN についての詳細は、Web サイト http://www.ibm.com/eserver/iseries/infocenter で **iSeries 400 Information Center** の中の『プログラミング』カテゴリーの『*CL および API*』の節を参照してください。

バインド可能 API の使用

バインド可能アプリケーション・プログラミング・インターフェース (API) は、すべての ILE 言語に使うことができます。場合によっては、特定の ILE 言語が提供する以上の追加機能を提供します。これらはまた、HLL に依存しないので、混合言語のアプリケーションで有用です。

バインド可能 API は、次のような広範囲にわたる機能を提供します。

- 活動化グループおよび制御流れ管理
- 記憶域管理
- 条件管理
- サービス・メッセージ
- ソース・デバッガー
- 数学関数
- 呼び出し管理
- 操作記述子アクセス

ILE RPG が使うのと同じ呼び出しメカニズムを使って ILE バインド可能 API にアクセスし、プロシージャー、すなわち CALLP 命令や CALLB 命令を呼び出します。 API が値を戻し、ユーザーがそれを使用したい場合には、式の中で API を呼び出してください。API のプロトタイプを定義するために必要となる情報については、Web サイト http://www.ibm.com/eserver/iseries/infocenter で **iSeries 400 Information Center** の中の『プログラミング』カテゴリーの『*CL および API*』の節を参照してください。図 72 はバインド可能 API の「呼び出し」例です。

```
D CEExxxx         PR                  EXTPROC('CEExxxx')
D   parm1   ...
D   ...
C                   CALLP   CEExxxx(parm1 : parm2 : ... :
                                    parmn : feedback)
 or
C                   CALLB   'CEExxxx'
C                   PARM                    parm1
C                   PARM                    parm2
                    ...
C                   PARM                    parmn
C                   PARM                    feedback
```

図 72. ILE バインド可能 API のサンプル呼び出し構文

ここで、

- CEExxxx はバインド可能 API の名前です。

- parm1、parm2、... parm*n* は、呼び出される API との間で受け渡しされる省略可能パラメーターまたは必須パラメーターです。
- feedback は、バインド可能 API の結果を示す、省略可能なフィードバック・コードです。

注: バインド可能 API は CRTBNDRPG コマンドに DFTACTGRP(*YES) を指定した場合には、使用できません。

バインド可能 API についての詳細は、Web サイト http://www.ibm.com/eserver/iseries/infocenter で **iSeries 400 Information Center** の中の『*プログラミング*』カテゴリーの『*CL および API*』の節を参照してください。

バインド可能 API の使用例

バインド可能 API の使用例については、次を参照してください。

- 104 ページの『サンプル・サービス・プログラム』(CEEDOD の使用例)
- 131 ページの『ILE バインド可能 API を使用したユーザー独自のヒープ管理』. (CEEGTST、CEEFRST、および CEECZST の使用例)
- 297 ページの『条件処理プログラムの使用』(CEEHDLR および CEEHDLU の使用例)
- 304 ページの『取り消し処理プログラムの使用』(CEERTX および CEEUTX の使用例)

グラフィックス・ルーチンの呼び出し

ILE RPG は CALL や CALLP 命令の使用をサポートして OS/400 グラフィックスを呼び出しますが、これには図形データ表示管理プログラム (GDDM®、描画用のグラフィック要素セット)、およびグラフィック表示ルーチン (ビジネス図表ルーチンのセット) が含まれています。演算項目 2 にはリテラルまたは名前付き固定情報 'GDDM' (変数ではない) を入れなければなりません。次のパラメーターを渡すためには、PLIST および PARM 命令を使用してください。

- 実行したいグラフィックス・ルーチンの名前。
- 指定されたグラフィックス・ルーチン用の適切なパラメーター。これらのパラメーターは、グラフィックス・ルーチンに必要なデータ・タイプのパラメーターでなければならないため、浮動形式を持つことはできません。

この CALL を処理するプロシージャーは、OS/400 グラフィックス・ルーチンを暗黙的に開始したり終了することはありません。

OS/400 グラフィックス、グラフィックス・ルーチンおよびパラメーターについて詳しくは、「*GDDM Programming Guide*」および「*GDDM Reference*」を参照してください。

注: CALL 命令を使用して、OS/400® グラフィックスを呼び出すことができます。またルーチンのプロトタイプを定義し、プロトタイプに EXTPGM キーワードを指定する場合には、CALLP を使用することもできます。 CALLB 命令を使用することはできません。日付、時刻、タイム・スタンプ、またはグラフィックス・フィールドを GDDM® に渡すことはできません。ポインターを渡すこともできません。

特殊なルーチンの呼び出し

ILE RPG は、CALL および PARM 命令や CALLP 命令を使って次の特殊ルーチンの使用をサポートします。

- メッセージ検索ルーチン (SUBR23R3)
- シフト文字付き 2 バイト・データの転送およびシフト文字の削除 (SUBR40R3)
- シフト文字付き 2 バイト・データの転送およびシフト文字の追加 (SUBR41R3)

注: CALLB 命令を使用してこれらの特殊サブルーチンを呼び出すことはできません。サブルーチンのプロトタイプを定義する場合には、CALLP を使用することができます。

メッセージ検索ルーチンはまだサポートされてはいますが、より強力な QMHRTVM メッセージ API を使用されるようお奨めします。

同様に、ルーチン SUBR40R3 および SUBR41R3 は互換上の理由からのみ継続されています。これらは、新しいグラフィック・データ・タイプを経由して RPG IV により提供されるグラフィックス・サポートのレベルを反映するために、更新はされません。

マルチスレッド化に関する考慮事項

一般的に、複数のスレッド内でアプリケーションを実行すると、アプリケーションのパフォーマンスが向上します。ILE RPG の場合には、通常これはあてはまりません。実際、モジュール・レベルでプロシージャーを逐次化することによってスレッド・セーフティーが実現された場合、マルチスレッド・アプリケーションのパフォーマンスは、単一スレッド・バージョンのパフォーマンスより低くなることがあります。

マルチスレッド化された環境内で ILE RPG プロシージャーを実行することは、そのアプリケーションにおいて他の必要性がある場合 (例えば、Domino 出口プログラムを書く場合、または Java から短期実行 RPG プロシージャーを呼び出す場合など) に限るようにお勧めします。Java から呼び出された長期実行 RPG プログラムの場合、RPG プログラムの別個のプロセスを使用するようにお勧めします。

THREAD(*SERIALIZE) 制御仕様書キーワードを指定すると、ILE RPG モジュールについてスレッド・セーフティーを実現する上で役立ちます。
THREAD(*SERIALIZE) を指定すると、ほとんどの変数とすべての内部制御構造が複数のスレッドによって不正にアクセスされないよう保護されます。スレッド・セーフ・モジュールは、モジュール内のプロシージャーに入るとロックされ、モジュール内のプロシージャーがまだ実行していないときはアンロックされます。このようにアクセスを逐次化すると、活動化グループ内のいずれのモジュール内でも、活動状態になれるスレッドは一度に 1 つだけになります。ただし、複数のモジュール間で共用される記憶域のスレッド・セーフティーの取り扱いは、まだ、プログラマーに任されています。これは、アプリケーション内にロジックを追加して記憶域へのアクセスを同期化することによって行います。例えば、共用ファイル、エクスポートされた記憶域やインポートされた記憶域、パラメーターのアドレスでアクセスさ

れた記憶域は、複数のスレッドからモジュールを介して共用することができます。このタイプの記憶域へのアクセスを同期化するためには、次のいずれか、または両方を行います。

- 共用資源が複数のスレッドから同時にアクセスされないようにアプリケーションを構成する
- 別個のスレッドから資源に同時にアクセスしようとする場合は、セマフォーまたは mutex といった機能を使用してアクセスを同期化する。詳細については、以下の URL で、プログラミングのトピックからマルチスレッド・アプリケーションの資料を参照してください。

  `http://www.ibm.com/eserver/iseries/infocenter`

複数のモジュールを介したデータの共用方法

THREAD(*SERIALIZE) 制御仕様書キーワードを使用してモジュールへのアクセスを逐次化すると、各モジュール内でグローバル・データへの順次アクセスは必ず行われますが、モジュールを介した共用データへの順次アクセスは必ずしも行われません。共用データに一度に 1 つのスレッドだけがアクセスできるようにするのは、プログラマーの役目です。

次の場合には、同じデータに 2 つ以上のモジュールがアクセスできます。

- 定義仕様書に EXPORT/IMPORT キーワードが使用されている場合
- ファイルがモジュールを介して共用される場合
- データがポインターに基づいており、そのポインターが複数のモジュールで使用できる状態の場合

例えば、モジュール A 内のプロシージャー A が、モジュール B 内のプロシージャー B にポインターを引き渡すと、プロシージャー B はそのポインターを静的変数内に保管します。このような場合、両方のモジュールは、モジュール A 内で実行するスレッドが基本記憶域にアクセスしているときにその記憶域にアクセスできます。プロシージャー B が戻ると、別のスレッドがモジュール B 内のプロシージャーを呼び出し、基本記憶域にアクセスできるようになります。モジュール A およびモジュール B 内で静的記憶域へのアクセスが逐次化されると、各モジュール内の同じ記憶域に同時にアクセスできなくなります。以下に、同じデータにアクセスできる 2 つのモジュールの例を示します。

```
 *---------------------------------------------------------------------------
 * .---------------.
 * |               |
 * | ある記憶域    |<---------------- 共有記憶域へのポインター
 * |               |                  (モジュール A の MyPtr の呼び出し)
 * '---------------'                  (モジュール B に静的変数として保管)
 * モジュール A
 * モジュール A のグローバル変数
D MyPtr          S             *
D SomeStorage    S          10A    based(MyPtr)
C          eval   SomeStorage = 'Init value'
C          callp  ProcB(MyPtr)                2
C          eval   SomeStorage = *BLANKS       3
 *---------------------------------------------------------------------------
 * モジュール B
 * モジュール B のグローバル変数
D SavedPtr       S             *
D SomeStorage    S          10A    based(SavedPtr)
 * モジュール B の ProcB
P ProcB          B                 export
D ProcB          PI
D   PtrParm                    *
C                 eval     SavedPtr = PtrParm          6
C                 return                               7
P                 E
 * モジュール B の ProcB2
P ProcB2         B                 export
D ProcB2         PI
D   PtrParm                    *
C                 if       SomeStorage = 'Init value'  8
C                 ....
C                 return
P                 E
```

図 73. マルチスレッド環境でのデータの共用例

ProcA が ProcB を呼び出す (行 2) と、モジュール A およびモジュール B は 1 つのスレッドが使用しているため、他のスレッドは MyPtr が指示する記憶域にアクセスできません。ProcB はポインターをモジュール B の静的記憶域に保管 (行 6) して、戻ります (行 7)。これで、モジュール B には活動状態のスレッドがなくなったので、別のスレッドがモジュール B を自由に呼び出せる状態です。別のスレッドが ProcB2 を呼び出した場合、最初のスレッドによる行 3 の処理と、2 番目のスレッドによる行 8 の処理が前後したり、同時に行われる可能性があります。これらのイベントの順序は定義されていません。SomeStorage = 'Init value' をテストするのに使用されるコードは、成功する場合もあれば、失敗する場合もあります。

共用データへのアクセスを同期するには、プログラム内でロジックを使用するか、C またはプラットフォーム関数によって提供される同期化手法を使用します。詳細については、以下の URL で、プログラミングのトピックからマルチスレッド・アプリケーションの資料を参照してください。

`http://www.ibm.com/eserver/iseries/infocenter`

モジュール間のデッドロックの防止方法

状況によっては、THREAD(*SERIALIZE) 制御仕様書キーワード以外の機能を使用してモジュールの同期化を制御する必要があります。例えば、2 つのプロシージャー PROC1 と PROC3 が同時に呼び出されている場合を考えてみます。再帰的呼び出し

が実際に行われなくても、PROC1 は PROC4 を呼び出す場合、MOD2 がアンロックするのを待ち、PROC3 は PROC2 を呼び出す場合、MOD1 がアンロックするのを待ちます。モジュールはそれぞれ他方のモジュール内でスレッドによってロックされるため、これらのプロシージャーは呼び出しを完了することができません。このタイプの問題は、モジュールへの呼び出しを逐次化した場合でも発生する可能性があるもので、デッドロックと呼ばれます。

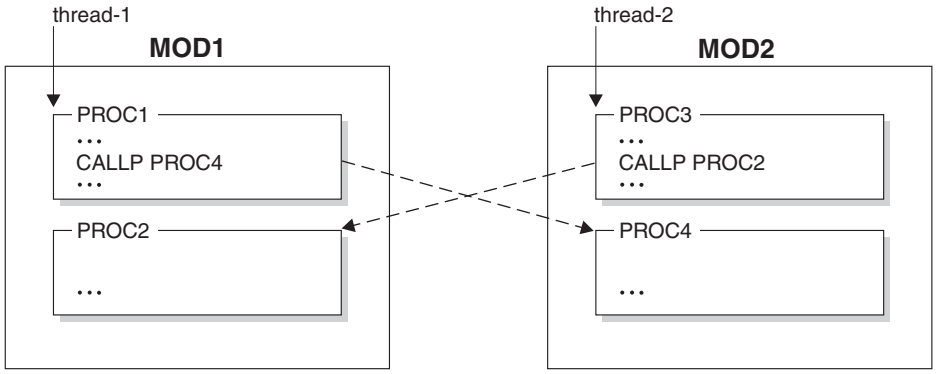


図 74. デッドロックの例

この例は、ILE RPG 同期化手法を用いて同時に同じモジュール内の複数のプロシージャーにアクセスすることができないことを示しています。

上の例のような問題を避け、スレッド・セーフのアプリケーションを確実にするためには、C またはプラットフォーム関数によって提供される手法を用いてモジュールの同期化を制御します。各スレッドについて、PROC1 または PROC3 の呼び出し元はいずれも、次のことを行う必要があります。

1. 現行スレッド以外のすべてのスレッドについてモジュールへのアクセスを、必ず同じ順序で (例えば、MOD1 の次に MOD2) 制御する
2. 現行スレッドで、モジュール内でプロシージャーを呼び出す (PROC1 および PROC3)
3. ステップ 1 の逆順で、すべてのスレッドについてモジュールへのアクセスを解放する (MOD2 の次に MOD1)

1 つのスレッドで MOD1 へのアクセス制限が正しく実行されます。MOD1 および MOD2 のすべてのユーザーは、この順序での MOD1 および MOD2 へのアクセスを制限するプロトコルを使用するため、最初のスレッドがモジュールへのアクセスを制限されているかぎり、他のスレッドは MOD1 または MOD2 内でプロシージャーを呼び出すことはできません。このような状況では、同じモジュール内の複数のプロシージャーに同時にアクセスすることはできますが、このモジュールは現行スレッドでしか使用できないため、スレッド・セーフです。

この方法は、共用記憶域へのアクセスを同期する場合にも適用できます。

第 11 章 RPG と e-business の世界

本章では、ILE RPG を e-business ソリューションの一部として使用する方法について説明します。内容は以下のとおりです。

- 『RPG と XML』
- 『RPG と MQSeries、V5.2』
- 『RPG と Java』

RPG と XML

Extensible Markup Language (XML) は、World Wide Web Consortium (W3C) によって開発された SGML のサブセットです。その目的は、一般的な SGML を Web 上で、現在 HTML で行なえるのと同じように提供し、受け取り、処理できるようにすることです。XML は、インプリメンテーションの容易性および SGML と HTML 双方の相互運用性を目指して設計されています。

XML の詳細については、`http://www.w3.org/XML` を参照してください。

XML Toolkit for iSeries (5733XT1) を使用すると、ILE RPG プログラムで新規の XML 文書の作成および既存文書の解析ができます。 XML は、データ・ストアとしても入出力メカニズムとしても使用することができます。

RPG と MQSeries、V5.2

MQSeries® を使用すると、プログラムは、同一プラットフォーム上または異なるプラットフォーム上の他のプログラムと、同じメッセージ処理製品を使用して通信することができます。 MQSeries はネットワーク・インターフェースを管理し、デリバリーを保証し、通信プロトコルを扱い、システムの問題の後のリカバリーを処理します。 MQSeries は 35 のプラットフォームにまたがって使用することができます。

MQSeries API を使用する iSeries RPG アプリケーションの例は、「*MQSeries アプリケーション・プログラミング・ガイド*, SC88-7253-11」を参照してください。この資料はオンラインで、**ibm.com**/software/ts/mqseries/library/manualsa/ で参照できます。

RPG と Java

Java と RPG についての概要

Java プログラム言語は、Sun Microsystems によって開発された高水準オブジェクト指向言語です。Java プログラムは、WebSphere Development Studio for iSeries の VisualAge® for Java コンポーネントを使用して開発することができます。

オブジェクト指向プログラミングにおいては、「メソッド」は「クラス」の一部として定義される、プログラム済みのプロシージャーです。「クラス」とはメソッド

と変数の集まりです。Java メソッドをユーザーの RPG プログラムから呼び出すことができます。ほとんどの Java メソッドは Java で書かれていますが、RPG などの他の高水準言語でメソッドを書くこともできます。これを「ネイティブ・メソッド」と呼びます。当セクションには、RPG から Java メソッドを呼び出したり、RPG ネイティブ・メソッドを書いたりするための情報があります。

オブジェクト・データ・タイプおよび CLASS キーワード

オブジェクトを保管できるフィールドは、**O** データ・タイプを使って宣言します。タイプ O のフィールドを宣言するには、D 仕様書の 40 桁目に O をコーディングし、オブジェクトのクラスを提供する CLASS キーワードを使用します。CLASS キーワードは次のとおり 2 つのパラメーターを受け入れます。

```
CLASS(*JAVA:class_name)
```

*JAVA はオブジェクトが Java オブジェクトであることを示します。Class_name は、オブジェクトのクラスを指定するもので、文字リテラルまたは名前の付いた固定情報でなければならず、かつクラス名は完全修飾されている必要があります。クラス名では、大文字と小文字が区別されます。

例えば、タイプ BigDecimal のオブジェクトを保持するフィールドを宣言するには、次のようにします。

```
D bdnum           S               O   CLASS(*JAVA:'java.math.BigDecimal')
```

タイプ String のオブジェクトを保持するフィールドを宣言するには、次のようにします。

```
D string          S               O   CLASS(*JAVA:'java.lang.String')
```

どちらのクラス名も完全に修飾されていること、および大文字小文字が Java クラスの大文字小文字と正確に一致していることに注意してください。

タイプ O のフィールドをデータ構造のサブフィールドとして定義することはできません。タイプ O のフィールドを持つ配列を定義することはできますが、タイプ O の実行時テーブルとコンパイル時テーブル、および実行時配列とコンパイル時配列は許されません。

Java メソッドのプロトタイピング

サブプロシージャーと同様に、Java メソッドも正確に呼び出すためにプロトタイプされる必要があります。ILE RPG コンパイラーは、メソッドの名前、所属するクラス、パラメーターのデータ・タイプと戻り値のデータ・タイプ、およびそのメソッドが静的メソッドであるかそうでないかを知る必要があります。

拡張された EXTPROC キーワードを使用して、メソッドの名前と所属するクラスを指定することができます。 Java メソッドのプロトタイピングの場合、EXTPROC キーワードの正しい形式は次のとおりです。

```
EXTPROC(*JAVA:class_name:method_name)
```

クラス名もメソッド名も、ともに文字定数であることが必要です。クラス名は完全に修飾された Java クラス名でなければならず、大文字小文字の区別があります。メソッド名は呼び出されるメソッドの名前でなければならず、大文字小文字の区別があります。

プロトタイプを作成するときは、Java で書かれるメソッドでも RPG で書かれるネイティブ・メソッドでも、いずれの場合も *JAVA を使用します。STATIC キーワードは、メソッドが静的であることを示すために使用します。

Java と RPG 定義とデータ・タイプ: メソッドのパラメーターと戻り値のデータ・タイプは、サブプロシージャーのプロトタイピングの時と同様に指定します。ただしデータ・タイプは、実際は Java データ・タイプにマップされます。次の表は、ILE RPG データ・タイプと Java データ・タイプとの対応表です。

Java データ・タイプ	ILE RPG データ・タイプ	RPG 定義
boolean	標識	N
byte[1]	整数	3I 0
	文字	1A
byte[]	文字の長さ > 1 (3 を参照)	nA
	文字の長さ = 1 の配列 (4 を参照)	1A DIM(x)
	日付	D
	時刻	T
	タイム・スタンプ	Z
short	2 バイト整数	5I 0
char	UCS-2 長さ = 1	1C
char[]	UCS-2 の長さ > 1 (3 を参照)	nC
	UCS-2 の長さ = 1 の配列 (4 を参照)	1C DIM(x)
int	4 バイト整数	10I 0
long	8 バイト整数	20I 0
float	4 バイト浮動	4F
double	8 バイト浮動	8F
any object	オブジェクト	O CLASS(x)
any array	同等タイプの配列 (4 を参照)	DIM(x)

注:

1. Java byte タイプが文字 (1A) との間で変換される時は、ASCII 変換が行なわれます。Java byte タイプが整数 (3I) データ・タイプとの間で変換される場合は、ASCII 変換は行なわれません。
2. Java でいかなるタイプを持つ配列の場合も、RPG で同等タイプの配列を定義することができますが、文字で長さが 1 より大きいか、または UCS-2 で長さが 1 より大きいデータ・タイプの配列は使用できないことに注意してください。
3. UCS-2 で長さが 1 より大きいデータ・タイプおよび文字で長さが 1 より大きいデータ・タイプの場合、VARYING キーワードを使用することができます。Java byte[] および char[] は固定長で宣言することはできないため、一般的に VARYING キーワードの使用をお勧めします。
4. Java 配列は固定長で宣言できないため、RPG の配列データ・タイプでは配列パラメーターとして一般に OPTIONS(*VARSIZE) をコーディングする必要があります。

ゾーン、パック、2 進、および符号なしのデータ・タイプは Java では使用できません。ゾーン、パック、2 進、または符号なしのフィールドをパラメーターとして渡した場合、コンパイラーは適切な変換を行ないます。ただしその結果として、切り捨てが行なわれるかまたは精度が失われる (あるいはその両方) 可能性があります。

メソッドを呼び出す際、コンパイラーが配列をパラメーターとして受け入れるのは、そのパラメーターが DIM キーワードを使用してプロトタイプされている場合のみです。

メソッドの戻り値またはパラメーターがオブジェクトである場合、プロトタイプに CLASS キーワードをコーディングすることにより、そのオブジェクトのクラスを提供する必要があります。指定されたクラス名は、オブジェクトが戻されたりパラメーターが渡されたりするクラス名になります (呼び出されるメソッドのクラスを指定するには、EXTPROC キーワードを使用します)。

メソッドを静的メソッドとして呼び出そうとしている場合は、プロトタイプに STATIC キーワードを指定する必要があります。メソッドがコンストラクターである場合は、 *CONSTRUCTOR をメソッドの名前として指定する必要があります。

Java では、値として渡すことができるのは以下のデータ・タイプのみです。

```
boolean
byte
int
short
long
float
double
```

これらのタイプのパラメーターは、プロトタイプで VALUE キーワードを指定する必要があります。

オブジェクトは参照によってのみ、渡すことができることに注意してください。VALUE キーワードはタイプ O と一緒に指定することはできません。配列は Java からはオブジェクトとして見なされるため、配列へのパラメーター・マッピングも参照によって渡される必要があります。これには文字配列およびバイト配列も含まれます。CONST キーワードが使用できます。

Java メソッドのプロトタイピングの例: この節では、Java メソッドのプロトタイピングの例をいくつか説明します。

例 1: Java Integer クラスには、*toString* と呼ばれる静的メソッドがあり、このメソッドは *int* パラメーターを受け取って String オブジェクトを戻します。これは、Java では次のように宣言されます。

```
static String  Integer.toString(int)
```

このメソッドは次のようにプロトタイプされるはずです。

```
# D tostring        PR             O   EXTPROC(*JAVA:
# D                                     'java.lang.Integer':
# D                                     'toString')
# D                                    CLASS(*JAVA:'java.lang.String')
# D                                    STATIC
# D    num                       10I 0 VALUE
```

EXTPROC キーワードが、このメソッドが Java メソッドであることを示します。また、メソッド名が「toString」で、「java.lang.Integer」というクラスにあることも示しています。

40 桁目の O と CLASS キーワードは、このメソッドがオブジェクトを戻すこと、およびそのオブジェクトのクラスが「java.lang.String」であることをコンパイラーに伝えます。

STATIC キーワードは、このメソッドが静的メソッドであること、つまりこのメソッドを呼び出すために Integer オブジェクトは必要ではないことを示します。

パラメーターのデータ・タイプは 10I と指定されており、これは Java の *int* データ・タイプにマップします。パラメーターが int であるため、これは値として渡されなければならず、VALUE キーワードが必要です。

例 2: Java Integer クラスには、*getInteger* と呼ばれる静的メソッドがあり、このメソッドはパラメーターとして String オブジェクトおよび Integer オブジェクトを受け取り、Integer オブジェクトを戻します。これは、Java では次のように宣言されます。

```
static Integer Integer.getInteger(String, Integer)
```

このメソッドは次のようにプロトタイプされるはずです。

```
D getint          PR              O   EXTPROC(*JAVA:
D                                      'java.lang.Integer':
D                                      'getInteger')
D                                     CLASS(*JAVA:'java.lang.Integer')
D                                     STATIC
D   string                        O   CLASS(*JAVA:'java.lang.String') CONST
D   num                           O   CLASS(*JAVA:'java.lang.Integer') CONST
```

このメソッドは 2 つのオブジェクトをパラメーターとして受け取ります。D 仕様書の 40 桁目には O がコーディングされ、CLASS キーワードが各オブジェクト・パラメーターのクラスを指定します。どちらのパラメーターも入力専用なので、CONST キーワードが指定されます。

例 3: Java Integer クラスには *shortValue* と呼ばれるメソッドが含まれており、このメソッドは自分自身の呼び出しのために使用された Integer オブジェクトの短縮表記を戻します。これは、Java では次のように宣言されます。

```
short shortValue()
```

このメソッドは次のようにプロトタイプされるはずです。

```
D shortval        PR             5I 0 EXTPROC(*JAVA:
D                                      'java.lang.Integer':
D                                      'shortValue'
```

このメソッドは静的メソッドでないため、STATIC キーワードは指定しません。このメソッドはパラメーターを取らないため、何もコーディングされていません。このメソッドを呼び出す際に、ユーザーが最初のパラメーターとして Integer インスタンスを指定します。戻り値は 5I と指定されており、これは Java short データ・タイプにマップされます。

例 4: Java Integer クラスには、*equals* と呼ばれるメソッドがあり、このメソッドはパラメーターとして Object を受け取って boolean を戻します。これは、Java では次のように宣言されます。

```
boolean equals(Object)
```

このメソッドは次のようにプロトタイプされるはずです。

```
D equals          PR             N   EXTPROC(*JAVA:
D                                     'java.lang.Integer':
D                                     'equals')
D  obj                           O   CLASS(*JAVA:'java.lang.Object')
```

戻り値は N と指定されており、これは Java boolean データ・タイプにマップされます。これは静的メソッドではないため、このメソッドへの呼び出しには 2 つのパラメーターがあり、インスタンス・パラメーターが最初にコーディングされます。

Java メソッドの ILE RPG からの呼び出し

この節では、Java メソッドを ILE RPG プログラムから呼び出す方法について説明します。

メソッドが静的メソッドでない場合、これを「インスタンス・メソッド」と呼びます。そのメソッドを呼び出すための特別な最初のパラメーターとして、オブジェクト・インスタンスがコーディングされなければなりません。例えば、インスタンス・メソッドが 1 つのパラメーターを伴ってプロトタイプされている場合であれば、ユーザーは 2 つのパラメーターを指定して呼び出さなければならず、その最初がインスタンス・パラメーターになるようにする必要があります。

以下のステップでは、ILE RPG から Java メソッドへの呼び出しについて説明します。

1. Java メソッドは、既存の命令コード CALLP (戻り値がないと予想される場合) および EVAL (戻り値があると予想される場合) を使用して呼び出すことができます。ユーザーの RPG プロシージャーが Java メソッドを呼び出そうとすると、RPG はその Java 仮想マシン (JVM) が開始済みであるかどうかをチェックします。まだ開始されていない場合は、RPG はユーザーに代わって JVM を開始します。 198 ページの『Java 仮想マシン (JVM) の作成』で説明されている JNI 機能を使用して、ユーザー自身が JVM を開始することもできます。
2. ユーザーが自分のクラス (あるいは通常の java.xxx クラス以外のすべてのクラス) を使用している場合は、Java メソッドを呼び出す前にユーザーの CLASSPATH 環境変数を必ずセットアップしておいてください。RPG がユーザーに代わって JVM を呼び出す時、RPG はユーザーの CLASSPATH 環境変数の中にあるクラスを標準クラスパスへ追加します。これによって、ユーザーが自分のクラスを使う時に Java はそれらのクラスを見つけることができます。CLASSPATH 環境変数を対話式に設定するには、次のようにします。

   ```
   ===>ADDENVVAR ENVVAR(CLASSPATH)
                 VALUE('/myclasses/:/xyzJava/classes/')
   ```

   ディレクトリーはコロンで分ける必要があります。
3. 通常、Java は自分自身のガーベッジ・コレクションを行ない、オブジェクトがいつ不要になるかを検出します。ユーザーが非ネイティブ RPG プロシージャーから Java コンストラクターを呼び出すことによってオブジェクトを作成する場合

は、Java はそのオブジェクトを破棄してよいということを知る方法がないため、それらを破棄しません。 194 ページの『複数のオブジェクトの同時解放を Java に通知』 で説明する JNI 機能を呼び出すことによって、複数のオブジェクトに対するガーベッジ・コレクションを同時に使用可能にできます。もうオブジェクトが不要であることをユーザーが知っている場合は、 195 ページの『一時オブジェクトの使用が終了したことを Java に伝える』で説明する JNI 機能を呼び出すことによって、このことを Java に伝える必要があります。

注意:
Java はスレッドを使用するため、Java と相互作用するすべてのモジュールに THREAD(*SERIALIZE) キーワードをコーディングしておく必要があります。 RPG は明らかに自動ストレージのみを使用するサブプロシージャーにあっても、静的ストレージに強く依存します。この静的ストレージを正しく取り扱うには THREAD(*SERIALIZE) が必要です。これは Java メソッドへの呼び出しを含むモジュールのみではなく、Java との相互作用中に呼び出される可能性があるモジュールすべてに適用されます。

さまざまな JNI 機能の詳細については、 194 ページの『Java を使用するためのその他の RPG コーディング』を参照してください。

例 1

この例では、目的は 2 つの BigDecimal 値を一緒に加算することです。これを行なうため、2 つの BigDecimal オブジェクトは BigDecimal クラス用のコンストラクターを呼び出してインスタンス化される必要があり、フィールドは BigDecimal オブジェクトを保管できるように宣言される必要があり、さらに BigDecimal クラスにある add() メソッドが呼び出される必要があります。

```
 *   String パラメーターを受け取る BigDecimal コンストラクターのプロトタイプ。
 *   新しい BigDecimal オブジェクトを戻します。
 *   string パラメーターはこのコンストラクターによって変更されないため、
 *   CONST キーワードをコーディングすることになります。これによって
 *   コンストラクターの呼び出しがより便利になります。
 *
D bdcreate1       PR              O   EXTPROC(*JAVA:
D                                       'java.math.BigDecimal':
D                                       *CONSTRUCTOR)
D    str                          O   CLASS(*JAVA:'java.lang.String')
D                                     CONST
 *
 *   double パラメーターを受け取る BigDecimal コンストラクターのプロトタイプ。
 *   8F は Java の double データ・タイプにマップするため、
 *   VALUE で渡す必要があります。BigDecimal オブジェクトを戻します。
 *
D bdcreate2       PR              O   EXTPROC(*JAVA:
D                                       'java.math.BigDecimal':
D                                       *CONSTRUCTOR)
D   double                       8F   VALUE
```

図 75. BigDecimal Java クラスを呼び出す RPG コーディング例 (1/2)

```
 *   BigDecimal オブジェクトを保管するフィールドを定義します。
 *
D bdnum1          S               0    CLASS(*JAVA:'java.math.BigDecimal')
D bdnum2          S               0    CLASS(*JAVA:'java.math.BigDecimal')
 *
 *   使用しているコンストラクターのいずれかが、String オブジェクトを
 *   必要としているため、これらのコンストラクターのいずれかを作成する必要が
 *   あります。byte 配列をパラメーターとして受け取る String コンストラクターを
 *   プロトタイプします。これは String オブジェクトを戻します。
 *
D makestring      PR              O   EXTPROC(*JAVA:
D                                      'java.lang.String':
D                                      *CONSTRUCTOR)
D    bytes                      30A   CONST VARYING
 *
 *  String オブジェクトを保管するフィールドを定義します。
 *
D string          S               O    CLASS(*JAVA:'java.lang.String')
 *
 *  BigDecimal 追加メソッドのプロトタイプ。BigDecimal オブジェクトをパラメーター
 *  として受け取り、BigDecimal オブジェクトを戻します (パラメーターおよび
 *  BigDecimal オブジェクトの合計が、この呼び出しを行なうために使用されます)。
 *
D add             PR              O    EXTPROC(*JAVA:
D                                      'java.math.BigDecimal':
D                                      'add')
D                                      CLASS(*JAVA:'java.math.BigDecimal')
D   bd1                           O    CLASS(*JAVA:'java.math.BigDecimal')
D                                      CONST
 *
 *  合計を保管するフィールドを定義します。
 *
D sum             S               O    CLASS(*JAVA:'java.math.BigDecimal')
D
D double          S              8F    INZ(1.1)
D fld1            S             10A
```

図 75. BigDecimal Java クラスを呼び出す RPG コーディング例 (2/2)

呼び出しを行なうコードは次の通りです。

```
 *  fld1 から String オブジェクトを作成するために、String クラスのコンストラクターを
 *  呼び出します。コンストラクターを呼び出しているため、
 *  最初のパラメーターとして String オブジェクトを渡す必要はありません。
 *
C                   EVAL      string = makestring('12345678901234567890123456789
 *
 *  インスタンス化したばかりの String オブジェクトを使用して、String パラメーターを
 *  受け取る BigDecimal コンストラクターを呼び出します。
 *
C                   EVAL      bdnum1 = bdcreate1(string)
 *
 *  double をパラメーターをして受け取る BigDecimal コンストラクターを
 *  呼び出します。
 *
C                   EVAL      bdnum2 = bdcreate2(double)

 *
 *  add メソッドを呼び出すことにより、2 つの BigDecimal オブジェクトを
 *  一緒に加算します。プロトタイプは、add が 1 つのパラメーターを
 *  受け取ることを示していますが、add は静的メソッドではないため、
 *  呼び出しを行なうために BigDecimal オブジェクトも渡す必要があり、
 *  かつそれは最初のパラメーターであることが必要です。
 *  bdnum1 は呼び出しを行なうために使用するオブジェクトで、
 *  bdnum2 はパラメーターです。
 *
C                   EVAL      sum = add(bdnum1:bdnum2)
 *  ここで sum には bdnum1 + bdnum2 の値を持つ BigDecimal
 *  オブジェクトが入ります。
```

図 76.

例 2

この例は、ILE RPG %TRIM 組み込み関数の代わりに trim() メソッドを使用して Java で TRIM を行なう方法について示します。String クラスにある trim() メソッドは静的メソッドではないため、これを呼び出すには String オブジェクトが必要です。

```
 *  トリムしたい String オブジェクトを保管するフィールドを定義します。
 *
D str             S               0   CLASS(*JAVA:'java.lang.String')
 *
 *  String クラス用のコンストラクターのプロトタイプ。
 *  コンストラクターは byte 配列を想定しています。
 *
D makestring      PR              0   EXTPROC(*JAVA:
D                                      'java.lang.String':
D                                      *CONSTRUCTOR)
D                                     CLASS(*JAVA:'java.lang.String')
D   parm                        10A
 *
 *  String を byte 配列に変換する String メソッド getBytes のプロトタイプ。
 *  変換後はこの byte 配列を alpha フィールドに保管します。
 *
D getBytes        PR            10A   EXTPROC(*JAVA:
D                                       'java.lang.String':
D                                       'getBytes') VARYING
 *
 *  String メソッド trim のプロトタイプ。これはパラメーターは取りませんが、
 *  静的メソッドではないため、String オブジェクトを使用して
 *  呼び出す必要があります。
 *
D trimstring      PR              0   EXTPROC(*JAVA:
D                                      'java.lang.String':
D                                      'trim')
D fld             S             10A    INZ('   hello    ') VARYING
```

図 77. trim() Java メソッドを使用する RPG コーディング例

この呼び出しは次のようにコーディングされます。

```
 * String コンストラクターを呼び出します。
 *
C                   EVAL      str = makestring(fld)
 *
 * String trim() メソッドを呼び出すことにより、ストリングをトリムします。
 * この str フィールドを、結果を保管するために再利用します。
 *
C                   EVAL      str = trimstring(str)
 *
 * ストリングをバイト配列に変換して戻し、
 * これを fld に保管します。
 *
C                   EVAL      fld = getBytes(str)
```

図 78. String コンストラクターに対する RPG 呼び出し

呼び出しを行なうのにオブジェクトが必要でない場合を除き、静的メソッドは同じ方法で呼び出されます。上記の getBytes() メソッドがもし静的メソッドであったならば、呼び出しは次の例のようになるはずです。

```
C                         EVAL      fld = getBytes()
```

メソッドが値を戻さない場合は、CALLP 命令コードを使用します。

オブジェクトの作成

非静的メソッドを呼び出すには、オブジェクトが必要です。オブジェクトのクラスは、メソッドを含んでいるクラスと同じでなければなりません。使用可能なオブジェクトが既にあっても、新たなオブジェクトをインスタンス化する必要がある場合があります。これはクラス・コンストラクターを使用して行ないます。クラス・コ

ンストラクターは静的メソッドでもインスタンス・メソッドでもありません。したがってクラス・コンストラクターにはインスタンス・パラメーターは必要ありません。特別なメソッド名である *CONSTRUCTOR が、コンストラクターのプロトタイピングを行なう時に使用されます。

例えば、クラス BigDecimal には float パラメーターを受け取るコンストラクターがあります。

このコンストラクターは次のようにプロトタイプされます。

```
# D bdcreate        PR              O   EXTPROC(*JAVA:
# D                                       'java.math.BigDecimal':
# D                                       *CONSTRUCTOR)
# D    dnum                        4F   VALUE
```

パラメーターは Java float データ・タイプにマップされるため、値による受け渡しをする必要があることに注意してください。

このコンストラクターは次のように呼び出します。

```
# D bd              PR              O   CLASS(*JAVA:
# D                                           'java.math.BigDecimal')
#  /free
#     bd = bdcreate(5.2E9);
#  /end-free
```

戻されるオブジェクトのクラスはコンストラクター自身のクラスと同じであるため、 CLASS キーワードはコンストラクターの場合には冗長になりますが、コーディングしてもかまいません。

独自クラスでのメソッドの呼び出し

独自の Java クラスを使用する場合、EXTPROC キーワードおよび CLASS キーワードで指定したクラスが、使用するクラスの名前になります。そのクラスがパッケージの一部である場合はキーワードにパッケージ情報を含めます。例えば、次の 2 つのクラスを考えてみてください。

```
class Simple
{
   static void method (void)
   {
      System.out.println ("Simple method");
   }
}

package MyPkg;

class PkgClass
{
   static void method (void)
   {
      System.out.println ("PkgClass method");
   }
}
```

図 79.

Simple クラス・ファイルが /home/myclasses/Simple.class の場合は、ディレクトリー /home/myclasses を CLASSPATH 環境変数に指定し、'Simple' をクラス名として RPG キーワードに指定します。

PkgClass クラス・ファイルが /home/mypackages/MyPkg/PkgClass.class の場合は、ディレクトリー /home/mypackages (パッケージが含まれるディレクトリー) を CLASSPATH 環境変数に指定し、 'MyPkg.PkgClass' (パッケージ修飾された Java クラス) をクラス名として RPG キーワードに指定します。

RPG キーワードのクラス名は Java クラスの import ステートメントで指定するものと同一の名前です。クラス・ファイルの位置、またはパッケージを含むディレクトリーの位置を指定するには、CLASSPATH 環境変数を使用します。

注: 注: クラスが jar ファイル内に入っている場合には、classpath で jar ファイル自身を指定します。

```
===> ADDENVVAR CLASSPATH '/home/myclasses:/home/mypackages:/home/myjarfiles/j1.jar'
```

```
D simpleMethod    PR                  EXTPROC(*JAVA
D                                           : 'Simple'
D                                           : 'method')
D                                     STATIC
D pkgMethod       PR                  EXTPROC(*JAVA
D                                           : 'Pkg.PkgClass'
D                                           : 'method')
D                                     STATIC
```

図 80. パッケージ内の Java メソッドのための RPG プロトタイプの作成

Java 仮想マシンのセットアップ方法の制御

RPG が Java 仮想マシン (JVM) を開始する際、JVM の開始方法を制御するオプションがいくつかあります。 iSeries Information Center の『*Java* システム・プロパティー』の節を参照してください。

- これらのオプションは SystemDefaults.properties ファイルに指定できます。
- CLASSPATH 環境変数を使用するとクラスパスを指定できます (上記参照)。
- これらのオプションは環境変数 QIBM_RPG_JAVA_PROPERTIES で指定できます。この環境変数で指定されたオプションは SystemDefaults.properties ファイル内のオプションを指定変更します。 java.class.path オプションを、この環境変数と CLASSPATH 環境変数の両方で指定した場合には、クラスパスに対してどちらが優先するかは未定義です。

  QIBM_RPG_JAVA_PROPERTIES 環境変数でオプションを指定するには、オプション中に現れない任意の文字を分離文字として、オプションを文字列中に順次にコーディングします。文字列の末尾には分離文字を置きます。例えば、以下のオプションを指定するとします。

  ```
  java.version=1.4
  os400.stderr=file:stderr.txt
  ```

この場合、以下のコマンドを使用して環境変数を追加します。

```
ADDENVVAR ENVVAR(QIBM_RPG_JAVA_PROPERTIES)
VALUE('-Djava.version=1.4;-Dos400.stderr=file:stderr.txt;')
```

オプション文字列が無効な場合、Java はそのオプションのいずれかをリジェクトすることがあります。ジョブ・ログ内のメッセージ JVAB55A に、どのオプションが無効であるかが示されます。この場合、RPG はオプションなしで JVM を再度開始しようとしますが、 java.class.path オプションが CLASSPATH 環境変数に指定されている場合、このオプションは引き続き組み込まれます。

RPG ネイティブ・メソッド

RPG ネイティブ・メソッドを定義するには、通常の Java メソッドの場合にプロトタイプをコーディングするのと同じ方法でプロトタイプをコーディングします。それから、通常どおりに RPG サブプロシージャーを書きます。ネイティブ・メソッドの場合、プロシージャー開始仕様書で EXPORT キーワードをコーディングする必要があります。

ネイティブ・メソッドは、ライブラリー・リストにあるサービス・プログラムの中に置かなければなりません。ネイティブ・メソッドを呼び出している Java クラスに、次のような静的ステートメントがなければなりません。

```
static
{
     System.loadLibrary ("MYSRVPGM");
}
```

これにより、Java はユーザーのネイティブ・メソッドを見つけることができます。*JAVA およびクラスを、ネイティブ・メソッドのプロトタイプのために EXTPROC キーワードに追加するのを除き、ユーザーは任意のサブプロシージャーと同じようにネイティブ・メソッドを書きます。 190 ページの図 81 は、ネイティブ・メソッドを呼び出す Java クラスの例です。

注意:
JVM の開始方法を制御するために環境変数を使用している場合は、 RPG プログラムが Java メソッドを呼び出す前に環境変数がジョブに存在している必要があります。 ADDENVVAR LEVEL(*SYS) を使用すると、環境変数はシステム・レベルで追加され、デフォルトでは、各ジョブはその環境変数を設定した状態で開始されます。これを行う場合は、システム上のアプリケーションが必要とする Java クラスを含むすべてのディレクトリーがクラスパスに含まれていることを確認してください。

```
class MyClass
{

   static
   {
     System.loadLibrary ("MYSRVPGM");
   }

  native boolean checkCust (byte custName[]);

   void anotherMethod ()
   {
      boolean found;
      // call the native method
      found = checkCust (str.getBytes());
    }
 }
```

図 81. ネイティブ・メソッドを呼び出す Java クラス

図 82 は RPG ネイティブ・メソッドのプロトタイプです。

```
D checkCust       PR              N   EXTPROC(*JAVA
D                                     : 'MyClass'
D                                     : 'checkCust')
D   custName                  100A   VARYING CONST
```

図 82. RPG ネイティブ・メソッド・プロトタイプ

ネイティブ・メソッド自身も他の任意のサブプロシージャーと同様にコーディングされます。図 83 は、RPG でコーディングされたネイティブ・メソッドの例です。

```
P checkCust       B                   EXPORT
D checkCust       PI              N
D   custName                  100A   VARYING CONST
 /free   chain custName  rec;
   return %found;
 /end-free
P checkCust       E
```

図 83. RPG でコーディングされたネイティブ・メソッドの例

Java はユーザーのサービス・プログラムをデフォルトの活動化グループから呼び出します。ユーザーのサービス・プログラムが活動化グループ *CALLER を指定して作成されている場合は、そのサービス・プログラムはデフォルトの活動化グループで実行されることになります。これは、次のような問題を招く場合があります。

- ユーザーが自分のネイティブ・メソッドをデバッグ中であり、かつそのコーディングを変更したい場合は、Java が新しいバージョンを参照する前にサインオフしてサインオンしなければならなくなります。
- デフォルトの活動化グループの中で実行されていない他の RPG コードから、サービス・プログラムの中にある他のプロシージャーを呼び出している場合は、「通常のプロシージャー」とネイティブ・メソッドとの間でグローバル変数を共用することはできなくなります。このシナリオは、ユーザーの RPG サービス・プログラムの中のプロシージャーがいくつかのグローバル変数をセットアップし

ている場合で、かつそのあとでそのサービス・プログラム内のネイティブ・メソッドを呼び出す Java クラスを呼び出す場合に、起こり得ます。これらのネイティブ・メソッドは、最初のプロシージャーがセットアップしたのと同じデータを参照しないことになります。

ユーザーが自分のネイティブ・メソッド内に Java オブジェクトを作成する場合、デフォルトではそれらはネイティブ・メソッドの戻りの時点で Java によって破棄されます。ネイティブ・メソッドの戻りの後もそれらのオブジェクトを使用したい場合 (例えば、後で別のネイティブ・メソッドから使用したい場合など) は、JNI ラッパー・プロシージャー getNewGlobalRef を呼び出すことにより、Java にグローバル参照を行ないたいことを伝える必要があります。グローバル参照を終えたら、JNI ラッパー・プロシージャー freeGlobalRef を呼び出せば、Java がそのオブジェクトを再利用することができます。これらのラッパー・プロシージャーについての詳細は、196 ページの『オブジェクトを永続にしたいことを Java に伝える』および 197 ページの『永続オブジェクトの使用が終了したことを Java に伝える』を参照してください。

RPG ネイティブ・メソッドが処理できない例外で異常終了した時は、RPG コンパイラーは例外を Java に投げます。例外はクラス java.lang.Exception となり、RPG nnnnn という形式になります。ここで、nnnnn は RPG 状況コードです。

```
try
{
   nativeMethod ();
}
catch (Exception exc)
{
   ...
}
```

非静的ネイティブ・メソッドにおけるインスタンス・パラメーターの入手

非静的ネイティブ・メソッドが呼び出される場合、Java がネイティブ・メソッドに渡すパラメーターのいずれかは、このメソッドが適用されるオブジェクトになります。これは「インスタンス・パラメーター」と呼ばれ、Java メソッドの中では「this」として参照されます。ネイティブ・メソッド自身の中で、組み込み関数 %THIS を使用してインスタンス・パラメーターを入手することができます。このパラメーターをユーザーの「プロシージャー・インターフェース」の中にコーディングしてはなりません。

Java からネイティブ・メソッドへの文字パラメーターの受け渡し

文字パラメーターを扱う場合には、次の 2 つの選択肢があります。

- Java コードをできるだけ簡潔にしたい場合は、Java ネイティブ・メソッド宣言の中で String としてそのパラメーターを定義します。ユーザーの RPG コードは、ストリングの値を自分で検索する必要があります (192 ページの『RPG におけるストリング・オブジェクトの使用』を参照)。
- 文字データをユーザーの RPG プログラムの中で即時に使用可能にしたい場合は、そのパラメーターを Java ネイティブ・メソッド宣言の中で byte 配列または char 配列としてコーディングし、さらに RPG プロトタイプの中でも文字フィー

ルド、UCS-2 フィールド、または Date、Time あるいは Timestamp としてコーディングします。このようにすると RPG がユーザーに代わって変換を行ないます。

RPG におけるストリング・オブジェクトの使用: RPG コードの中に String オブジェクトがある場合、図 84 にあるコーディングを使用してその長さと内容を検索することができます。

```
D stringBytes     PR           100A   VARYING
D                                     EXTPROC(*JAVA
D                                           : 'java.lang.String'
D                                           : 'getBytes')
D stringLength    PR                  like(jint)
D                                     EXTPROC(*JAVA
D                                           : 'java.lang.String'
D                                           : 'length')
D string          S                   like(jstring)
D len             S                   like(jint)
D data            S            100A   VARYING
 /free      len = stringLength (string);
      data = stringBytes (string);
      if (len > %len(data));
            error ('Actual string was too long');
        endif;
 /end-free
```

図 84. String オブジェクトの長さと内容を Java から検索する

ユーザーは、getBytes メソッドからの戻り値を、Java String の中にあるデータの長さについてユーザー自身が知っていることに基づいて長さを選んで、任意の長さ (可変または非可変のいずれでも) の文字データとして定義することができます。また、String オブジェクトの形式が正しいことが確実であれば、戻り値を Date、Time または Timestamp として定義することもできます。

他の方法として、getBytes の代わりに getChars メソッドを呼び出して、ストリング値を UCS-2 値として検索することもできます。

RPG から Java を呼び出したときのコーディング・エラー

Java リソースの解放エラー

コンストラクターを呼び出すことで、またはオブジェクトを戻すメソッドを呼び出すことで、 Java オブジェクトを作成した場合、作成したオブジェクトは解放しない限り存在し続けます。オブジェクトが解放されるのは次の場合です。

1. RPG プログラムが、オブジェクトを解放する JNI 関数を呼び出したとき (194 ページの『Java を使用するためのその他の RPG コーディング』を参照)。
2. Java から固有メソッドへの呼び出し中にオブジェクトが作成された場合に固有メソッドが返されたとき。
3. JVM が終了したとき。

Java メソッドを呼び出す RPG プロシージャーが RPG 固有のメソッドではない場合、および RPG プロシージャーが、自分が作成したオブジェクトの解放に責任を持たない場合、そのジョブは結果的にそれ以上オブジェクトを作成できなくなる可能性があります。

次のようなコード・フラグメントがあるとします。

```
strObject = newString ('abcde');
strObject = trim (strObject);
data = getBytes (strObject);
freeLocalRef (strObject);
```

このコードはオブジェクトの解放を引き受けているようですが、実際にはこのコードは 2 つのオブジェクトを作成します。 1 つ目のオブジェクトは newString() 呼び出しによって作成され、 2 つ目のオブジェクトは trim() 呼び出しによって作成されます。このコード・フラグメントを訂正するには、以下の 2 つの方法があります。

1. 複数のオブジェクトを同時に解放:

```
beginObjGroup();
strObject = newString ('abcde');
strObject = trim (strObject);
data = getBytes (strObject);
endObjGroup();
```

2. 使用されているオブジェクトすべてを追跡し、それらを個別に解放:

```
strObject = newString ('abcde');
trimmedStrObject = trim (strObject);
data = getBytes (trimmedStrObject);
freeLocalRef (strObject);
freeLocalRef (trimmedStrObject);
```

Java メソッドをほかの Java メソッドのパラメーターとして呼び出すと、別の問題が起こることがあります。次の例では、String パラメーターを取るコンストラクターから BigDecimal オブジェクトを作成しています。

```
bigDec = newBigDecimal (newString ('12.345'));
...
freeLocalRef (bigDec);
```

このコードの問題は、パラメーターの指定により String オブジェクトが作成されたのに、 RPG プロシージャーはこのオブジェクトを解放できないことです。この問題を訂正するには、Java を呼び出す RPG コードの前に `beginObjGroup()` を呼び出し、後で `endObjGroup()` を呼び出すか、以下のようにコーディングします。

```
tempObj = newString ('12.2345');
bigDec = newBigDecimal (tempObj);
freeLocalRef (tempObj);
...
freeLocalRef (bigDec);
```

存在しないオブジェクトの使用

固有のメソッドに静的な Object 変数 (定義に STATIC キーワード) がある場合、または固有のメソッドが静的なグローバル Object 変数 (メイン・ソース・セクションで宣言した変数) を使用している場合、その Object 変数は、その固有のメソッドが次に呼び出されるまで値を保持し続けます。しかし、デフォルトでは、Java は固有メソッドの呼び出し時に作成されたオブジェクトをすべて解放します (Java がオブジェクトを解放しないようにする方法については、 194 ページの『Java を使用するためのその他の RPG コーディング』を参照してください)。

RPG の ″Object″ は実際には数値オブジェクト参照です。 Java オブジェクトが解放されても、この数値オブジェクト参照を再利用することができます。 RPG 固有のメソッドが、明示的に解放の禁止が指定されていない静的な Object 変数を参照すると、次のいずれかが発生する可能性があります。

1. 数値オブジェクト参照が再使用されていない場合、オブジェクト参照は無効となる可能性があります。
2. オブジェクト参照は再使用されている可能性がありますが、参照しているのは別のオブジェクトであるため、 RPG 固有のメソッドでそのオブジェクトを使用しようとしても、エラーが発生する可能性があります。

オブジェクトの不正な再使用問題を防ぐには、次のようにします。

- 静的記憶域で Object 変数を宣言しない。代わりに、STATIC キーワードを使用せずに、サブプロシージャーのローカル記憶域ですべての Object 変数を宣言します。
- 固有のメソッドから戻る前に、すべての静的オブジェクト参照を明示的に *NULL に設定する。
- 固有のメソッドを入力するときに、すべての静的オブジェクト参照を明示的に初期値に設定する。

Java を使用するためのその他の RPG コーディング

通常は Java によって処理される機能でも、ILE RPG を Java とともに使用する場合には、ユーザーの RPG コードで処理される必要があるものがいくつかあります。RPG コンパイラーがこれらのいくつかについてはユーザーに代わって処理しますが、一部についてはユーザー自身が処理しなければならないものもあります。この節では、この処理を行ないいくつかのサンプル RPG ラッパーを示し、これらを呼び出す方法とタイミングを説明し、JNI 例外を処理する方法を提案します。

これらの JNI ラッパー関数を保持するために作成するモジュールは以下のステートメントで始める必要があります。

```
H thread(*serialize)
H nomain
 /define OS400_JVM_12
 /copy qsysinc/qrpglesrc,jni
```

次の JNI 機能用 RPG ラッパーを説明します。

- 『複数のオブジェクトの同時解放を Java に通知』
- 195 ページの『一時オブジェクトの使用が終了したことを Java に伝える』
- 196 ページの『オブジェクトを永続にしたいことを Java に伝える』
- 197 ページの『永続オブジェクトの使用が終了したことを Java に伝える』
- 198 ページの『Java 仮想マシン (JVM) の作成』
- 198 ページの『JNI 環境ポインターの入手』

複数のオブジェクトの同時解放を Java に通知

RPG コードの、Java を使用するセクションの前で JNI 関数 PushLocalFrame を呼び出し、 RPG コードの最後のセクションで PopLocalFrame を呼び出すと、複数のローカル参照を同時に解放できます。 PopLocalFrame を呼び出すと、PushLocalFrame の呼び出し以降に作成されたローカル参照すべてが解放されます。これらの JNI 関数のパラメーターについて詳しくは、http://java.sun.com にある JNI の資料を参照してください。

```
D JNI_GROUP_ADDED...
D                 c                   0
D JNI_GROUP_NOT_ADDED...
D                 c                   -1
D JNI_GROUP_ENDED...
D                 c                   0
 *-------------------------------------------------------------
 * beginObjGroup - 後にまとめて削除できる新しいオブジェクトの
 *                 グループの開始
 *-------------------------------------------------------------
P beginObjGroup   b                   export
D beginObjGroup   pi            10i 0
D   env                           *   const
D   capacityParm                10i 0 value options(*nopass)
D rc              s             10i 0
D capacity        s             10i 0 inz(100)
  /free
  JNIENV_p = env;
  if (%parms >= 2);
   capacity = capacityParm;
  endif;
  rc = PushLocalFrame (JNIENV_p : capacity);
  if (rc <> 0);
   return JNI_GROUP_NOT_ADDED;
  endif;
  return JNI_GROUP_ADDED;
 /end-free
P beginObjGroup   e
 *-------------------------------------------------------------
 * endObjGroup - 最後に開始されたオブジェクトのグループの
 *               終了
 *-------------------------------------------------------------
P endObjGroup     b                   export
D endObjGroup     pi            10i 0
D   env                           *   const
D   refObject
P                     o   class(*java:'java.lang.Object')
D                                 const
D                                 options(*nopass)
D   newObject
P                     o   class(*java:'java.lang.Object')
D                                 options(*nopass)
D retVal          s               o   class(*java:'java.lang.Object')
D refObject       s                   like(refObjectP) inz(*null)
D newObject       s                   like(newObjectP)
 /free
  JNIENV_p = env;
  if %parms() >= 2;
   refObject = refObjectP;
  endif;
  newObject = PopLocalFrame (JNIENV_p : refObject);
  if %parms() >= 3;
    newObjectP = newObject;
  endif;
  return JNI_GROUP_ENDED;
 /end-free
P endObjGroup     e
```

注: このラッパーを呼び出すには、JNI 環境ポインター (下記の 198 ページの『JNI 環境ポインターの入手』で説明) が必要です。

一時オブジェクトの使用が終了したことを Java に伝える

Java コンストラクターを使用してオブジェクトを作成した場合、またはユーザーにオブジェクトを戻す Java メソッドを呼び出した場合、このオブジェクトは、Java のガーベッジ・コレクションがもうユーザーがそのオブジェクトを必要としていな

いと知ったときに破棄されるまでの間だけ、使用可能であることになります。これは、ネイティブ・メソッドの場合にそのネイティブ・メソッドが戻る場合に行なわれますが、非ネイティブ RPG プロシージャーの場合には、ユーザーが明示的に Java に対してもうそのオブジェクトが不要であることを知らせない限りは、行なわれません。これは、RPG ラッパー・プロシージャー freeLocalRef を呼び出すことによって行ないます。

```
CALLP  freeLocalRef (JNIEnv_P : string);
```

図 85 に、freeLocalRef のサンプル・ソース・コードを示します。

```
/*-------------------------------------------------------*/
/* freeLocalRef                                          */
/*-------------------------------------------------------*/
P freeLocalRef...
P                 B                   EXPORT
D freeLocalRef...
D                 PI
D   env                           *    VALUE
D   localRef                      O    CLASS(*JAVA
D                                          : 'java.lang.Object')
D                                      VALUE

  /free
#    jniEnv_P = env;
#    DeleteLocalRef (env : localRef);
#  /end-free

P freeLocalRef...
P                 E
```

図 85. freeLocalRef のソース・コード

注: このラッパーを呼び出すには、JNI 環境ポインター (下記の 198 ページの『JNI 環境ポインターの入手』で説明) が必要です。

オブジェクトを永続にしたいことを Java に伝える

ユーザーにパラメーターとして渡されたか、または Java メソッドあるいはコンストラクターを呼び出すことによって作成されたか、いずれかの Java オブジェクトに対して、ユーザーが参照を行なう場合で、かつそのオブジェクトをユーザーのネイティブ・メソッドが戻った後も使用したい場合は、Java に対してそのオブジェクトを永続または「グローバル」にしたいということを伝える必要があります。これは、RPG ラッパー・プロシージャー getNewGlobalRef を呼び出し、結果をグローバル変数に保管することによって行ないます。

```
EVAL   globalString = getNewGlobalRef (JNIENV_P : string);
```

197 ページの図 86 に、getNewGlobalRef のサンプル・ソース・コードを示します。

```
/*-------------------------------------------------------*/
/* getNewGlobalRef                                       */
/*-------------------------------------------------------*/
P getNewGlobalRef...
P                 B                   EXPORT
D getNewGlobalRef...
D                 PI             O    CLASS(*JAVA
D                                       : 'java.lang.Object')
D   env                          *    VALUE
D   localRef                     O    CLASS(*JAVA
D                                       : 'java.lang.Object')
D                                     VALUE

 /free
          jniEnv_P = env;
     return NewGlobalRef (env : localRef);
 /end-free
P getNewGlobalRef...
P                 E
```

図 86. getNewGlobalRef のソース・コード

注: このラッパーを呼び出すには、JNI 環境ポインター (下記の 198 ページの『JNI 環境ポインターの入手』で説明) が必要です。

永続オブジェクトの使用が終了したことを Java に伝える

グローバル参照を作成した場合で、そのオブジェクトが既に必要ではなくなったことがわかっている場合は、ユーザー側に関しては、このオブジェクトを次回 Java がガーベッジ・コレクションを行なう時点で破棄してよい、ということを Java に対して伝える必要があります。 (オブジェクトは、グローバル参照の対象でなくなり、かつ Java それ自身の中で他から参照されなくなった場合にのみ、破棄されます。) Java に、ユーザーがもうオブジェクトへの参照を必要としなくなったことを伝えるためには、RPG ラッパー・プロシージャー freeGlobalRef を呼び出します。

```
CALLP  freeGlobalRef (JNIEnv_P : globalString);
```

図 87 に、freeGlobalRef のサンプル・ソース・コードを示します。

```
     /*-------------------------------------------------------*/
     /* freeGlobalRef                                         */
     /*-------------------------------------------------------*/
     P freeGlobalRef...
     P                 B                   EXPORT
     D freeGlobalRef...
     D                 PI
     D   env                          *    VALUE
     D   globalRef                    O    CLASS(*JAVA
     D                                         : 'java.lang.Object')
     D                                       VALUE

      /free
         jniEnv_P = env;
         DeleteGlobalRef (env : globalRef);
      /end-free
     P freeGlobalRef...
     P                 E
```

図 87. freeGlobalRef のソース・コード

注: このラッパーを呼び出すには、JNI 環境ポインター (下記の 『JNI 環境ポインターの入手』で説明) が必要です。

Java 仮想マシン (JVM) の作成

ユーザーの RPG コードが Java メソッドを呼び出す準備ができた時点でまだ JVM が作成されていない場合は、RPG は、デフォルトのクラスパスに加えてユーザーの CLASSPATH 環境変数の中のクラスパスを使用して、ユーザーの代わりに JVM を作成します。ただし、ユーザー自身が JVM を作成したい場合は、199 ページの図 88 の最後の部分にあるこのコーディングの例を参照することができます。

JNI 環境ポインターの入手

何らかの JNI 機能を呼び出す必要がある場合は、QSYSINC/QRPGLESRC からの /COPY ファイル JNI を使用します。 JNI 機能のほとんどは、プロシージャー・ポインターを介して呼び出されます。プロシージャー・ポインターは、「JNI 環境ポインター」と呼ばれるポインターにそれ自身が基づいているデータ構造の一部です。このポインターは JNI /COPY ファイルの中で JNIEnv_P と呼ばれています。このポインターを入手するには、JNI ラッパー・プロシージャー getJniEnv を呼び出します。

```
EVAL   JNIEnv_P = getJniEnv();
```

199 ページの図 88 に、getJniEnv のサンプル・ソース・コードを示します。

```
*------------------------------------------------------------------
* getJniEnv - JVM への接続または JVM の開始
*
* パラメーター:
* 1. inputOpts - 文字列の最後の文字で分離された
*                オプションの文字列。
*                例
*                  -Djava.pool.size=800;-Dcompile=none;
*       - JVM が既に開始されている場合は無視されます
*       - クラスパスは CLASSPATH 環境変数から取得されますが、
*         -Djava.class.path オプションを使用して
*         このプロシージャーに渡すこともできます。
*    呼び出しの例:
*    env = getJniEnv()                      // デフォルトを使用
*    env = getJniEnv('-Djava.poolsize=800;'
*                  + '-Dcompile=none;')     // 2 つのオプションを指定
*    env = getJniEnv('-Djava.poolsize=800!'
*                  + '-Dcompile=none!')     // 分離文字として ! を使用
*
*------------------------------------------------------------------
P getJniEnv       b                   export
D getJniEnv       pi              *
D   inputOpts                 65535a   varying const options(*nopass)
D env             s
*   inz(*null)
/free
      env = attachJvm();
      if (env = *null);
       if %parms() = 0
       or %len(inputOpts) = 0;
        env = startJvm();
       else;
        env = startJvm(inputOpts);
       endif;
      endif;
      return env;
/end-free
P getJniEnv       e
```

図 88. getJniEnv のソース・コード (1/6)

```
*------------------------------------------------------------
* startJvm - JVM の開始を試行
*
* パラメーター:
* 1. inputOpts - オプションの文字列は文字列の最初の文字で
*                分離されます
*              - JVM が開始済みの場合は無視されます
*------------------------------------------------------------
  P startJvm        b                   export
D startJvm        pi              *
D   inputOptsP                65535a   varying const options(*nopass)
D initArgs        ds                   likeds(JavaVMInitArgs)
D options         ds                   likeds(JavaVMOption) occurs(10)
D jvm             s                    like(JavaVM_p)
D env             s                    like(JNIENV_p) inz(*null)
D rc              s              10i 0
D len             s              10i 0
D prefix          s             100a   varying
D pOptions        s                *   inz(%addr(options))
D i               s              10i 0
D classpath       S           65535a   varying
   * For handling the input options
D splitChar       s               1a
D pos             s              10i 0
D nextPos         s              10i 0
D inputOpts       s           65535a   varying
D inputOptsPtr    s                *
D freeThisOccur   s                n   dim(%elem(options)) inz(*off)
  /free
  monitor;
  initArgs = *allx'00';
  JNI_GetDefaultJavaVMInitArgs (%addr(initArgs));
  initArgs.version = JNI_VERSION_1_2;
  // 必要な場合は classpath オプションを追加
  classpath = getClasspath();
  if (%len(classpath) > 0);
   initArgs.nOptions = initArgs.nOptions + 1;
   %occur(options) = initArgs.nOptions;
   freeThisOccur(initArgs.nOptions) = *on;
   options = *allx'00';

   prefix = '-Djava.class.path=:';
   len = %len(prefix) + %len(classpath) + 1;
   options.optionString = %alloc (len);
   %str(options.optionString : len) =
                                          cvtToAscii(prefix)
              + cvtToAscii(classpath);
  endif;
```

図 88. getJniEnv のソース・コード (2/6)

```
      // そのほかの受け渡されたオプションを追加
    if  %parms > 0
 and %len(inputOptsP) > 0;
  inputOpts = cvtToAscii(inputOptsP);
  inputOptsPtr = %addr(inputOpts) + 2;
  splitChar = %subst(inputOpts : %len(inputOpts) : 1);
  pos = 1;
  dow pos <= %len(inputOpts);
   nextPos = %scan(splitChar : inputOpts : pos);
   len = nextPos - pos;
   %subst(inputOpts : nextPos : 1) = x'00';
   initArgs.nOptions = initArgs.nOptions + 1;
   %occur(options) = initArgs.nOptions;
   options = *allx'00';
   options.optionString = inputOptsPtr + pos - 1;

   pos = nextPos + 1;
  enddo;
 endif;

    if initArgs.nOptions > 0;
  initArgs.options = pOptions;
 endif;

 rc = JNI_CreateJavaVM (jvm : env : %addr(initArgs));
 if (rc <> 0);
  env = *null;
 endif;

    rc = JNI_CreateJavaVM (jvm : env : %addr(initArgs));
 if (rc <> 0);
  env = *null;
 endif;

 on-error;
  env = *null;
 endmon;

 // このオプション用に割り振られたストレージすべてを解放

 for i = 1 to initArgs.nOptions;
  if (freeThisOccur(i));
    %occur(options) = i;
    dealloc(n) options.optionString;
  endif;
 endfor;

 return env;

/end-free
P startJvm        e
```

図 88. getJniEnv のソース・コード (3/6)

```
*------------------------------------------------------------
* attachJvm - JVM への接続を試行
*------------------------------------------------------------
P attachJvm       b                   export
D attachJvm       pi              *
D attachArgs      ds                  likeds(JavaVMAttachArgs)
D jvm             s                   like(JavaVM_p) dim(1)
D nVms            s                   like(jsize)
D env             s               *   inz(*null)
D rc              s            10i 0
  /free

 monitor;
  rc = JNI_GetCreatedJavaVMs(jvm : 1 : nVms);
  if (rc <> 0);
   return *null;
  endif;
  if (nVms = 0);
   return *null;
  endif;
  JavaVM_P = jvm(1);
  attachArgs = *allx'00';
  attachArgs.version = JNI_VERSION_1_2;
  rc = AttachCurrentThread (jvm(1) : env : %addr(attachArgs));
  if (rc <> 0);
   env = *null;
  endif;

 on-error;
  env = *null;
 endmon;

   return env;
  /end-free
P attachJvm       e
```

図 88. getJniEnv のソース・コード (4/6)

```
*------------------------------------------------------------
* getClasspath - CLASSPATH 環境変数を取得
*------------------------------------------------------------
P getClasspath    B                   export
D getClasspath    PI         65535A   varying
D Qp0zGetEnvNoInit...
D                 PR              *   extproc('Qp0zGetEnvNoInit')
D   name                          *   value options(*string)
D envVarP         S               *
/free
 envvarP = Qp0zGetEnvNoInit('CLASSPATH');
 if (envvarP = *NULL);
  return '';
 else;
  return %str(envvarP : 65535);
 endif;
/end-free
P getClasspath    E
```

図 88. getJniEnv のソース・コード (5/6)

```
*---------------------------------------------------------------
* cvtToAscii - EBCDIC から ASCII への変換
*---------------------------------------------------------------
P cvtToAscii      B                   export
D cvtToAscii      PI         65535A   varying
D   input                    65535A   const varying
D QDCXLATE        PR                  extpgm('QDCXLATE')
D   len                         5P 0 const
D   cnvData                  65535A   options(*varsize)
D   cnvTbl                      10A   const
D   cnvLib                      10A   const
D retval          S                   like(input)
D retvalRef       S               1A  based(retvalPtr)
/free
 retval = input;
 retvalPtr = %addr(retval) + 2;   // set ptr after the length part
 QDCXLATE(%len(retval): retvalRef : 'QASCII': 'QSYS');
 return retval;
/end-free

P cvtToAscii      E
```

図 88. getJniEnv のソース・コード (6/6)

JNI 例外の処理

ILE RPG では、例外が発生すると例外メッセージを出して知らせます。プログラムは明示的に例外をチェックする必要はありません。代わりに、例外が発生した場合に制御を入手する例外処理プログラムをコーディングすることができます。ユーザーは、ユーザー自身の JNI 呼び出しを行なう時に JNI 例外を自分で処理する必要があるだけです。JNI 機能の呼び出しの結果、処理されない Java 例外が発生した場合、付随する例外メッセージはありません。その代わり、JNI プログラマーは JNI 機能を呼び出した後は毎回、例外が発生したかどうかをチェックする必要があります。これは、Java Exception オブジェクト (または JNI の値が 0 になる Java null オブジェクト) を戻す ExceptionOccurred JNI 機能を呼び出すことにより行ないます。いったん例外が発生したことを判別したら、行なえる JNI 呼び出しは ExceptionClear と ExceptionDescribe のみです。ExceptionClear を実行した後、ユーザーは解放されて再度 JNI 呼び出しを行なえるようになります。ExceptionClear を呼び出す前に非例外 JNI 呼び出しを行なった場合は、例外は消失し、もはや詳細を知ることができなくなります。RPG は JNI 例外を必ず RPG 例外に変換します (その時実行中だった RPG 機能に応じて、RNX030x メッセージのいずれかが出されます)。

ヒント

前述の JNI ラッパー・プロシージャーのユーザー独自のバージョンに、このようなタイプの例外処理コードを組み込むこともできます。

追加の考慮事項

一般的な実行時エラー

コンパイラーはコンパイル時にはクラスの解決を行ないません。実行時にクラスが見付からない場合、実行時エラーが発生します。これは、Java 環境から *UnresolvedLinkException* オブジェクトを受け取ったことによって示されます。

コンパイラーは、コンパイル時にはパラメーターのタイプのチェックは行ないません。プロトタイプと呼び出されるメソッドとの間に矛盾があった場合、エラーを実行時に受け取ることになります。

デバッグのヒント

Java オブジェクトは、RPG においてはオブジェクト参照子として見られます。このオブジェクト参照子は整数値で、ポインターのように動作します。通常のオブジェクト参照子は正の値で、1 から始まって順に割り当てられます。JNI 機能 NewGlobalRef を使用して作成することができるグローバル参照は、負の値になります。これらの値は、最小の負の数 (-2147483647) から始まって順に割り当てられます。

通常、これらの値は RPG コードの中では見えませんが、この情報が RPG コードをデバッグする時には役に立つことがあります。

RPG での String オブジェクトの作成

String オブジェクトを Java メソッドに渡す必要がある場合は、次のようにして String オブジェクトを作成することができます。

```
D newString       PR              O   EXTPROC(*JAVA
D                                     : 'java.lang.String'
D                                     : *CONSTRUCTOR)
D   value                     65535A   CONST VARYING
D string          S                   like(jstring)
  /free
     string = newString ('abcde');
     ...
  /end-free
```

ストリングを、UCS-2 データまたはグラフィック・データとして指定して作成したい場合は、以下のコーディングを使用します。

```
D newStringC      PR              O   EXTPROC(*JAVA
D                                     : 'java.lang.String'
D                                     : *CONSTRUCTOR)
D   value                     16383C   CONST VARYING
D string          S                   like(jstring)
D graphicData     S               15G
D ucs2Data        S              100C
  /free
     string = newStringC (%UCS2(graphicData));
     ...
     string = newStringC (ucs2Data);
  /end-free
```

呼び出された Java メソッドからスローされた例外に関する情報の取得

RPG が呼び出した Java メソッドが例外で終了した場合、RPG は Java 例外とシグナル・エスケープ・メッセージ RNX0301 を処理します。このメッセージには例外

のストリング値が含まれますが、Java が例外で終了するメソッドを呼び出した場合に通常使用可能なトレース情報は含まれません。

Java 例外トレース情報を参照するには、次のコマンドを実行します。

1. ADDENVVAR ENVVAR(QIBM_USE_DESCRIPTOR_STDIO) VALUE('Y')

   注: このステップは JVM を開始する前に行う必要があります。

2. SystemProperties.default ファイル内の os400.stderr オプションが file:myfilename に設定されていることを確認してください。例えば os400.stderr=file:/home/mydir/stderr.txt などです。 188 ページの『Java 仮想マシンのセットアップ方法の制御』を参照してください。

   注: このステップは JVM を開始する前に行う必要があります。

3. ADDENVVAR ENVVAR(QIBM_RPG_JAVA_EXCP_TRACE) VALUE('Y')

   注: このステップはいつ実行しても構いません。 RPG が行っている例外トレースを停止するには、環境変数を除去するか、または環境変数に 'Y' 以外の値を設定します。

4. 例外が発生すると、トレース情報は os400.stderr オプションで指定したファイルに書き込まれます。

拡張 JNI コーディング

RPG IV コンパイラーの Java メソッド呼び出しのサポート、および RPG ネイティブ・メソッドの作成のサポートでは、ほとんどすべての JNI コーディングは、RPG プログラマーからは見えません。ただし、RPG のサポートが最も効果的であるとは限りません。例えば、RPG は呼び出しの時およびネイティブ・メソッドの入り口と出口において必ず、RPG と Java との間の配列の変換を行ないますが、パフォーマンスを向上させるにはユーザー自身で配列の変換を行なった方がよい場合もあります。

RPG のサポートは、ユーザーに Java メソッドへのアクセスを提供しているに過ぎません。クラスの中のフィールドにアクセスしたい場合は Java クラスに対して「入手」メソッドと「設定」メソッドを追加するか、または JNI コーディングをすることが必要になります (206 ページの『Java クラスの中のフィールドへのアクセス』を参照)。

206 ページの図 89 は、RPG における JNI 呼び出しの例です。

```
 /COPY JNI
D objectId        s                   like(jobject)
D methodId        s                   like(jmethodID)
D string          s                   like(jstring)
D parms           s                   like(jvalue) dim(2)

  /free
     jvalue_P = %addr(parms(1));
     jvalue.i = 10;
     jvalue_P = %addr(parms(2));
     jvalue.l = string;
     CallVoidMethodA (JNIEnv_P : objectId : methodId : parms);
  /end-free
```

図 89. RPG における JNI 呼び出し

ポインター JNIEnv_P およびポインター jvalue_P は JNI /COPY ファイルの中で定義されていることに注意してください。

Java 文字データの変換

Java では、文字データは EBCDIC ではなく ASCII であるため、FindClass などの JNI 機能を呼び出すには、クラス名、メソッド名、およびフィールド名は必ず ASCII にしておく必要があります。Java から来る文字データも ASCII なので、これをユーザーの RPG プログラムの中で使用するには、おそらく EBCDIC への変換をすることになります。RPG コンパイラーはこれらの変換をユーザーに代わって行ないますが、ユーザー自身が JNI 呼び出しを行なう場合は ASCII と EBSDIC 間の変換を行なう必要があります。

Java クラスの中のフィールドへのアクセス

RPG がサポートするのは Java メソッドの呼び出しのみです。Java フィールドへのアクセスはサポートしません。通常、フィールドは「get」メソッドおよび「set」メソッドを通じてアクセスされますが、JNI 呼び出しを使用してフィールドにアクセスすることも可能です。ここでは、 Java クラスまたはオブジェクトのフィールドへのアクセスに必要な JNI 呼び出しの例を示します。

注: この例は JNI を使用する例となることを目的としたものです。「get」メソッドや「set」メソッドを使用するよりも、フィールドに直接にアクセスすることを推奨する、という意味ではありません。

```
 *---------------------------------------------------------------------
 * この例は、クラスまたはオブジェクトのフィールドにアクセスするための
 * JNI の使用法を示しています。
 *
 * このプログラムは Rectangle オブジェクトを作成し、JNI 呼び出しを
 * 使用して、width 変数と height 変数に直接アクセスします。
 *
 * この特定の例においては、getWidth()、getHeight、
 * setWidth() および setHeight() メソッドを使用して
 * これらのフィールドにアクセスし、JNI 呼び出しの使用は避けます。
 *---------------------------------------------------------------------
H THREAD(*SERIALIZE)
 /DEFINE JNI_COPY_FIELD_FUNCTIONS
 /COPY JNI
 /COPY JNIRPG_PR
 *---------------------------------------------------------------------
 * JAVA クラスおよびメソッド
 *---------------------------------------------------------------------
```

図 90. Java クラスおよびオブジェクトのフィールドにアクセスするための JNI の使用 (1/3)

```
D Rectangle       C                   'java.awt.Rectangle'
D NewRectangle    PR              O   EXTPROC(*JAVA
D                                     : Rectangle
D                                     : *CONSTRUCTOR)
D     x                       10I 0 VALUE
D     y                       10I 0 VALUE
D     width                   10I 0 VALUE
D     height                  10I 0 VALUE
 *---------------------------------------------------------------
 * JAVA 名の ASCII 表記を用いた定数
 *---------------------------------------------------------------
 * これらの値を判別する 1 つの方法は、文字値を UCS-2 に変換するために
 * %UCS2 を使用して、結果をデバッガーの中で
 * 16 進で表示するという方法です。
 *
 * ASCII 値は UCS-2 文字に 2 バイトおきに入っています。
 *
 *     例えば、%UCS2('abc') = X'006100620063'
 *                              --  --  --
 *     'abc' の ASCII 表記は X'616263' です。
 *---------------------------------------------------------------
D ASCII_I         C                   x'49'
D ASCII_x         C                   x'78'
D ASCII_y         C                   x'79'
D ASCII_width     C                   X'7769647468'
D ASCII_height    C                   X'686569676874'
 * JNI はスラッシュを区切り文字として使用するため、これは
 *「java/awt/Rectangle」であって「java.awt.Rectangle」ではないことに注意。
D ASCII_Rectangle...
D                 C                   X'6A6176612F6177742F52656-
D                                     374616E676C65'
 *---------------------------------------------------------------
 * 取り消し処理
 *---------------------------------------------------------------
D EnableCanHdlr   PR                  EXTPROC('CEERTX')
D   Handler                       *   CONST PROCPTR
D   CommArea                      *   CONST OPTIONS(*OMIT)
D   Feedback                    12A   OPTIONS(*OMIT)
D CanHdlr         PR
D   CommArea                      *   CONST
 *---------------------------------------------------------------
 * 変数およびプロシージャー
 *---------------------------------------------------------------
D rect            s               O   CLASS(*JAVA : Rectangle)
D x               S             10I 0
D y               S             10I 0
D rectClass       S                   LIKE(jclass)
D fieldId         S                   LIKE(jfieldID)
D msg             S             52A
D Cleanup         PR
 *---------------------------------------------------------------
 * 取り消し処理プログラムを使用可能にし終結処理が完了したことを確認
 *---------------------------------------------------------------
C                   CALLP     EnableCanHdlr (%PADDR(CanHdlr)
C                                          : *OMIT : *OMIT)
 *---------------------------------------------------------------
 * x,y 座標 (5, 15) を使用して、幅 100 で高さが 200 の
 * 新しい Rectangle を作成します。
 *---------------------------------------------------------------
C                   EVAL      rect = NewRectangle (5 : 15 : 100 : 200)
 *---------------------------------------------------------------
 * Rectangle のフィールドにアクセスするための JNI 機能を呼び出す準備
 *---------------------------------------------------------------
```

図 90. Java クラスおよびオブジェクトのフィールドにアクセスするための JNI の使用 (2/3)

```
C                   EVAL      JNIEnv_P = getJniEnv ()
C                   EVAL      rectClass = FindClass (JNIEnv_P
C                                                  : ASCII_Rectangle)
 *-------------------------------------------------------------------
 * Rectangle の幅および高さを検索するための JNI 機能の呼び出し
 *-------------------------------------------------------------------
C                   eval      fieldId = GetFieldID (JNIEnv_P
C                                                 : rectClass
C                                                 : ASCII_width
C                                                 : ASCII_I)
C                   eval      width = GetIntField (JNIEnv_P
C                                                  : rect
C                                                  : fieldId)
C                   eval      fieldId = GetFieldID (JNIEnv_P
C                                                 : rectClass
C                                                 : ASCII_height
C                                                 : ASCII_I)
C                   eval      height = GetIntField (JNIEnv_P
C                                                  : rect
C                                                  : fieldId)
C                   eval      msg = 'The rectangle has dimensions ('
C                                 + %trim(%editc(width : '1'))
C                                 + ', '
C                                 + %trim(%editc(height : '1'))
C                                 + ')'
C     msg           dsply
 *-------------------------------------------------------------------
 * Cleanup プロシージャーを呼び出します。
 *-------------------------------------------------------------------
C                   callp     Cleanup()
C                   eval      *INLR = '1'
 *-------------------------------------------------------------------
 * Cleanup。* - 必要であればオブジェクトを解放します。
 *-------------------------------------------------------------------
P Cleanup         B
C                   if        rect <> *NULL and
C                             JNIEnv_P <> *NULL
C                   callp     DeleteLocalRef(JNIEnv_P : rect)
C                   endif
C                   eval      rect = *NULL
C                   eval      JNIEnv_P = *NULL
P Cleanup         E
 *-------------------------------------------------------------------
 * 取り消し処理プログラム。終結処理が完了したことを確認。
 *-------------------------------------------------------------------
P CanHdlr         B
D CanHdlr         PI
D   CommArea                      *   CONST
C                   callp     Cleanup()
P CanHdlr         E
```

図 90. Java クラスおよびオブジェクトのフィールドにアクセスするための JNI の使用 (3/3)

RPG *JAVA プロトタイプではなく JNI を使用する Java メソッドの呼び出し

最初の 3 つのパラメーターは、必ず次と同じになります。

1. JNI 環境ポインター
2. オブジェクト (インスタンス・メソッドの場合) またはクラス (静的メソッドの場合)

3. メソッド

メソッド固有のパラメーターは、これら 3 つのパラメーターの後に、3 つの異なる方法のいずれかでコーディングされます。例えば、メソッドが値を戻さない場合 (戻りタイプは「void」)。

CallVoidMethod:
同じメソッドを何度も呼び出そうとしている場合はこの方法を選択します。これはメソッドを非常に容易に呼び出せるからです。この方法では、パラメーターは普通に渡されることを想定しています。この JNI 機能を呼び出すには、RPG は CallVoidMethod プロトタイプを JNI/COPY ファイルからコピーして、追加のパラメーターをコーディングします。例えば

```
D CallMyMethod    PR                  EXTPROC(*CWIDEN
D                                     : JNINativeInterface.
D                                       CallVoidMethod_P)
D env                                 LIKE(JNIEnv_P) VALUE
D obj                                 LIKE(jobject) VALUE
D methodID                            LIKE(jmethodID) VALUE
D len                                 LIKE(jint) VALUE
D str                                 LIKE(jstring) CONST

...

CallMyMethod (JNIEnv_P : objectId : methodId : 10 : string);
```

図 91. CallVoidMethod を呼び出すサンプル RPG コーディング

CallVoidMethodA:
メソッドを呼び出す場合に別のプロトタイプを作成したくない場合には、この方法を選択します。この方法では、配列の各要素が 1 つのパラメーターを保持する、jvalue 構造の配列を想定しています。前述の 206 ページの図 89 はこの例です。

CallVoidMethodV:
これは RPG コードの中では使用してはなりません。これは C 構成を想定したもので、RPG でコーディングするのはきわめて不向きです。

実際にどの機能を呼び出すかは、戻り値のタイプによって決まります。例えば、メソッドが整数を戻す場合であれば、 CallIntMethodA を使用することになります。これらの機能のクラスおよび methodID パラメーターを入手するには、FindClass と GetMethodID または GetStaticMethodID を使用します。

注: JNI を直接に呼び出す場合、クラス名はピリオド (.) ではなくスラッシュ (/) を区切り文字として使用して指定する必要があります。例えば、「java/lang/String」であって、「java.lang.String」ではありません。

PCML の使用による Java からの RPG プログラムの呼び出し

Java から RPG プログラムまたはプロシージャーを呼び出す場合には、 RPG プログラムまたはプロシージャーのパラメーターを記述したプログラム呼び出しマークアップ言語 (PCML) ソース・ファイルを使用することができます。 Java アプリケーションは、 PCML ソース・ファイルの参照を持つ ProgramCallDocument オブジェクトを作成することで PCML を使用することができます。

INFOSTMF コンパイラー・パラメーターと一緒に PGMINFO(*PCML) コンパイラー・パラメーターを指定して、生成された PCML を受け取る IFS 出力ファイルの名前を指定すると、 ILE RPG コンパイラーは、ILE RPG プログラムまたはモジュールの PCML ソースを生成します。 CRTBNDRPG の場合、 PCML は *ENTRY PLIST つまりメイン・プロシージャーの「プロシージャー・インターフェース」の内容に基づいて生成されます。 CRTRPGMOD の場合も、 PCML はエクスポート・サブプロシージャー (Java 固有のメソッドは除く) の「プロシージャー・インターフェース」に基づいて生成されます。

CRTRPGMOD を使用し、サービス・プログラムを作成する場合は、ProgramCallDocument クラスの setPath(String) メソッドを使用して、 Java コードでサービス・プログラムを指定します。次に、例を示します。

```
AS400 as400;
ProgramCallDocument pcd;
String path = "/QSYS.LIB/MYLIB.LIB/MYSRVPGM.SRVPGM";
as400 = new AS400 ();
pcd = new ProgramCallDocument (as400, "myModule");
pcd.setPath ("MYFUNCTION", path);
pcd.setValue ("MYFUNCTION.PARM1", "abc");
rc = pcd.callProgram("MYFUNCTION");
```

PCML の制約事項

PCML では、パラメーターおよび戻り値の型について、次の制約事項が適用されます。

- PCML では次のデータ型はサポートされません。
  - 日付
  - 時刻
  - タイム・スタンプ
  - ポインター
  - プロシージャー・ポインター
  - 1 バイト整数
  - 8 バイト符号なし整数
- 数値で渡される戻り値およびパラメーターとして有効なのは、 4 バイト整数 (10i 0) のみです。
- 可変長配列、および可変長サブフィールドを含むデータ構造はサポートされません。
- *ENTRY PLIST のパラメーターとしてデータ構造を使用する場合、またはプロトタイプされたパラメーターを LIKEDS で定義する場合は、次の PCML 制約事項が適用されます。
  - データ構造内で、サブフィールドの重複は認められません。
  - サブフィールドは、正しい順序でコーディングしなければなりません。つまり、各サブフィールドの開始位置は、直前のサブフィールドの終了位置の後ろでなければなりません。
  - サブフィールドとサブフィールドの間にギャップがある場合、構造に対して生成される PCML は、型 ″char″ の ″_unnamed_1″、″_unnamed_2″ などの名前のサブフィールドから構成されます。

- RPG に、出力専用パラメーターの概念はありません。 CONST または VALUE がコーディングされていないパラメーターの使用法は、 ″inputoutput″ です。 inputoutput パラメーターの場合、ProgramCallDocument クラスは、そのパラメーターの入力値が設定されていなければプログラムを呼び出すことができません。 パラメーターが本当に出力パラメーターの場合は、 PCML を編集して ″inputoutput″ を ″output″ に変更する必要があります。

上記制約条件のいずれかに違反するプログラムまたはモジュールで PCML を生成すると、コンパイルは失敗します。 PCML は生成されますが、コメントとしてエラー・メッセージが含まれます。例えば、パラメーターとして Date フィールドを使用すると、そのパラメーターに対する PCML は次のようになります。

```
<data name="DATE" type="   " length="10" usage="input" />
<!-- Error: unsupported data type -->
```

第 3 部 デバッグおよび例外処理

この部では、以下の方法について説明します。

- ILE ソース・デバッガーの使用による、ILE アプリケーション・プログラムのデバッグ
- 例外を処理するプログラムの作成
- ダンプの入手

第 12 章 プログラムのデバッグ

デバッグによって、プログラム内の実行時エラーを検出、診断、および除去することができます。ILE ソース・デバッガーを使用して、ILE プログラムおよび OPM プログラムをデバッグすることができます。

WebSphere Development Studio Client for iSeries を使用します。これは推奨される方法であり、プログラムのデバッグに関する説明は、製品のオンライン・ヘルプに記載されています。統合 iSeries デバッガーを使用すると、サーバー上で実行されるプログラムをワークステーション上のグラフィカル・ユーザー・インターフェースからデバッグできます。また、デバッガーを実行する前にソース内にブレークポイントを直接設定することもできます。統合 iSeries デバッガー・クライアント・ユーザー・インターフェースでは、プログラム実行を制御することもできます。例えば、プログラムの実行、行の設定、ウォッチができ、またエントリー・ポイント・ブレークポイントの設定、プログラム命令のステップスルー、変数の値の表示、およびコール・スタック内容の表示ができます。また、異なる言語で記述された複数のアプリケーションであっても、単一のデバッガー・ウィンドウでデバッグできます。デバッグする各セッションは、「デバッグ」ビュー内に個別に表示されます。

この章では、ILE ソース・デバッガーを使用して次のことを行う方法について説明します。

- デバッグのために ILE RPG プログラムを準備する
- デバッグ・セッションを開始する
- デバッグ・セッションでプログラムを追加および除去する
- デバッグ・セッションからプログラム・ソースを表示する
- ブレークポイントおよびウォッチ条件を設定および除去する
- プログラムをステップスルーする
- フィールドの値を表示し変更する
- フィールドの属性を表示する
- 簡略名をフィールド、式、またはデバッグ・コマンドに対応させる

プログラムのデバッグおよびテスト中、既存の実際のデータが影響を受けることのないように、プログラムがテスト・データの入っているテスト・ライブラリーを指定するようにライブラリー・リストを変更してください。

次のコマンドのいずれかを使用して、プロダクション・ライブラリー中のデータベース・ファイルが、意図せずに変更されてしまうのを防ぐことができます。

- デバッグ開始 (STRDBG) コマンドを使用して、UPDPROD パラメーターをデフォルトの *NO のままにしておきます。
- デバッグ変更 (CHGDBG) コマンドを使用して、 UPDPROD パラメーターを *NO に指定します。
- 「モジュール・ソースの表示」画面の SET デバッグ・コマンドを使用して、UPDPROD NO を指定します。

(プログラムまたはサービス・プログラムのデバッグに必要な権限、および最適化レベルの効果を含め) ILE ソース・デバッガーについて詳しくは、「*ILE 概念*」のデバッグの章を参照してください。

デバッガーの使用に慣れていない場合は、以下のステップに従ってプログラムを作成およびデバッグしてください。プログラム PROOF 用のソースはすべてのシステムの QGPL 内にあります。

1. ===> CRTBNDRPG QTEMP/PROOF DBGVIEW(*ALL)
2. ===> STRDBG QTEMP/PROOF
3. 計算行にカーソルを置き、F6 を押してブレークポイントを設定します。
4. F12 で「DSPMODSRC」画面を終了します。
5. ===> CALL QTEMP/PROOF

   ブレークポイント行が強調表示されたソースが再度表示されます。
6. カーソルをプログラム・ソース (定義仕様書、入力仕様書、計算仕様書または出力仕様書) 内の変数上に移動し、F11 を押します。画面の下部に変数の値が表示されます。
7. F10 を押してプログラムの残りをステップスルーするか、または F12 を押して最後まで実行します。
8.

ブレークポイントを設定した後に、プログラムを直接呼び出す必要はありません。最終的にプログラムを呼び出すことになるアプリケーションを開始します。

プログラム全体をステップスルーする場合には、入出力仕様書をステップスルーすることになります。入出力仕様書をスキップするには、ヘッダー仕様書またはプログラムのコンパイル時に OPTION(*NODEBUGIO) を指定します。

これらのステップの詳細は本章内で後述します。

ILE ソース・デバッガー

ILE ソース・デバッガーはプログラム・オブジェクトおよびサービス・プログラムのエラーを検出し、それらを除去するために使われます。デバッグ・データを含む ILE プログラムでデバッグ・コマンドを使用して、次のことを行うことができます。

- プログラム・ソースを表示するか、あるいはデバッグ・ビューを変更する。
- ブレークポイントおよびウォッチ条件を設定および除去する。
- 指定した数のステートメントをステップスルーする。
- フィールド、構造、および配列の値を表示または変更する。
- 簡略名をフィールド、式、またはデバッグ・コマンドに対応させる。

ソース・デバッガーを使えるようになるには、CRTRPGMOD または CRTBNDRPG を使ってモジュール・オブジェクトやプログラム・オブジェクトを作る時にデバッグ・ビューを選択しなければなりません。デバッガーの開始後に、ブレークポイントを設定して、プログラムを呼び出すことができます。

ブレークポイントまたはステップ・コマンドのためにプログラムが停止した時には、プログラムが停止した個所で、関連するモジュール・オブジェクトのビューが画面に表示されます。この時点でフィールド値の表示や変更などの他の処置を実行することができます。

注: プログラムが最適化されていた場合には、フィールドを表示することはできますが、その値は信頼できないことがあります。フィールドまたはデータ構造の内容が正しい (現在の) 値であるようにするためには、適切な定義仕様書に NOOPT キーワードを指定してください。最適化レベルを変更するには、98 ページの『最適化レベルの変更』を参照してください。

デバッグ・コマンド

ILE ソース・デバッガーには使用可能な多くのデバッグ・コマンドがあります。デバッグ・コマンドおよびそのパラメーターは、「モジュール・ソースの表示」画面や式評価画面の最下行に表示されるデバッグ・コマンド行に入力します。これらのコマンドは、大文字、小文字、あるいはその組み合わせで入力することができます。

注: デバッグ・コマンド入力行に入力するデバッグ・コマンドは CL コマンドではありません。

デバッグ・コマンドを以下にリストします。

コマンド	説明
ATTR	変数の属性を表示することができます。属性は、デバッグ記号テーブルに記録された変数のサイズおよびタイプです。
BREAK	テスト中のプログラムのある位置に無条件または条件付きいずれかのブレークポイントを入力することができます。条件付きジョブ・ブレークポイントを入力するには、BREAK *行番号* WHEN *式* を使います。
CLEAR	条件付きブレークポイントおよび無条件ブレークポイントを除去するか、あるいは 1 つまたはすべての活動状態のウォッチ条件を除去することができます。
DISPLAY	EQUATE コマンドを使用して、割り当てられた名前および定義を表示することができます。また、現在「モジュール・ソースの表示」画面に示されているもの以外のソース・モジュールを表示することができます。モジュール・オブジェクトは現行のプログラム・オブジェクト中に存在していなければなりません。
EQUATE	簡単に使用できるよう、式、変数、またはデバッグ・コマンドを名前に割り当てることができます。
EVAL	変数の値を表示または変更したり、あるいは式、レコード、構造、または配列の値を表示することができます。
QUAL	後続の EVAL または WATCH コマンドに現れる変数の有効範囲を定義することができます。現在、これは ILE RPG には適用されません。
SET	実動実行ファイルを更新する機能、検出操作は大文字小文字の区別

をするかどうかの指定、あるいは OPM ソース・デバッグ・サポートを使用可能にするかどうかの指定など、デバッグ・オプションを変更することができます。

STEP デバッグ中のプロシージャーの 1 つ以上のステートメントを実行することができます。

TBREAK テスト中のプログラムのある位置に、条件付きまたは無条件いずれかの現行スレッドのブレークポイントを入力することができます。

THREAD 「デバッグされたスレッドの処理」画面で作業を表示するか、現行スレッドを変更することができます。

WATCH 指定された記憶域位置の内容を現行値から変更する際にブレークポイントを要求することができます。

FIND 現在表示されているモジュールで正方向または逆方向の検索を行い、指定した行番号、ストリング、またはテキストを見付けます。

UP 表示されているソースのウィンドウを入力した量だけ、ビューの先頭の方に移動します。

DOWN 表示されているソースのウィンドウを入力した量だけ、ビューの終わりの方に移動します。

LEFT 表示されているソースのウィンドウを入力した文字数分だけ、左に移動します。

RIGHT 表示されているソースのウィンドウを入力した文字数分だけ、右に移動します。

TOP 最初の行を表示するように、ビューを位置付けます。

BOTTOM 最後の行を表示するように、ビューを位置付けます。

NEXT 現在表示されているソースの次のブレークポイントに、ビューを位置付けます。

PREVIOUS 現在表示されているソースの前のブレークポイントに、ビューを位置付けます。

HELP 使用可能なソース・デバッガー・コマンドについてのオンライン・ヘルプ情報を表示します。

ILE ソース・デバッガーのオンライン・ヘルプには、デバッグ・コマンドの記述、それらの可能な簡略形の説明、各コマンドの構文図が示されます。また、ソース・デバッガーを使用して変数の表示や変更をする、各 ILE 言語の例も示されています。

ILE RPG のオンライン・ヘルプ情報にアクセスするには、以下のステップに従います。

1. STRDBG ライブラリー名/プログラム名を入力します。ここで*プログラム名* とはライブラリー、*ライブラリー名* のデバッグ・データを持つ任意の ILE プログラムです。
2. ステップ 1 につづいてソース・ビューが表示されない場合には、ソース・ビューを表示するために DSPMODSRC を入力します。
3. PF1 (ヘルプ) を押します。

4. EVAL コマンド・ヘルプを出すには、カーソルを EVAL に置き、実行キーを押します。
5. 式のヘルプを出すには、カーソルを式に置き、実行キーを押します。
6. RPG 言語例を出すには、カーソルを RPG 言語に置き、実行キーを押します。
7. 表示されるヘルプ画面から、変数の表示、テーブルの表示、複数回繰り返しデータ構造の表示などの、RPG 関連のトピックを選択することができます。

デバッグのためのプログラムの準備

プログラムまたはモジュールをデバッグする場合には、そのプログラムまたはモジュールに使用可能なデバッグ・データがなければなりません。デバッグ・データはコンパイル時に作成されるので、CRTBNDRPG または CRTRPGMOD を使用してこれを作成する時に、モジュールにデバッグ・データを入れるかどうかを指定します。コンパイル時にどのタイプのデータを作成するか (作成する場合) を指示するために、これらのコマンドで DBGVIEW パラメーターを使用します。

モジュールに関連付けることができるデバッグ・データのタイプは、**デバッグ・ビュー**として参照されます。デバッグしたい各モジュールに次のビューの 1 つを作成することができます。それらは次のとおりです。

- ルート・ソース・ビュー
- コピー・ソース・ビュー
- リスト・ビュー
- ステートメント・ビュー

CRTBNDRPG と CRTRPGMOD のデフォルト値は、共にステートメント・ビューを作成します。このビューでは、前のリリースに最も近いレベルのデバッグ・サポートが提供されます。

モジュールにデバッグ・データを含めたくない場合、あるいはコンパイル時刻を短縮したい場合には、モジュールが作成される時に DBGVIEW(*NONE) を指定してください。しかし、デバッグ・データが使用可能でない時には、定様式ダンプはプログラム変数の値をリストしません。

含まれるデバッグ・データのタイプによって、モジュールまたはプログラムの記憶域所要量が、いくぶん変化することに注意してください。DBGVIEW パラメーターの次の値は、2 次記憶域所要量についてその影響が大きくなる順序でリストしてあります。

1. *NONE
2. *STMT
3. *SOURCE
4. *COPY
5. *LIST
6. *ALL

デバッグ・データを持つモジュールを作成し、それをプログラム・オブジェクト (*PGM) にバインドした後で、プログラムのデバッグを開始することができます。

注: ILE ソース・デバッガーを使ってデバッグするには、OPM プログラムは OPTION(*SRCDBG) または OPTION(*LSTDBG) を指定してコンパイルする必要があります。詳しくは、223 ページの『ILE ソース・デバッガーの開始』を参照してください。

デバッグ・ビューは次の表にまとめてあります。

表 28. デバッグ・ビュー

デバッグ・ビュー	デバッグ・データ	DBGVIEW パラメーター値
なし	デバッグ・データなし	*NONE
ステートメント・ビュー (デフォルト値)	ソースは表示されない (コンパイラー・リストのソース・セクションのステートメント番号を使う)	*STMT
ルート・ソース・ビュー	ルート・ソース・メンバーの情報	*SOURCE
コピー・ソース・ビュー	ルート・ソース・メンバーおよび /COPY メンバーの情報	*COPY
リスト・ビュー	コンパイラー・リスト (OPTION パラメーターによって異なる)	*LIST
すべて	ルート・ソース、コピー・ソース、およびリスト・ビューからのデータ	*ALL

ルート・ソース・ビューの作成

ルート・ソース・ビューは、ルート・ソース・メンバーからのテキストを含みます。このビューには /COPY メンバーは入っていません。さらに、ルート・ソース・メンバーが DDM ファイルである場合には、このビューは使用することができません。

モジュールの作成時に、CRTRPGMOD または CRTBNDRPG コマンドの DBGVIEW パラメーターに *SOURCE、*COPY または *ALL オプションを使用することによってモジュールをデバッグするためのルート・ソース・ビューを作成します。

コンパイラーは、モジュール・オブジェクト (*MODULE) のコンパイル中に、ルート・ソース・ビューを作成します。ルート・ソース・ビューは、メンバーのテキストをモジュール・オブジェクトにコピーするのでなく、ルート・ソース・メンバーのテキストの位置の参照を使用して作成されます。このため、これらのメンバーのモジュール作成とこれらのメンバーから作成されたモジュールのデバッグの間で、ルート・ソース・メンバーの変更、名前の付け直し、あるいは移動をしてはいけません。これを行った場合には、これらのソース・メンバーのビューを使用できなくなることがあります。

例えば、CRTBNDRPG を使用してプログラム DEBUGEX のルート・ソース・ビューを作成するためには、次を入力します。

```
CRTBNDRPG PGM(MYLIB/DEBUGEX) SRCFILE(MYLIB/QRPGLESRC)
          TEXT('ILE RPG/400 program DEBUGEX')
          DBGVIEW(*SOURCE)
```

CRTRPGMOD を使用して、モジュール DBGEX のルート・ソース・ビューを作成するには、次を入力します。

```
CRTRPGMOD MODULE(MYLIB/DBGEX) SRCFILE(MYLIB/QRPGLESRC)
          TEXT('Entry module for program DEBUGEX')
          DBGVIEW(*SOURCE)
```

どちらかの作成コマンドで DBGVIEW(*SOURCE) を指定すると、デバッグ・モジュール DBGEX のルート・ソース・ビューが作成されます。デフォルトでは、その他の追加情報と共に /COPY メンバーおよび拡大 DDS のコンパイラー・リストが作成されます。

コピー・ソース・ビューの作成

コピー・ソース・ビューには、ソースのテキストに展開されるすべての /COPY メンバーのテキストだけでなく、ルート・ソース・メンバーからのテキストも含まれています。コピー・ビューを使用すると、ルート・ソース・ビューを使用してプログラムのルート・ソース・メンバーをデバッグし、コピー・ソース・ビューを使用してプログラムの /COPY メンバーをデバッグすることができます。

DBGVIEW(*COPY) によって生成されるルート・ソース・メンバーのビューは DBGVIEW(*SOURCE) により生成されるもの同じです。ソース・ファイルが DDM ファイルの場合には、ルート・ソース・ビューの場合と同様に、コピー・ソース・ビューは使用できません。

DBGVIEW パラメーターの *COPY または *ALL オプションを使用して、コピー・ソース・ビューを作成してモジュールをデバッグします。

コンパイラーはモジュール・オブジェクト (*MODULE) のコンパイル中に、コピー・ビューを作成します。コピー・ソースは、メンバーのテキストをビューにコピーするのでなく、ソース・メンバー (ルート・ソース・メンバーと /COPY メンバーの両方) のテキストの位置の参照を使用して作成します。このため、モジュール・オブジェクトの作成時とこれらのメンバーから作成されたモジュールのデバッグ時の間で、ソース・メンバーの変更、名前の付け直し、あるいは移動をしてはいけません。これを行った場合には、これらのソース・メンバーのビューを使用できなくなることがあります。

例えば、/COPY メンバーが入っているプログラム TEST1 のソース・ビューを作成するためには、次を入力します。

```
CRTBNDRPG PGM(MYLIB/TEST1) SRCFILE(MYLIB/QRPGLESRC)
          TEXT('ILE RPG/400 program TEST1')
          DBGVIEW(*COPY)
```

どちらかの作成コマンドで DBGVIEW(*COPY) を指定すると、デバッグ・モジュール TEST1 の /COPY メンバーのルート・ソース・ビューが作成されます。デフォルトでは、コンパイラー・リストが作成されます。OPTION(*SHOWCPY) がデフォルト値なので、コンパイラー・リストに /COPY メンバーも含まれます。

リスト・ビューの作成

リスト・ビューには、ILE RPG コンパイラーによって生成されたコンパイラー・リストのテキストと類似のテキストがあります。リスト・ビューに入っている情報は、OPTION(*SHOWCPY)、OPTION(*EXPDDS)、および OPTION(*SRCSTMT) が

いずれかの作成コマンドについて指定されているかどうかによって異なります。OPTION(*SHOWCPY) の場合はリストに /COPY メンバーが含まれ、OPTION(*EXPDDS) の場合は外部記述ファイルが含まれます。OPTION(*SRCSTMT) を指定すると、コンパイラー・リストの行番号の代わりにステートメント番号を使用してプログラム・オブジェクトをデバッグすることができます。

注: コンパイラー・リストで使用できる情報で、リスト・ビューに表示されないものがあります。例えば、コンパイラー・リストに (INDENT パラメーターを使用して) 字下げを指定した場合、リスト・ビューでは字下げは示されません。また、コンパイラー・リストに OPTION(*SHOWSKP) を指定した場合には、リスト・ビューにはスキップされたステートメントは現れません。

モジュールの作成時に、CRTRPGMOD または CRTBNDRPG コマンドの DBGVIEW パラメーターの *LIST または *ALL を使用して、モジュールをデバッグするためのリスト・ビューを作成します。

コンパイラーは、モジュール・オブジェクト (*MODULE) の生成時に、リスト・ビューを生成します。リスト・ビューは、適切なソース・メンバーのテキストをモジュール・オブジェクトにコピーすることによって作成されます。リスト・ビューが作成されてしまうと、基礎となっているソース・メンバーに対する従属性はなくなります。

例えば、拡張 DDS の入っているプログラム TEST1 のリスト・ビューを作成するためには、次を入力します。

```
CRTBNDRPG PGM(MYLIB/TEST1) SRCFILE(MYLIB/QRPGLESRC)
          SRCMBR(TEST1)  OUTPUT(*PRINT)
          TEXT('ILE RPG/400 program TEST1')
          OPTION(*EXPDDS) DBGVIEW(*LIST)
```

いずれかの作成コマンドで DBGVIEW パラメーターに DBGVIEW(*LIST) を指定し、OPTION パラメーターに *EXPDDS を指定すると、TEST1 のソースをデバッグするための拡張 DDS を持つリスト・ビューが作成されます。OUTPUT(*PRINT) と OPTION(*EXPDDS) が共にデフォルト値であるということに注意してください。

ステートメント・ビューの作成

ステートメント・ビューによって、モジュール・オブジェクトはステートメント番号とデバッグ・コマンドを使ったデバッグが可能となります。ソースは表示されませんので、コンパイラー・リストのソース・セクションに示されるステートメント番号を使用しなければなりません。言い換えると、このビューを効果的に使用するにはコンパイラー・リストが必要であるということです。また、デバッグ用に生成されるステートメント番号は、OPTION パラメーターに *SRCSTMT または *NOSRCSTMT が指定されるかどうかによって異なります。*NOSRCSTMT は、ステートメント番号が順番に割り当てられ、コンパイラー・リストのソース・セクションの左端の桁に行番号として表示されることを意味します。*SRCSTMT は、デバッグのためにステートメント番号を生成する際にコンパイラーが SEU 順序番号およびソース ID を使用するよう要求できるようにします。ステートメント番号は、コンパイラー・リストの右端の桁に示されます。

モジュールをデバッグするには、モジュールの作成時に、 CRTRPGMOD または CRTBNDRPG コマンドの DBGVIEW パラメーターの *STMT オプション を使用してステートメント・ビューを作成します。

次の場合にはこのビューを使用してください。

- モジュールまたはプログラムのデバッグが必要な場合に、記憶域の制約があるが、モジュールまたはプログラムをコンパイルし直したくない時
- コンパイル済みオブジェクトを他のユーザーに送り、デバッガーを使用してコード中の問題を診断できるようにしたいが、それらのユーザーに実際のコードを見せたくない時

例えば、CRTBNDRPG を使用してプログラム DEBUGEX のステートメント・ビューを作成するためには、次を入力します。

```
CRTBNDRPG PGM(MYLIB/DEBUGEX) SRCFILE(MYLIB/QRPGLESRC)
          TEXT('ILE RPG/400 program DEBUGEX')
```

CRTRPGMOD を使用してモジュールのステートメント・ビューを作成するためには、次を入力します。

```
CRTRPGMOD MODULE(MYLIB/DBGEX) SRCFILE(MYLIB/QRPGLESRC)
          TEXT('Entry module for program DEBUGEX')
```

デフォルトでは、コンパイラー・リストおよびステートメント・ビューが作成されます。コンパイラー・リストを使用して、ステートメント番号を確認し、デバッグ・コマンドを使用してプログラムをデバッグします。

いずれかの作成コマンドのデフォルトの値が変更されている場合には、明示的に DBGVIEW(*STMT) および OUTPUT(*PRINT) を指定しなければなりません。

ILE ソース・デバッガーの開始

デバッグ・ビュー (ステートメント、ソース、コピー、またはリスト) を作成したら、アプリケーションのデバッグを開始することができます。ILE ソース・デバッガーを開始するには、デバッグ開始 (STRDBG) コマンドを使います。デバッガーが開始すると、デバッグ・モード終了 (ENDDBG) コマンドを入力するまで活動状態のままになります。

最初に、STRDBG コマンドのプログラム (PGM) パラメーターを使用して最大 20 個のプログラム・オブジェクトをデバッグ・セッションに追加することができます。これは OPM プログラムまたは ILE プログラムの組み合わせが可能です (OPM プログラムのコンパイル方法およびデバッグ環境の設定値に応じて、 ILE ソース・デバッガーを使用してそれらのプログラムをデバッグすることができます)。さらに、STRDBG コマンドのサービス・プログラム (SRVPGM) パラメーターを使って最初に、20 までのサービス・プログラム・オブジェクトをデバッグ・セッションに加えることができます。サービス・プログラムをデバッグする規則は、プログラムをデバッグする規則と同じです。

- プログラムやサービス・プログラムにはデバッグ・データがなければならない。
- プログラムやサービス・プログラムをデバッグ・セッションに含めるために、それらに対する *CHANGE 権限をユーザーが持っている。

注: COPY またはルート・ソース・ビューを使用してプログラムをデバッグする場合には、ソース・コードはデバッグ中のプログラム・オブジェクトと同じシステムになければなりません。さらに、ソース・コードはコンパイル時点と同じ名前のライブラリー/ファイル (メンバー) になければなりません。

ILE プログラムの場合、デバッグ・データがあれば入り口モジュールが表示され、そうでない場合は、デバッグ・データを持つ ILE プログラムにバインドされた最初のモジュールが表示されます。

OPM プログラムの場合、プログラムにデバッグ・データがあるときに、OPMSRC パラメーターが *YES であれば、STRDBG コマンドで最初に指定されたプログラムが表示されます。すなわち、OPM プログラムがデバッグ・セッションにある場合は、次の条件が満たされれば ILE ソース・デバッガーを使って、それをデバッグすることができます。

1. OPM プログラムが、OPTION(*LSTDBG) または OPTION(*SRCDBG) を指定してコンパイルされている。 (3 種類の OPM 言語、すなわち RPG、COBOL、および CL 言語がサポートされます)。RPG プログラムおよび COBOL プログラムは、*LSTDBG または *SRCDBG を指定してコンパイルすることができますが、CL プログラムは *SRCDBG を指定してコンパイルしなければなりません。
2. ILE デバッグ環境が、OPM プログラムを受け入れるように設定されている。STRDBG コマンドに OPMSRC(*YES) を指定すればこれを設定することができます (システムのデフォルト値は OPMSRC(*NO) です)。

これら 2 つの条件が満たされないときは、OPM システム・デバッガーを使って OPM プログラムをデバッグする必要があります。

*LSTDBG または *SRCDBG なしで コンパイルされた OPM プログラムが指定され、サービス・プログラムが指定された場合、デバッグ・データを持っていれば、サービス・プログラムが表示されます。デバッグ・データがなければ DSPMODSRC 画面は空になります。ILE プログラムとサービス・プログラムが指定されると、ILE プログラムが表示されます。

STRDBG の例

例えば、サンプル・デバッグ・プログラム DEBUGEX および呼び出し先プログラム RPGPGM のデバッグ・セッションを開始するには、次のように入力します。

```
STRDBG PGM(MYLIB/DEBUGEX MYLIB/RPGPGM) OPMSRC(*YES)
```

「モジュール・ソースの表示」画面が 225 ページの図 92 に示すように現れます。DEBUGEX は、2 つのモジュール、RPG モジュール DBGEX と C モジュール cproc で構成されます。DBGEX、cproc、および RPGPGM のソースは、 267 ページの『デバッグ用サンプル・ソースの例』を参照してください。

入り口モジュールにルート・ソース、コピー、またはリスト・ビューがある場合には、最初のプログラムの入り口モジュールのソースが画面に示されます。この場合には、プログラムは DBGVIEW(*ALL) を使用して作成され、それに続いてメイン・モジュール DBGEX のソースが表示されます。

```
                          モジュール・ソースの表示
PROGRAM:   DEBUGEX        LIBRARY:   MYLIB          MODULE:   DBGEX
     1       *============================================================
     2       *  DEBUGEX - ILE RPG OS/400 用ソースでの ILE ソース・デバッグ
     3       *            プログラムの使用法を示すように設計されたプログラム。
     4       *            各種のデータ・タイプおよびデータ構造の例を示します。
     5       *
     6       *            また、定様式ダンプ例の作成にも使用できます。
     7       *============================================================
     8
     9       *------------------------------------------------------------
    10       * DEBUG キーワードは定様式ダンプ機能を可能にします。
    11       *------------------------------------------------------------
    12      H DEBUG
    13
    14       *------------------------------------------------------------
    15       * 各種の ILE RPG OS/400 用データ・タイプの独立フィールドの定義
                                                                  続く...
デバッグ . . .   ____________________________________________________________
_____________________________________________________________________________
F3=終了プログラム  F6=停止点の追加/消去  F10=ステップ  F11=変数の表示
F12=再開       F17=ウォッチ変数   F18=ウォッチの処理   F23=表示出力
```

図 92. プログラム DEBUGEX の「モジュール・ソースの表示」画面

注: STRDBG コマンドのサービス・プログラム (SRVPGM) パラメーターを使って、最大 20 までのサービス・プログラムを最初にデバッグ・セッションに追加することができます。モジュールの処理リスト (F14) のオプション 1 (追加) を使用するか、それを STEP INTO デバッグ・コマンドの一部としてソース・デバッガーに追加させることによって、ILE サービス・プログラムをデバッグ・セッションに追加することもできます。

デバッグ・オプションの設定

デバッグ・セッションを開始した後で、以下のデバッグ・オプションを設定あるいは変更することができます。

- プログラムのデバッグ中にデータベース・ファイルを更新できるかどうか (このオプションは、STRDBG コマンドの UPDPROD パラメーターと対応しています)。
- FIND を使ったテキスト検索は大文字小文字の区別をするかどうか。
- OPM プログラムが ILE ソース・デバッガーを使用してデバッグされるかどうか (このオプションは OPMSRC パラメーターと対応しています)。

デバッグ・コマンド SET を使用してデバッグ・オプションを変更すると、対応するパラメーターが STRDBG コマンドに指定されていれば、このパラメーターの値に影響を与えます。デバッグ変更 (CHGDBG) コマンドを使用して、デバッグ・オプションの設定を行うこともできます。しかし、OPMSRC オプションは CHGDBG コマンドで変更することはできません。OPMSRC はデバッグ SET コマンドによってのみ、変更することができます。

ILE プログラムを処理するデバッグ・セッション中であり、しかも、デバッグ・データが利用できる OPM プログラムもデバッグする必要があるとします。ILE ソース・デバッガーが OPM プログラムを受け入れることができるようにするには、次のステップに従ってください。

1. STRDBG の入力後に現行画面が「モジュール・ソースの表示」画面になっていない 場合には、次の入力をします。

```
DSPMODSRC
```

「モジュール・ソースの表示」画面が表示されます。

2. タイプ

   ```
   SET
   ```

3. 「デバッグ・オプションの設定」画面が表示されます。この画面では、*OPM* ソース・デバッグ・サポート・フィールドに Y (はい) を入力して、実行キーを押して、「モジュール・ソースの表示」画面に戻ってください。

これで、モジュール処理画面を使用するか、あるいはそのプログラムへの呼び出しステートメントを処理することにより、OPM プログラムを追加できるようになりました。

デバッグ・セッションでのプログラムの追加/除去

デバッグ・セッションの開始後に、デバッグ・セッションでプログラムの追加と除去ができます。プログラムをデバッグ・セッションに加える、またはそこから除去するには、そのプログラムに対する *CHANGE 権限が必要です。

ILE プログラムについては、DSPMODSRC コマンドのモジュールの処理リストのオプション 1 (プログラムの追加) を使います。ILE プログラムまたはサービス・プログラムを除去するには、同じ画面のオプション 4 を使います。 ILE プログラムまたはサービス・プログラムを除去すると、そのプログラムのすべてのブレークポイントは除去されます。デバッグ・セッションに一度に入れたり除去したりできる ILE プログラムまたはサービス・プログラムの数に制限はありません。

OPM プログラムの場合には、 OPMSRC に指定された値に応じて 2 つの選択があります。STRDBG、SET デバッグ・コマンド、あるいは CHGDBG のいずれかを使用して OPMSRC(*YES) を指定している場合には、モジュール処理画面を使用して OPM プログラムの追加もしくは除去を行います (モジュール名は OPM プログラムにリストされていないことに注意してください)。OPMSRC(*YES) が指定されているときにデバッグ・セッションに入れることができる OPM プログラムの数に、制限はありません。

OPMSRC(*NO) を指定している場合には、プログラム追加 (ADDPGM) コマンドまたはプログラム除去 (RMVPGM) コマンドを使用する必要があります。
OPMSRC(*NO) が指定されているときに、一度にデバッグ・セッションに入れることができる OPM プログラムは 20 個だけです。

注: ILE デバッグ・セッションと OPM デバッグ・セッションの両方からのデバッグ・データを使用して OPM プログラムをデバッグすることはできません。OPM プログラムが既に OPM デバッグ・セッションに入っている場合には、まず最初にプログラムをこのセッションから除去してから、 ILE デバッグ・セッションに追加したり、呼び出しステートメントからステップインしなければなりません。同様に、プログラムを OPM デバッグ・セッションからデバッグしたい場合には、まず最初にプログラムを ILE デバッグ・セッションから除去する必要があります。

デバッグ・セッションへのサービス・プログラムの追加例

この例では、既に前に開始しているデバッグ・セッションへ、サービス・プログラム CVTTOHEX を追加することができます。(サービス・プログラムの説明については、104 ページの『サンプル・サービス・プログラム』を参照してください)。

1. 現行画面が「モジュール・ソースの表示」画面ではない 場合には、次を入力してください。

   ```
   DSPMODSRC
   ```

   「モジュール・ソースの表示」画面が表示されます。
2. F14 (モジュール・リストの処理) キーを押して、図 93 に示されている「モジュール・リストの処理」画面を表示してください。
3. サービス・プログラム CVTTOHEX を追加するためには、画面の最初の行で次のように入力してください。OPT 欄に 1 (プログラムの追加)、*PROGRAM/MODULE* フィールドに CVTTOHEX、 *LIBRARY* フィールドに MYLIB。*PGM から *SRVPGM までのデフォルトのプログラム・タイプを変更し、実行キーを押してください。
4. F12 (取消し) を押して「モジュール・ソースの表示」画面に戻ります。

```
                          モジュール・リストの処理
                                                    システム :   ISERIES1
オプションを入力して、実行キーを押してください。
  1=プログラムの追加   4=プログラムの除去   5=モジュール・ソースの表示
  8=モジュール停止点の処理
OPT     PROGRAM/MODULE     LIBRARY        TYPE
 1       cvttohex          mylib          *SRVPGM
         RPGPGM            MYLIB          *PGM
         DEBUGEX           MYLIB          *PGM
           DBGEX                          *MODULE     選択
           CPROC                          *MODULE
                                                                    終わり
コマンド
===>
F3=終了    F4=プロンプト    F5=最新表示    F9=コマンド複写    F12=取消し
```

図 93. デバッグ・セッションへの ILE サービス・プログラムの追加

ILE プログラムのデバッグ・セッションからの除去例

この例では、ILE プログラム CVTHEXPGM とサービス・プログラム CVTTOHEX をデバッグ・セッションから除去します。

1. 現行画面が「モジュール・ソースの表示」画面ではない 場合には、次を入力してください。

   ```
   DSPMODSRC
   ```

   「モジュール・ソースの表示」画面が表示されます。
2. F14 (モジュール・リストの処理) キーを押して 228 ページの図 94 に示されている「モジュール・リストの処理」画面を表示してください。
3. この画面では、CVTHEXPGM と CVTTOHEX の横にある行に 4 (プログラムの除去) を入力して、実行キーを押してください。
4. F12 (取消し) を押して「モジュール・ソースの表示」画面に戻ります。

```
                          モジュール・リストの処理
                                                        システム :   ISERIES1
オプションを入力して、実行キーを押してください。
  1=プログラムの追加    4=プログラムの除去    5=モジュール・ソースの表示
  8=モジュール停止点の処理
OPT      PROGRAM/MODULE     LIBRARY        TYPE
               *LIBL      *PGM
 4         CVTHEXPGM         MYLIB          *PGM
             CVTHEXPG                       *MODULE
 4         CVTTOHEX          MYLIB          *SRVPGM
             CVTTOHEX                       *MODULE
           RPGPGM            MYLIB          *PGM
           DEBUGEX           MYLIB          *PGM
             DBGEX                          *MODULE      選択
             CPROC                          *MODULE
                                                                        終わり
コマンド
===>_____________________________________________________________________
F3=終了    F4=プロンプト    F5=最新表示    F9=コマンド複写    F12=取消し
```

図 94. ILE プログラムのデバッグ・セッションからの除去

プログラム・ソースの表示

「モジュール・ソースの表示」画面には、一度に 1 モジュール・オブジェクトずつ ILE プログラム・オブジェクトのソースが表示されます。 ILE モジュール・オブジェクトが次のデバッグ・ビュー・オプションのいずれかを使用してコンパイルされている場合には、そのモジュール・オブジェクトのソースを表示することができます。

- DBGVIEW(*SOURCE)
- DBGVIEW(*COPY)
- DBGVIEW(*LIST)
- DBGVIEW(*ALL)

以下の条件が満たされる場合には、OPM プログラムのソースを表示することができます。

1. OPM プログラムが、OPTION(*LSTDBG) または OPTION(*SRCDBG) を指定してコンパイルされている。 (*LSTDBG を指定してコンパイルできるのは RPG プログラムと COBOL プログラムだけです)。
2. ILE デバッグ環境が、OPM プログラムを受け入れるように設定されている。すなわち OPMSRC の値が *YES です (システムのデフォルト値は OPMSRC(*NO) です)。

「モジュール・ソースの表示」画面に表示されるものを変更するには、次の 2 つの方法があります。

- 別のモジュールに変更する
- モジュールのビューを変更する

ビューを変更すると、ILE ソース・デバッガーが変更先のビューの位置と等価の位置をマップします。モジュールを変更する時に、表示されたビューの実行可能ステートメントがメモリーに保管され、モジュールが再び表示された時に示されます。ブレークポイントを設定させる行番号が強調表示されます。ブレークポイント、ス

テップ、またはメッセージによりプログラムが停止し、画面が表示されると、ブレークポイントが起こった場所のステートメントが強調表示されます。

別のモジュールの表示

「モジュール・ソースの表示」画面に表示されるモジュール・オブジェクトを変更するためには、モジュール・リストの処理のオプション 5 (モジュール・ソースの表示) を使用してください。モジュール・リストの処理には、「モジュール・ソースの表示」画面から F14 (モジュール・リストの処理) キーを押すことによってアクセスします。

ILE プログラム・オブジェクトでこのオプションを使うと、ルート・ソース、コピー、またはリスト・ビューの入力モジュールが (もしあれば) 表示されます。そうでない場合には、デバッグ・データを持つプログラム・オブジェクトにバインドされた、最初のモジュール・オブジェクトが表示されます。OPM プログラム・オブジェクトでこのオプションを使用する場合には、ソースまたはリスト・ビューが表示されます (使用可能な場合)。

別のモジュール・オブジェクトを表示するもう 1 つの方法として、 DISPLAY デバッグ・コマンドを使用します。デバッグ・コマンド入力行に以下を入力します。

```
DISPLAY MODULE module-name
```

モジュール・オブジェクトのモジュール*名* が表示されます。このモジュール・オブジェクトは、デバッグ・セッションに追加されたプログラム・オブジェクトに存在していなければなりません。

例えば、モジュール・ソースの表示オプションを使用して 225 ページの図 92 中のモジュール DBGEX からモジュール cproc に変更するためには、次のステップに従ってください。

1. モジュールを処理するために、DSPMODSRC を入力して実行キーを押してください。「モジュール・ソースの表示」画面が表示されます。
2. F14 (モジュール・リストの処理) を押して、モジュール・リストの処理を表示します。 230 ページの図 95 はサンプルの表示を示します。
3. cproc を選択するには、その横に 5 (モジュール・ソースの表示) を入力し実行キーを押してください。ルート・ソースが使用可能なので、 230 ページの図 96 に示されたように表示されます。ルート・ソースを使用できなかった場合には、デバッグ・データをもつプログラム・オブジェクトにバインドされた最初のモジュール・オブジェクトが表示されます。

```
                          モジュール・リストの処理
                                                        システム :   ISERIES1
オプションを入力して、実行キーを押してください。
  1=プログラムの追加   4=プログラムの除去   5=モジュール・ソースの表示
  8=モジュール停止点の処理
OPT      PROGRAM/MODULE     LIBRARY        TYPE
                *LIBL      *PGM
          RPGPGM            MYLIB          *PGM
          DEBUGEX           MYLIB          *PGM
            DBGEX                          *MODULE      選択
 5          CPROC                          *MODULE
                                                                        終わり
コマンド
===> ____________________________________________________________________________
F3=終了    F4=プロンプト    F5=最新表示    F9=コマンド複写    F12=取消し
```

図 95. 別のモジュールへの変更

```
                          モジュール・ソースの表示
PROGRAM:   DEBUGEX        LIBRARY:   MYLIB          MODULE:   CPROC
     1       #include <stdlib.h>
     2       #include <string.h>
     3       #include <stdio.h>
     4       extern char EXPORTFLD[6];
     5
     6       char *c_proc(unsigned int size, char *inzval)
     7       {
     8          char *ptr;
     9          ptr = malloc(size);
    10          memset(ptr, *inzval, size);
    11          printf("import string: %6s.¥n",EXPORTFLD);
    12          return(ptr);
    13       }
                                                                        終わり
デバッグ . . .   ____________________________________________________________
_____________________________________________________________________________
F3=終了プログラム    F6=停止点の追加/消去    F10=ステップ    F11=変数の表示
F12=再開             F17=ウォッチ変数    F18=ウォッチの処理    F23=表示出力
```

図 96. ILE C プロシージャー cproc のソース・ビュー

モジュールのビューの変更

モジュールを作成する時に指定する値によって異なる、ILE RPG モジュールのいくつかのビューを表示することができます。それらは次のとおりです。

- ルート・ソース・ビュー
- コピー・ソース・ビュー
- リスト・ビュー

「ビューの選択」画面により、「モジュール・ソースの表示」画面に示されるモジュール・オブジェクトのビューを変更することができます。「ビューの選択」画面は、F15 (ビューの選択) キーを押すことによって「モジュール・ソースの表示」画面でアクセスすることができます。「ビューの選択」画面は 231 ページの図 97 に示されています。現在のビューがウィンドウの最上部にリストされ、使用可能な他のビューはその下に表示されます。作成時に使われたデバッグ・オプションによって、プログラム・オブジェクトの各モジュール・オブジェクトは、使用可能な異なるビューのセットをもつことがあります。

例えば、モジュールのビューをルート・ソースからリストに変更するためには、次のステップに従ってください。

1. DSPMODSRC を入力して、実行キーを押してください。「モジュール・ソースの表示」画面が表示されます。
2. F15 (ビューの選択) キーを押してください。「ビューの選択」ウィンドウが図 97 に示されます。

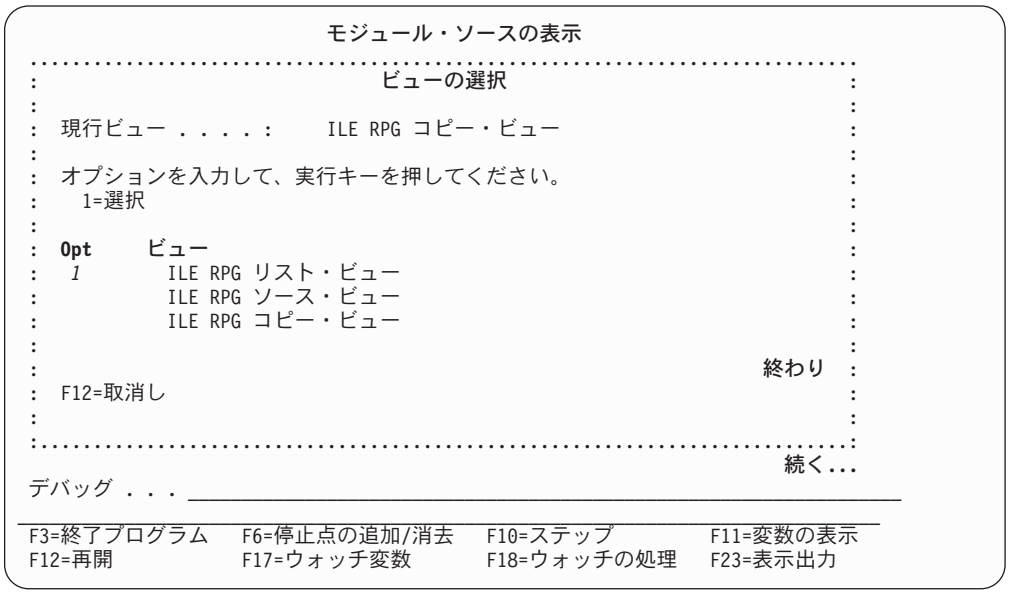


図 97. モジュールのビューの変更

現在のビューがウィンドウの最上部にリストされ、使用可能な他のビューはその下に表示されます。作成に使用したデバッグ・オプションによって、プログラム中の各モジュールがビューの異なるセットをもつことがあります。

注: モジュールが DBGVIEW(*ALL) で作成された場合には、ビューの選択ウィンドウに使用可能な 3 つのビュー、すなわちルート・ソース、コピー、およびリストの各ビューが表示されます。モジュールに /COPY メンバーがない場合には、コピー・ビューはルート・ソース・ビューと同じになります。

3. リスト・ビューの横に 1 を入力し、実行キーを押してください。「モジュール・ソースの表示」画面が表示され、リスト・ビューでモジュールが表示されます。

ブレークポイントの設定と除去

ブレークポイントを使用して、実行時に特定の点でプログラム・オブジェクトを停止することができます。**無条件ブレークポイント**は、特定のステートメントでプログラム・オブジェクトを停止します。**条件付きブレークポイント**は、特定のステートメントで特定の条件が満たされた時にプログラム・オブジェクトを停止します。

ブレークポイントにはジョブとスレッドの 2 つのタイプがあります。スレッド化されたアプリケーションの各**スレッド**は、同時に同じ場所に自分自身のブレークポイントを持つことができます。ジョブおよびスレッドのブレークポイントは両者とも、条件付きにも無条件にすることができます。一般に、ジョブ・ブレークポイン

トにはデバッグ・コマンドと機能キーのある組み合わせがあり、スレッド・ブレークポイントには別の組み合わせがあります。ブレークポイントについて説明するこのセクションのこれ以降では、特に断りがない限り、ブレークポイントという言葉はジョブとスレッドの両方のブレークポイントを指すものとします。

注: ブレークポイントは、デフォルトの OPTION(*DEBUGIO) が指定された場合には、入出力仕様書について自動的に生成されます。ブレークポイントを生成したくない場合には、OPTION(*NODEBUGIO) を指定してください。

プログラムの実行に先立ちブレークポイントを設定します。プログラム・オブジェクトの停止時に、「モジュール・ソースの表示」画面が表示されます。ブレークポイントが起こった行に位置付けられたソースに該当するモジュール・オブジェクトが表示されます。この行は強調表示されます。この時点でフィールドを評価し、より多くのブレークポイントを設定し、任意のデバッグ・コマンドを実行することができます。

ブレークポイントを使用する前に、ブレークポイントについて、次の特性を知っておいてください。

- ブレークポイントがあるステートメントに設定されている時には、そのステートメントが処理される*前に* ブレークポイントが起こります。
- 条件付きブレークポイントの設定されているステートメントに達すると、そのステートメントが処理される*前に*、そのブレークポイントに関連した条件式が評価されます。式が真の場合には、そのブレークポイントが効力をもち、その行でプログラムが停止します。
- ブレークポイントを設定したい行が実行可能ステートメントでない場合には、ブレークポイントは次の実行可能なステートメントに設定されます。
- ブレークポイントが迂回された場合には、そのブレークポイントは処理されません。
- ブレークポイント機能は、デバッグ・コマンドによって指定されます。ブレークポイント機能には次のものがあります。
  - プログラム・オブジェクトへのブレークポイントの追加
  - プログラム・オブジェクトからのブレークポイントの除去
  - ブレークポイント情報の表示
  - ブレークポイントに達した後でのプログラム・オブジェクトの実行の再開
  - ジョブまたはスレッド・ブレークポイントのいずれかを同じ時にある指定した場所で持つことができますが、両者一緒には持てません。

ブレークポイントを設定した後にモジュールのビューを変更する場合には、ブレークポイントの行番号はソース・デバッガーによって新しいビューにマップされます。

ステートメント・ビューで作成されたモジュールまたはプログラムをデバッグしている場合には、コンパイラー・リストから得たステートメント番号を使用してブレークポイントの設定、または除去することができます。ステートメント番号の使用の詳細については、 239 ページの『ステートメント番号を使用したジョブ・ブレークポイントの設定および除去』を参照してください。

無条件ジョブ・ブレークポイントの設定および除去

以下を使って、無条件ジョブ・ブレークポイントを設定、または除去することができます。

- 「モジュール・ソースの表示」画面からの F13 (モジュール・ブレークポイントの処理) または、F6 (ブレークポイントの追加/消去)
- ジョブ・ブレークポイントを設定する BREAK デバッグ・コマンド
- ジョブ・ブレークポイントを除去する CLEAR デバッグ・コマンド
- 「モジュール・ブレークポイントの処理」画面

無条件ジョブ・ブレークポイントの設定および除去する最も簡単な方法は、F6 (ブレークポイントの追加 / 消去) を使うことです。この機能キーはキーとして機能するので、カーソルのある行にブレークポイントが既に設定されている場合にはその行からブレークポイントを除去します。

F13 (モジュール・ブレークポイントの処理) を使って無条件ジョブ・ブレークポイントを除去するには、「モジュール・ソースの表示」画面から F13 (モジュール・ブレークポイントの処理) を押します。ブレークポイントを設定または除去することができるオプションのリストが表示されます。4 (消去) を選ぶと、ジョブ・ブレークポイントがその行から除去されます。

無条件ジョブ・ブレークポイントを設定・除去するもう 1 つの方法は、BREAK および CLEAR デバッグ・コマンドを使うことです。BREAK デバッグ・コマンドを使って無条件ジョブ・ブレークポイントを設定するには、デバッグ・コマンド行に、

```
BREAK line-number
```

と入力します。変数の *line-number* は、ブレークポイントを設定したいモジュール・オブジェクトの現在表示されているビューの行番号です。

CLEAR デバッグ・コマンドを使って無条件ジョブ・ブレークポイントを除去するためには、デバッグ・コマンド行に、

```
CLEAR line-number
```

と入力します。変数の *line-number* は、ブレークポイントを除去したいモジュール・オブジェクトの現在表示されているビューの行番号です。ジョブ・ブレークポイントを消去する際は、すべてのスレッドから消去されます。

無条件ジョブ・ブレークポイントの設定例

この例では F6 (ブレークポイントの追加/消去) を使って、無条件ジョブ・ブレークポイントを設定します。ブレークポイントは最初の実行可能な演算仕様書に設定されるので、各種のフィールドおよびデータ構造を表示することができます。

1. モジュールを処理するためには、DSPMODSRC を入力して、実行キーを押してください。「モジュール・ソースの表示」画面が表示されます。
2. 表示されたモジュールでジョブ・ブレークポイントを設定したいときは、ステップ 3 (234 ページ) を続けます。別のモジュールでジョブ・ブレークポイントを設定したいときは、

   ```
   DISPLAY MODULE module-name
   ```

とデバッグ・コマンド行で入力します。ここで、*module-name* は表示したいモジュールの名前です。

3. 最初の演算仕様書で無条件ブレークポイントを設定するために、カーソルを行 88 に置いてください。
4. F6 (ブレークポイントの追加/消去) キーを押してください。行 88 にブレークポイントがなければ、図 98 に示すように、無条件ブレークポイントがその行に設定されます。その行にブレークポイントがある場合には、除去されます。

   注: *最初の* 演算仕様書でブレークポイントが必要なのですから、カーソルを演算仕様書の開始前のいずれかの行に位置付けることもでき、行 88 は最初の実行可能ステートメントであるためにブレークポイントはその行に設定されることになります。

```
                          モジュール・ソースの表示
PRPGRAM:   DEBUGEX        LIBRARY:   MYLIB          MODULE:   DBGEX
    84        *------------------------------------------------------------------
    85        * 'A' をデータ構造 DS2 に転送します。転送後、DS2 の
    86        * 最初のオカレンスには 10 桁の 'A' が入っています。
    87        *------------------------------------------------------------------
    88       C                   MOVE      *ALL'a'       DS2
    89
    90        *------------------------------------------------------------------
    91        * DS2 のオカレンスを 2 に変更して、'B' を DS2 に転送すると
    92        * 最初の 10 バイトが 'A' になり、次の 10 バイトが 'B' になります。
    93        *------------------------------------------------------------------
    94       C     2             OCCUR     DS2
    95       C                   MOVE      *ALL'b'       DS2
    96
    97        *------------------------------------------------------------------
    98        * FILD1A は FILD1 のオーバーレイ・フィールドです。FILD1 は初期設定。
                                                                        続く...
デバッグ . . . _____________________________________________________________________
_____________________________________________________________________________
F3=終了プログラム  F6=停止点の追加/消去  F10=ステップ  F11=変数の表示
F12=再開           F17=ウォッチ変数       F18=ウォッチの処理   F23=表示出力
停止点が行 88 に追加されました。
```

図 98. 無条件ジョブ・ブレークポイントの設定

5. ブレークポイントが設定されたら、F3 (終了) キーを押して「モジュール・ソース表示」画面を終了してください。ブレークポイントは除去されません。
6. プログラムを呼び出してください。ブレークポイントに達するとプログラムは停止し、ブレークポイントのある行が強調表示されて、「モジュール・ソースの表示」画面が再び表示されます。この時点で、プログラムを実行するか、あるいは処理を再開することができます。

無条件スレッド・ブレークポイントの設定および除去

以下を使って、無条件スレッド・ブレークポイントを設定または除去することができます。

- 「モジュール・ブレークポイントの処理」画面
- 現行スレッドのスレッド・ブレークポイントを設定する TBREAK デバッグ・コマンド
- スレッド・ブレークポイントを除去する CLEAR デバッグ・コマンド

「モジュール・ブレークポイントの処理」画面を使って、無条件スレッド・ブレークポイントを設定するには、次のようにします。

- *OPT* フィールドに 1 (追加) を入力します。
- *スレッド・フィールドにスレッド識別子を入力します。
- 残りのフィールドを、無条件ジョブ・ブレークポイントと同様に埋めます。
- 実行 (キー) を押します。

注: スレッド・フィールドは、SPAWN コマンドの DEBUG オプションが 1 より大きいか等しい場合に表示されます。

TBREAK デバッグ・コマンドの構文は、BREAK デバッグ・コマンドと同じです。BREAK デバッグ・コマンドはすべてのスレッドの同じ位置でジョブ・ブレークポイントを設定するのに対して、TBREAK デバッグ・コマンドは単一スレッド、すなわち現行スレッドでスレッド・ブレークポイントを設定します。

現行スレッドとは、現在デバッグされているスレッドのことです。デバッグ・コマンドはこのスレッドに対して出されます。ブレークポイントのようなデバッグ停止が起こると、現行スレッドはデバッグ停止が起こったスレッドに設定されます。現行スレッドを変更するには、デバッグ THREAD コマンドおよび「デバッグされたスレッドの処理」画面を使うことができます。

無条件スレッド・ブレークポイントを除去するには、CLEAR デバッグ・コマンドを使います。スレッド・ブレークポイントが消去される時は、現行スレッドのみが対象です。

条件付きジョブ・ブレークポイントの設定および除去

以下を使って、条件付きジョブ・ブレークポイントを設定または除去することができます。

- 「モジュール・ブレークポイントの処理」画面
- ジョブ・ブレークポイントを設定する BREAK デバッグ・コマンド
- ブレークポイントを除去するための CLEAR デバッグ・コマンド

注: 条件付きブレークポイント用にサポートされる関係演算子は、<、>、=、<=、>=、および <> (等しくない) です。

条件付きジョブ・ブレークポイントを設定または除去できる 1 つの方法は、「モジュール・ブレークポイントの処理」画面によるものです。「モジュール・ブレークポイントの処理」画面には、「モジュール・ソースの表示」画面で F13 (モジュール・ブレークポイントの処理) キーを押すことによってアクセスします。この画面は、条件付きまたは無条件ジョブ・ブレークポイントの追加、または除去のいずれかを許可するオプションのリストを提供します。画面の例を 237 ページの図 99 に示します。

ジョブ・ブレークポイントを条件付きにするには、*条件* フィールドに条件式を指定します。ジョブ・ブレークポイントを設定しようとする行が実行可能ステートメントではない場合、ブレークポイントは次の実行可能ステートメントに設定されます。

スレッド列を表示するには、実行キー を押す前にスレッド・フィールドに *JOB と入力します。

すべてのジョブ・ブレークポイントを指定し終えたら、プログラムを呼び出します。「モジュール・ソースの表示」画面から F21 (コマンド入力) を使用してコマンド行でプログラム・オブジェクトを呼び出すか、あるいはこの画面を出た後でプログラムを呼び出すことができます。

条件付きジョブ・ブレークポイントが設定されたステートメントに達すると、ジョブ・ブレークポイントに関連した条件式が、そのステートメントが実行される前に評価されます。結果が偽の場合には、プログラム・オブジェクトは実行し続けます。結果が真の場合には、プログラム・オブジェクトが停止し、「モジュール・ソースの表示」画面が表示されます。この時点でフィールドを評価し、より多くのブレークポイントを設定し、任意のデバッグ・コマンドを実行することができます。

条件付きブレークポイントを設定および除去するもう 1 つの方法は、BREAK および CLEAR デバッグ・コマンドの使用です。

BREAK デバッグ・コマンドを使用して条件付きブレークポイントを設定するためには、デバッグ・コマンド行に、

```
BREAK line-number WHEN expression
```

と入力します。変数の *line-number* は、ブレークポイントを設定したいモジュール・オブジェクトの現在表示されているビューの行番号で、*expression* はブレークポイントが出てくる時に評価される条件付き式です。条件式ブレークポイントに対しサポートされている比較演算子は、このセクションの始めに注記されています。

非数字の条件式ブレークポイントの式では、比較が起こる前に、より短い式が暗黙にブランクで埋め込まれます。この暗黙の埋め込みは国別言語分類順序 (NLSS) 変換の前に起こります。NLSS の詳細については、238 ページの『国別言語分類順序 (NLSS)』を参照してください。

CLEAR デバッグ・コマンドを使用して条件付きブレークポイントを除去するためには、デバッグ・コマンド行に

```
CLEAR line-number
```

と入力します。変数の *line-number* は、ブレークポイントを除去したいモジュール・オブジェクトが現在表示されているビューの行番号です。

F13 を使った条件付きジョブ・ブレークポイントの設定例

この例では、F13 (モジュール停止点の処理) を使って条件付きジョブ・ブレークポイントを設定します。

1. 条件付きジョブ・ブレークポイントを設定するには、F13 (モジュール停止点の処理) を押します。「モジュール停止点の処理」画面が表示されます。
2. この画面では、リストの最初の行に 1 (追加) を入力して条件付きブレークポイントを追加します。
3. *IN02='1' の時に条件付きブレークポイントを行 127 に設定するには、*行* フィールドに 127、*条件* フィールドに *IN02='1' と入力します。

4. スレッド列を表示するには、実行キーを押す前にスレッド・フィールドに *JOB と入力します。

   図 99 は、条件付きブレークポイントの追加後の「モジュール停止点の処理」画面を示します。

```
                          モジュール停止点の処理
                                                          システム :   TORASD80
プログラム . . :   DEBUGEX              ライブラリー . :   MYLIB
  モジュール . :     DBGEX              タイプ . . . . :   *PGM
オプションを入力して，実行キーを押してください。
  1= 追加    4= 消去

OPT       行          条件
          127         *in02='1'
          88
          102

                                                                    終わり
コマンド
===> ____________________________________________________________________________
F3= 終了    F4=プロンプト   F5= 最新表示    F9=コマンド複写    F12= 取消し
 停止点が行 127 に追加されました。
```

図 99. 条件付きジョブ・ブレークポイントの設定

   条件付きジョブ・ブレークポイントが行 127 に設定されます。式はステートメントの実行前に評価されます。結果が真の場合には (例では *IN02='1' の場合)、プログラムが停止し、「モジュール・ソースの表示」画面が表示されます。結果が偽の場合には、プログラムは実行し続けます。

   既存のブレークポイントは常に、同じ位置に入力された新しいブレークポイントによって置き換えられます。

5. ブレークポイントの設定後に、F12 (取消し) キーを押して「モジュール・ブレークポイントの処理」画面を出てください。F3 (終了プログラム) キーを押して、ILE ソース・デバッガーを終了してください。ブレークポイントは除去されません。
6. プログラムを呼び出してください。ブレークポイントに達するとプログラムが停止し、「モジュール・ソースの表示」画面が再び表示されます。この時点で、プログラムを実行するか、あるいは処理を再開することができます。

BREAK コマンドを使用した条件付きジョブ・ブレークポイントの設定例

この例では、日付フィールド BigDate に特定の値がある時にプログラムが停止させたいものとします。BREAK コマンドを使って条件付きジョブ・ブレークポイントを指定するには、次のようにします。

1. 「モジュール・ソースの表示」画面から次の入力をします。

```
break 128 when BigDate='1994-09-30'
```

   条件付きジョブ・ブレークポイントが行 128 に設定されます。

2. ブレークポイントの設定後、F3 (終了プログラム) キーを押して ILE ソース・デバッガーを終了してください。ブレークポイントは除去されません。
3. プログラムを呼び出してください。ブレークポイントに達するとプログラムが停止し、「モジュール・ソースの表示」画面が再び表示されます。

```
                          モジュール・ソースの表示
PROGRAM:   DEBUGEX       LIBRARY:   MYLIB          MODULE:   DBGEX
   122
   123        *-------------------------------------------------------------
   124        * 次の SETON 命令後に、*IN02 = '1'.
   125        *-------------------------------------------------------------
   126       C                   SETON
   127       C                   IF        *IN02
   128       C                   MOVE      '1994-09-30'  BigDate
   129       C                   ENDIF
   130
   131        *-------------------------------------------------------------
   132        * ARRY の 2 番目のセルに新しい値を入れる。
   133        *-------------------------------------------------------------
   134       C                   MOVE      4             Arry
   135
   136        *-------------------------------------------------------------
                                                                      続く...
デバッグ. . break 128 when BigDate='1994-09-30'_____________________________
_____________________________________________________________________________
F3=終了プログラム  F6=停止点の追加/消去  F10=ステップ  F11=変数の表示
F12=再開       F17=ウォッチ変数   F18=ウォッチの処理   F23=表示出力
```

図 100. BREAK コマンドを使用した条件付きジョブ・ブレークポイントの設定

国別言語分類順序 (NLSS)

非数値のブレークポイント条件式は次の 2 つのタイプに分かれます。

- Char- 8: 各文字が 8 ビットからなります。

  これは、RPG データ・タイプの文字、日付、時刻、およびタイム・スタンプに対応しています。

- Char- 16: 各文字が 16 ビットからなります (DBCS)。

  これは、RPG 図形データ・タイプに対応しています。

NLSS はタイプ Char-8 の非数値のブレークポイント条件式に対してだけ適用されます。非数値のブレークポイント条件式の可能な組み合わせについては、 239 ページの表 29を参照してください。

ソース・デバッガーにより使用されるタイプ Char-8 の式の分類順序テーブルは、CRTRPGMOD または CRTBNDRPG コマンドの SRTSEQ パラメーターで指定される分類順序テーブルです。

解決された分類順序テーブルが *HEX の場合には、分類順序テーブルは使用されません。したがって、ソース・デバッガーは分類順序を決定するのにその文字の 16 進数値の桁を使用します。そうでない場合には、比較する前に各バイトに重みを割り当てるために、指定された分類順序テーブルが使用されます。シフトアウト / シフトインの文字間、およびそれを含むバイトには重みは割り当てられ**ません**。これは ILE RPG が比較を扱う方法とは異なります。シフトアウト/シフトイン文字を含めすべての文字に重みが割り当てられます。

注:

1. 制御仕様書の ALTSEQ (*SRC) によって指定されるもう 1 つの順序は、ILE ソース・デバッガーには使えません。代わりに、ソース・デバッガーは *HEX 分類順序テーブルを使用します。
2. 分類順序テーブルの名前はコンパイル時に保管されます。デバッグ時に、分類順序テーブルをアクセスするためソース・デバッガーはコンパイルで保管された名前を使用します。コンパイル時に指定された分類順序テーブルが *HEX または *JOBRUN 以外の何かに解決された場合には、この分類順序テーブルがデバッグの開始前に変えられない ということが重要です。損傷があるか、あるいは削除されていてテーブルをアクセスできない場合には、ソース・デバッガーは *HEX 分類順序テーブルを使用します。

表 29. 非数値条件付きブレークポイントの式

タイプ	可能な式
Char-8	• 文字フィールドと文字フィールドの比較 • 文字フィールドと文字リテラル[1] の比較 • 文字フィールドと 16 進リテラル[2] の比較 • 文字リテラル[1] と文字フィールドの比較 • 文字リテラル[1] と文字リテラル[1] の比較 • 文字リテラル[1] と 16 進リテラル[2] の比較 • 16 進リテラル[2] と文字フィールド[1] の比較 • 16 進リテラル[2] と文字リテラル[1] の比較 • 16 進リテラル[2] と 16 進リテラル[2] の比較
Char-16	• 図形フィールドと図形フィールドの比較 • 図形フィールドと図形リテラル[3] の比較 • 図形フィールドと 16 進リテラル[2] の比較 • 図形リテラル[3] と図形フィールドの比較 • 図形リテラル[3] と図形リテラル[3] の比較 • 図形リテラル[3] と 16 進リテラル[2] の比較 • 16 進リテラル[2] と図形フィールドの比較 • 16 進リテラル[2] と図形リテラル[3] の比較

注:

1. 文字リテラルは 'abc' の形式になっています。
2. 16 進リテラルは X'16 進数字'の形式になっています。
3. 図形リテラルは G'oK1K2i' の形式となっています。シフトアウトは o で、シフトインは i で表されます。

ステートメント番号を使用したジョブ・ブレークポイントの設定および除去

問題のモジュールのコンパイラー・リストから分かるステートメント番号を使って、条件付きまたは無条件のジョブ・ブレークポイントを設定、または削除します。DBGVIEW(*STMT) を用いて作成されたモジュールをデバッグしたい場合には、これが必要です。

BREAK デバッグ・コマンドを使って無条件ジョブ・ブレークポイントを設定するには、デバッグ・コマンド行に、

```
BREAK procedure-name/statement-number
```

と入力します。変数 *procedure-name* は、ブレークポイントを設定するプロシージャーの名前です。ILE RPG では、1 つのモジュールにつき複数のプロシージャーが許されるため、*procedure-name* は、メイン・プロシージャーの名前でも、モジュール内のサブプロシージャーの 1 つでもかまいません。変数 *statement-number* はブレークポイントを設定したいコンパイラー・リストからのステートメント番号です。

注: ソース・リスト内のステートメント番号には、OPTION(*NOSRCSTMT) が指定された場合には行番号、また、OPTION(*SRCSTMT) が指定された場合にはステートメント番号というラベルが付けられます。例えば、図 101 は、OPTION(*NOSRCSTMT) が指定された場合のリストのセクション例を示しています。図 102 は、OPTION(*SRCSTMT) が指定された場合の同じセクションを示しています。

```
行  <----------------------------- ソースの仕様 -------------------------------------------------><------- 注記 ------> SRC SEQ
番号   ....1....+....2....+<-------- 26 - 35 -------->....4....+....5....+....6....+....7....+....8....+....9....+...10 ID  番号
                        ソ ー ス ・ リ ス ト
     1 C                   MOVE                     '123'         BI_FLD1                                                000100
     2 C                   SETON                                                       LR----                            000200
       * * * * *   ソ ー ス の 終 わ り   * * * * *
```

*図 101. OPTION(*NOSRCSTMT) が指定された場合のリストのセクション例*

```
SEQ    <-------------------------- ソースの仕様 -------------------------------------------------><------- 注記 ------> STATEMENT
番号   ....1....+....2....+<-------- 26 - 35 -------->....4....+....5....+....6....+....7....+....8....+....9....+...10 番号
                        ソ ー ス ・ リ ス ト
000100 C                   MOVE                     '123'         BI_FLD1                                                000100
000200 C                   SETON                                                       LR----                            000200
       * * * * *   ソ ー ス の 終 わ り   * * * * *
```

*図 102. OPTION(*SRCSTMT) が指定された場合のコンパイラー・リストのセクション例*

この例では、プロシージャー TEST についてブレークポイントを設定するのにステートメント・ビューが使用されています。*NOSRCSTMT リストを使用してモジュールのブレークポイントを設定するためには、次のように入力します。

```
BREAK TEST/2
```

*SRCSTMT リストを使用してモジュールのブレークポイントを設定するためには、次のように入力します。

```
BREAK TEST/200
```

いずれの場合も、ブレークポイントは 'SETON   LR----' 行に設定されます。

```
                         モジュール・ソースの表示
PROGRAM:   TEST          LIBRARY:   MYLIB          MODULE:   TEST
  (ソースを使用することができない)
                                                                    終わり
デバッグ. . .   break TEST/2______________________________________________

F3=終了プログラム   F6=停止点の追加/消去   F10=ステップ     F11=変数の表示
F12=再開            F17=ウォッチ変数   F18=ウォッチの処理   F23=表示出力
停止点がプロシージャー TEST のステートメント 2 に追加されました。
```

図 103. ステートメント・ビューを使用したブレークポイントの設定

その他のすべてのデバッグ・ビューについて、デバッガー内でプログラムの行番号のほかにステートメント番号を使用できます。例えば、以下のリスト・ビューのサブプロシージャーの始めにブレークポイントを設定するには、次のように入力します。

```
BREAK 34
```

または

```
BREAK FmtCust/2600
```

いずれの場合も、ブレークポイントは 'P FmtCust   B' 行に設定されます。

```
                         モジュール・ソースの表示
PROGRAM:   MYPGM         LIBRARY:   MYLIB          MODULE:   MYPGM
    33     002500  * Begin-procedure
    34     002600 P FmtCust         B
    35     002700 D FmtCust         PI            25A
    36     002800  * Procedure-interface (same as the prototype)
    37     002900 D    FirstName                  10A
    38     003000 D    LastName                   15A
    39     003100 D    ValidRec                     N
    40     003200  * Calculations
    41     003300 C                   IF        ValidRec = '0'
    42     003400 C                   RETURN    %TRIMR(FirstName) + ' ' + Last
    43     003500 C                   ENDIF
    44     003600 C                   RETURN    'Last Customer'
    45     003700  * End-procedure
    46     003800 P                 E
    47           *MAIN PROCEDURE EXIT
                                                                    終わり
デバッグ . . .   BREAK fmtcust/2600________________________________________

F3=プログラムの終了  F6=停止点の追加/消去   F10=ステップ     F11=変数の表示
F12=再開             F17=ウォッチ変数   F18=ウォッチの処理   F23=表示出力
停止点が行 34 に追加されました。
```

*図 104. ステートメント番号を使用したブレークポイントと、OPTION(*SRCSTMT) の指定されたリスト・ビューの設定*

BREAK デバッグ・コマンドを使って条件付きジョブ・ブレークポイントを設定するには、デバッグ・コマンド行に次のように入力します。

```
BREAK procedure-name/statement-number WHEN expression
```

と入力します。変数の *procedure-name* および *statement-number* は無条件ブレークポイントのものと同じです。変数の *expression* はブレークポイントに達した時に評価される条件式です。

CLEAR デバッグ・コマンドを使って無条件ブレークポイントまたは条件付きブレークポイントを除去するためには、デバッグ・コマンド行に次のように入力します。

```
CLEAR procedure-name/statement-number
```

と入力します。

条件付きスレッド・ブレークポイントの設定および除去

以下を使って、条件付きスレッド・ブレークポイントを設定または除去することができます。

- 「モジュール・ブレークポイントの処理」画面
- 現行スレッドの条件付きスレッド・ブレークポイントを設定する TBREAK デバッグ・コマンド
- 条件付きスレッド・ブレークポイントを除去する CLEAR デバッグ・コマンド

モジュール・ブレークポイントの処理画面の使用

「モジュール・ブレークポイントの処理」画面を使って条件付きスレッド・ブレークポイントを設定するには、次のようにします。

1. *OPT* フィールドに 1 (追加) を入力します。
2. スレッド・フィールドにスレッド識別子を入力します。
3. 残りのフィールドを、条件付きジョブ・ブレークポイントと同様に埋めます。
4. 実行 (キー) を押します。

「モジュール・ブレークポイントの処理」画面を使って条件付きスレッド・ブレークポイントを除去するには、次のようにします。

1. 除去したいブレークポイントの次の *OPT* フィールドに 4 (消去) を入力します。
2. 実行 (キー) を押します。

TBREAK または CLEAR デバッグ・コマンドの使用

TBREAK デバッグ・コマンドの構文は BREAK デバッグ・コマンドの場合と同じ構文を使います。BREAK デバッグ・コマンドはすべてのスレッドの同じ位置で条件付きジョブ・ブレークポイントを設定するのに対して、TBREAK デバッグ・コマンドは現行スレッドで条件付きスレッド・ブレークポイントを設定する点が異なります。

条件付きスレッド・ブレークポイントを除去するには、CLEAR デバッグ・コマンドを使います。条件付きスレッド・ブレークポイントが消去される時は、現行スレッドのみが対象です。

ジョブとスレッドすべてのブレークポイントの除去

「モジュール・ソースの表示」画面に表示されるモジュール・オブジェクトを持つプログラム・オブジェクトから、条件付きおよび無条件のジョブおよびスレッド・ブレークポイントのすべてを、CLEAR PGM デバッグ・コマンドを使って除去することができます。デバッグ・コマンドを使用するためには、デバッグ・コマンド行に

```
CLEAR PGM
```

と入力します。プログラムにバインドされたすべてのモジュールからブレークポイントが除去されます。

ウォッチ条件の設定および除去

プログラムの実行時に、式または変数の現行値が変更されるかどうかをモニターするためには、**ウォッチ条件**を使用します。ウォッチ条件の設定は、条件付きブレークポイントの設定とほぼ同じですが、次のような重要な相違点が 1 つあります。

- ウォッチ条件では、ウォッチされている式または変数が現行値から変更されると、ただちにプログラムが停止します。
- 条件付きジョブ・ブレークポイントがプログラムを停止させるのは、変数が条件で指定された値に変わる場合に限られます。

デバッガーは、ウォッチ条件の設定時に算出される**記憶域のアドレス**の内容によって、式または変数をウォッチします。記憶域アドレスの内容が、ウォッチ条件が設定された時、または最後のウォッチ条件が発生した時の値から変わった時にプログラムは停止します。

注: ウォッチ条件が登録された後、ウォッチされている記憶域の位置における新しい内容が、対応する式または変数の新しい現行値として保管されます。ウォッチされている記憶域の位置の新しい内容がそれ以降に変更されると、次のウォッチ条件が登録されます。

ウォッチの特性

ウォッチを行う前に、ウォッチについて次の特性を知っておいてください。

- ウォッチはシステム全体で行われます。同時に活動状態にできるウォッチの最大数は 256 です。この数には、システムが設定したウォッチの数が含まれています。

  システム全体の使用状況に応じて、所定の時間に設定できるウォッチ条件の数が制限されることがあります。システム全体で活動状態のウォッチの最大数を超えているときに、さらにウォッチ条件を設定しようとすると、エラー・メッセージが出され、このウォッチ条件は設定されません。

  注: 式または変数がページ境界にまたがっている場合、記憶域位置のモニターには内部的に 2 つのウォッチ条件が使用されます。したがって、システム全体で同時にウォッチできる式または変数の最大数は、 128 から 256 までです。
- ウォッチ条件が設定できるのは、デバッグによりプログラムが停止されている場合に、ウォッチされる式または変数が有効範囲内にあるときだけです。これにあてはまらない場合にウォッチが要求されると、対応する呼び出しスタック項目が存在しないことを示すエラー・メッセージが出されます。
- ウォッチ条件が設定された後は、ウォッチされる記憶域位置のアドレスは変更されません。したがって、仮の位置にウォッチが設定されると、ウォッチ条件の疑似通知が出される場合があります。

  この例として、ILE RPG サブプロシージャーの自動記憶域があり、これはサブプロシージャーの終了後に再使用することができます。

  ウォッチされる変数がもはや有効範囲内にない場合であっても、ウォッチ条件が登録される場合があります。そのため、ウォッチ条件が報告されたからといって、変数が有効範囲内にあると見なすことはできません。

- 同一ジョブ内の 2 つのウォッチ位置がオーバーラップしてはなりません。異なるジョブの 2 つのウォッチ位置は、同一の記憶域アドレスから開始してはなりません。そうでない場合は、オーバーラップが使用できます。こうした制約に違反すると、エラー・メッセージが出されます。

  注: ウォッチされる記憶域の位置が、このウォッチ条件を設定したジョブとは異なるジョブで変更される場合、この変更は無視されます。
- コマンドが正常に実行された後、セッション内のプログラムがウォッチされる記憶域位置の内容を変更すると、アプリケーションは停止し、**モジュール・ソース表示**画面が表示されます。

  プログラムにデバッグ・データがある場合、使用可能なソース・テキスト・ビューがあれば表示されます。記憶域位置における内容の変更が検出されたときに実行されようとしていたステートメントのソース行が強調表示され、ウォッチ条件が満たされたことを示すメッセージが出されます。

  プログラムをデバッグすることができない場合、画面のテキスト部分はブランクになります。
- 適格なプログラムがウォッチを停止させると、このプログラムは、自動的にデバッグ・セッションに追加されます。
- 同一プログラム・ステートメントが、複数のウォッチ条件に該当した場合には、最初のウォッチ条件のみが報告されます。
- デバッグにサービス・ジョブを使用している時、すなわちあるジョブから別のジョブをデバッグする時にもウォッチ条件を設定できます。

ウォッチ条件の設定

ウォッチ条件を設定する前に、プログラムをデバッグの制御下で停止する必要があり、しかもウォッチを行いたい式または変数が有効範囲内になければなりません。

- グローバル変数をウォッチするには、ウォッチ条件の設定前に、変数が定義されているプログラムが活動状態であることを確認する必要があります。
- ローカル変数をウォッチするには、ウォッチ条件を設定する前にその変数が定義されるプロシージャーに飛び込む必要があります。

次を使用して、ウォッチ条件を設定することができます。

- F17 (ウォッチ変数)。カーソルが置かれている変数のウォッチ条件を設定します。
- パラメーターの有無に関係なく、WATCH デバッグ・コマンド。

WATCH コマンドの使用

WATCH コマンドを使用する場合には、単一のコマンドとして入力しなければなりません。同一のコマンド入力行で他のデバッグ・コマンドを使用することはできません。

- 下記に表示される**ウォッチの処理**画面にアクセスするには、パラメーターを指定せず、デバッグのコマンド入力行で次をタイプします。

  ```
  WATCH
  ```

```
                          ウォッチの処理
                                                  システム :  DEBUGGER
 オプションを入力して、実行キーを押してください。
   4=消去 5=表示
 OPT     NO        変数                      アドレス           長さ
  -      1         SALARY                    080090506F027004   4
                                                                 終わり
コマンド
===>_____________________________________________________________________
F3=終了   F4=プロンプト   F5=最新表示    F9=検索    F12=取消し
```

図 105. 「ウォッチの処理」画面の例

ウォッチの処理画面は、デバッグ・セッションで現在活動状態にあるすべてのウォッチを表示します。この画面からウォッチを消去し表示することができます。オプション 5 の**表示**を選択すると、下記の**ウォッチ表示**ウィンドウに、現在活動状態にあるウォッチについての情報が表示されます。

```
                          ウォッチの処理
.....................................................
:                    ウォッチの表示                 :  DEBUGGER
:                                                   :
:  ウォッチ番号 ....:   1                           :
:  アドレス.........:   080090506F027004            :
:  長さ   ..........:   4                           :
:  ヒットの数 ......:   0                           :
:                                                   :
:  ウォッチ送信時の範囲    :                        :
:    プログラム/ライブラリー/タイプ: PAYROLL  ABC  *PGM  :
:                                                   :
:    モジュール.....:    PAYROLL                    :
:    プロシージャー.:    PAYROLL                    :
:    変数 ..........:    SALARY                     :
:                                                   :
:  F12=取消し                                       :
:                                                   :
.....................................................
                                                                 終わり
コマンド
===>_____________________________________________________________________
F3=終了   F4=プロンプト   F5=最新表示    F9=検索    F12=取消し
```

図 106. 「ウォッチの表示」ウィンドウの例

- ウォッチの対象となる変数または式を指定するには、デバッグ・コマンド行に次のように入力します。

  ```
  WATCH expression
  ```

  と入力します。

  このコマンドは、expression の値が現行値から変更される場合に、ブレークポイントを設定するよう要求します。

  注: 式は、ウォッチの対象となる記憶域位置のアドレスを判別するのに使用され、割り当て可能な位置に分析解決する必要があります。例えば以下のように指定します。

  ```
  %SUBSTR(X 1 5)
  ```

  ウォッチにおける式の変数の有効範囲は、直前に出した QUAL コマンドによって定義されます。

- ウォッチ条件を設定し、あわせてウォッチの長さを指定するには、次のように、

  ```
  WATCH expression : watch length
  ```

  とデバッグ・コマンド行に入力します。

  ウォッチごとに、最大 128 バイトの連続記憶域をモニターし、比較することができます。128 バイトの最大長を超えると、ウォッチ条件は設定されず、デバッガーがエラー・メッセージを出します。

  デフォルトでは、式タイプの長さは、ウォッチ比較命令の長さでもあります。**ウォッチの長さパラメーター**は、このデフォルト値を一時変更します。このパラメーターは式のバイト数を判別するので、このバイト数を比較して、値が変更されたかどうかを判別します。

  例えば、ウォッチの長さパラメーターを指定せずに 4 バイト整数が変数として指定されている場合、比較の長さは 4 バイトです。ただし、ウォッチの長さパラメーターを指定すると、ウォッチの長さを判別する際の式の長さを一時変更します。

活動状態のウォッチの表示

システム全体における活動状態のウォッチのリストを表示し、どのジョブにウォッチが設定されているかを表示するには、デバッグ・コマンド入力行に次を入力します。

```
DSPDBGWCH
```

とデバッグ・コマンド行に入力します。下記の**デバッグ・ウォッチの表示**画面が表示されます。

```
                          デバッグ・ウォッチの表示
                                                    システム :  DEBUGGER
 ------------ジョブ------------          番号       長さ    アドレス
 MYJOBNAME1  MYUSERPRF1  123456           1          5     080090506F027004
 JOB4567890  PRF4567890  222222           1          8     09849403845A2C32
 JOB4567890  PRF4567890  222222           2          2     098494038456AA00
 JOB         PROFILE     333333          14          4     040689578309AF09
 SOMEJOB     SOMEPROFIL  444444           3          4     005498348048242A
                                                                  終わり
続行するためには、実行キーを押してください。
F3=終了   F5=最新表示   F12=取消し
```

図 107. 「デバッグ・ウォッチの表示」画面の例

注: この画面は、システムが設定したウォッチ条件は表示しません。

ウォッチ条件の除去

ウォッチの除去は、以下の方法で行うことができます。

- WATCH キーワードを指定して CLEAR コマンドを使用すると、 1 つまたはすべてのウォッチを選択的に終了させます。例えば、**ウォッチの数**によって識別されるウォッチを消去するには、次を入力します。

  ```
  CLEAR WATCH watch-number
  ```

  ウォッチの数は、**ウォッチの処理**画面から入手することができます。

セッションのウォッチをすべて消去するには、次のように入力します。

```
CLEAR WATCH ALL
```

とデバッグ・コマンド行に入力します。

注: CLEAR PGM コマンドは、表示されるモジュールを含むプログラムのブレークポイントをすべて除去しますが、ウォッチには影響を与えません。ウォッチ条件を除去するには、CLEAR コマンドの WATCH キーワードを明示的に使用する必要があります。

- CL デバッグ・モード終了 (ENDDBG) コマンドは、ローカル・ジョブまたはサービス・ジョブ内のウォッチを除去します。

  注: 異常時には ENDDBG が自動的に呼び出され、影響を受けたウォッチがすべて除去されるようにします。

- iSeries システムの初期プログラム・ロード (IPL) は、システム全体のウォッチ条件をすべて除去します。

ウォッチ条件の設定例

この例では、プログラム MYLIB/PAYROLL の変数 SALARY をウォッチします。ウォッチ条件を設定するには、次のように入力します。

```
WATCH SALARY
```

とデバッグ行に入力して、ウォッチの長さのデフォルト値を受け入れます。

変数 SALARY の値をこれ以降に変更すると、アプリケーションは停止し、図 108 に図示されているように、**モジュール・ソースの表示**画面が表示されます。

```
                              モジュール・ソースの表示
PROGRAM:   PAYROL          LIBRARY:   MYLIB       MODULE:  PAYROLL
    52  C                  eval       cnt = 1
    53  C                  dow        (cnt < EMPMAX)
    54  C                  eval       Pay_exmpt(cnt) = eflag(cnt)
    55  C                  eval       cnt = cnt + 1
    56  C                  enddo
    57  C
    58  C                  eval       index = 1
    59  C                  dow        index <= cnt
    60  C                  if         Pay_exmpt(index) = 1
    61  C                  eval       SALARY = 40 * Pay_wage(index)
    62  C                  eval       numexmpt = numexmpt + 1
    63  C                  else
    64  C                  eval       SALARY = Pay_hours(index)*Pay_wage(index)
    65  C                  endif
    66  C                  eval       index = index + 1
    67  C                  enddo
                                                                  続く...
デバッグ . . .
F3=終了プログラム  F6=停止点の追加/消去  F10=ステップ  F11=変数の表示
F12=再開       F17=ウォッチ変数   F18=ウォッチの処理   F23=表示出力
行 65 の監視番号 1 ,   変数 : SALARY
```

図 108. WATCH (ウォッチ) は正常に設定されたことを伝えるメッセージの例

- ウォッチ変数の変更が検出されたステートメントの行番号が強調表示されます。これは通常、変数を変更したステートメントの後に続く 最初の実行可能行です。

- ウォッチ条件が満たされたことを示すメッセージが出されます。

  注: テキスト・ビューが利用できない場合、ブランクの**モジュール・ソースの表示**画面が表示され、メッセージ域に上と同じメッセージが出されます。

  次のプログラムは、ILE デバッグ環境に追加することはできません。

  1. デバッグ・データのない ILE プログラム
  2. 非ソース・デバッグ・データだけがある OPM プログラム
  3. デバッグ・データのない OPM プログラム

最初の 2 つのプログラムの場合は、停止したステートメント番号が渡されます。3 番目のプログラムの場合は、停止した MI 命令が渡されます。下に示すブランクの**モジュール・ソースの表示**画面の下に情報が表示されます。行番号ではなく、ステートメントまたは命令番号が表示されます。

```
                              モジュール・ソースの表示
(ソースを表示することができない)
F3=終了プログラム  F12=再開  F14=モジュール・リストの処理  F18=ウォッチの処理
F21=コマンド入力  F22=ステップイン  F24=キーの続き
命令 18 の監視番号 1 ,   変数: SALARY
```

図 109. 「モジュール・ソース表示」パネルの例

プログラム・オブジェクトのステップスルー

ブレークポイントに達した後で、プログラム・オブジェクトの指定された数のステートメントを実行し、次にプログラムを再び停止して「モジュール・ソースの表示」画面に戻ることができます。これを行うには、ILE ソース・デバッガーのステップ機能を使います。プログラム・オブジェクトはプログラムが停止したモジュール・オブジェクトの次のステートメントで実行を再開します。通常、ブレークポイントはプログラム・オブジェクトを停止するために使用されます。

ブレークポイントは、プログラムの呼び出し前およびプログラムのステップスルー中に設定できます。ブレークポイントは、デフォルトの OPTION(*DEBUGIO) が指定された場合には、入出力仕様書について自動的に生成されます。このオプションを選択した場合、READ ステートメントに STEP を指定すると、入力仕様書は停止されます。OPTION(*NODEBUGIO) を指定すると、入出力仕様書についてブレークポイントを生成しないよう選択できます。

OPM プログラムに利用可能なデバッグ・データがある場合、およびデバッグ・セッションが OPM プログラムを受け入れてデバッグを行う場合には、 OPM プログラムにステップインすることができます。

次を使用してプログラム・オブジェクトをステップスルーすることができます。

- 「モジュール・ソースの表示」画面の F10 (ステップ) または F22 (ステップイン)
- STEP デバッグ・コマンド

一度に 1 ステートメントずつプログラム・オブジェクトをステップスルーする最も単純な方法は、「モジュール・ソースの表示」画面で F10 (ステップ) または F22 (ステップイン) を使用することです。 F10 (ステップ) または F22 (ステップイン)

を押すと、「モジュール・ソースの表示」画面に示されたモジュール・オブジェクトの次のステートメントが実行され、プログラム・オブジェクトが再び停止します。

注: F10 (ステップ) または F22 (ステップイン) を使用する時にステップスルーするステートメントの数を指定することはできません。 F10 (ステップ) または F22 (ステップイン) を押すと、シングル・ステップが実行されます。

プログラム・オブジェクトをステップスルーするもう 1 つの方法は、 STEP デバッグ・コマンドを使用する方法です。 STEP デバッグ・コマンドによって、シングル・ステップで複数のステートメントを実行することができます。STEP デバッグ・コマンドを使用して実行するステートメントのデフォルトの数は 1 つです。 STEP デバッグ・コマンドを使用してプログラム・オブジェクトをステップスルーするには、デバッグ・コマンド行に次を入力します。

```
STEP number-of-statements
```

と入力します。変数 *number-of-statements* は、プログラム・オブジェクトがもう再度停止する前に次のステップで実行したいプログラム・オブジェクトのステートメントの数です。 例えば、

```
STEP 5
```

プログラム・オブジェクトの次ぎ 5 つのステートメントが実行され、次のプログラム・オブジェクトが再度停止し、「モジュール・ソースの表示」画面が表示されます。

別のプログラムまたはプロシージャーへの呼び出しステートメントがデバッグ・セッションに出てきた場合には、次のようにできます。

- 呼び出しステートメントをステップオーバーする。または
- 呼び出しステートメントにステップインする。

ILE RPG の呼び出しステートメントには、次の命令のいずれかが含まれています。

- CALL
- CALLB
- CALLP
- 拡張演算項目 2 フィールドに式のある命令で、その式にはプロシージャーに対する呼び出しが入っています。

呼び出しステートメントの**ステップオーバー**を選択した場合には、現行プロシージャー内に留まります。呼び出しステートメントはシングル・ステップとして処理され、カーソルはその呼び出し後に次のステップに移動します。ステップオーバーはデフォルトのステップ・モードです。

呼び出しステートメントへの**ステップイン**を選択した場合には、呼び出しステートメント内部の各ステートメントがシングル・ステップとして実行されます。指定されたステップの数に従って、ステップ・コマンドは呼び出しステートメント内で終了することがありますが、この場合には、呼び出しステートメントのソースが「モジュール・ソースの表示」画面に表示されます。

注: RPG サブルーチンではステップオーバーまたはステップインすることはできません。しかし、サブプロシージャーではステップオーバーおよびステップインすることができます。

呼び出しステートメントのステップオーバー

次を使用して、呼び出しステートメントをステップオーバーすることができます。

- 「モジュール・ソースの表示」画面の F10 (ステップ)
- STEP OVER デバッグ・コマンド

「モジュール・ソースの表示」画面の F10 (ステップ) を使って、デバッグ・セッションの呼び出しステートメントをステップオーバーすることができます。実行する呼び出しステートメントが別のプログラム・オブジェクトに対する CALL 命令である場合には、F10 (ステップ) を押すことによって、呼び出しているプログラム・オブジェクトが再び停止する前に、呼び出し先プログラム・オブジェクトを完了するために実行されます。同様に、呼び出しステートメントが、式でプロシージャーが呼び出される EVAL 命令である場合は、呼び出し側プログラムまたはプロシージャーが再度停止する前に、プロシージャーに対する呼び出しを含め、完全な EVAL 命令が実行されます。

もう 1 つの方法として STEP OVER デバッグ・コマンドを使用して、デバッグ・セッションの呼び出しステートメントをステップオーバーすることができます。STEP OVER デバッグ・コマンドを使用するためには、デバッグ・コマンド行に次を入力します。

```
STEP number-of-statements OVER
```

と入力します。変数 *number-of-statements* は、処理が再び停止する前に次のステップで実行するステートメントの数です。この変数を省略した場合、デフォルトの値は 1 です。

呼び出しステートメントへのステップイン

次を使用して、呼び出しステートメントへステップインすることができます。

- 「モジュール・ソースの表示」画面の F22 (ステップイン)
- STEP INTO デバッグ・コマンド

「モジュール・ソースの表示」画面の F22 (ステップイン) を使って、デバッグ・セッションの呼び出し先プログラムまたはプロシージャーにステップインすることができます。実行する次のステートメントが別のプログラムまたはプロシージャーへの呼び出しステートメントの場合には、F22 (ステップイン) を押すことによって、呼び出し先プログラムまたはプロシージャーの最初の実行可能なステートメントを実行することができます。次に、呼び出し先プログラムまたはプロシージャーが「モジュール・ソースの表示」画面に表示されます。

注: 呼び出し先プログラムまたはプロシージャーは、それが「モジュール・ソースの表示」画面に表示されるように、それに関連したデバッグ・データを持っていなければなりません。

もう 1 つの方法として STEP INTO デバッグ・コマンドを使用して、デバッグ・セッションで呼び出しステートメントをステップインすることができます。STEP INTO デバッグ・コマンドを使用するためには、デバッグ・コマンド行に次を入力します。

```
STEP number-of-statements INTO
```

と入力します。変数 *number-of-statements* は、処理が再び停止する前に次のステップで実行するステートメントの数です。この変数を省略した場合、デフォルトの値は 1 です。

実行されるステートメントの 1 つに呼び出しステートメントが入っている場合には、デバッガーが呼び出し先プログラムまたはプロシージャーにステップインします。呼び出し先プログラムまたはプロシージャーの各ステートメントはステップでカウントされます。ステップが呼び出し先プログラムまたはプロシージャーで終了する場合には、この呼び出し先プログラムまたはプロシージャーが「モジュール・ソースの表示」画面に表示されます。例えば、

```
STEP 5 INTO
```

とデバッグ・コマンド行に入力すると、プログラム・オブジェクトの次の 5 つのステートメントが実行されます。 3 番目のステートメントが別のプログラム・オブジェクトに対する CALL 命令である場合には、呼び出し側プログラム・オブジェクトの 2 つのステートメントが実行され、呼び出し先プログラム・オブジェクトの最初の 3 つのステートメントが実行されます。

DEBUGEX の例では、プロシージャー c_proc を呼び出す EVAL 命令で STEP INTO を入力した (または F22 を押した) 場合に、C モジュールにステップインします。

STEP INTO コマンドは、CL CALL コマンドを同様に処理します。呼び出しの後で、これを利用してプログラムをステップスルーすることができます。ソース・デバッガーを開始した後、「モジュール・ソースの表示」画面から、次の入力をします。

```
STEP 1 INTO
```

これでステップ・カウントは 1 に設定されます。F12 を使ってコマンド行に戻り、次にプログラムを呼び出します。プログラムは、デバッグ・データをもつ最初のステートメントで停止します。

ヒント

サブプロシージャーの実行の直前または直後にデータを表示するためには、サブプロシージャーを開始および終了するプロシージャー仕様書にブレークポイントを置きます。

F22 を使用した OPM プログラムのステップインの例

この例では、F22 (ステップイン) を使用してプログラム DEBUGEX から OPM プログラム RPGPGM にステップインします。

1. 「モジュール・ソースの表示」画面が DBGEX のソースを表示します。

2. CALL 命令の前の最後の実行可能ステートメントである無条件ブレークポイントを行 102 に設定するために、Break 102 を入力して、実行キーを押してください。
3. F3 (プログラム終了) を押して、「モジュール・ソースの表示」画面を終わりにします。
4. プログラムを呼び出してください。プログラムは 図 110 に示すようにブレークポイント 102 で停止します。

```
                          モジュール・ソースの表示
PROGRAM:   DEBUGEX        LIBRARY:   MYLIB          MODULE:   DBGEX
    98       * FLD1A は FLD1 のオーバーレイ・フィールドです。FLD1 は 'ABCDE' に
    99       * 初期設定されているので、FLD1A(1) の値は 'A' です。次の
   100       * MOVE 命令の後の FLD1A(1) の値は '1' になります。
   101       *-----------------------------------------------------------------
   102      C                   MOVE      '1'           FLD1A(1)
   103
   104       *-----------------------------------------------------------------
   105       * 別のプログラム・オブジェクトであるプログラム RPGPGM を
             * 呼び出します。
   106       *-----------------------------------------------------------------
   107      C     PLIST1        PLIST
   108      C                   PARM                    PARM1
   109      C                   CALL      'RPGPGM'      PLIST1
   110
   111       *-----------------------------------------------------------------
   112       * メイン・プロシージャーから EXPORTFLD をインポートする C_PROC を
             * 呼び出します。
                                                                  続く...
デバッグ . . .  ____________________________________________________________
F3=終了プログラム  F6=停止点の追加/消去  F10=ステップ          F11=変数の表示
F12=再開           F17=ウォッチ変数       F18=ウォッチの処理    F23=表示出力
行 102 が停止点。
```

図 110. RPGPGM にステップインする前の DBGEX の「モジュール・ソースの表示」画面

5. F22 (ステップイン) を押してください。プログラムの 1 ステートメントが実行され、 253 ページの図 111 に示すように RPGPGM の「モジュール・ソースの表示」画面が表示されます。

   この場合、RPGPGM の最初の実行可能ステートメントが処理されてから (行 13)、プログラムが停止します。

   注: F22 を使用する時にはステップスルーするステートメントの数を指定することはできません。F22 キーを押すと、1 つのステップが実行されます。

```
                          モジュール・ソースの表示
PROGRAM:   RPGPGM         LIBRARY:   MYLIB
     1       *====================================================================
     2       *  RPGPGM - DEBUGEX によって呼び出されるプログラム。ILE ソース・
     3       *           デバッグ・プログラムの STEP 機能を説明しています。
     4       *
     5       *  このプログラムは DEBUGEX から INPUTPARM を受け取り、
     6       *  それを表示してから戻ります。
     7       *====================================================================
     8
     9      D INPUTPARM       S              4P 3
    10
    11      C     *ENTRY        PLIST
    12      C                   PARM                    INPUTPAEM
    13      C     InputParm     DSPLY
    14      C                   SETON
                                                                        終わり
デバッグ . . .  ____________________________________________________________________
F3=終了プログラム  F6=停止点の追加/消去  F10=ステップ          F11=変数の表示
F12=再開           F17=ウォッチ変数                 F18=ウォッチの処理  F23=表示出力
行 13 でステップが完了した。
```

図 111. RPGPGM へのステップイン

ILE ソース・デバッガーが OPM プログラムを受け入れるように設定されていない場合、あるいはデバッグ・データが利用できない場合には、ソースが利用不能であることを知らせるメッセージを持つ、ブランクの「モジュール・ソースの表示」画面が表示されます (OPM プログラムが OPTION(*SRCDBG) または OPTION(*LSTDBG) を使用してコンパイルされている場合には、このプログラムにはデバッグ・データがあります)。

サブプロシージャーへのステップインの例

この例では、F22 (ステップイン) を使用してサブプロシージャー Switch にステップインしますが、これはモジュール DEBUGEX にあります。

1. 「モジュール・ソースの表示」画面が DBGEX のソースを表示します。
2. CALLP 命令の前の最後の実行可能ステートメントである行 120 に無条件ブレークポイントを設定するために `Break 120` を入力して実行キーを押してください。
3. F3 (プログラム終了) を押して、「モジュール・ソースの表示」画面を終わりにします。
4. プログラムを呼び出してください。プログラムはブレークポイント 119 で停止します。
5. F22 (ステップイン) を押してください。呼び出しステートメントが実行されてから、 254 ページの図 112 に示されたように表示はサブプロシージャーに移動します。RPGPGM の最初の実行可能ステートメントが処理 (行 13) されてから処理が停止します。

```
                          モジュール・ソースの表示
PROGRAM:   DEBUGEX        LIBRARY:   MYLIB          MODULE:   DBGEX
   141
   142        *============================================================
   143        * サブプロシージャー SWITCH の定義
   144        *============================================================
   145       P SWITCH          B
   146       D SWITCH          PI
   147       D   PARM                        1A
   148        *------------------------------------------------------------
   149        * デバッグのためのローカル変数の定義
   150        *------------------------------------------------------------
   151       D Local           S               5A      INZ('aaaaa')
   152
   153       C                   IF        Parm = '1'
   154       C                   EVAL      Parm = '0'
   155       C                   ELSE
デバッグ . . .   ____________________________________________________________
F3=終了プログラム  F6=停止点の追加/消去  F10=ステップ           F11=変数の表示
F12=再開           F17=ウォッチ変数      F18=ウォッチの処理     F23=表示出力
行 145 でステップが完了した。
```

図 112. サブプロシージャー SWITCH へのステップイン

データおよび式の表示

フィールド、データ構造、および配列の内容を表示したり、式を評価することができます。表示または評価には次の 2 つの方法があります。

- F11 (変数の表示)
- EVAL デバッグ・コマンド

DS.SUBF 形式の簡単な修飾名の場合は、次のコマンドのいずれかを使用して、変数を表示または変更することができます。

```
EVAL SUBF OF DS
EVAL DS.SUBF
```

複雑な修飾名の場合は、ドット修飾形式の名前を使用します。

```
EVAL FAMILY.CHILD(2).PETS.PET(3).NAME
```

EVAL コマンドで使われるフィールドの有効範囲は、ILE C のような言語の QUAL コマンドを使って定義することができます。ただしこのコマンドは現在、ILE RPG に適用することはできません。

注: EVAL デバッグ・コマンドには使用可能な外部名がないので、戻り値を表示することができません。

データまたは式を表示する最も簡単な方法は、「モジュール・ソースの表示」画面で F11 (変数の表示) を使用することです。F11 (変数の表示) を使ってフィールドを表示するためには、表示したいフィールドにカーソルを置き、F11 (変数の表示) を押してください。フィールドの現在の値は「モジュール・ソースの表示」画面の最下部のメッセージ行に表示されます。

構造、レコード、または配列を評価している場合には、F11 (変数の表示) を押した時に戻されるメッセージが数行にわたることがあります。数行にわたるメッセージ

がテキスト全体のメッセージを示す「評価式」画面で表示されます。「評価式」画面でメッセージの表示を済ませたら、実行キーを押して「モジュール・ソースの表示」画面に戻ります。

EVAL デバッグ・コマンドを使用してデータを表示するためには、デバッグ・コマンド行に

```
EVAL field-name
```

と入力します。変数の *field-name* は、表示または評価したいフィールド、データ構造、または配列の名前です。 EVAL デバッグ・コマンドが「モジュール・ソースの表示」画面から入力され、値が単一行に表示できる場合には、メッセージ行に値が表示されます。そうでない場合には、これは「評価式」画面に表示されます。

図 113 には、EVAL デバッグ・コマンドを使用してサブフィールド LastName の内容を表示する例です。

```
                          モジュール・ソースの表示
PROGRAM:   DEBUGEX        LIBRARY:   MYLIB          MODULE:   DBGEX
    61       D  LastName                    10A   INZ('Jones    ')
    62       D  FirstName                   10A   INZ('Fred     ')
    63
    64        *---------------------------------------------------------------------
    65        * 呼び出されたプロシージャーと c_proc と switch のプロトタイプの定義
    66        *---------------------------------------------------------------------
    67       D c_proc          PR             *   EXTPROC('c_proc')
    68       D   size                        10U 0 VALUE
    69       D   inzval                       1A   CONST
    70
    71       D Switch          PR
    72       D   Parm                         1A
    73
    74        *---------------------------------------------------------------------
    75        * 非プロトタイプ呼び出しのパラメーターの定義
                                                                        続く...
デバッグ . . .   eval LastName_____________________________________________________
_____________________________________________________________________________
 F3=終了プログラム  F6=停止点の追加/消去  F10=ステップ          F11=変数の表示
 F12=再開           F17=ウォッチ変数      F18=ウォッチの処理    F23=表示出力
 LASTNAME = 'Jones     '
```

図 113. EVAL デバッグ・コマンドを使用したフィールドの表示

256 ページの図 114 には RPG フィールドのいろいろなタイプの EVAL コマンドの使用例が示されています。これらのフィールドは 268 ページの図 122 のソースに基づいています。ソース・デバッガーのオンライン・ヘルプには追加の例も提供されています。

```
スカラー・フィールド                RPG 定義
> EVAL String                              6A   INZ('ABCDEF')
  STRING = 'ABCDEF'
> EVAL Packed1D0                           5P 2 INZ(-93.4)
  PACKED1D0 = -093.40
> EVAL ZonedD3D2                           3S 2 INZ(-3.21)
  ZONEDD3D2 = -3.21
> EVAL Bin4D3                              4B 3 INZ(-4.321)
  BIN4D3 = -4.321
> EVAL Int3                                3I 0 INZ(-128)
  INT3 = -128
> EVAL Int5                                5I 0 INZ(-2046)
  INT5 = -2046
> EVAL Int10                              10I 0 INZ(-31904)
  INT10 = -31904
> EVAL Int20                              20I 0 INZ(-463972)
  INT20 = -463972
> EVAL Unsigned3                           3U 0 INZ(128)
  UNSIGNED3 = 128
> EVAL Unsigned5                           5U 0 INZ(2046)
  UNSIGNED5 = 2046
> EVAL Unsigned10                         10U 0 INZ(31904)
  UNSIGNED10 = 31904
> EVAL Unsigned20                         20U 0 INZ(463972)
  UNSIGNED20 = 463972
> EVAL DBCSString                          3G   INZ(G'~BBCCDD~')
  DBCSSTRING = '"BBCCDD"'
> EVAL NullPtr                              *   INZ(*NULL)
  NULLPTR = SYP:*NULL
基底付きフィールド
> EVAL String                              6A   INZ('ABCDEF')
  STRING = 'ABCDEF'
> EVAL BasePtr                              *   INZ(%ADDR(String))
  BASEPTR = SPP:C01947001218
> EVAL BaseString                          6A   BASED(BasePtr)
  BASESTRING = 'ABCDEF'
日付、時刻、タイム・スタンプ・フィールド
> EVAL BigDate                              D   INZ(D'9999-12-31')
  BIGDATE = '9999-12-31'
> EVAL BigTime                              T   INZ(T'12.00.00')
  BIGTIME = '12.00.00'
> EVAL BigTstamp                            Z   INZ(Z'9999-12-31-12.00.00.000000
  BIGTSTAMP = '9999-12-31-12.00.00.000000'
```

図 114. モジュール DBGEX に基づくサンプルの EVAL コマンド

変数評価時の予期しない結果

デバッグの最中に変数が予期しない値を表示したら、次のいずれかが該当していないかどうか調べてください。

- モジュールが最適化されている。モジュールが最適化されている場合は、デバッガーは変数の最新の値を示さないことがあります。またデバッガーを使って変数を変更した場合、その変更の効果はプログラムの実行順には反映されないことがあります。
- 入力フィールドの一部がファイルから読み込まれていない。通常、プログラムで使われていない入力フィールドは入力操作による影響を受けません。制御仕様書に DEBUG キーワードを指定すると、すべての入力フィールドが読み込まれます。

配列の内容の表示

EVAL で配列名を指定すると配列全体が表示されます。配列の 1 つの要素を表示するには、表示したい要素の指標を括弧内に指定してください。

ある範囲の要素を指定するためには、次の範囲表記法を使用してください。

```
EVAL field-name (n...m)
```

変数 *field-name* は配列の名前で、変数 *n* は範囲の始まりを示す数値で、変数 *m* は範囲の終わりを示す数値です。

図 115 は、DBGEX 中で配列に EVAL を使用した例を示したものです。

```
> EVAL Arry                              3S 2 DIM(2) INZ(1.23)
  ARRY(1) = 1.23    ** 配列全体の表示 **
  ARRY(2) = 1.23
> EVAL Arry(2)      ** 2 番目の要素の表示 **
  ARRY(2) = 1.23
> EVAL Arry(1..2)   ** 要素の範囲の表示 **
  ARRY(1) = 1.23
  ARRY(2) = 1.23
```

図 115. 配列に対するサンプルの EVAL コマンド

テーブルの内容の表示

テーブルに対して EVAL を使用すると、現在のテーブル要素が表示されます。範囲の表記法を使用してテーブル全体を表示することができます。例えば、3 要素テーブルを表示するには次を入力します。

```
EVAL TableA(1..3)
```

%INDEX 組み込み関数を使って現行の要素を変更することができます。テーブル指標の値を決定するには、次のコマンドを入力してください。

```
EVAL _QRNU_TABI_name
```

ここでは *name* は該当のテーブル名を表します。

258 ページの図 116 は DBGEX 中でテーブルに EVAL を使用した例を示したものです。

```
                                     3    DIM(3) CTDATA
                                     コンパイル時データ:  **
> EVAL TableA             ** 現行指標での                    aaa
  TABLEA = 'aaa'             値の表示                        bbb
                                                             ccc
> EVAL TableA(1)          ** 指標 1 の指定 **
  TABLEA(1) = 'aaa'
> EVAL TableA(2)          ** 指標 2 の指定 **
  TABLEA(2) = 'bbb'
> EVAL _QRNU_TABI_TableA ** 現行指標の値の表示 **
  _QRNU_TABI_TABLEA = 1
> EVAL TableA(1..3)       ** テーブル全体を指定 **
  TABLEA(1) = 'aaa'
  TABLEA(2) = 'bbb'
  TABLEA(3) = 'ccc'
> EVAL TableA=%INDEX(3)   ** 現行指標を 3 へ変更 **
> EVAL TableA
  TABLEA = 'ccc'
```

図 116. テーブルに対するサンプルの EVAL コマンド

データ構造の表示

独立したフィールドと同じように、データ構造またはそのサブフィールドの内容を表示します。内容全体を参照するために EVAL の後にデータ構造名を使用するか、あるいはサブセットを参照するためにサブフィールド名を使用するだけです。

データ構造が修飾されている場合は、次の表記のいずれかを使用してサブフィールドを指定します。

```
EVAL subfield-name OF datastructure-name
EVAL datastructure-name.subfield-name:
```

例えば、修飾されたデータ構造 INFO のサブフィールド NAME を表示するには、以下のいずれかを入力します。

```
EVAL NAME OF INFO
EVAL NAME OF INFO EVAL INFO.NAME
```

複数回繰り返しデータ構造を表示する時に、データ構造名に対して EVAL を使用すると、現行指標を使ってサブフィールドが表示されます。特定のオカレンスを指定するには、データ構造名の後の括弧内に指標を指定してください。例えば、DS1 の 2 番目の内容を表示するには次を入力してください。

```
EVAL DS1(2)
```

同様に、サブフィールドの特定のオカレンスの内容を表示するためには、指標表記法を使用してください。

現行指標の値を決定するには、次のコマンドを入力してください。

```
EVAL _QRNU_DSI_name
```

ここでは *name* は該当のデータ構造名を表します。

あるサブフィールドが別のサブフィールドの配列オーバーレイとして定義されている場合に、そのオーバーレイ・サブフィールドの内容を表示するためには、%INDEX 組み込み関数を使用してオカレンスを指定し、指標表記法を使用して配列を指定することができます。

配列オーバーレイであるサブフィールドを表示するもう 1 つの方法として、次の表記法を使用することができます。

```
EVAL subfield-name(occurrence-index,array-index)
```

ここで、変数 *subfield-name* は表示したいサブフィールドの名前で、 *occurrence-idex* は表示する配列オカレンスの番号、 *array-index* は表示する要素の番号です。

図 117 は、DBGEX で定義されたデータ構造に対して EVAL を使用するいくつかの例を示したものです。

```
** データ構造名またはサブフィールド名を入力できることに注意してください。**
> EVAL DS3
  TITLE OF DS3 = 'Mr.  '                  5A   INZ('Mr.  ')
  LASTNAME OF DS3 = 'Jones     '         10A   INZ('Jones    ')
  FIRSTNAME OF DS3 = 'Fred      '        10A   INZ('Fred     ')
> EVAL LastName
  LASTNAME = 'Jones     '
> EVAL DS1                                    OCCURS(3)
  FLD1 OF DS1 = 'ABCDE'                   5A   INZ('ABCDE')
  FLD1A OF DS1(1) = 'A'                   1A   DIM(5) OVERLAY(Fld1)
  FLD1A OF DS1(2) = 'B'                   5B 2 INZ(123.45)
  FLD1A OF DS1(3) = 'C'
  FLD1A OF DS1(4) = 'D'
  FLD1A OF DS1(5) = 'E'
  FLD2 OF DS1 = 123.45
> EVAL _QRNU_DSI_DS1     ** 現行指標値の判別 **
  _QRNU_DSI_DS1 = 1
> EVAL DS1=%INDEX(2)     ** DS1 のオカレンスの変更 **
  DS1=%INDEX(2) = 2
> EVAL Fld1              ** サブフィールドの表示   **
  FLD1 = 'ABCDE'            (現行オカレンス)
> EVAL fld1(2)
  FLD1(2) = 'ABCDE'         (2 番目のオカレンス)
> EVAL Fld1a             ** 配列オーバーレイ・サブフィールドの表示 **
  FLD1A OF DS1(1) = 'A'     (現行オカレンス)
  FLD1A OF DS1(2) = 'B'
  FLD1A OF DS1(3) = 'C'
  FLD1A OF DS1(4) = 'D'
  FLD1A OF DS1(5) = 'E'
> EVAL Fld1a(2,1)        ** 2 番目のオカレンスの表示、最初の要素   **
  FLD1A(2,1) = 'A'
> EVAL Fld1a(2,1..2)     ** 2 番目のオカレンスの表示、1 番目 ～ 2 番目の要素 **
  FLD1A(2,1) = 'A'
  FLD1A(2,2) = 'B'
#> EVAL QUALDS.ID_NUM              ** 修飾された DS のサブフィールドの表示
#     QUALDS.ID_NUM = 1100022
#> EVAL LIKE_QUALDS.ID_NUM         ** 別の DS の同じサブフィールドの表示
#     LIKE_QUALDS.ID_NUM = 0
#> EVAL LIKE_QUALDS.COUNTRY(1)     ** 修飾された DS からの配列要素
#     LIKE_QUALDS.COUNTRY(1) = 'CANADA'
#> EVAL cust(1).parts.item(2).Id_Num       ** 複雑な構造のサブフィールドを表示
#     CUST(1).PARTS.ITEM(2).ID_NUM = 15
```

図 117. データ構造に対する EVAL の使用

サブフィールドが定義されていないデータ構造を表示するためには、以下に説明する EVAL の文字表示機能を使用しなければなりません。

標識の表示

標識は 1 バイトの文字フィールドとして定義されます。*INLR のような標識を除いて、'*INxx' または '*IN(xx)'として標識を表示することができます。システムが標識を配列として保管するため、範囲表記法を使用して標識のすべてまたは標識の一部のサブセットを表示することができます。例えば、EVAL *IN を入力した場合には、01 から 99 までの標識のリストが取り出されます。標識 *IN01 から *IN06 までを表示するには、EVAL *IN(1..6) を入力します。

図 118 は、DBGEX で設定された標識を使用してこれらの各方法を示したものです。

```
> EVAL IN02
  識別コードが存在していない。
> EVAL *IN02
  *IN02 = '1'
> EVAL *IN(02)
  *IN(02) = '1'
> EVAL *INLR
  *INLR = '0'
> EVAL *IN(LR)
  識別コードが存在していない。
> EVAL *IN(1..6)        ** ある範囲の標識の表示 **
  *IN(1) = '0'
  *IN(2) = '1'
  *IN(3) = '0'
  *IN(4) = '1'
  *IN(5) = '0'
  *IN(6) = '1'
```

図 118. 配列に対するサンプルの EVAL コマンド

16 進値としてのフィールドの表示

EVAL デバッグ・コマンドを使用してフィールドの値を 16 進形式で表示することができます。16 進形式で変数を表示するためには、デバッグ・コマンド行に次を入力します。

```
EVAL field-name: x number-of-bytes
```

と入力します。変数 *field-name* は 16 進形式で表示したいフィールドの名前です。'x' は、フィールドを 16 進形式で表示することを指定します。変数 *number-of-bytes* は表示されるバイト数を指示します。'x' の後に長さが指定されていない場合には、フィールドのサイズが長さとして使用されます。最小 16 バイトが常に表示されます。フィールドの長さが 16 バイトより短い場合には、残りのスペースが 16 バイト境界に達するまで*ゼロで埋められ* ます。

例えば、フィールド String は 6 文字ストリングとして定義されています。最初の 3 桁の 16 進等価値を検索するには、次のように入力します。

```
EVAL String: x 3
   Result:
   00000     C1C2C3.. ........ ........ ........   - ABC.............
```

文字形式でのフィールドの表示

EVAL デバッグ・コマンドを使用してフィールドを文字形式で表示することができます。文字形式で変数を表示するためには、デバッグ・コマンド行に次を入力します。

```
EVAL field-name: c number-of-characters
```

と入力します。変数 *field-name* は文字形式で表示したいフィールドの名前です。'c' は、表示する文字数を指定します。

例えば、プログラム DEBUGEX で、データ構造 DS2 にはサブフィールドは定義されていません。いくつかの MOVE 命令が値をサブフィールドに移動します。

サブフィールドが定義されていないので、データ構造を表示することはできません。したがって、その内容を表示するために、EVAL の文字表示機能を使用することができます。

```
EVAL DS2:C 20              Result:   DS2:C 20 = 'aaaaaaaaaabbbbbbbbbb'
```

UCS-2 データの表示

UCS-2 フィールドについて表示される値は、読み取り可能な文字に変換されています。例えば、UCS-2 フィールドが %UCS2('abcde') に設定されている場合には、そのフィールドについて表示される値は 'abcde' です。EVAL にサフィックス :u を使用すると、任意のフィールドに UCS-2 データを表示することができます。

可変長フィールドの表示

可変長フィールドに EVAL fldname を使用すると、そのフィールドのデータ部分だけが示されます。フィールドについて :c または :x などのサフィックスを使用すると、長さを含むフィールド全体が示されます。可変長フィールドの現行の長さを判別するためには、EVAL fldname:x を使用します。この長さは、2 進数形式の、最初の 4 桁の 16 進数字です。長さを得るには、この値を 10 進数に変換する必要があります。例えば、結果が 003DF1F2... の場合、長さは 003D で、これは (3 * 16) + 13 = 61 となります。

ポインターがアドレス指定するデータの表示

ポインターが指す対象を表示したい場合には、:c または :x の接尾部を指定した EVAL コマンドを使用することができます。例えば、ポインター・フィールド PTR1 が 10 バイトの文字データを指す場合には、以下を指定することにより、この 10 バイトの内容が表示されます。

```
EVAL PTR1:c 10
```

この 10 バイトの内容が表示されます。

また、次を使用すると 16 進で内容を表示することもできます。

```
EVAL PTR1:x 10
```

ポインターがアドレス指定するデータが、パック・データや 2 進データなどの印刷可能形式で保管されていない場合には、これは特に便利です。

null 可能フィールドの表示

EVAL デバッグ・コマンドを使用して null 可能フィールドの空標識を表示することができます。空標識は、QRNU_NULL_ フィールド名という名前の内部変数 (複数回繰り返しデータ構造の指標変数とほぼ同じ) です。配列が null 可能である場合には、フィールド名を配列の名前にすることができます。

デバッガーが null 可能フィールドを表示すると、このフィールドが null と見なされているかどうかにかかわらず、フィールドの内容が表示されます。例えば、FLD1 が null 可能であり、現在 null であるとします。 EVAL _QRNU_NULL_FLD1 の結果は '1' であり、空標識がオンであったとしても、EVAL FLD1 は FLD1 の現在の内容を表示します。

```
EVAL _QRNU_NULL_FLD1     Result:  _QRNU_NULL_FLD1 = '1'
EVAL FLD1                Result:  FLD1 = 'abcde'
```

デバッグ組み込み関数の使用

ILE ソース・デバッガーを使用している時に、次の組み込み関数を使うことができます。

%SUBSTR
ストリング・フィールドのサブストリング

%ADDR
フィールドのアドレスの検索

%INDEX
テーブルまたは複数回繰り返しデータ構造の指標の変更

%VARS
指定されたパラメーターを変数として識別

%SUBSTR 組み込み関数によって、ストリング変数のサブストリングを作成することができます。最初のパラメーターはストリング識別コード、2 番目のパラメーターは開始桁、そして 3 番目のパラメーターは 1 バイトまたは 2 バイトの文字数でなければなりません。さらに、2 番目と 3 番目のパラメーターは正の整数リテラルでなければなりません。パラメーターは 1 つまたは複数のスペースで区切ります。

次のことを行うために %SUBSTR 組み込み関数を使用してください。

- 文字フィールドの一部を表示する
- 文字フィールドの一部を割り当てる
- 条件付きブレークポイント式のいずれかの側で文字フィールドの一部を使用する

263 ページの図 119 は、 268 ページの図 122 のソースに基づく %SUBSTR の使用例のいくつかを示します。

```
 > EVAL String
   STRING = 'ABCDE '
** ストリングの最初の 2 文字の表示 **
 > EVAL %substr (String 1 2)
   %SUBSTR (STRING 1 2) = 'AB'
 > EVAL TableA
   TABLEA = 'aaa'
** 最初のテーブル要素の最初の文字の表示 **
 > EVAL %substr(TableA 1 1)
   %SUBSTR(TABLEA 1 1) = 'a'
 > EVAL BigDate
   BIGDATE = '1994-10-23'
** String を BigDate の最初 4 桁と等しくなるように設定 **
 > EVAL String=%substr(BigDate 1 4)
   STRING=%SUBSTR(BIGDATE 1 4) = '1994  '
 > EVAL Fld1           (5 文字)
   FLD1 = 'ABCDE'
 > EVAL String               (6 文字)
   STRING = '123456'
** String の 2 ～ 5 桁を Fld1 の最初の 4 桁と等しく設定 **
 > EVAL %substr(String 2 4) = %substr(Fld1 1 4)
   %SUBSTR(STRING 2 4) = %SUBSTR(FLD1 1 4) = 'ABCD'
 > EVAL String
   STRING = '1ABCD6'
** %SUBSTR は文字ストリングまたはグラフィック・ストリングでのみ可能 ! **
 > EVAL %substr (Packed1D0 1 2)
   String type error occurred.
```

図 119. DBGEX を使用した %SUBSTR の例

現行索引を変更するために、%INDEX 組み込み関数を使用できますが、この場合指標は機能名の後の括弧内に指定します。%INDEX の例は、 258 ページの図 116 および 259 ページの図 117 のテーブルの項にあります。

注: %INDEX は現行の指標を指定された値に変更します。したがって、EVAL ステートメントの後でテーブルまたは複数回繰り返しデータ構造を参照するソースが予定していたものとは異なる指標で操作される可能性があります。

変数名がデバッグ・コマンド名のいずれかと競合する時は、%VARS デバッグ組み込み関数を使います。例えば、EVAL %VAR(EVAL) は EVAL という名前の変数を評価するのに使用できますが、EVAL EVAL は構文エラーになります。

フィールドの値の変更

割り当て演算子 (=) を指定した EVAL コマンドを使用してフィールドの値を変更することができます。

EVAL コマンドで使用されるフィールドの有効範囲は、QUAL コマンドを使用して定義します。しかし、ILE RPG モジュールに含まれるフィールドはすべてがグローバルな有効範囲なので、それらを特別に定義する必要はありません。

フィールドの値を変更するためには、デバッグ・コマンド行に

```
EVAL field-name = value
```

と入力します。*field-name* は変数の名前であり、 *value* は、変数 *field-name* に割り当てたい識別コード、リテラル、または固定情報値です。例えば

```
EVAL COUNTER=3
```

によって、*COUNTER* の値は 3 に変わり、また「モジュール・ソースの表示」画面のメッセージ行に

```
COUNTER=3 = 3
```

と表示されます。

EVAL デバッグ・コマンドを使用して、フィールドに数値、英字、および英数字データを割り当ててください。割り当て式で %SUBSTR 組み込み関数を使用することもできます。

文字フィールドに値を割り当てる時には、次の規則が適用されます。

- 割り当て元の式の長さが割り当て先の式の長さより短い場合には、割り当て先の式でデータが左寄せにされ、残りの桁にブランクが埋め込まれます。
- 割り当て元の式の長さが割り当て先の式の長さより長い場合には、割り当て先の式でデータが左寄せにされ、割り当て先の式の長さまでに切り捨てられます。

注: 図形フィールドには次のものを割り当てることができます。

- 別の図形フィールド
- G'oK1K2i ' の形式の図形リテラル
- X'16 進数' 形式の 16 進リテラル

UCS-2 フィールドの変更は、16 進定数を使用して行います。例えば、%UCS2('AB') = U'00410042' であるため、デバッガー内で UCS-2 フィールドを 'AB' という UCS-2 形式に設定するためには、EVAL ucs2 = X'00410042' を使用します。

可変長フィールドは、例えば、EVAL varfldname = 'abc' を使用して割り当てることができます。これにより、フィールドのデータ部分は 'abc' に、長さ部分は 3 に設定されます。データを変更せずに長さ部分を設定するためには、長さを 16 進数値 (例えば、11 は X'000B') で表し、 EVAL %SUBSTR(varfldname 1 2) = X'000B' を使用します。

フィールドにリテラルを割り当てる時には、以下の通常の RPG 規則が適用されます。

- 文字リテラルは引用符で囲まなければなりません。
- 図形リテラルは G'oDDDDi' として指定し、ここで o はシフトアウト、i はシフトインを表します。
- 16 進リテラルは引用符で囲み、前に 'x' がなければなりません。
- 数値リテラルは引用符で囲んではなりません。

注: EVAL デバッグ・コマンドを使用して形象定数をフィールドに割り当てることはできません。形象定数は EVAL デバッグ・コマンドではサポートされていません。

265 ページの図 120 は、 268 ページの図 122 のソースに基づいてフィールドの値を変更するいくつかの例を示します。ソース・デバッガーのオンライン・ヘルプには追加の例も提供されています。

```
** ターゲットの長さ = ソースの長さ **
 > EVAL String='123456'     (6 文字)
   STRING='123456' = '123456'
 > EVAL ExportFld           (6 文字)
   EXPORTFLD = 'export'
 > EVAL String=ExportFld
   STRING=EXPORTFLD = 'export'
** ターゲットの長さ < ソースの長さ **
 > EVAL String               (6 文字)
   STRING = 'ABCDEF'
 > EVAL LastName            (10 文字)
   LASTNAME='Williamson' = 'Williamson'
 > EVAL String=LastName
   STRING=LASTNAME = 'Willia'
** ターゲットの長さ > ソースの長さ **
 > EVAL String               (6 文字)
   STRING = '123456'
 > EVAL TableA               (3 文字)
   TABLEA = 'aaa'
 > EVAL String=TableA
   STRING=TABLEA = 'aaa   '
** %SUBSTR の使用 **
 > EVAL BigDate
   BIGDATE = '1994-10-23'
 > EVAL String=%SUBSTR(BigDate 1 4)
   STRING=%SUBSTR(BIGDATE 1 4) = '1994  '
** サブストリング・ターゲットの長さ > サブストリング・ソースの長さ **
 > EVAL string = '123456'
   STRING = '123456' = '123456'
 > EVAL LastName='Williamson'
   LASTNAME='Williamson' = 'Williamson'
 > EVAL String = %SUBSTR(Lastname 1 8)
   STRING = %SUBSTR(LASTNAME 1 8) = 'Willia'
** サブストリング・ターゲットの長さ < サブストリング・ソースの長さ **
 > EVAL TableA
   TABLEA = 'aaa'
 > EVAL String
   STRING = '123456'
 > EVAL String=%SUBSTR(TableA 1 4)
   サブストリングがストリングの終わりを超える。** エラー **
 > EVAL String
   STRING = '123456'
```

図 120. DBGEX に基づくフィールドの値の変更の例

フィールドの属性の表示

属性 (ATTR) デバッグ・コマンドを使用してフィールドの属性を表示することができます。属性は、デバッグ記号テーブルに記録された変数のサイズ (バイト数) およびタイプです。

266 ページの図 121 は、 268 ページの図 122 のソースに基づいてフィールドの属性を表示するいくつかの例を示します。ソース・デバッガーのオンライン・ヘルプには追加の例も提供されています。

```
> ATTR NullPtr
  TYPE = PTR, LENGTH = 16 BYTES
> ATTR ZonedD3D2
  TYPE = ZONED(3,2), LENGTH = 3 BYTES
> ATTR Bin4D3
  TYPE = BINARY, LENGTH = 2 BYTES
> ATTR Int3
  TYPE = INTEGER, LENGTH = 1 BYTES
> ATTR Int5
  TYPE = INTEGER, LENGTH = 2 BYTES
> ATTR Unsigned10
  TYPE = CARDINAL, LENGTH = 4 BYTES
> ATTR Unsigned20
  TYPE = CARDINAL, LENGTH = 8 BYTES
> ATTR Float4
  TYPE = REAL, LENGTH = 4 BYTES
> ATTR Float8
  TYPE = REAL, LENGTH = 8 BYTES
> ATTR Arry
  TYPE = ARRAY, LENGTH = 6 BYTES
> ATTR tablea
  TYPE = FIXED LENGTH STRING, LENGTH = 3 BYTES
> ATTR tablea(2)
  TYPE = FIXED LENGTH STRING, LENGTH = 3 BYTES
> ATTR BigDate
  TYPE = FIXED LENGTH STRING, LENGTH = 10 BYTES
> ATTR DS1
  TYPE = RECORD, LENGTH = 9 BYTES
> ATTR SpcPtr
  TYPE = PTR, LENGTH = 16 BYTES
> ATTR String
  TYPE = FIXED LENGTH STRING, LENGTH = 6 BYTES
> ATTR *IN02
  TYPE = CHAR, LENGTH = 1 BYTES
> ATTR DBCSString
  TYPE = FIXED LENGTH STRING, LENGTH = 6 BYTES
```

図 121. DBGEX に基づくフィールド属性の表示例

フィールド、式、またはコマンドと名前の等値化

EQUATE デバッグ・コマンドを使用して名前をフィールド、式またはデバッグ・コマンドと等値化することによって簡略名を使用できます。これにより、その名前を単独であるいは別の式の中で使用することができます。別の式で使用した場合には、名前の値は式が評価される前に判別されます。これらの名前はデバッグ・セッションが終了するか、名前が除去されるまで活動状態のままになります。

名前をフィールド、式、またはデバッグ・コマンドと等値化するためには、デバッグ・コマンド行に次を入力します。

```
EQUATE shorthand-name definition
```

と入力します。*shorthand-name* はフィールド、式、またはデバッグ・コマンドと等値化したい名前で、*definition* はその名前と等値化するフィールド、式、またはデバッグ・コマンドです。

例えば、*COUNTER* と呼ばれるフィールドの内容を表示する *DC* と呼ばれる簡略名を定義するためには、デバッグ・コマンド入力行に次を入力してください。

```
EQUATE DC EVAL COUNTER
```

と入力します。デバッグ・コマンド行に *DC* が入力されるたびに、コマンド EVAL *COUNTER* が実行されます。

EQUATE コマンドで入力できる最大桁数は 144 です。定義が指定されず、前の EQUATE コマンドが名前を定義している場合には、前の定義が除去されます。名前が前に定義されていなかった場合には、エラー・メッセージが表示されます。

デバッグ・セッションで EQUATE デバッグ・コマンドによって定義されている名前を表示するためには、デバッグ・コマンド行に次を入力してください。

```
DISPLAY EQUATE
```

と入力します。式の評価画面に活動状態の名前のリストが表示されます。

ILE RPG のソース・デバッグの各国語サポート

ILE RPG のソース・デバッグの各国言語サポートで処理している時に存在する次の状態に注意してください。

- 「モジュール・ソースの表示」画面にビューが表示されている時に、ソース・デバッガーはすべてのデータをデバッグ・ジョブのコード化文字セット識別子 (CCSID) に変換します。
- フィールドにリテラルを割り当てる時には、ソース・デバッガーは引用符付きリテラル (例えば、'abc') の CCSID 変換を行いません。また、引用符付きリテラルは大文字と小文字を区別されます。

NLS の制約事項について詳しくは、「*ILE 概念*」のデバッグの章を参照してください。

デバッグ用サンプル・ソースの例

268 ページの図 122 は、プログラム DEBUGEX のメイン・プロシージャーのソースを示します。この章に示されているほとんどの例および画面はこのソースに基づいています。 271 ページの図 123 および 272 ページの図 124 は、呼び出されるプログラム RPGPGM とプロシージャー cproc をそれぞれに示します。

プログラム DEBUGEX は ILE ソース・デバッガーと ILE RPG 定様式ダンプの異なる側面を示すように設計されています。サンプルのダンプは次の章にあります。

次のステップは、プログラム DEBUGEX がこれら例で使用されるためにどのように作成されたかを説明したものです。

1. 268 ページの図 122 のソースを使ってモジュール DBGEX を作るには、次の入力をします。

   ```
   CRTRPGMOD MODULE(MYLIB/DBGEX) SRCFILE(MYLIB/QRPGLESRC) DBGVIEW(*ALL)
             TEXT('Main module for Sample Debug Program')
   ```

   使用可能な異なるビューを表示するために、DBGVIEW(*ALL) が選択されました。

2. 272 ページの図 124 のソースを使用して C モジュールを作成するためには、次を入力します。

   ```
   CRTCMOD MODULE(MYLIB/cproc) SRCFILE(MYLIB/QCLESRC) DBGVIEW(*SOURCE)
           TEXT('C procedure for Sample Debug Program')
   ```

3. プログラム DEBUGEX を作成するためには、次を入力してください。

```
CRTPGM PGM(MYLIB/DEBUGEX) MODULE(MYLIB/DBGEX MYLIB/CPROC)
           TEXT('Sample Debug Program')
```

最初のモジュール DBGEX はこのプログラムの入力モジュールです。このプログラムは、呼び出された時に新しい活動化グループ (すなわち *NEW) で実行されます。

4. 271 ページの図 123 のソースを使用して呼び出し先 RPG プログラムを作成するためには、次を入力します。

```
CRTBNDRPG PGM(MYLIB/RPGPGM) DFTACTGRP(*NO)
          DBGVIEW(*SOURCE) ACTGRP(*NEW)
          TEXT('RPG program for Sample Debug Program')
```

作成した RPGPGM はデフォルトの OPM 活動化グループで実行することができます。しかし、これを DEBUGEX と同じ活動化グループで実行するよう決定し、また DEBUGEX は一時活動化グループしか必要としないので両方のプログラムに *NEW が選択されました。

```
     *===================================================================*
     *  DEBUGEX - ILE RPG ソースでの ILE ソース・デバッグ・
     *            プログラムの使用法を示すように設計されたプログラム。
     *            各種のデータ・タイプおよびデータ構造の例を示します。
     *
     *            また、定様式ダンプ例の作成にも使用できます。
     *===================================================================*
     *-------------------------------------------------------------------*
     * DEBUG キーワードは定様式ダンプ機能を可能にします。
     *-------------------------------------------------------------------*
    H DEBUG
     *-------------------------------------------------------------------*
     * 各種の ILE RPG データ・タイプの独立フィールドの定義
     *-------------------------------------------------------------------*
    D String          S              6A   INZ('ABCDEF')
    D Packed1D0       S              5P 2 INZ(-93.4)
    D ZonedD3D2       S              3S 2 INZ(-3.21)
    D Bin4D3          S              4B 3 INZ(-4.321)
    D Bin9D7          S              9B 7 INZ(98.7654321)
    D DBCSString      S              3G   INZ(G'"BBCCDD"')
    D UCS2String      S              5C   INZ(%UCS2('ucs-2'))
    D CharVarying     S              5A   INZ('abc') VARYING
    D Int3            S              3I 0 INZ(-128)
    D Int5            S              5I 0 INZ(-2046)
    D Int10           S             10I 0 INZ(-31904)
    D Int20           S             20I 0 INZ(-463972)
    D Unsigned3       S              3U 0 INZ(128)
    D Unsigned5       S              5U 0 INZ(2046)
    D Unsigned10      S             10U 0 INZ(31904)
    D Unsigned20      S             20U 0 INZ(463972)
    D Float4          S              4f   INZ(7.2098)
    D Float8          S              8f   INZ(-129.0978652)
    D DBCSString      S              3G   INZ(G'"BBCCDD"')
```

図 122. モジュール DBGEX のソース (1/4): DBGEX はプログラム DEBUGEX のメイン・モジュールです。

```
 *   ポインター
D NullPtr         S               *   INZ(*NULL)
D BasePtr         S               *   INZ(%ADDR(String))
D ProcPtr         S               *   ProcPtr INZ(%PADDR('c_proc'))
D BaseString      S              6A   BASED(BasePtr)
D BaseOnNull      S             10A   BASED(NullPtr)
 *
D Spcptr          S               *
D SpcSiz          C                   8
 *   日付、時刻、タイム・スタンプ
D BigDate         S               D   INZ(D'9999-12-31')
D BigTime         S               T   INZ(T'12.00.00')
D BigTstamp       S               Z   INZ(Z'9999-12-31-12.00.00.000000')
 *   Array
D Arry            S              3S 2 DIM(2) INZ(1.23)
 *   Table
D TableA          S              3    DIM(3) CTDATA
 *---------------------------------------------------------------*
 * 各種のデータ構造の定義
 *---------------------------------------------------------------*
D DS1             DS                  OCCURS(3)
D  Fld1                           5A   INZ('ABCDE')
D  Fld1a                          1A   DIM(5) OVERLAY(Fld1)
D  Fld2                           5B 2 INZ(123.45)
 *
D DS2             DS             10    OCCURS(2)
 *
D DS3             DS
D  Title                          5A   INZ('Mr.  ')
D  LastName                      10A   INZ('Jones    ')
D  FirstName                     10A   INZ('Fred     ')

D QUALDS          DS                  QUALIFIED
D  Id_Num                         8S 0
D  Country                       20A   DIM(10)
D LIKE_QUALDS     DS                  LIKEDS(QUALDS)
D itemInfo        DS                  QUALIFIED
D   ID_Num                       10I 0
D   name                         25A
D items           DS                  QUALIFIED
D   numItems                     10I 0
D   item                              LIKEDS(itemInfo) DIM(10)
D cust            DS                  QUALIFIED DIM(10)
D   name                         50A
D   parts                             LIKEDS(items)
 *---------------------------------------------------------------*
 * 呼び出されるプロシージャー c_proc および switch のプロトタイプの定義
 *---------------------------------------------------------------*
D c_proc          PR              *   EXTPROC('c_proc')
D   size                        10U 0 VALUE
D   inzval                        1A   CONST
D Switch          PR
D   Parm                          1A
 *---------------------------------------------------------------*
 * 非プロトタイプ呼び出しの場合のパラメーターの定義。
 *     PARM1 は RPGPROG プログラムの呼び出し時に使用されます。
 *---------------------------------------------------------------*
D PARM1           S               4P 3 INZ(6.666)
D EXPORTFLD       S               6A   INZ('export') EXPORT
```

図 122. モジュール DBGEX のソース (2/4): DBGEX はプログラム DEBUGEX のメイン・モジュールです。

```
 *===================================================================*
 * 値の変更または他のオブジェクトの呼び出しのための命令。
 *===================================================================*
 *-------------------------------------------------------------------*
 * 'a' をデータ構造 DS2 に転送します。転送後、
 * DS2 の最初のオカレンスには 10 桁の 'a' が入っています。
 *-------------------------------------------------------------------*
C                   MOVE      *ALL'a'       DS2
 *-------------------------------------------------------------------*
 * DS2 のオカレンスを 2 に変更して、'b' を DS2 に転送すると、
 * 最初の 10 バイトは 'a' で 2 番目の 10 バイトが 'b' になります。
 *-------------------------------------------------------------------*
C     2             OCCUR     DS2
C                   MOVE      *ALL'b'       DS2
 *-------------------------------------------------------------------*
 * Fld1a は Fld1 のオーバーレイ・フィールドです。Fld1 は
 * 'ABCDE' に初期化されているので、Fld1a(1) の値は 'A' です。
 * 次の MOVE 命令の後の Fld1a(1) の値は '1' になります。
 *-------------------------------------------------------------------*
C                   MOVE      '1'           Fld1a(1)
 *-------------------------------------------------------------------*
 * 別のプログラム・オブジェクトであるプログラム RPGPGM を呼び出します。
 *-------------------------------------------------------------------*
C     Plist1        PLIST
C                   PARM                    Parm1
C                   CALL      'RPGPGM'      Plist1
 *-------------------------------------------------------------------*
 * メイン・プロシージャーから ExportFld をインポートする
 * c_proc を呼び出します。
 *-------------------------------------------------------------------*
C                   EVAL      SpcPtr = c_proc(SpcSiz : 'P')
 *-------------------------------------------------------------------*
 * 標識の値を反転する、ローカル・サブプロシージャー Switch を
 * 呼び出します。
 *-------------------------------------------------------------------*
C                   EVAL      *IN10 = '0'
C                   CALLP     Switch(*in10)
```

図 122. モジュール DBGEX のソース (3/4)： DBGEX はプログラム DEBUGEX のメイン・モジュールです。

```
     *-----------------------------------------------------------------*
     * 次の SETON 命令の後、*IN02 = 1 になります。
     *-----------------------------------------------------------------*
    C                   SETON                                      020406
    C                   IF        *IN02 = '1'
    C                   MOVE      '1994-09-30'  BigDate
    C                   ENDIF
     *-----------------------------------------------------------------*
     * Arry の 2 番目のセルに新しい値を入れます。
     *-----------------------------------------------------------------*
    C                   MOVE      4             Arry
     *-----------------------------------------------------------------*
     *  ここで定様式ダンプを開始し、LR をオンに設定して戻ります。
     *-----------------------------------------------------------------*
    C                   DUMP
    C                   SETON                                        LR
     *=================================================================*
     * サブプロシージャー Switch を定義します。
     *=================================================================*
    P Switch          B
    D Switch          PI
    D   Parm                        1A
     *-----------------------------------------------------------------*
     * デバッグ目的のローカル変数を定義します。
     *-----------------------------------------------------------------*
    D Local           S              5A      INZ('aaaaa')
    C                   IF        Parm = '1'
    C                   EVAL      Parm = '0'
    C                   ELSE
    C                   EVAL      Parm = '1'
    C                   ENDIF
    P Switch          E
     *=================================================================*
     * テーブルのコンパイル時データ・セクション
     *=================================================================*
**
aaa
bbb
ccc
```

図 122. モジュール DBGEX のソース (4/4): DBGEX はプログラム DEBUGEX のメイン・モジュールです。

```
     *=================================================================*
     *  RPGPGM - ILE ソース・デバッガーの STEP 機能を説明するための、   *
     *           DEBUGEX によって呼び出されるプログラム。              *
     *                                                                 *
     *  このプログラムは DEBUGEX からパラメーター InputParm を         *
     *  受け取り、表示してから戻ります。                               *
     *=================================================================*
    D InputParm       S              4P 3
    C     *ENTRY        PLIST
    C                   PARM                    InputParm
    C     InputParm     DSPLY
    C                   SETON                                        LR
```

図 123. OPM プログラム RPGPGM のソース

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
extern char EXPORTFLD[6];
char *c_proc(unsigned int size, char *inzval)
{
   char *ptr;
   ptr = malloc(size);
   memset(ptr, *inzval, size);
   printf("import string: %6s.¥n",EXPORTFLD);
   return(ptr);
}
```

図 124. C プロシージャー cproc のソース： cproc は DBGEX によって呼び出されます。

第 13 章 例外の処理

この章では、ILE RPG 例外処理機能の働きと以下の使用法について説明します。

- 例外処理プログラム
- ILE RPG 特有の処理プログラム
- ILE 条件処理プログラム
- 取り消し処理プログラム

ILE RPG は、次のタイプの例外処理プログラムをサポートします。

- RPG 特有の処理プログラム、例えばエラー標識、'E' 命令コード拡張の使用、MONITOR グループ、あるいは *PSSR または INFSR エラー処理サブルーチンの使用など。
- ILE 条件処理プログラム、ILE 条件処理プログラムとのバインドが可能な API CEEHDLR を使用して実行時に登録するユーザー作成例外処理プログラム。
- プロシージャーが異常終了した時に使用できる ILE 取り消し処理プログラム。

ある種の計画した例外処理を利用すると、不要な異常終了 (すなわち機能チェックに関連するもの) の数を最小にすることができるので、ほとんどのプログラムにとって役立ちます。ILE 条件処理プログラムによっても、混合言語のアプリケーション中の例外を一貫性のある方法で処理することができます。

RPG 例外処理プログラムを使用して、RPG アプリケーションで生ずる可能性のあるほとんどの状況を処理することができます。RPG が提供する最低レベルの例外処理は、ある種の操作でエラー標識を使用することです。エラー標識の使用法を学習するためには、この章の次の項を参照してください。

- 276 ページの『ILE RPG 例外処理』
- 283 ページの『エラー標識または 'E' 命令コード拡張の指定』
- 287 ページの『ファイル・エラー処理 (INFSR) サブルーチンの使用』
- 284 ページの『MONITOR グループの使用』
- 291 ページの『プログラム・エラー処理サブルーチンの使用』

さらに ILE 例外処理の機能方法については、次を参照してください。

- 274 ページの『例外処理の概要』(一般概念について)
- 283 ページの『RPG 特有の処理プログラムの使用』
- 「*ILE 概念*」のエラー処理に関するセクション

例外処理および RPG サイクルについての情報は、「*WebSphere Development Studio: ILE RPG 解説書*」を参照してください。

注: 本書では、「例外処理」という用語は、例外処理とエラー処理の両方を指すために用いられています。ただし、「エラー」という用語は、RPG 用語の「エラー標識」および「エラー処理サブルーチン」と同様のコンテキストで用いられています。

例外処理の概要

例外処理とは次のような処理のことです。

- 実行時エラーの結果として出された例外メッセージを調べる
- 例外が受け取られた (すなわち処理された) ことを示すために例外を任意に修正する
- 例外情報をコードの一部に渡して必要な処理を取ることによって任意に例外から回復する

実行時エラーが起こった時には、例外メッセージが生成されます。例外メッセージは起こったエラーのタイプによって、次のいずれかのタイプになります。

***ESCAPE** 重大エラーが検出されたことを示します。

***STATUS** プログラムによって行われている処理の状況を説明します。

***NOTIFY** 呼び出し側プログラムからの訂正処置または応答を必要とする状態を説明します。

機能チェック 前記 3 つの状態の 1 つが起こったが、処理されていないことを示します。

例外メッセージは呼び出しスタック項目と関連しています。さらに各呼び出しスタック項目はその項目に定義されている例外処理プログラムのリストと関連しています (呼び出しスタックについて詳しくは、 141 ページの『呼び出しスタック』 を参照してください)。

275 ページの図 125 は、OPM プログラムがいくつかのモジュール、したがっていくつかのプロシージャーから構成される ILE プログラムを呼び出す、呼び出しスタックを示します。後に続く説明では、この図を参照してください。

一般に、例外が起こった時には、呼び出しスタック項目に関連した処理プログラムが例外を処理できるようになります。例外がリスト上のどの処理プログラムによっても処理されなかった場合には、例外は未処理と見なされ、その点で、未処理例外について次のようなデフォルトの処置が取られます。

1. 例外が機能チェックである場合には、その呼び出しスタック項目がスタックから取り除かれます。
2. 例外が前の呼び出しスタック項目に移されます (パーコレートされます)。
3. この呼び出しスタック項目で例外処理が再開されます。

前の呼び出しスタック項目で例外を処理できるようにする処置を**パーコレーション**といいます。パーコレーションは、例外が処理されるか、あるいは制御境界に達するまで続けられます。**制御境界**は、直前の呼び出しスタック項目が別の活動化グループであるか、**あるいは** OPM プログラムである呼び出しスタック項目です。 275 ページの図 125 では、プロシージャー P1 が制御境界です。

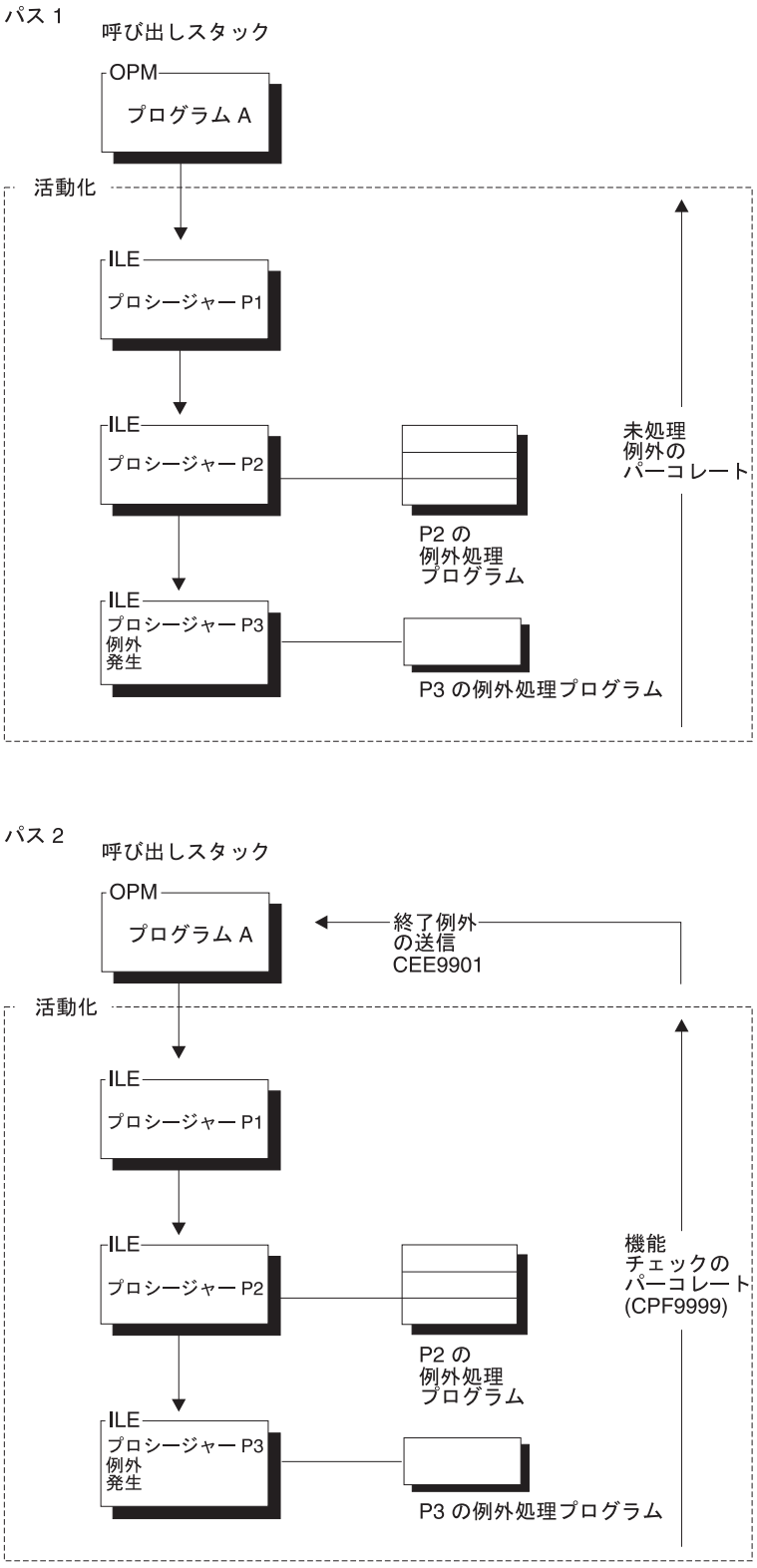


図 125. 呼び出しスタックおよび例外メッセージのパーコレーション

OPM では、例外メッセージは呼び出しスタック上で活動状態の*プログラム* に関連しています。例外が関連の例外処理プログラムによって処理されなかった場合には、例外を受け取った同じ呼び出しスタック項目に機能チェックが送られます。例

外が処理不能のままである場合には、項目が除去され、機能チェックがパーコレートされます。例外が処理されるまで、この処理が反復されます。

ILE では、例外メッセージは、呼び出しスタックで活動状態の*プロシージャー* と関連します。例外がパーコレートされる時には、例外は機能チェックに変換されません。各呼び出しスタック項目は、制御境界に達するまで元の例外を処理できるようになります。制御境界に達して初めて、例外は機能チェックに変換され、その時点から例外を受け取ったプロシージャーから例外処理がやり直されます。この時、各呼び出しスタック項目が機能チェックを処理できるようになります。制御境界に達し、例外がまだ未処理である場合には、一般障害例外メッセージ CEE9901 が制御境界でプロシージャーの呼び出し元に送られます。さらに、メッセージを処理しなかった呼び出しスタック項目はすべて除去されます。

ILE RPG 例外処理

ILE RPG は、次の 4 つの例外処理メカニズムを提供します。

- エラー標識、または 'E' 命令コード拡張処理プログラム
- MONITOR グループ
- エラー処理サブルーチン処理プログラム
- デフォルトの例外処理プログラム

RPG では例外はプログラムとファイルの 2 つに分類されています。これによって呼び出されるエラー処理サブルーチンのタイプが決定されます。プログラム例外の例として、ゼロによる除算、範囲外配列指標、負数の SQRT があります。ファイル例外の例としては未定義のレコード・タイプや装置エラーなどがあります。

RPG が例外を処理するように指示する方法は 5 つあります。それらは次のとおりです。

1. 該当する命令コードの演算仕様書の 73 ～ 74 桁目にエラー標識を指定する。
2. 適切な命令コードに命令コード拡張 'E' を指定する。
3. 例外を生成するコードを MONITOR グループの中に組み込む。
4. ファイル例外について、ファイル仕様書の INFSR キーワードによって定義されるファイル・エラー処理サブルーチンをコーディングする。ファイル・エラー処理サブルーチンは、メイン・ソース・セクションでしかコーディングすることはできません。サブプロシージャーで使用されたファイルには INFSR をコーディングすることはできません。
5. プログラム例外について、*PSSR という名前のプログラム・エラー処理サブルーチンをコーディングする。*PSSR は、それがコーディングされているプロシージャーに対して固有であることに注意してください。これはメイン・プロシージャーの *PSSR が、メイン・プロシージャーに関連したプログラム・エラーだけを処理することを意味しています。同様に、サブプロシージャーの *PSSR はサブプロシージャーのエラーだけを処理します。

メイン・プロシージャー内の例外処理プログラム

例外がメイン・プロシージャーで起こると ILE RPG は次のことを行います。

1. 演算仕様書にエラー標識が指定されていて、例外がその命令で予期されるものである場合:

a. 標識をオンに設定する。
b. 例外を処理する。
c. 制御は次の ILE RPG 命令で再開する。

2. 'E' 命令コード拡張が演算仕様書上にあり、この例外がその命令で予期される場合。
   a. 組み込み関数 %STATUS および %ERROR の戻り値が設定される。

      注: 'E' 拡張が指定されていなくても何か例外が発生すると、%STATUS は設定されます。

   b. 例外を処理する。
   c. 制御は次の ILE RPG 命令で再開する。
3. エラー標識も 'E' 拡張もなく、例外を生成するコードが MONITOR グループの MONITOR ブロックにある場合は、制御は MONITOR グループ内の ON-ERROR セクションに渡されます。
4. エラー標識または 'E' 拡張がなく、例外を処理できる活動状態の MONITOR グループがなく、かつ
   - *PSSR エラー処理サブルーチンをコーディングしていて、例外がプログラム例外である場合、

     あるいは、
   - ファイルの INFSR エラー処理サブルーチンをコーディングしていて、例外が入出力例外である場合、

   このときは、その例外が処理され、エラー処理サブルーチンの最初のステートメントで制御が再開されます。
5. エラー標識、'E' 拡張、またはエラー処理サブルーチンがコーディングされておらず、かつ例外を処理できる活動状態の MONITOR グループがない場合は、RPG のデフォルトのエラー処理プログラムが呼び出されます。
   - 例外が機能チェックでない 場合には、例外はパーコレートされます。
   - 例外が機能チェックである場合には、照会メッセージが表示されます。'G' または 'R' オプションを選択した場合には、機能チェックが処理され、プロシージャーの適切な時点 ('G' の場合には *GETIN あるいは 'R' の場合には例外を受け取った同じ演算仕様書) で制御が再開されます。そうでない場合には、機能チェックがパーコレートされ、プロシージャーが異常終了します。

RPG のデフォルト処理プログラムの詳細については、 280 ページの『未処理例外』を参照してください。

サブプロシージャー内の例外処理プログラム

サブプロシージャー内の例外処理プログラムはメイン・プロシージャーとは異なり、次の方法があります。

- ユーザーは INFSR サブルーチンのコーディングを行なえないため、ファイル・エラーの処理にはエラー標識、'E' 命令コード拡張、または MONITOR グループを使わなければなりません。
- デフォルトの処理プログラムはありません。言い換えるとユーザーは、照会メッセージを見ることはありません。

サブプロシージャー内の例外処理は、サブプロシージャー用に生成された RPG サイクル・コードがないために、基本的にメイン・プロシージャーと異なっています。結果として、サブプロシージャー用のデフォルトの例外処理プログラムがないので、対応するメイン・プロシージャーのデフォルトの処理プログラムが呼び出された場合には、サブプロシージャーの異常終了となります。これは次のことを意味します。

- サブプロシージャー内の *PSSR サブルーチンの ENDSR 命令の演算項目 2 はブランクでなければなりません。メイン・プロシージャー内のブランクの演算項目 2 は、結果としてデフォルトの処理プログラムに制御が渡されることになります。サブプロシージャーでは、ENDSR に達した場合には、サブプロシージャーが異常終了し、サブプロシージャーの呼び出し元に RNX9001 が通知されます。
- *PSSR がなく、機能チェックが起こった場合には、プロシージャーが呼び出しスタックから除去され、例外が呼び出し元にパーコレートされます。
- サブプロシージャー内のエラーについての照会メッセージは出されないので、一部の入出力エラーに使用可能な '再試行' 機能へはアクセスできません。サブプロシージャー内でレコード・ロック・エラーが予想される場合にはエラー標識、または 'E' 拡張をコーディングして、状況がロックされているレコードに関連しているかどうかを検査する必要があります。

PSDS および INFDS がモジュールの有効範囲をもっていることに注意してください。メイン・プロシージャーおよびサブプロシージャーともにこれらをアクセスすることができます。

ヒント

*PSSR は、それがコーディングされているプロシージャーに対してローカルであるので、共通のエラー処理ルーチンとするためには、エラーを処理するためのプロシージャーをコーディングし、各ローカル *PSSR からそのプロシージャーを呼び出すことができます。

OPM と ILE RPG 例外処理との違い

ほとんどの部分で OPM RPG と ILE RPG では、例外処理は同じように作動します。重要な相違は処理不能例外の部分にあります。

OPM では、例外が起こり、RPG 特有の処理プログラムが活動状態でない場合には、照会メッセージが出されます。ILE では、これは、例外が機能チェックである場合にのみ起こります。機能チェックでない場合には、例外はプロシージャーまたはプログラムの呼び出し元に渡され、適格なより高位の呼び出しスタック項目が例外を処理できるようになります。例えば、次の例を考えてください。

- PGM A が PGM B を呼び出し、次に PGM B が PGM C を呼び出します。
- PGM B では呼び出しに対するエラー標識がコーディングされています。
- PGM C にはエラー標識も *PSSR エラー処理サブルーチンもコーディングされていません。
- PGM C が例外を受け取ります。

OPM では照会メッセージは PGM C に対して出されます。ILE では、例外は PGM C によって処理されないため PGM B にパーコレートされます。PGM B のエラー標識はオンになって PGM B がエラーを処理できるようになり、その過程で PGM C が異常終了します。照会メッセージはありません。

PGM C に *PSSR エラー処理サブルーチンがコーディングされている場合には、OPM でも ILE でも、例外は PGM C によって処理され、エラー処理サブルーチンが実行されます。

注: ILE RPG によって出される照会メッセージは OPM RPG と同様、'RPG' ではなく 'RNQ' の接頭部で始まります。

特定のエラーの場合には一定の処理上の相違が存在します。詳しくは、447 ページの『付録 A. OPM RPG/400 と AS/400 用 ILE RPG との動作上の相違点』を参照してください。

例外処理プログラムの使用

アプリケーションの例外処理機能を計画するということは、次のような決定を下すことです。

1. RPG 特有のエラー処理の手段 (例えばエラー標識、'E' 拡張、エラー処理サブルーチンなど) を使うか、ILE API CEEHDLR を使って登録する、別の例外処理ルーチンを作るかの決定。両方を使用するように選択することもできます。
2. 回復処置、すなわち別個の例外処理ルーチンを使用する場合にプログラムが処理を再開する時点についての決定。

さらに、例外処理プログラムを計画する時には次のことにも留意してください。

- 例外処理プログラムの優先順位
- ネストされた例外
- 未処理例外に対するデフォルトの処置
- 最適化レベルの影響

例外処理プログラムの優先順位

例外処理プログラムの優先順位は、言語特有のエラー処理と ILE 条件処理プログラムの両方を使うとき、重要になります。ILE RPG プロシージャーの場合、例外処理プログラムの優先順位は次のようになります。

1. エラー標識、または 'E' 拡張処理プログラム
2. MONITOR グループ
3. ILE 条件処理プログラム
4. 入出力エラー処理サブルーチン・ハンドラー (ファイル・エラーの場合) およびプログラム・エラー処理サブルーチン・ハンドラー (その他すべてのエラーの場合)
5. RPG の未処理例外用のデフォルトの処理プログラム (メイン・プロシージャーのみ)

ネストされた例外

例外はネストされることがあります。ネストされた例外とは、別の例外の処理中に起こる例外のことです。これが起こった時には、最初の例外の処理が一時的に中断されます。例外処理は、最新に生成された例外から再開されます。

未処理例外

未処理例外とは、最初に例外を受け取った呼び出しスタック項目に関連した例外処理プログラムによって処理されなかった例外のことです。例外が未処理である時には、次のいずれかの処置が取られます。

メッセージ・タイプがメイン・プロシージャーに*関連した* **機能チェック** (CPF9999) である場合には、 RPG のデフォルトの処理プログラムは元の状態を説明する照会メッセージを出します。

- D (ダンプ) または C (取り消し) オプションを選択した場合には、最初に例外を受け取ったプロシージャーが終了し、機能チェックが呼び出し元にパーコレートされます。
- R (再試行) または G (入力取得) オプションを選択した場合には、機能チェックが処理され、例外処理が終了し、プロシージャーが *GETIN (G を選択した場合) あるいは例外が起こった入出力操作 (R を選択した場合) で処理を再開します。例えば、レコードのロックのために読み取りが正常に行われなかった場合には、読み取り操作が再試行されます。

他のタイプのメッセージの場合には、例外はプロシージャーの呼び出し元に対する呼び出しスタックにパーコレートされます。そのプロシージャーは例外を提示され、それを処理できるようになります。そのプロシージャーが例外を処理しない場合には、制御境界に達するまで、例外が呼び出しスタックにパーコレートされ、制御境界に達すると、例外は機能チェックに変換され、上記に説明したように例外処理がやり直されます。

未処理エスケープ・メッセージの例

次のシナリオで、エスケープ・メッセージが出され、それが起こったプロシージャーで処理できなかった時に起こるイベントについて説明します。このシナリオでは次のことが想定されています。

1. PGM1 と PGM2 という 2 つのプログラムがあり、これらは同じ活動化グループで実行されます。それぞれには PRC1 と PRC2 という対応するプロシージャーが含まれています。
2. PRC1 は PGM2 を動的に呼び出し、PRC2 が制御を受け取ります。
3. PRC1 の CALL 命令コードには呼び出しに対するエラー標識があります。
4. PRC2 には RPG 例外処理プログラムがコーディングされていません。すなわち、SUBST 命令に対するエラー標識がコーディングされておらず、また *PSSR エラー処理サブルーチンもありません。
5. PRC2 には演算項目 1 が負数である SUBST 命令があります。

PGM1 が PGM2 を呼び出す時に、SUBST 命令が試みられると、例外メッセージ RNX0100 が生成されます。 281 ページの図 126 は、このシナリオと発生するイベントを説明しています。

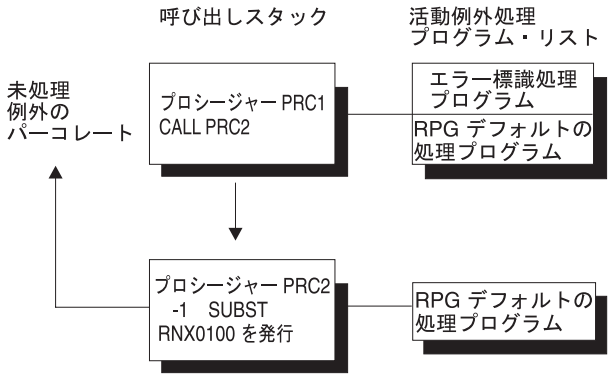


図 126. 処理不能エスケープ・メッセージのシナリオ

ここで次のことが起こります。

1. PRC2 の SUBST 命令ではエラー標識も活動状態の MONITOR グループもなく、*PSSR エラー処理サブルーチンもコーディングされていないために、PRC2 はプログラム・エラーを処理することができず、エラーは処理不能となります。
2. エラーは機能チェックでないので、PRC1 にパーコレートされます (呼び出しスタックを押し上げます)。
3. PRC1 は同じ例外メッセージを受け取り (処理し)、CALL 命令のエラー標識をオンに設定しますが、その影響で PRC2 が終了します。
4. ここで処理は、PRC1 の CALL ステートメントの後のステートメントから続行されます。

注: 説明した同じ例外処理イベントは、プロシージャー呼び出し (CALLB 命令) にも適用します。

未処理機能チェックの例

次のシナリオでは、メイン・プロシージャーで機能チェックが起こり、処理されなかった時に起こるイベントについて説明します。このシナリオでは次のことが想定されています。

1. PGM1 と PGM2 という 2 つのプログラムがあり、それぞれ対応する PRC1 と PRC2 というプロシージャーを含んでいます。
2. PRC1 は PGM2 を動的に呼び出し、PRC2 が制御を受け取ります。
3. PRC1 の CALL 命令コードにはエラー標識がコーディングされていません。
4. PRC2 には RPG 例外処理プログラムがコーディングされていません。すなわち、エラー標識がコーディングされておらず、活動状態の MONITOR グループもなく、また *PSSR エラー処理サブルーチンもありません。
5. PRC2 にはポインター・アドレス・エラーがあります。

PGM1 が PGM2 を呼び出すと、基底ポインターが null として定義されているので、ポインター・エラーが起こります。したがって、MCH1306 が生成されます。PRC2 が制御境界を超えて例外をパーコレートしようとすると、機能チェックが起こります。 282 ページの図 127 は、このシナリオと発生するイベントを説明しています。

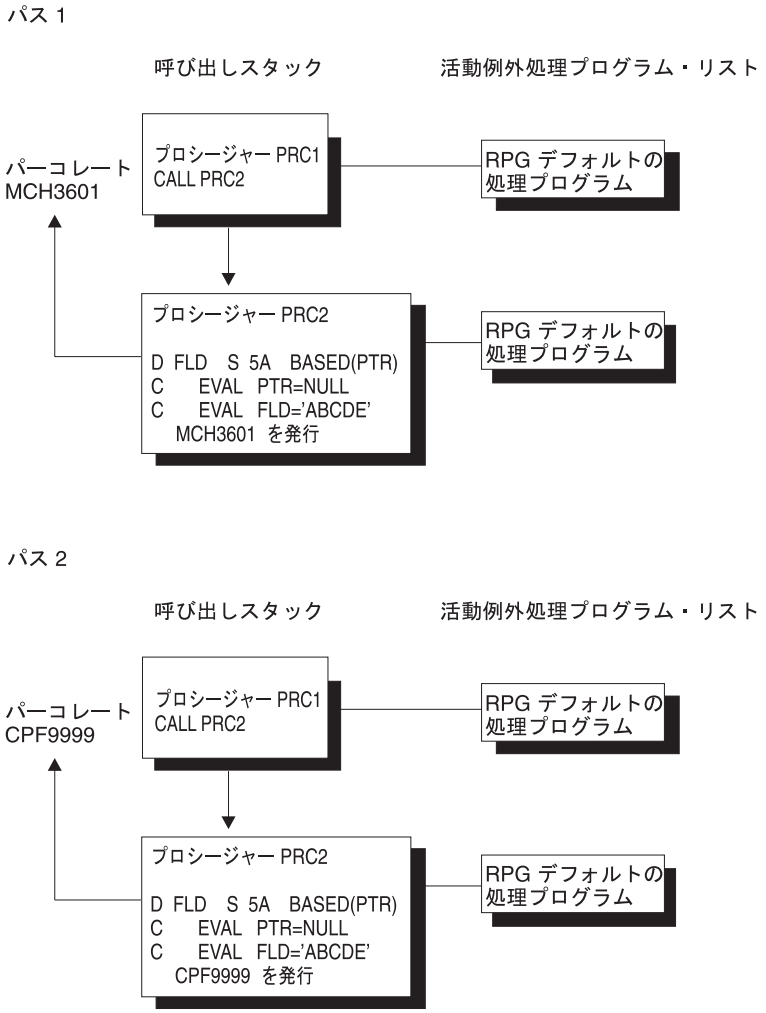


図 127. 処理不能の機能チェックのシナリオ

ここで次のことが起こります。

1. PRC2 にはエラー処理プログラムがないので、PRC2 は機能チェックを処理することができず、例外は処理不能となります。
2. 機能チェックであるために、元の状態を説明する照会メッセージが出されます。
3. 照会メッセージに対する応答によっては、PRC2 が終了し、例外が PRC1 にパーコレートされるか (応答が 'C' の場合)、あるいは処理が PRC2 で続行されることがあります (応答が 'G' の場合)。

最適化に関する考慮事項

*FULL で最適化されたプログラムの実行中に、最適化プログラムは、頻繁に使用される値をマシン・レジスターに保持し、通常のプログラム処理時には、事前定義点でのみ記憶域に復元します。例外処理でこの通常処理が中断され、結果的にレジスターに入っているプログラム変数がその割り当て記憶域位置に戻されないことがあります。

特に、例外が起こった場合には、変数に現在の値が入っていないことがあり、次の 1 つを使用して回復しなければなりません。

- MONITOR グループ
- *PSSR エラー処理サブルーチン
- INFSR エラー処理サブルーチン
- ユーザー定義の例外処理プログラム
- 照会メッセージの実行 ('G') オプション
- 照会メッセージの再試行 ('R') オプション

ILE RPG は、全最適化でも標識に現在の値が入るように標識を自動的に定義します。フィールドまたはデータ構造の内容が正しい (現在の) 値であるようにするためには、適切な定義仕様書に NOOPT キーワードを指定してください。

NOOPT キーワードについて詳しくは、「*WebSphere Development Studio: ILE RPG 解説書*」を参照してください。最適化の詳細については、98 ページの『最適化レベルの変更』を参照してください。

RPG 特有の処理プログラムの使用

ILE RPG は、HLL 固有の処理プログラムを使用可能にして例外から回復することのできる 4 つの方法を提供します。

1. エラー標識または 'E' 命令コード拡張
2. MONITOR グループ
3. INFSR エラー処理サブルーチン
4. *PSSR エラー処理サブルーチン

適切なデータ構造をコーディングし、対応するデータ構造フィールドを照会することによって、起こったエラーについての詳しい情報を入手することができます。

エラー標識の代わりに 'E' 拡張を使っている場合は、関連プログラムおよびファイル・エラー情報は %STATUS および %ERROR 組み込み関数を使って取り出すことができます。

この項には、これらの RPG 構造のそれぞれを使用する方法の例をいくつか示してあります。*PSSR と INFSR エラー処理サブルーチン、EXSR 命令コード、および INFDS と PSDS データ構造について詳しくは、「*WebSphere Development Studio: ILE RPG 解説書*」に説明があります。

エラー標識または 'E' 命令コード拡張の指定

エラー標識が使えるコードは、'E' 命令コード拡張も使えます。CALLP 命令の場合、エラー標識が使えなくても 'E' 拡張が使えます。このことは根本的には同じ、2 つの ILE RPG エラー処理方式を提供することになります。エラー標識または 'E' 拡張のいずれか (両方ではない) を使って、例外を処理することができます。

注: エラー標識や 'E' 拡張が、ある命令にコーディングされていても、起こったエラーがその命令に関係ない場合 (例えば、CHAIN 命令の配列指標エラー)、どのエラー標識や 'E' 拡張も無視されます。エラーは、他のプログラム・エラーと同様に取り扱われます。

RPG エラー標識処理プログラムを使用可能にするには、表 30 にリストされた命令コード (CALLP を除く) の 73 ～ 74 桁目にエラー標識を指定します。その命令で例外が起こった場合には標識がオンに設定され、適切なデータ構造 (PSDS または INFDS) が更新され、制御が次の順番の命令に戻ります。ここで標識をテストして、取る処置を決定することができます。

'E' 命令コード拡張処理プログラムを使用可能にするには、表 30 の命令コードのいずれかに 'E' (または 'e') を指定します。'E' 拡張をコーディングすると、例外に対する組み込み関数、%ERROR および %STATUS によって戻される値に影響します。その操作が始まる前に、これらの組み込み関数によって戻される値はゼロに設定されます。その操作で例外が起こった場合には、これらの組み込み関数の戻り値はそれによって更新され、適切なデータ構造 (PSDS または INFDS) が更新され、制御が次の順番の命令に戻ります。次に、これらの組み込み関数を使って戻り値をテストし、取るべき処置を決定することができます。

表 30. 拡張 'E' またはエラー標識を 73 ～ 74 桁目で使用可能にする命令コード

ACQ (e)	ADDDUR (e)	ALLOC (e)	CALL (e)
CALLB(d e)	CALLP (e m/r)[1]	CHAIN (e n)	CHECK (e)
CHECKR (e)	CLOSE (e)	COMMIT (e)	DEALLOC(e/n)
DELETE (e)	DSPLY (e)	EXFMT (e)	EXTRCT (e)
FEOD (e)	IN (e)	NEXT (e)	OCCUR (e)
OPEN (e)	OUT (e)	POST (e)	READ (e n)
READC (e)	READE (e n)	READP (e n)	READPE (e n)
REALLOC (e)	REL (e)	RESET (e)	ROLBK (e)
SCAN (e)	SETGT (e)	SETLL (e)	SUBDUR (e)
SUBST (e p)	TEST (e d/t/z)	UNLOCK (e)	UPDATE (e)
WRITE (e)	XLATE (e p)		

注:

1. CALLP (e m/r) は拡張演算項目 2 の命令コードで、エラー標識を持つことはできません。しかし、この命令コードに 'e' を指定することによってプログラム状況およびエラー条件は、判別することができます。

ある命令コードにエラー標識か 'E' 拡張を指定した時には、EXSR 命令によってファイル・エラー処理サブルーチン (INFSR) またはプログラム・エラー処理サブルーチン (*PSSR) を明示的に呼び出すことができます。EXSR 命令によって INFSR または *PSSR が明示的に呼び出され、ENDSR 命令の演算項目 2 がブランクであるか、あるいは指定されたフィールドの値がブランクの場合には、制御は EXSR 命令の後の次の順番の命令に戻されます。

MONITOR グループの使用

MONITOR グループは、状況コードに基づいて、条件付きエラー処理を実行します。エラーが発生すると、MONITOR グループの中の適切な ON-ERROR グループに制御を渡します。

MONITOR ブロックのすべてのステートメントがエラーなしに処理された場合は、ENDMON ステートメントの次のステートメントへ制御が渡されます。

MONITOR グループは演算の中の任意の場所に指定することができます。IF、DO、SELECT、または他の MONITOR グループの中でネストすることもできます。IF、DO、および SELECT グループは、MONITOR グループの中でネストすることができます。

MONITOR が別の MONITOR グループの中でネストされている場合、エラーが起こると、最も内部のグループが最初に考慮されます。その MONITOR グループがそのエラー条件を処理しない場合は、次のグループが考慮されます。

MONITOR 命令ではレベル標識が使用され、MONITOR グループが合計演算の一部であることを示します。文書化を目的としてレベル標識を ON-ERROR または ENDMON 命令に指定することもできますが、このレベル標識は無視されることになります。

MONITOR ステートメントでは、条件付け標識を使用することができます。条件付け標識が満たされない場合には、MONITOR グループの ENDMON 命令の後のステートメントに即時に制御が渡されます。ON-ERROR 命令に単独で条件付け標識を使用することはできません。

MONITOR ブロックがサブプロシージャーの呼び出しを含んでおり、かつそのサブプロシージャーにエラーがある場合は、そのサブプロシージャーのエラー処理が優先順位を有します。例えば、サブプロシージャーに *PSSR サブルーチンがあれば呼び出されます。呼び出しを含む MONITOR グループは、サブプロシージャーがエラー処理に失敗し、00202 の「呼び出し中のエラー」状況で呼び出しが失敗した場合にのみ、考慮されることになります。

MONITOR グループはサブルーチンで発生したエラーを処理します。サブルーチンが自分自身の MONITOR グループを持っている場合は、それらが最初の考慮されます。

MONITOR ブロックの中での分岐命令は許されませんが、ON-ERROR ブロックの中であれば許されます。

MONITOR ブロックの中での LEAVE 命令または ITER 命令は、その MONITOR ブロックを含んでいるすべての活動状態の DO グループに対して適用されます。MONITOR ブロックの中での LEAVESR 命令または RETURN 命令は、その MONITOR ブロックを含んでいるすべてのサブルーチン、サブプロシージャー、またはプロシージャーに対して適用されます。

各 ON-ERROR ステートメントでは、その ON-ERROR グループがどのエラー条件を処理するかを指定します。次のいずれの組み合わせでも、コロンで区切って指定することができます。

nnnnn 状況コード

***PROGRAM** 00100 ～ 00999 のすべてのプログラム・エラー状況コードを処理します。

***FILE** 01000 ～ 09999 のすべてのファイル・エラー状況コードを処理します。

***ALL** 00100 ～ 09999 のプログラム・エラー・コードとファイル・エラー・コードの両方を処理します。これはデフォルトです。

00100 ～ 09999 の範囲外の状況コード (例えば 0 ～ 99 のコード) は、モニターの対象にはなりません。これらの値を ON-ERROR グループに対して指定することはできません。また、使用されているコンパイラーの特定バージョンに対して無効ないかなる状況コードも、指定することはできません。

複数の ON-ERROR グループが同一の状況コードをカバーしている場合は、最初の 1 つだけが使用されます。この理由により、特定の状況コードの後に *ALL などの特殊値を指定する必要があります。

ON-ERROR グループの中で発生するいかなるエラーも、MONITOR グループによって処理されることはありません。エラーを処理するには、ON-ERROR グループの中で MONITOR グループを指定することができます。

```
 * MONITOR ブロックは、READ ステートメントと IF グループで
* 構成されています。
* - 最初の ON-ERROR ブロックは、ファイルがオープンされていない
*   場合に READ 命令に対して出される状況 1211 を処理します。
* - 2 番目の ON-ERROR ブロックは、その他のすべてのファイル・エラーを処理します。
* - 3 番目の ON-ERROR ブロックは、ストリング命令状況コード 00100
*   および配列指標状況コード 00121 を処理します。
* - 4 番目の ON-ERROR ブロック (演算項目 2 が *ALL) は、
*   指定された ON-ERROR 命令では処理されないエラーを
*   処理します。
*
* MONITOR ブロックでエラーが発生しない場合は、制御は
* ENDIF から ENDMON に渡されます。
C                   MONITOR
C                   READ      FILE1
C                   IF        NOT %EOF
C                   EVAL      Line = %SUBST(Line(i) :
C                                       %SCAN('***': Line(i)) + 1)
C                   ENDIF
C                   ON-ERROR   1211
C                    ... ファイルがオープンしないの処理
C                   ON-ERROR   *FILE
C                    ... その他のファイル・エラーの処理
C                   ON-ERROR   00100 : 00121
C                    ... ストリング・エラーおよび配列指標エラーの処理
C                   ON-ERROR
C                    ... その他のすべてのエラーの処理
C                   ENDMON
```

図 128. MONITOR 命令

エラー処理サブルーチンの使用

エラー処理サブルーチンを作成する時には、次の 2 つのことを行います。

1. RPG サブルーチン・エラー処理プログラムの活動化

   このサブルーチン・エラー処理プログラムは例外を処理して、制御をユーザーのサブルーチンに渡します。

2. 回復処置の指定 (任意指定)

   エラー処理サブルーチンを使用して、起こったエラーに基づいて特定の処置を取るか、あるいは一般処置 (例えば、すべてのエラーに照会メッセージを出すなど) を取ることができます。

エラー処理サブルーチンには次の考慮事項が適用されます。

- EXSR 命令の演算項目 2 にエラー処理サブルーチンの名前を指定して、そのサブルーチンを明示的に呼び出すことができます。
- サブルーチンの ENDSR 命令の演算項目 2 に値を指定して、メイン・プロシージャーの処理が再開される点を制御することができます。サブプロシージャーでは、ENDSR の演算項目 2 はブランクでなければなりません。サブプロシージャーが異常終了しないようにするためには、ENDSR 命令の前に GOTO または RETURN 命令のいずれかを使用してください。
- エラー処理サブルーチンが呼び出される場合には、 RPG エラー処理サブルーチン処理プログラムが既に例外を処理しています。このように、エラー処理サブルーチンに対する呼び出しは、プログラム処理への戻りを反映しています。サブルーチンの実行中に例外が起こった場合には、そのサブルーチンが再び呼び出されます。この問題を避けるようにサブルーチンをコーディングしていない限り、プロシージャーはループします。

  このようなループを避けるためのエラー処理サブルーチンのコーディング方法については、 294 ページの『エラー処理サブルーチンでのループの防止』を参照してください。

ファイル・エラー処理 (INFSR) サブルーチンの使用

メイン・プロシージャーでファイル・エラーまたは例外を処理するためには、ファイル・エラー処理 (INFSR) サブルーチンを作成することができます。ファイル例外が起こると、次のようになります。

1. INFDS が更新されます。
2. 次のもので例外が起こった場合には、ファイル・エラー処理サブルーチン (INFSR) が制御を受け取ります。
   - 暗黙の (プライマリーまたはセカンダリー) ファイル操作
   - 73 ～ 74 桁目に標識の指定されていない明示ファイル操作

ファイル・エラー処理サブルーチンは複数のファイルでエラーを処理することができます。

次の制約事項が適用されます。

- プログラムの開始または終了時に (例えば、サイクルの開始時の暗黙オープンで) ファイル例外が起こった場合、制御は、エラー処理サブルーチン処理プログラムではなく、ILE RPG のデフォルトの値の例外処理プログラムに渡されます。したがってファイル・エラー処理サブルーチンは処理されません。
- 命令に関連しないエラー (例えば、CHAIN 命令での配列指標エラー) が起こった場合には、INFSR エラー処理サブルーチンは無視されます。エラーは、他のプログラム・エラーと同様に取り扱われます。
- INFSR はサブプロシージャーとして使用されているファイルでは、エラーを処理することができません。

プログラムにファイル・エラー処理サブルーチンを追加するためには、次のステップを行ってください。

1. ファイル仕様書のキーワード INFSR の後にサブルーチンの名前を記入してください。サブルーチン名は *PSSR とすることができますが、これはこのファイルでの例外でプログラム・エラー処理サブルーチンに制御が渡されることを示します。
2. 任意指定で、キーワード INFDS を使用してファイル仕様書でファイル情報データ構造を指定してください。
3. 演算項目 1 にキーワード INFSR に指定したものと同じサブルーチン名のある BEGSR 命令を指定してください。
4. 戻り点がある場合にはそれを指定して、サブルーチンの ENDSR 命令にそれをコーディングしてください。演算項目 2 の有効な項目については、296 ページの『ENDSR 命令での戻り点の指定』を参照してください。
5. ファイル・エラー処理サブルーチンの残りをコーディングしてください。ファイル・エラー処理サブルーチンで ILE RPG コンパイラー命令を使用することができますが、エラーのあった同じファイルに入出力命令を使用することは望ましくありません。ENDSR 命令は、ファイル・エラー処理サブルーチンの最後の仕様書でなければなりません。

289 ページの図 129 には、INFSR エラー処理サブルーチンを使った例外処理の例が示してあります。プログラム TRNSUPDT は単純な在庫更新プログラムです。このプログラムはトランザクション・ファイル TRANSACT を使用して在庫マスター・ファイル PRDMAS を更新します。入出力エラーが起こった場合には、INFSR エラー処理サブルーチンが呼び出されます。レコード・ロック・エラーの場合には、そのレコードがバックログ・ファイルに書き出されます。そうでない場合には、照会メッセージが出されます。

PRDMAS のファイル仕様書は INFDS を識別するとともに、それに関連した INFSR を識別するということに注意してください。

TRANSACT ファイルの各レコードに対して次のことが行われます。

1. トランザクション製品番号を使用して製品マスター・ファイル中で該当するレコードが見付けられます。
2. レコードが見付かった場合には、在庫数量が更新されます。
3. UPDATE 命令でエラーが起こった場合には、制御が INFSR エラー処理サブルーチンに渡されます。
4. レコードが見付からなかった場合には、エラー報告書に製品番号が書き出されます。

```
     *===================================================================*
     * TRNSUPDT: このプログラムは簡単な在庫更新プログラムです。          *
     * トランザクション・ファイル (TRANSACT) が連続的に処理されます。    *
     * トランザクションにある製品番号が、マスター・ファイル (PRDMAS)     *
     * にランダムにアクセスするキーとして使用されます。                  *
     * 1. レコードが見つかった場合には、在庫数量が                       *
     *    更新されます。                                                 *
     * 2. レコードが見つからなかった場合には、報告書にエラーが           *
     *    印刷されます。                                                 *
     * 3. 現在レコードがロックされている場合には、トランザクションは     *
     *    トランザクション受注残ファイルに書き出され、                  *
     *    後で処理されます。                                             *
     * 4. それ以外の予期しないエラーが起こった場合には、実行時エラー・   *
     *    メッセージが出されます。                                       *
     *===================================================================*
     *-------------------------------------------------------------------*
     * ファイルの定義:                                                   *
     *   1) PRDMAS      - 製品マスター・ファイル                         *
     *   2) TRANSACT    - トランザクション・ファイル                     *
     *   3) TRNBACKLG   - トランザクション受注残ファイル                 *
     *   2) PRINT       - エラー報告書                                   *
     *-------------------------------------------------------------------*
    FPRDMAS    UF   E           K DISK
    F                                     INFSR(PrdInfsr)
    F                                     INFDS(PrdInfds)
    FTRANSACT  IP   E             DISK
    FTRNBACKLG O    E             DISK
    FPRINT     O    F   80        PRINTER
     *-------------------------------------------------------------------*
     * ファイル PRDMAS のファイル情報データ構造を定義します。           *
     * *STATUS フィールドは取るべき処置を決めるために使用されます。     *
     *-------------------------------------------------------------------*
    D PrdInfds        DS
    D  PrdStatus        *STATUS
     *-------------------------------------------------------------------*
     * 予期される例外をリストします。                                    *
     *-------------------------------------------------------------------*
    D ErrRecLock      C                   CONST(1218)
```

図 129. ファイル例外処理の例 (1/2)

```
 *---------------------------------------------------------------*
 * トランザクションの製品番号を使用して、製品マスター・ファイルに  *
 * アクセスします。                                              *
 *---------------------------------------------------------------*
C     TRNPRDNO      CHAIN     PRDREC                         10
 *---------------------------------------------------------------*
 * レコードが見つかった場合、マスター・ファイルの数量を更新します。*
 *---------------------------------------------------------------*
C                   IF        NOT *IN10
C                   SUB       TRNQTY        PRDQTY
C                   UPDATE    PRDREC
 *---------------------------------------------------------------*
 * レコードが見つからない場合には、エラー報告書を書き出します。    *
 *---------------------------------------------------------------*
C                   ELSE
C                   EXCEPT    NOTFOUND
C                   ENDIF
C                   SETON                                      LR
 *---------------------------------------------------------------*
 * エラー処理ルーチン。                                          *
 *---------------------------------------------------------------*
C     PrdInfsr      BEGSR
 *---------------------------------------------------------------*
 * 現在マスター・レコードがロックされている場合、トランザクション・*
 * レコードを受注残ファイルに書き次のトランザクションにスキップ。  *
 *---------------------------------------------------------------*
C     PrdStatus     DSPLY
C                   IF        (PrdStatus = ErrRecLock)
C                   WRITE     TRNBREC
C                   MOVE      '*GETIN'      ReturnPt          6
 *---------------------------------------------------------------*
 * 予期しないエラーが起こった場合、照会メッセージが出されます。    *
 *---------------------------------------------------------------*
C                   ELSE
C                   MOVE      *BLANK        ReturnPt
C                   ENDIF
C                   ENDSR     ReturnPt
 *---------------------------------------------------------------*
 * エラー報告書の形式。
 *---------------------------------------------------------------*
OPRINT     E            NOTFOUND
O                       TRNPRDNO
O                                          29 'NOT IN PRDMAS FILE'
```

図 129. ファイル例外処理の例 (2/2)

制御がエラー処理サブルーチンに渡されると、次のことが起こります。

- エラーの原因がレコード・ロックである場合には、そのレコードが受注残ファイルに書き出され、次のトランザクションで制御が主要部分に戻されます (戻り点として *GETIN を介します)。
- エラーの原因が何か他の理由である場合には、ReturnPt にブランクが転送されます。これにより、RPG のデフォルトの処理プログラムに制御が渡されます。その点での回復処置は、エラーの性質によります。

レコード・ロック・エラーの検査は、PRDMAS の INFDS の *STATUS サブフィールドを、レコード・ロック状況コードの値によって定義される ErrRecLock フィールドと突き合わせることによって行われるということに注意してください。他のエラーを定義し、そのエラーを検査し、適切な処置を取ることによって、他のタイプの入出力エラーを処理できるように INFSR を拡張することができます。

プログラム・エラー処理サブルーチンの使用

プログラム・エラーまたは例外を処理するためには、プログラム・エラー処理サブルーチン (*PSSR) を作成することができます。プログラム・エラーが起こると、次のようになります。

1. プログラム状況データ構造が更新されます。
2. 命令コードの 73 ～ 74 桁目に標識が指定されていない 場合には、エラーが処理され、制御が *PSSR に移されます。

   ファイル仕様書でキーワード INFSR の後に *PSSR を指定することによって、ファイル・エラーの後でプログラム・エラー処理サブルーチンに明示的に制御を渡すことができます。

モジュールのプロシージャーのどれか (あるいはすべて) に *PSSR をコーディングすることができます。各 *PSSR は、それがコーディングされているプロシージャーに固有です。

プログラムに *PSSR エラー処理サブルーチンを追加するためには、次のステップを行ってください。

1. 定義仕様書の 23 桁目に S を指定することによって、任意指定でプログラム状況データ構造 (PSDS) を指定してください。
2. *PSSR の演算項目 1 で BEGSR 命令を指定してください。
3. 戻り点がある場合にはそれを指定して、サブルーチンの ENDSR 命令にそれをコーディングしてください。サブプロシージャーでは、演算項目 2 はブランクでなければなりません。演算項目 2 の有効な項目については、 296 ページの『ENDSR 命令での戻り点の指定』を参照してください。
4. プログラム・エラー処理サブルーチンの残りをコーディングしてください。プログラム・エラー処理サブルーチンでは、どんな ILE RPG コンパイラー命令でも使用することができます。ENDSR 命令は、プログラム・エラー処理サブルーチンの最後の仕様書でなければなりません。

292 ページの図 130 には、メイン・プロシージャーのプログラム・エラー処理サブルーチンの例を示してあります。

```
 *---------------------------------------------------------------*
 *  プログラム状況データ構造の対応する部分を定義します。          *
 *---------------------------------------------------------------*
D Psds           SDS
D  Loc             *ROUTINE
D  Err             *STATUS
D  Parms           *PARMS
D  Name            *PROC
 *---------------------------------------------------------------*
 *  コーディングの本体部分
 *  ゼロによる除算が行なわれるとエラーが発生します。
 *  *PSSR サブルーチンに制御が渡されます。
 *---------------------------------------------------------------*
 *===============================================================*
 * *PSSR: メイン・プロシージャーのエラー処理サブルーチン。状況が
 *        102 であるかどうかをチェックすることにより、ゼロによる
 *        除算エラーをチェックします。エラーがあれば、除数に 1 を
 *        加算し、*GETIN を ReturnPt に転送して続行します。
 *===============================================================*
C     *PSSR         BEGSR
C                   IF        Err = 102
C                   ADD       1             Divisor
C                   MOVE      '*GETIN'      ReturnPt          6
 *---------------------------------------------------------------*
 *       予期しないエラーが起こったため、
 *       *CANCL を ReturnPt に転送してプロシージャーを終了させます。
 *---------------------------------------------------------------*
C                   ELSE
C                   MOVE      '*CANCL'      ReturnPt
C                   ENDIF
C                   ENDSR     ReturnPt
```

*図 130. メイン・プロシージャーの *PSSR サブルーチンの例*

プログラム状況データ構造は、定義仕様書で定義されます。事前定義のサブフィールドの *STATUS, *ROUTINE, *PARMS, および *PROGRAM が指定され、各サブフィールドには名前が割り当てられています。

*PSSR エラー処理サブルーチンは、演算仕様書でコーディングされています。プログラム・エラーが起こると、ILE RPG は制御を *PSSR エラー処理サブルーチンに渡します。このサブルーチンは、除数がゼロの除算命令が原因で例外が起こったかどうかを判別します。そうであった場合には、除数 (Divisor) に 1 が加算され、フィールド ReturnPt にリテラル '*DETC' が転送され、明細演算ルーチンの始めからプログラムが処理を再開することを示します。

例外がゼロによる除算でなかった場合には、リテラル '*CANCL' がフィールド ReturnPt に転送され、プロシージャーが終了します。

293 ページの図 131 および 294 ページの図 132 では、サブプロシージャーの類似したプログラム・エラー処理サブルーチンをコーディングする方法を示しています。1 つの例として、GOTO を、他の例としては、RETURN 命令をコーディングします。

```
 *---------------------------------------------------------------*
 *  サブプロシージャー定義の開始
 *---------------------------------------------------------------*
P SubProc         B
D SubProc         PI             5P 0
 ...
 *---------------------------------------------------------------*
 *  回復コードを含むコーディング本体
 *---------------------------------------------------------------*
C      TryAgain     TAG
C      X            DIV       Divisor       Result
C                   Return    Result
 *---------------------------------------------------------------*
 *  ゼロによる除算が行なわれるとエラーが発生します。
 *  *PSSR サブルーチンに制御が渡されます。
 *---------------------------------------------------------------*
C     *PSSR         BEGSR
 *---------------------------------------------------------------*
 * これがゼロによる除算のエラーの場合は、除数に 1 を加算して、
 * 再試行します。
 *---------------------------------------------------------------*
C                   IF        Err = 102
C                   ADD       1             Divisor
C                   GOTO      TryAgain
C                   ENDIF
 *---------------------------------------------------------------*
 * 制御が ENDSR に達すると、プロシージャーは正常に実行されません。
 *---------------------------------------------------------------*
C                   ENDSR
P                 E
```

*図 131. GOTO のある *PSSR サブルーチン のサブプロシージャーの例*

```
 *---------------------------------------------------------------*
 *  サブプロシージャー定義の開始
 *---------------------------------------------------------------*
P SubProc         B
D SubProc         PI             5P 0
 ...
 *---------------------------------------------------------------*
 *  除算命令を含むコーディング本体
 *---------------------------------------------------------------*
C      X          DIV       Divisor       Result
C                 Return    Result
 *---------------------------------------------------------------*
 *  ゼロによる除算が行なわれるとエラーが発生します。
 *  *PSSR サブルーチンに制御が渡されます。
 *---------------------------------------------------------------*
C     *PSSR        BEGSR
 *---------------------------------------------------------------*
 * これがゼロによる除算エラーの場合、サブプロシージャーから
 * 0 が戻されます。
 *---------------------------------------------------------------*
C                 IF        Err = 102
C                 RETURN    0
C                 ENDIF
 *---------------------------------------------------------------*
 * 制御が ENDSR に達すると、プロシージャーは正常に実行されません。
 *---------------------------------------------------------------*
C                 ENDSR
P                 E
```

*図 132. RETURN のある *PSSR サブルーチン のサブプロシージャーの例*

エラー処理サブルーチンでのループの防止

前の例では、*PSSR でエラーが起こるため、ループの起こる可能性は少なくなっています。しかし、*PSSR がどのように書かれるかによって、*PSSR の処理中に例外が起こるとループが起こる可能性があります。

このようなループを避ける 1 つの方法として、サブルーチンに初回の実行を示すスイッチを設定することができます。サブルーチンの初回の実行でない場合には、ENDSR 命令の演算項目 2 に *CANCL などの適切な戻り点を指定することができます。

295 ページの図 133 には、*PSSR サブルーチン内でのループを避ける方法を示すために例外を生成するように設計されたプログラム NOLOOP を示してあります。このプログラムは例外を次の 2 回生成します。

1. コードの本体で *PSSR に制御権を渡すため
2. *PSSR 内部で潜在的にループを起こすため

```
 *====================================================================*
 * NOLOOP:  *PSSR サブルーチンの反復を回避する方法を示します。      *
 *====================================================================*
 *--------------------------------------------------------------------*
 * エラーを起こすために使用する配列                                   *
 *--------------------------------------------------------------------*
D Arr1            S             10A   DIM(5)
 *--------------------------------------------------------------------*
 * 配列の範囲外エラーを生成し、制御を *PSSR に渡します。              *
 *--------------------------------------------------------------------*
C                   Z-ADD     -1            Neg1              5 0
C                   MOVE      Arr1(Neg1)    Arr1(Neg1)
C                   MOVE      *ON           *INLR
 *====================================================================*
 * *PSSR: プロシージャーのエラー処理サブルーチン。PSSR の反復を検出*
 *        するために変数 InPssr を使用します。                        *
 *        反復を検出した場合には、プロシージャーを *CANCL します。  *
 *====================================================================*
C     *PSSR         BEGSR
C                   IF        InPssr = 1
C                   MOVE      '*CANCL'      ReturnPt          6
C                   Z-ADD     0             InPssr            1 0
C                   ELSE
C                   Z-ADD     1             InPssr
 *                                                                  *
 *         サブルーチンがプロシージャーを取り消す方法を見るために  *
 *         ここで PSSR に別のエラーを生成します。                   *
 *                                                                  *
C                   MOVE      Arr1(Neg1)    Arr1(Neg1)
 *                                                                  *
 *         Neg1 がまだ負の場合には、次の 2 つの命令は処理されない  *
 *         ことに注意してください。                                 *
 *                                                                  *
C                   MOVE      '*GETIN'      ReturnPt
C                   Z-ADD     0             InPssr
C                   ENDIF
C                   ENDSR     ReturnPt
```

図 133. エラー処理サブルーチンでのループの防止

図 133 のソースを使ってプログラムを作成し、そのデバッグを始めるには、次の入力をします。

```
CRTBNDRPG PGM(MYLIB/NOLOOP) DBGVIEW(*SOURCE)
STRDBG PGM(MYLIB/NOLOOP)
```

*PSSR サブルーチンの BEGSR 行に中断点を設定することにより、*PSSR サブルーチンをステップスルーすることができます。

プログラムを呼び出すと、次のことが起こります。

1. プログラムが負の指標を使用して配列に MOVE 命令を実行しようとすると、例外が起こります。制御が *PSSR に渡されます。
2. これが *PSSR の最初の実行であるので、変数 In_Pssr はまだオンに設定されていません。将来のループを防止するために変数 In_Pssr がオンに設定されます。
3. 処理は、*PSSR 内の ELSE の後の MOVE で続行されます。再び例外が起こり、*PSSR の処理が新たに始まります。
4. 今回の実行では変数 In_Pssr が既に 1 に設定されています。これはサブルーチンがループ状態にあることを示しているので、ReturnPt フィールドを *CANCL に設定することによってプロシージャーが取り消されます。

5. ENDSR 命令が制御を受け取り、プロシージャーが取り消されます。

ループを避けるためにここで用いた方法は、INFSR エラー処理サブルーチン内でも使用することができます。

ENDSR 命令での戻り点の指定

メイン・プロシージャーで INFSR または *PSSR エラー処理サブルーチンを使用する時には、ENDSR ステートメントの演算項目 2 として次の 1 つを入力することによって、プログラムが処理を再開する戻り点を示すことができます。この項目は、値が次の戻り点の 1 つを指定する 6 桁の文字フィールド、リテラル、名前付き固定情報、配列要素、またはテーブル名でなければなりません。

注: 戻り点をリテラルとして指定する場合には、リテラルはアポストロフィで囲み、大文字 (例えば *detl ではなく *DETL) で入力しなければなりません。フィールドまたは配列要素に指定する場合には、フィールドまたは配列要素の中でこの値は左寄せにしなければなりません。

***DETL** 明細行の始めから続行

***GETIN** 入力レコード取得ルーチンから続行

***TOTC** 合計演算の始めから続行

***TOTL** 合計行の始めから続行

***OFL** オーバーフロー行の始めから続行

***DETC** 明細演算の始めから続行

***CANCL** プログラムの処理の取り消し

ブランク ILE RPG のデフォルトの例外処理プログラムへ制御を戻します。演算項目 2 がブランクの値である時および 演算項目 2 が指定されていない時にこれが起こります。サブルーチンが EXSR 命令によって呼び出され、演算項目 2 がブランクの場合には、次の順番の命令に制御が渡されます。

INFSR または *PSSR サブルーチンの ENDSR 命令の実行後に ILE RPG コンパイラーは、演算項目 2 に指定されたフィールドまたは配列要素をブランクにリセットします。演算項目 2 がブランクに設定されているので、サブルーチン内で起こった例外に最も適した戻り点を指定することができます。

サブルーチンの終わりでこのフィールドがブランクである場合には、INFSR または *PSSR サブルーチンが EXSR 命令で呼び出された場合を除いて、サブルーチンの実行に続いて、ILE RPG のデフォルトの例外処理プログラムが制御を受け取ります。サブルーチンが EXSR 命令で呼び出され、ENDSR 命令の演算項目 2 がブランクである場合には、制御は EXSR 命令の次の順番の命令に戻ります。

注: サブプロシージャーでは、ENDSR に演算項目 2 を指定することはできません。サブプロシージャーで処理を再開したい場合には、サブプロシージャーの本体の TAG に GOTO 命令を使用します。かわりに、*PSSR では RETURN 命令をコーディングすることができます。その後で、サブプロシージャーは呼び出し元に戻ります。

ILE 条件処理プログラム

ILE 条件処理プログラムとは、ILE 条件処理プログラム登録 (CEEHDLR) バインド可能 API を使って実行時に登録される、例外処理プログラムのことです。条件処理プログラムは、例外を処理するか、パーコレートするか、あるいはプロモートするために使用されます。例外は、ILE 条件の形で条件処理プログラムに提示されます。複数の ILE 条件処理プログラムを登録することができます。ILE 条件処理プログラムは、ILE 条件処理プログラム登録抹消 (CEEHDLU) バインド可能 API を使って登録を抹消することができます。

ILE 条件処理プログラムを使いたくなるには、いくつかの理由があります。

- ユーザー独自の処理プログラムで例外を処理することによって、言語特有の処理を回避することができます。

  これにより、さまざまな ILE HLL のモジュールのアプリケーションに、同じ例外処理手法を提供することができます。
- この API を使用して例外処理を呼び出しスタック項目まで拡大することができます。

  ILE バインド可能 API CEEHDLR は、それを含む呼び出しまで拡大されます。これは、登録解除するか、あるいはそのプロシージャーから制御が戻されるまで効力をもっています。

  注: 明細、合計、またはサブルーチン演算からの任意の CEEHDLR API 呼び出しによって、条件処理プログラムが、すべての入力、演算、または出力操作を含めて、プロシージャー全体に対して活動状態になります。しかし、これはサブプロシージャーに影響せず、CEEHDLR を呼び出すサブプロシージャーもメイン・プロシージャーに影響しません。

  サブプロシージャーが反復して呼び出される場合には、CEEHDLR を呼び出す呼び出しだけがそれによる影響を受けます。すべての呼び出しに対して条件処理プログラムを活動状態にしたい場合には、CEEHDLR は各呼び出しごとに呼び出されなければなりません。

ILE 条件処理プログラムの使い方に関する情報は、「*ILE 概念*」を参照してください。

条件処理プログラムの使用

次の例は以下のことを行う方法を示したものです。

1. RPG「範囲外」エラーを処理する条件処理プログラムをコーディングする
2. 条件処理プログラムを登録する
3. 条件処理プログラムの登録を取り消す
4. *PSSR エラー処理サブルーチンをコーディングする

この例は、次の 2 つのプロシージャーから成っています。

- RPGHDLR。範囲外サブストリング・エラーを処理するためのユーザー作成条件処理プログラムからなります。
- SHOWERR。RPGHDLR プロシージャーをテストします。

SHOWERR は主に RPGHDLR の働きを示すために設計されていますが、2 つが結合したプロシージャーは、ILE 例外処理が「どのように」行われるかを判別する上でも有用です。両方のプロシージャーは、処理されて時に起こる「処置」を QSYSPRT に書き出します。調べたい、ILE 例外処理の別の側面をシミュレートするために、これらのプロシージャーを変更することができます。

299 ページの図 134 にはプロシージャー RPGHDLR のソースを示してあります。このプロシージャーは 3 つのプロシージャー・パラメーターを定義します。すなわち、ILE 条件トークンの構造、SHOWERR と RPGHDLR 間の連絡域へのポインター、および可能な処置、再開またはパーコレートの指示が入るフィールドです (RPGHDLR は例外をプロモートしません)。

RPGHDLR の基本ロジックは次のとおりです。

1. メッセージ ID をテストすることによって範囲外エラーであるかを調べるためのテストをする。
   - 範囲外エラーの場合および SHOWERR が範囲外エラーが無視されることを指示している場合には、'Handling...' を QSYSPRT に書き出して、処置を「再開」に設定します。
   - 範囲外エラーでない場合には、'Percolating' を QSYSPRT に書き出して、処置を「パーコレート」に設定します。
2. 戻る。

```
 *================================================================*
 * RPGHDLR: RPG 例外処理プロシージャー。                          *
 *          このプロシージャーは次のことを実行します。            *
 *          RPG 範囲外エラー (RNX0100) の場合には、               *
 *             例外を処理します。                                 *
 *          そうでない場合には、                                  *
 *             例外をパーコレートします。                         *
 *          また、実行したことを印刷します。                      *
 *                                                                *
 * 注:      これは SHOWERR プロシージャーの例外処理プロシージャー *
 *          です。                                                *
 *================================================================*
FQSYSPRT   O    F  132        PRINTER

D RPGHDLR         PR
D  Parm1                           LIKE(CondTok)
D  Parm2                         *
D  Parm3                       10I 0
D  Parm4                           LIKE(CondTok)

 *----------------------------------------------------------------*
 * プロシージャー・パラメーター                                   *
 * 1. 入力: 条件トークン構造                                      *
 * 2. 入力: 以下を含んでいる連絡域へのポインター                  *
 *       a. 処理されているプロシージャーの PSDS へのポインター    *
 *       b. ストリング・エラーが有効かどうかを示す標識            *
 * 3. 出力: 例外に対して実行されるべき処置を                      *
 *          識別するコード                                        *
 * 4. 出力: 条件をプロモートすることに決めた場合の新しい条件。    *
 *          この処理プログラムは再開とパーコレートしか実行しない *
 *          ので、このパラメーターを無視します。                  *
 *----------------------------------------------------------------*
D RPGHDLR         PI
D  InCondTok                       LIKE(CondTok)
D  pCommArea                     *
D  Action                      10I 0
D  OutCondTok                      LIKE(CondTok)
```

図 134. 範囲外サブストリング・エラーの条件処理プログラムのソース (1/2)

```
D CondTok         DS                  BASED(pCondTok)
D   MsgSev                       5I 0
D   MsgNo                        2A
D                                1A
D  MsgPrefix                     3A
D  MsgKey                        4A

D CommArea        DS                  BASED(pCommArea)
D   pPSDS                         *
D   AllowError                   1N

D PassedPSDS      DS                  BASED(pPSDS)
D   ProcName              1     10

 *
 * 処置コード:
 *
D Resume          C                   10
D Percolate       C                   20

 *----------------------------------------------------------------*
 *      入力条件トークンをポイントする。                          *
 *----------------------------------------------------------------*
C                   EVAL      pCondTok = %ADDR(InCondTok)

 *----------------------------------------------------------------*
 * サブストリング・エラーの場合は、ELSE パーコレートを処理します。 *
 * メッセージ番号 (MsgNo) の値は 16 進であることに注意。          *
 *----------------------------------------------------------------*
C                   EXCEPT
C                   IF        MsgPrefix = 'RNX' AND
C                             MsgNo     = X'0100' AND
C                             AllowError = '1'
C                   EXCEPT    Handling
C                   EVAL      Action    = Resume
C                   ELSE
C                   EXCEPT    Perclating
C                   EVAL      Action = Percolate
C                   ENDIF
C                   RETURN

 *================================================================*
 * プロシージャー出力                                             *
 *================================================================*
OQSYSPRT   E
O                                           'HDLR: In Handler for '
O                        ProcName
OQSYSPRT   E             Handling
O                                           'HDLR: Handling...'
OQSYSPRT   E             Perclating
O                                           'HDLR: Percolating...'
```

図 134. 範囲外サブストリング・エラーの条件処理プログラムのソース (2/2)

302 ページの図 135 は、条件処理プログラム RPGHDLR が登録されているプロシージャー SHOWERR のソースを示します。

プロシージャー・パラメーターには、RPGHDLR へのプロシージャー・ポインター、およびモジュールの PSDS へのポインターを含む連絡域へのポインター、そして範囲外ストリング・エラーを無視できるかどうかを示す標識が含まれます。さらに、エラーとなる可能性がある配列 ARR1 の定義、および ILE バインド可能 API の CEEHDLR および CEEHDLU によって使用されるパラメーター・リストの識別を必要とします。

プログラムの基本論理は次のとおりです。

1. サブルーチン RegHndlr を使用して処理プログラム RPGHDLR を登録する。このサブルーチンは CEEHDLR API を呼び出し、それに RPGHDLR を指すプロシージャー・ポインターを渡します。
2. RPGHDLR に対して、範囲外エラーが許されることを指示してから、範囲外サブストリング・エラーを生成し、次いで、RPGHDLR が予期せぬ範囲外ストリング・エラーを許さないように標識をオフに設定する。

   処理プログラム RPGHDLR は自動的に呼び出されます。この処理プログラムは例外を処理し、エラーの後の次の*機械語* 命令で処理が再開されることを示します。次の機械語命令が次の RPG 命令の始めにない場合があるということに注意してください。
3. 範囲外配列エラーを生成する。

   再び RPGHDLR が自動的に呼び出されます。しかし今回は例外を処理できないので、それをプロシージャーに関連した次の例外処理プログラム、すなわち *PSSR エラー処理サブルーチンにパーコレートします。

   *PSSR がプロシージャーを取り消します。
4. CEEHDLU の呼び出しを介して条件処理プログラム RPGHDLR を登録解除する。
5. 戻る。

RPGHDLR プロシージャーと同様に、SHOWERR は QSYSPRT に書き出しを行って、処理時に何が起こったかを示します。

```
 *====================================================================*
 * SHOWERR:       ユーザー定義の例外処理プログラムを使用した          *
 *                例外処理を示します。                                *
 *====================================================================*
FQSYSPRT   O    F  132        PRINTER

 *--------------------------------------------------------------------*
 * CEEHDLR API のパラメーター定義は次のとおりです。最初の            *
 * パラメーターは例外を処理するプロシージャーへのプロシージャー・    *
 * ポインターです。2 番目は例外処理プロシージャーに渡される          *
 * 連絡域へのポインターです。この例では、                            *
 * この連絡域にはこのモジュールの PSDS へのポインターと、            *
 * エラーが許されるかどうかを示す                                    *
 * 標識が含まれています。                                            *
 *                                                                   *
 * このプログラム (SHOWERR) が処理されたいかなるエラーも無視しない *
 * ようにするため、RPGHDLR が「許可」するであろうエラーを起こす     *
 * 可能性のあるすべての命令の後で「エラー」標識をチェックします。    *
 * また、プログラムの終了時に、いかなるエラーも見逃さなかった       *
 * ことを確認するためにチェックを行ないます。                        *
 *--------------------------------------------------------------------*
D pConHdlr        S               *   PROCPTR
D                                     INZ(%paddr('RPGHDLR'))

 *--------------------------------------------------------------------*
 * 連絡域                                                            *
 *--------------------------------------------------------------------*
D CommArea        DS                  NOOPT
D   pPsds                          *   INZ(%ADDR(DSPsds))
D   AllowError                    1N   INZ('0')

 *--------------------------------------------------------------------*
 * PSDS                                                              *
 *--------------------------------------------------------------------*
D DSPsds         SDS                  NOOPT
D   ProcName        *PROC

 *--------------------------------------------------------------------*
 * エラーを起こすために使用される変数                                *
 *--------------------------------------------------------------------*
D Arr1            S             10A   DIM(5)
D Num             S              5P 0

 *--------------------------------------------------------------------*
 * CEEHDLR インターフェース
 *--------------------------------------------------------------------*
D CEEHDLR         PR
D   pConHdlr                       *   PROCPTR
D   CommArea                       *   CONST
D   Feedback                      12A   OPTIONS(*OMIT)

 *--------------------------------------------------------------------*
 * CEEHDLU インターフェース
 *--------------------------------------------------------------------*
D CEEHDLU         PR
D   pConHdlr                       *   PROCPTR
D   Feedback                      12A   OPTIONS(*OMIT)
```

図 135. 条件処理プログラム登録のソース (1/3)

```
 *----------------------------------------------------------------*
 * 処理プログラムを登録し、エラーを生成します。                   *
 *----------------------------------------------------------------*
C                   EXSR      RegHndlr

 *----------------------------------------------------------------*
 *      サブストリング・エラーを生成します。                      *
 *      これはこの例で「許される」エラーです (RPGHDLR は、エラーの *
 *      後で次の命令に戻るための制御を許可するという、例外を処理  *
 *      します)。                                                 *
 *      RPGHDLR は "AllowError" 標識がオンに設定されているとき以外 *
 *      エラーを許しません。これは、例えば SCAN 命令が後で        *
 *      SHOWERR に追加された場合、RPGHDLR はデフォルトでエラーを  *
 *      許可しないことを保証します。                              *
 *----------------------------------------------------------------*
C                   Z-ADD     -1            Num
C                   EVAL      AllowError = '1'
C     Num           SUBST     'Hello'       Examp           10
C                   EVAL      AllowError = '0'

 *----------------------------------------------------------------*
 *      処理プログラムによって例外が処理され、ここで              *
 *      制御が再開されます。                                      *
 *----------------------------------------------------------------*
C                   EXCEPT    ImBack

 *----------------------------------------------------------------*
 *      配列の範囲外エラーを生成します。                          *
 *      これはこの例では「予想された」エラーではありません。      *
 *----------------------------------------------------------------*
C                   Z-ADD     -1            Num
C                   MOVE      Arr1(Num)     Arr1(Num)

 *----------------------------------------------------------------*
 *      この例外は処理プログラムでは処理されないので、            *
 *      制御はここには戻りません。この例外は                      *
 *      パーコレートされ、制御は *PSSR で再開されます。           *
 *----------------------------------------------------------------*

 *----------------------------------------------------------------*
 *      処理プログラムの登録解除                                  *
 *      注: 処理プログラムが登録解除される前に例外が起こった場合  *
 *      は、プロシージャーが取り消される時に自動的に              *
 *      登録解除されます。                                        *
 *----------------------------------------------------------------*
C                   EXSR      DeRegHndlr
C                   SETON                                      LR

 *================================================================*
 * RegHdlr - 処理プログラムを登録する API の呼び出し             *
 *================================================================*
C     RegHndlr      BEGSR
C                   CALLP     CEEHDLR(pConHdlr : %ADDR(CommArea) : *OMIT)
C                   ENDSR

 *================================================================*
 * DeRegHndlr - 処理プログラムを登録解除する API の呼び出し      *
 *================================================================*
C     DeRegHndlr    BEGSR
C                   CALLP     CEEHDLU(pConHdlr : *OMIT)
C                   ENDSR
```

図 135. 条件処理プログラム登録のソース (2/3)

```
 *===================================================================*
 * *PSSR: プロシージャーのエラー処理サブルーチン                     *
 *===================================================================*
C     *PSSR         BEGSR
C                   EXCEPT    InPssr
C                   EXCEPT    Cancelling
 C                   ENDSR     '*CANCL'

 *===================================================================*
 * プロシージャー出力                                                *
 *===================================================================*
OQSYSPRT   E            ImBack
O                                           'I''m Back'
OQSYSPRT   E            InPssr
O                                           'In PSSR'
OQSYSPRT   E            Cancelling
O                                           'Cancelling...'
```

図 135. 条件処理プログラム登録のソース (3/3)

これらのプロシージャーを試したい場合には、次のステップに従ってください。

1. 299 ページの図 134 に示されたソースを使用してプロシージャー RPGHDLR を作成するために、次を入力してください。

   ```
   CRTRPGMOD MODULE(MYLIB/RPGHDLR)
   ```

2. 302 ページの図 135 に示されたソースを使用してプロシージャー SHOWERR を作成するために、次を入力してください。

   ```
   CRTRPGMOD MODULE(MYLIB/SHOWERR)
   ```

3. プログラム ERRORTEST を作成するために、次を入力してください。

   ```
   CRTPGM PGM(MYLIB/ERRORTEST) MODULE(SHOWERR RPGHDLR)
   ```

4. プログラム ERRORTEST を実行するために、次を入力してください。

   ```
   OVRPRTF FILE(QSYSPRT) SHARE(*YES)
   CALL PGM(MYLIB/ERRORTEST)
   ```

   出力を以下に示します。

   ```
   HDLR: In Handler for SHOWERR
   HDLR: Handling...
   I'm Back
   HDLR: In Handler for SHOWERR
   HDLR: Percolating...
   In PSSR
   Cancelling...
   ```

取り消し処理プログラムの使用

取り消し処理プログラムは、呼び出しスタック項目が通常の戻り以外の何らかの理由によって打ち切られた時に、終結処置および回復処置のための制御を受け取れるようにするという重要な機能を提供します。例えば、プロシージャーがシステム要求 '2' によって、あるいは照会メッセージに 'C' (取り消し) の応答があったために終了した時に、制御を受け取るために取り消し処理プログラムを必要とする場合があります。

呼び出しスタック項目終了ユーザー出口プロシージャー登録 (CEERTX) および、呼び出しスタック項目終了ユーザー出口プロシージャー (CEEUTX) ILE バインド可能 API は、ユーザー定義ルーチンが登録される呼び出しスタック項目が取り消された

時に実行する、このユーザー定義ルーチンを自動的に登録する方法を提供します。登録されると、呼び出しスタック項目が除去されるか、あるいは CEEUTX が使用禁止にするために呼び出されるまで、取り消し処理プログラムは有効のままとなっています。ILE バインド可能 API についての詳細は、Web サイト http://www.ibm.com/eserver/iseries/infocenter で **iSeries 400 Information Center** の中の『*プログラミング*』カテゴリーの『*CL および API*』の節を参照してください。

図 136 は、サブプロシージャーの取り消し処理プログラムの活動可能化、およびコーディングの例を示します (また、取り消し処理プログラムは同じ方法でメイン・プロシージャーを使用可能にすることができます)。

```
 *---------------------------------------------------------------
 * 取り消し処理プログラムのプロトタイプを定義します。
 * このプロシージャーはローカル・プロシージャーです。
 *---------------------------------------------------------------
D CanHdlr         PR
D   pMsg                          *
 *---------------------------------------------------------------
 * 取り消し処理プログラムを使用可能にするための
 * メッセージを、表示します。
 *---------------------------------------------------------------
D Enabler         PR
 *---------------------------------------------------------------
 * Enabler を呼び出すためのサブプロシージャーのプロトタイプを
 * 定義します。
 *---------------------------------------------------------------
D SubProc         PR
 *---------------------------------------------------------------
 * メイン・プロシージャー。SubProc を 3 回呼び出します。
 *---------------------------------------------------------------
C                   CALLP     SubProc
C                   CALLP     SubProc
C                   CALLP     SubProc
C                   SETON                                        LR
 *---------------------------------------------------------------
 * プロシージャー SubProc。Enabler を呼び出します。この呼び出しは
 * 失敗することになるので、エラーを処理するローカル *PSSR サブ
 * ルーチンを定義します。
 *---------------------------------------------------------------
P SubProc         B
C                   CALLP     Enabler
 *---------------------------------------------------------------
 * PSSR に RETURN 命令があるため、メイン・プロシージャーからの
 * SubProc に対する呼び出しは失敗します。
 *---------------------------------------------------------------
C     *PSSR         BEGSR
C     'Subproc PSSR'DSPLY
C                   RETURN
C                   ENDSR
P SubProc         E
```

図 136. サブプロシージャーの取り消し処理プログラムの活動可能化、およびコーディング (1/3)

```
 *---------------------------------------------------------------
 * プロシージャー Enabler。このプロシージャーは取り消し処理
 * プログラムを使用可能にしてから、Enabler が取り消される原因
 * となるエラーを受け取ります。
 *---------------------------------------------------------------
P Enabler         B
 * ローカル変数
D Handler         S               *   PROCPTR  INZ(%PADDR('CANHDLR'))
D Msg             S             20A
D pMsg            S               *   INZ(%ADDR(Msg))
D Zero            S              5P 0 INZ(0)
D Count           S              5I 0 INZ(0) STATIC
D Array           S              1A   DIM(2)
 *---------------------------------------------------------------
 * 取り消し処理プログラムを使用可能にします。このプロシージャーが
 * 取り消された時にプロシージャー 'CANHDLR' が呼び出されます。
 *---------------------------------------------------------------
C                   CALLB     'CEERTX'
C                   PARM                    Handler
C                   PARM                    pMsg
C                   PARM                    *OMIT
 *---------------------------------------------------------------
 * このプロシージャーは 3 回呼び出されます。最初の 2 回は、取り消し
 * 処理プログラムが使用可能になった時にエラーを受け取ります。
 *---------------------------------------------------------------
C                   EVAL      Count = Count + 1
C                   SELECT
C                   WHEN      Count = 1
C                   EVAL      Msg = 'Divide by zero'
C                   EVAL      Zero = Zero / Zero
C                   WHEN      Count = 2
C                   EVAL      Msg = 'String error'
C     'A'           SCAN      'ABC':Zero    Zero
 *---------------------------------------------------------------
 * 3 回目の呼び出しでは取り消し処理プログラムが使用不可にされます。
 * 配列指標エラーのためにプロシージャーは失敗しますが、
 * 処理プログラムは呼び出されません。
 *---------------------------------------------------------------
C                   WHEN      Count = 3
C                   CALLB     'CEEUTX'
C                   PARM                    Handler
C                   PARM                    *OMIT
C                   EVAL      Msg = 'Array index error'
C                   EVAL      Array(Zero) = 'x'
C                   ENDSL
P Enabler         E
```

図 136. サブプロシージャーの取り消し処理プログラムの活動可能化、およびコーディング (2/3)

```
     *---------------------------------------------------------------
     * 取り消し処理プログラムを定義します。パラメーターは、表示される
     * メッセージを示す「連絡域」へのポインターです。
     *---------------------------------------------------------------
    P CanHdlr         B
    D CanHdlr         PI
    D   pMsg                          *
     *---------------------------------------------------------------
     * 入力ポインター pMsg に基づいてフィールドを定義します。
     *---------------------------------------------------------------
    D Msg             S             20A         BASED(pMsg)
     *---------------------------------------------------------------
     * 処理プログラムを使用可能にしたプロシージャーが設定した
     * メッセージを表示します。
     *---------------------------------------------------------------
    C     'Cancel Hdlr 'DSPLY                   Msg
    P CanHdlr         E
```

図 136. サブプロシージャーの取り消し処理プログラムの活動可能化、およびコーディング (3/3)

次は、プログラム CANHDLR からの出力を示しています。プロシージャー SubProc の *PSSR が 3 回呼び出されていますが、3 番目のエラーの前に使用不可になったために、取り消し処理プログラムは 2 回だけ呼び出されています。

```
DSPLY  Cancel Hdlr     ゼロによる除算
DSPLY  Subproc PSSR
DSPLY  Cancel Hdlr     ストリング・エラー
DSPLY  Subproc PSSR
DSPLY  Subproc PSSR
```

図 137. CANHDLR プログラムからの出力

ILE CL が通知および状況メッセージを監視する際の問題

ILE RPG プロシージャーが同一活動化グループ内の ILE CL プロシージャーによって呼び出される時に、呼び出し元が状況メッセージまたは通知メッセージについて監視している場合には、ILE CL の呼び出し元は、この ILE RPG プロシージャーが無視しようとしていた通知メッセージまたは状況メッセージにより、制御権を得る場合があります。

例えば、ILE RPG プロシージャーがプリンター・ファイルにレコードを書き込むときに、実際のプリンター・ファイルのレコード長が、RPG プロシージャーで宣言された短いレコード長である場合には、通知メッセージ CPF4906 が RPG プロシージャーに送られます。RPG 例外処理はこのメッセージをパーコレートするため、デフォルトの値応答の 'I' がこのメッセージを無視します。これを使用すると、出力操作が正常に続行され、 RPG プロシージャーが次の命令に進みます。

ただし、ILE CL MONMSG が制御を得ると、制御は、MONMSG の処置または ILE CL プロシージャーの次のステートメントにただちに渡されます。

注: この問題が生じる場合、メッセージを監視しているプロシージャーが、 RPG プロシージャーの直接の呼び出し元である必要はありません。

この問題は、多くの場合 ILE CL 呼び出し元の MONMSG で発生しますが、CEEHDLR を使用して使用可能になっている ILE 条件処理プログラムを使用する ILE RPG を含めて、通知メッセージと状況メッセージを監視できる、他の ILE 言語でも生じる可能性があります。

この問題が検出される場合、これを避けるには次の 2 つの方法があります。

1. 呼び出し元が ILE RPG プロシージャーとは異なる活動化グループにいるようにする。
2. RPG プロシージャーで ILE 条件処理プログラムを使用可能にする。処理プログラムでは、無視したいメッセージであれば、このメッセージを処理するように指定してください。そうでない場合には、パーコレートを指定します。

   この処理プログラムの汎用性を高めて、重大度 0 (通知) と重大度 1 (警告) のメッセージをすべて無視するようにすることもできます。

   図 138 は、CPF4906 を無視する ILE 条件処理プログラムの例を示します。

```
 *---------------------------------------------------------------
 * 処理プログラム定義
 *---------------------------------------------------------------
D Action          S             10I 0
D Token           DS
D   MsgSev                       5I 0
D   MsgNo                        2A
D                                1A
D   Prefix                       3A
D                                4A
 *---------------------------------------------------------------
 * 処置
 *---------------------------------------------------------------
D Handle          C                   10
D Percolate       C                   20
 *---------------------------------------------------------------
 * 重大度
 *---------------------------------------------------------------
D Info            C                   0
D Warning         C                   1
D Error           C                   2
D Severe          C                   3
D Critical        C                   4
C     *ENTRY        PLIST
C                   PARM                    Token
C                   PARM                    dummy             1
C                   PARM                    Action
 *---------------------------------------------------------------
 * これが CPF4906 であれば通知メッセージを処理、それ以外は
 * パーコレート
 *---------------------------------------------------------------
C                   IF        Prefix = 'CPF' AND
C                             MsgNo = X'4906'
C                   EVAL      Action = Handle
C                   ELSE
C                   EVAL      Action = Percolate
C                   ENDIF
C                   RETURN
```

図 138. CPF4906 を無視する ILE 条件処理プログラム

図 139 は、すべての状況および通知メッセージを無視する場合に演算をコーディングする方法を示します。エスケープ・メッセージおよびファンクション・チェックは重大度 2 以上になります。

```
     *----------------------------------------------------------------
     * 情報メッセージまたは警告メッセージを処理、それ以外は
     * パーコレート
     *----------------------------------------------------------------
    C                   IF        MsgSev <= Warning
    C                   EVAL      Action = Handle
    C                   ELSE
    C                   EVAL      Action = Percolate
    C                   ENDIF
    C                   RETURN
```

図 139. 状況および通知メッセージを無視する方法

第 14 章 ダンプの入手

この章では、ILE RPG 定様式ダンプの入手方法について説明するとともに、定様式ダンプのサンプルを示しています。

ILE RPG 定様式ダンプの入手

プロシージャーの実行中にプロシージャーの ILE RPG 定様式ダンプ (記憶域の印刷出力) を作成するためには、次のことを行うことができます。

- 演算仕様書に 1 つまたは複数の DUMP 命令コードをコーディングする。
- 実行時メッセージに D または F オプションで応答する。また、自動応答でもダンプを使用可能にすることができます。資料「*CL プログラミング*」の『システム応答リスト』の説明を参照してください。

定様式ダンプには、フィールドの内容、データ構造の内容、配列およびテーブルの内容、ファイル情報データ構造、およびプログラム状況データ構造が含まれます。ダンプは QPPGMDMP という名前のファイルに書き出されます (システムの異常時ダンプはファイル QPSRVDMP に書き出されます)。

ILE RPG 実行時メッセージに F オプションで応答すると、そのダンプにはオープン・データ・パス (ODP、データ管理制御ブロック) の 16 進表示も含まれます。

ダンプ情報には、モジュールと関連したグローバル・データが含まれます。メイン・プロシージャーが活動状態であるかどうかによって、グローバル・データは、*INZSR の処理中に割り当てられた値を表さない場合があります。プログラムが複数のプロシージャーから成っている場合には、定様式ダンプ中の情報はダンプ要求時に活動状態であったすべての プロシージャーについての情報も反映します。プロシージャーが活動状態になっていない場合には、自動記憶域の変数の値は有効ではありません。プロシージャーがまだ呼び出されていない場合には、静的記憶域はまだ初期設定されません。プロシージャーが反復して呼び出される場合には、最新呼び出しの情報だけが表示されます。

ダンプ・データが使用できない場合、次の 2 つの原因があります。

- プログラム・オブジェクトが、デバッグ・ビュー *NONE を指定して作成されている場合。ダンプに含まれるのは、PSDS (プログラム状況データ構造)、ファイル情報、および *IN 標識だけになります。
- 単一の変数または構造が 16 MB を超えるダンプ・データを必要とする場合。これは一般に、5 MB より大きい変数または構造の場合に発生します。

| 定様式ダンプ内の、プログラムの変数の値をユーザーから隠蔽するには、以下のいずれかを行います。

- プログラム識別情報を除去することにより、プログラム内のデバッグ・データをなくします。
- ユーザーに対して、プログラムは実行できるが定様式ダンプは実行できない権限を付与します。 OPM RPG プログラムでは、これは *OBJOPR 権限と

*EXECUTE 権限を付与することで実現できます。 ILE RPG プログラムでは、これは *USE 権限を付与することで実現できます。

DUMP 命令コードの使用

ソースの演算に 1 つ以上の DUMP 命令コードをコーディングして、ILE RPG の定様式ダンプを作成することができます。DUMP 命令コードが実行されると常に、新しい QPPGMDMP スプール・ファイルが作成されます。

DUMP 命令について、次のことに注意してください。

- DUMP 命令が定様式ダンプを作成するかどうかを判断するには、DUMP 命令の命令拡張と、制御仕様書の DEBUG キーワードをチェックする必要があります。定様式ダンプは、DUMP 命令で (A) 拡張が指定されているか、または DEBUG キーワードがパラメーターなしでまたはパラメーター *YES で指定された場合に、生成されます。 DUMP 命令に (A) 拡張が指定されておらず、かつ DEBUG キーワードが指定されなかったか、または (*YES) パラメーターが指定された場合には、DUMP 命令のエラーが検査され、ステートメントがリストに印刷されますが、DUMP 命令は処理されません。
- DUMP 命令が条件付けられている場合には、その条件が満たされて始めて DUMP 命令が実行されます。
- DUMP 命令が GOTO 命令によって迂回される場合には、DUMP 命令は実行されません。

定様式ダンプの例

次の図は DBGEX に類似のモジュールの定様式ダンプの例を示します (267 ページの『デバッグ用サンプル・ソースの例』を参照)。定様式ダンプでデータ・バッファーの扱われ方を示すために、出力ファイル QSYSPRT を追加しています。

この例のダンプは、全定様式ダンプです。すなわち、照会メッセージが 'F' で回答された時に作成されます。

プログラム状況情報

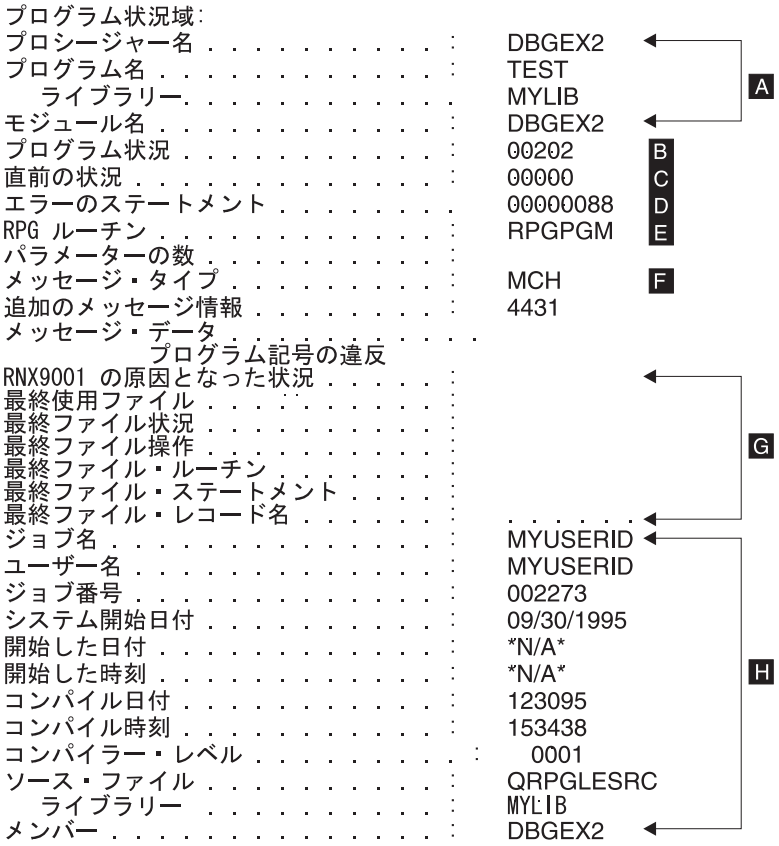


図 140. 定様式ダンプのプログラム状況情報セクション

A プロシージャー識別: プロシージャー名、プログラム名とライブラリー名、およびモジュール名。

B 現行状況コード。

C 前の状況コード。

D エラーのある ILE RPG ソース・ステートメント。

E 例外またはエラーが起こった ILE RPG ルーチン。

F マシン例外の CPF または MCH。

G 例外またはエラーが起こる前にプログラムで最後に使用されたファイルについての情報。この場合には、ファイルは使用されていません。

H プログラム情報。'*N/A*' は、プログラム中で情報が使用可能でないフィールドを示します。これらのフィールドは、PSDS に含まれている場合にのみ更新されます。

フィードバック域

```
 INFDS ファイルのフィードバック  I
ファイル . . . . . :  QSYSPRT
ファイル・オープン . . . . . :  YES
EOF のファイル . . . . . :  NO
ファイル状況 . . . . . :  00000
ファイル命令 . . . . . :  OPEN I
ファイル・ルーチン . . . . . :  *INIT
ステートメント番号 . . . . . :  *INIT
レコード名  . . . . . :
メッセージ識別子 . . . . . :
オープン・フィードバック  J
ODP タイプ . . . . . :  SP
ファイル名 . . . . . :  QSYSPRT
  ライブラリー. . . . . :  QSYS
メンバー . . . . . :  Q501383525                .
スプール・ファイル . . . . . :  Q04079N002
  ライブラリー. . . . . :  QSPL
スプール・ファイル番号 . . . . . :  7
1 次レコード長 . . . . . :  80
入力ブロック長 . . . . . :  0
出力ブロック長 . . . . . :  80
装置クラス . . . . . :  PRINTER
ページ当たり行数 . . . . . :  66
行当たり桁数 . . . . . :  132
重複キー使用可能 . . . . . :  *N/A*
転送するレコード数 . . . . . :  1
オーバーフロー行 . . . . . :  60
ブロック・レコードの増分 . . . . . :  0
ファイル共用可能 . . . . . :  NO
DDS で作成された装置ファイル . . . . . :  NO
IGC または図形可能ファイル . . . . . :  NO
ファイル・オープン・カウント . . . . . :  1
独立標識域 . . . . . :  NO
ユーザー・バッファー . . . . . :  NO
オープン識別コード . . . . . :  Q04079N002
最大レコード長 . . . . . :  0
ジョブに範囲指定された ODP . . . . . :  NO
最大プログラム装置 . . . . . :  1
定義された現行プログラム装置 . . . . . :  1
装置名 . . . . . :  *N
装置記述名 . . . . . :  *N
装置クラス . . . . . :  '02'X
装置タイプ . . . . . :  '08'X

共通入出力フィードバック  K
PUT の数 . . . . . :  0
GET の数 . . . . . :  0
PUT/GET の数 . . . . . :  0
その他の入出力の数 . . . . . :  0
現行の操作 . . . . . :  '00'X
レコード形式 . . . . . :
装置クラスおよびタイプ . . . . . :  '0208'X
装置名 . . . . . :  *N
最終レコードの長さ . . . . . :  80
検索されたレコード数 . . . . . :  80
最後の入出力レコードの長さ . . . . . :  0
現行ブロック・カウント . . . . . :  0

印刷装置のフィードバック:
現在行番号 . . . . . :  1
現行ページ . . . . . :  1
メジャー戻りコード . . . . . :  00
マイナー戻りコード . . . . . :  00

出力バッファー:
   0000   00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000     *                       *
   0020   00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000     *                       *
   0040   00000000  00000000  00000000  00000000                                             *                       *
```

図 141. 定様式ダンプのフィードバック域セクション

I これは INFDS のファイル・フィードバック・セクションです。ファイル・タイプに適用可能なフィールドだけが印刷されます。INFDS フィードバック・セクションの残りは、プログラムで宣言されている場合にだけ更新されるので、ダンプされません。

J これはファイルのファイル・オープン・フィードバック情報です。フィールドの詳細については、Web サイト http://www.ibm.com/eserver/iseries/infocenter で **iSeries 400 Information Center** の中の『データベースおよびファイル・システム』カテゴリーの『*DB2 UDB for iSeries*』の節を参照してください。

K これは、ファイルの共通入出力フィードバック情報です。フィールドの説明は、上記の Web サイトを参照してください。

全定様式ダンプによる情報

```
 オープン・データ・パス:
   0000   64800000  00001AF0  00001B00  000000B0  00000140  000001C6  00000280  000002C0      *       0              F       *
   0020   00000530  00000000  00000000  00000380  00000000  06000000  00000000  00000000      *                              *
   0040   00008000  00000000  003AC02B  A00119FF  000006C0  00003033  00000000  00000000      *                              *
   0060   80000000  00000000  003AC005  CF001CB0  00000000  00000000  00000000  00000000      *                              *
   0080   80000000  00000000  003AA024  D0060120  01900000  00010000  00000050  00000000      *                           &    *
   00A0   1F000000  00000000  00000000  00000000  E2D7D8E2  E8E2D7D9  E3404040  D8E2E8E2      *                SPQSYSPRT   QSYS*
   00C0   40404040  4040D8F0  F4F0F7F9  D5F0F0F2                                              *      Q04079N002QSPL         &  *
 オープン・フィードバック:
   0000   E2D7D8E2  E8E2D7D9  E3404040  D8E2E8E2  40404040  4040D8F0  F4F0F7F9  D5F0F0F2      *SPQSYSPRT   QSYS      Q04079N002*
   0020   D8E2D7D3  40404040  40400007  00500000  D8F5F0F1  F3F8F3F5  F2F50000  00000000      *QSPL         &  Q501383525      *
   0040   00500002  00000000  42008400  00000000  0000D5A4  00100000  00000008  00000000      * &        d       Nu            *
   0060   00000000  00000000  00000100  3C000000  0005E000  5CD54040  40404040  40400001      *                     *N         *
   0080   00000000  00001300  00000000  00000000  00010001  5CD54040  40404040  40400000      *                     *N         *
   00A0   07100000  00000000  00450045  00450045  07A10045  00450045  00700045  00450045      *                                *
   00C0   00450045  00450045  002F0030  00040005  5CD54040  40404040  40400208  00000000      *                *N              *
   00E0   20000000  00000000  00000000  00000000  00000000  00000001  C2200000  00059A00      *                         B      *
   0100   00000000  00000000  00000000  00000000  00000000  4040                              *                                *
 共通入出力フィードバック:
   0000   00900000  00000000  00000000  00000000  00000000  00000000  00000000  00000208      *                                *
   0020   5CD54040  40404040  40400000  00500000  00000000  00000000  00000000  00000000      **N           &                  *
   0040   00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000      *                                *
   0060   00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000      *                                *
   0080   00000000  00000000  00000000  00000000                                              *                                *
 装置の入出力フィードバック:
   0000   00010000  00010000  00000000  00000000  00000000  00000000  00000000  00000000      *                                *
   0020   0000F0F0  0001                                                                      *  0000                          *
```

図 142. 全定様式ダンプに提供される情報

ILE RPG 照会メッセージに F オプションで応答した場合には、このファイルに関連した共通オープン・データ・パスおよびフィードバック域がダンプに含まれます。

データ情報

```
 ILE RPG/400 定様式ダンプ
モジュール名 . . . . . . . . . . . . . :   DBGEX2
最適化レベル . . . . . . . . . . . . . :   *NONE          L  M
停止標識:
H1 '0'   H2 '0'   H3 '0'   H4 '0'   H5 '0'   H6 '0'   H7 '0'   H8 '0'   H9 '0'
コマンド/機能キーの標識:
KA '0'    KB '0'    KC '0'    KD '0'    KE '0'    KF '0'    KG '0'    KH '0'    KI '0'    KJ '0'
KK '0'    KL '0'    KM '0'    KN '0'    KP '0'    KQ '0'    KR '0'    KS '0'    KT '0'    KU '0'
KV '0'    KW '0'    KX '0'    KY '0'
制御レベル標識:
L1 '0'    L2 '0'    L3 '0'    L4 '0'    L5 '0'    L6 '0'    L7 '0'    L8 '0'    L9 '0'
オーバーフロー標識:
OA '0'    OB '0'    OC '0'    OD '0'    OE '0'    OF '0'    OG '0'    OV '0'
外部標識:
U1 '0'    U2 '0'    U3 '0'    U4 '0'    U5 '0'    U6 '0'    U7 '0'    U8 '0'
一般標識:
01 '0'    02 '1'    03 '0'    04 '1'    05 '0'    06 '1'    07 '0'    08 '0'    09 '0'    10 '0'
11 '0'    12 '0'    13 '0'    14 '0'    15 '0'    16 '0'    17 '0'    18 '0'    19 '0'    20 '0'
21 '0'    22 '0'    23 '0'    24 '0'    25 '0'    26 '0'    27 '0'    28 '0'    29 '0'    30 '0'
31 '0'    32 '0'    33 '0'    34 '0'    35 '0'    36 '0'    37 '0'    38 '0'    39 '0'    40 '0'
41 '0'    42 '0'    43 '0'    44 '0'    45 '0'    46 '0'    47 '0'    48 '0'    49 '0'    50 '0'
51 '0'    52 '0'    53 '0'    54 '0'    55 '0'    56 '0'    57 '0'    58 '0'    59 '0'    60 '0'
61 '0'    62 '0'    63 '0'    64 '0'    65 '0'    66 '0'    67 '0'    68 '0'    69 '0'    70 '0'
71 '0'    72 '0'    73 '0'    74 '0'    75 '0'    76 '0'    77 '0'    78 '0'    79 '0'    80 '0'
81 '0'    82 '0'    83 '0'    84 '0'    85 '0'    86 '0'    87 '0'    88 '0'    89 '0'    90 '0'
91 '0'    92 '0'    93 '0'    94 '0'    95 '0'    96 '0'    97 '0'    98 '0'    99 '0'
内部標識:
LR '0'    MR '0'    RT '0'    1P '0'
 N
NAME                 ATTRIBUTES          VALUE
_QRNU_DSI_DS1        INT(10)             1                '00000001'X       O
_QRNU_DSI_DS2        INT(10)             2                '00000002'X
_QRNU_NULL_ARR       CHAR(1)             DIM(8)      P
                       (1-2)             '1'              'F1'X
                       (3)               '0'              'F0'X
                       (4)               '1'              'F1'X
                       (5-6)             '0'              'F0'X
                       (7)               '1'              'F1'X
                       (8)               '0'              'F0'X
_QRNU_NULL_FLDNULL   CHAR(1)             '1'              'F1'X
_QRNU_TABI_TABLEA    INT(10)             1                '00000001'X       Q
ARR                  CHAR(2)             DIM(8)
                       (1-3)             'AB'             'C1C2'X
                       (4-7)             '  '             '4040'X
                       (8)               '1'              'F1'X
ARRY                 ZONED(3,2)          DIM(2)
                       (1-2)             1.24             'F1F2F4'X
BASEONNULL           CHAR(10)            NOT ADDRESSABLE
BASEPTR              POINTER             SPP:E30095A62F001208
BASESTRING           CHAR(6)             'ABCDEF'         'C1C2C3C4C5C6'X
BIGDATE              DATE(10)            '1994-09-30'     'F1F9F9F460F0F960F3F0'X
BIGTIME              TIME(8)             '12.00.00'       'F1F24BF0F04BF0F0'X
BIGTSTAMP            TIMESTAMP(26)       '9999-12-31-12.00.00.000000'
                     VALUE IN HEX        'F9F9F9F960F1F260F3F160F1F24BF0F04BF0F04BF0F0F0F0F0F0'X
BIN4D3               BIN(4,3)            -4.321           'EF1F'X
BIN9D7               BIN(9,7)            98.7654321       '3ADE68B1'X
DBCSSTRING           GRAPHIC(3)          ' BBCCDD '       'C2C2C3C3C4C4'X
```

図 143. 定様式ダンプのデータ・セクション (1/2)

```
DS1                 DS                  OCCURS(3)   R
 OCCURRENCE(1)
   FLD1             CHAR(5)             '1BCDE'           'F1C2C3C4C5'X
   FLD1A            CHAR(1)             DIM(5)
                      (1)               '1'               'F1'X
                      (2)               'B'               'C2'X
                      (3)               'C'               'C3'X
                      (4)               'D'               'C4'X
                      (5)               'E'               'C5'X
   FLD2             BIN(5,2)            123.45            '00003039'X
 OCCURRENCE(2)
   FLD1             CHAR(5)             'ABCDE'           'C1C2C3C4C5'X
   FLD1A            CHAR(1)             DIM(5)
                      (1)               'A'               'C1'X
                      (2)               'B'               'C2'X
                      (3)               'C'               'C3'X
                      (4)               'D'               'C4'X
                      (5)               'E'               'C5'X
   FLD2             BIN(5,2)            123.45            '00003039'X
 OCCURRENCE(3)
   FLD1             CHAR(5)             'ABCDE'           'C1C2C3C4C5'X
   FLD1A            CHAR(1)             DIM(5)
                      (1)               'A'               'C1'X
                      (2)               'B'               'C2'X
                      (3)               'C'               'C3'X
                      (4)               'D'               'C4'X
                      (5)               'E'               'C5'X
   FLD2             BIN(5,2)            123.45            '00003039'X
DS2                 CHAR(10)            DIM(2)     S
                      (1)               'aaaaaaaaaa'      '81818181818181818181'X
                      (2)               'bbbbbbbbbb'      '82828282828282828282'X
DS3                 DS                          T
  FIRSTNAME         CHAR(10)            'Fred      '      'C6998584404040404040'X
  LASTNAME          CHAR(10)            'Jones     '      'D1969585A24040404040'X
  TITLE             CHAR(5)             'Mr.  '           'D4994B4040'X
EXPORTFLD           CHAR(6)             'export'          '85A7979699A3'X
FLDNULL             ZONED(3,1)          24.3              'F2F4F3'X
FLOAT1              FLT(4)              1.234500000000E+007   U
                    VALUE IN HEX        '4B3C5EA8'X
FLOAT2              FLT(8)              3.962745000000E+047
                    VALUE IN HEX        '49D15A640A93FCFF'X
INT10               INT(10)             -31904            'FFFF8360'X
INT5                INT(5)              -2046             'F802'X
NEG_INF             FLT(8)              -HUGE_VAL  V
                    VALUE IN HEX        'FFF0000000000000'X
NOT_NUM             FLT(4)              *NaN       W
                    VALUE IN HEX        '7FFFFFFF'X
NULLPTR             POINTER             SYP:*NULL
PACKED1D0           PACKED(5,2)         -093.40           '09340D'X
PARM1               PACKED(4,3)         6.666             '06666F'X
POS_INF             FLT(8)              HUGE_VAL   X
                    VALUE IN HEX        '7FF0000000000000'X
PROCPTR             POINTER             PRP:A00CA02EC200   Y
SPCPTR              POINTER             SPP:A026FA0100C0
SPCSIZ              BIN(9,0)            000000008.        '00000008'X
STRING              CHAR(6)             'ABCDEF'          'C1C2C3C4C5C6'X
TABLEA              CHAR(3)             DIM(3)
                      (1)               'aaa'             '818181'X
                      (2)               'bbb'             '828282'X
                      (3)               'ccc'             '838383'X
UNSIGNED10          UNS(10)             31904             '00007CA0'X
UNSIGNED5           UNS(5)              2046              '07FE'X
ZONEDD3D2           ZONED(3,2)          -3.21             'F3F2D1'X
Local variables for subprocedure SWITCH:  Z
NAME                ATTRIBUTES          VALUE
_QRNL_PSTR_PARM     POINTER             SYP:*NULL
LOCAL               CHAR(5)             '    '            '0000000000'X
PARM                CHAR(1)             NOT ADDRESSABLE
          * * * * *   R P G   ダンプの終わり   * * * * *
```

図 143. 定様式ダンプのデータ・セクション (2/2)

L 最適化レベル。

M 一般標識 1 ～ 99 およびその現在の状況 ('1' はオン、'0' はオフ)。標識 *IN02、*IN04、および *IN06 はまだ設定されていないということに注意してください。

N アルファベット順にリストされ、プロシージャー別にグループ化された、ユ

ーザー変数の始まり部分。サブプロシージャーに対してローカルなデータは自動記憶域に保管され、サブプロシージャーが活動状態でない限り使用可能になりません。すべての変数の 16 進数値が表示されることに注意してください。131 桁を超える長さの名前はダンプ・リストで、複数行に分割して表示されます。名前全体が、行の末尾に文字 '...' を付けて印刷されます。名前の最後の部分が 21 文字を超える場合は、属性と値のリストは、次の行から開始します。

O 複数回繰り返しデータ構造指標を含む、内部定義フィールド。

P null 可能フィールドに空標識を含む、内部定義フィールド。

Q テーブルの指標を含む内部定義フィールド。

R 複数回繰り返しデータ構造。

S サブフィールドのないデータ構造は文字ストリングとして表示されます。

T データ構造サブフィールドは、*定義される順序ではなく、アルファベット順に* リストされます。サブフィールド定義中のギャップは示されません。

U 4 バイトおよび 8 バイトの浮動フィールド。

V 負の無限大を示します。

W 値が有効な浮動小数点数ではないことを示す「非数値」を意味します。

X 正の無限大を示します。

Y 属性は、基底ポインターとプロシージャー・ポインターを区別しません。

Z サブプロシージャー内の内部データはメイン・ソース・セクションとは別個にリストされます。

第 4 部 ファイルおよび装置の処理

この部では、ILE RPG プログラムのファイルと装置の使い方を説明します。特に、次のことを行う方法を示します。

- ファイルを装置に関連付ける
- ファイルを定義する (プログラム記述または外部記述として)
- ファイルを処理する
- データベース・ファイルにアクセスする
- 外部接続装置にアクセスする
- 対話式アプリケーションを作成する

注: 用語「RPG IV プログラム」とは、RPG IV で書かれた 1 つ以上のプロシージャーを含む ILE プログラムのことです。

第 15 章 ファイルの定義

ファイルはプログラムと入出力に使用される装置の間の接続リンクとして機能します。システム上の各ファイルは、ファイルの特性を記述し、ファイルに関連したデータが、どのようにレコードおよびフィールドに編成されているかを記述するファイル記述をもっています。

プログラムが任意の入出力操作を実行するためには、プログラムが参照するファイル記述、使用される入出力装置のタイプ、およびデータの編成方法を指定しなければなりません。この章では以下の概念を示します。

- ファイル記述と入出力装置との関連付け
- 外部記述ファイルの定義
- プログラム記述ファイルの定義
- データ管理命令

異なる装置タイプでの外部記述ファイルおよびプログラム記述ファイルの使用法についての説明は、後続の章にあります。

ファイル記述と入出力装置との関連付け

iSeries 上のすべての入出力操作の主要な要素はファイルです。システムは次のファイル・タイプをサポートしています。

データベース・ファイル
システム上でデータを永続的に保管できるようにします。

装置ファイル
外部に接続する装置にアクセスすることができます。表示装置ファイル、PRINTER ファイル、テープ・ファイル、ディスケット・ファイル、および ICF ファイルが含まれます。

保管ファイル
ディスク上で保管データを保管するために使用されます。

DDM ファイル
遠隔システム上に保管されているデータ・ファイルをアクセスできるようにします。

各入出力装置は、上記のタイプの 1 つの対応するファイル記述をもっており、プログラムはそのファイル記述を使用してその装置をアクセスします。実際の装置関連は、ファイルが処理された時に作成されます。データは、ファイルが処理用に使用された時に装置に対して読み取られたり書き出されたりします。

RPG によって、SPECIAL ファイルの使用を通して、システムが直接サポートしていないファイルおよび装置にアクセスすることもできます。SPECIAL ファイルでは、ファイルに対する名前およびファイル用のデータ管理の関連を処理するプログラムを指定しなければなりません。その他のタイプのファイルでは、これは RPG およびオペレーティング・システムが処理します。

プログラムが使用するファイル記述をオペレーティング・システムに指示するためには、使用する各ファイルごとにファイル仕様書の 7 ～ 16 桁目にファイル名 を指定します。36 ～ 42 桁目には RPG 装置名 を指定します。装置名は、関連付けられたファイルで使用することができる RPG 操作を定義します。装置名としては DISK、PRINTER、WORKSTN、SEQ、または SPECIAL のいずれかを使うことができます。図 144 は、画面 (WORKSTN) ファイル FILEX のファイル仕様書を示します。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++++++++++
FFILEX     CF   E             WORKSTN
```

図 144. RPG プログラムでの表示装置ファイルの識別

これは、装置名 (36～42 桁目に指定) ではなく、実際の装置の仕様書が入っている OS/400 ファイル記述を指すのはファイル名であるということに注意してください。

RPG 装置タイプは次のとおり上のファイル・タイプと対応しています。

表 31. RPG 装置タイプと iSeries のファイル・タイプとの相互関連

RPG 装置タイプ	iSeries ファイル・タイプ
DISK	データベース・ファイル、保管ファイル、DDM ファイル
PRINTER	プリンター・ファイル
WORKSTN	表示装置ファイル、ICF ファイル
SEQ	テープ、ディスケット、保管、プリンター、データベース
SPECIAL	N/A

図 145 は、図 144 にコーディングされている RPG ファイル名 FILEX と表示装置ファイルのシステム・ファイル記述との関連を図示したものです。

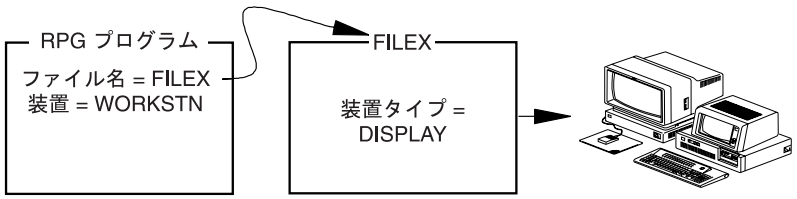


図 145. ファイル名と表示装置ファイル記述との関連付け

コンパイル時に、一定の RPG 操作は特定の RPG 装置名だけに有効です。この点では、RPG 操作は装置従属です。装置従属性の 1 つの例は、EXFMT 命令コードは WORKSTN 装置の場合にのみ有効です。

その他の命令コードは装置とは独立しています。これは、その命令コードがどの装置タイプでも使用できることを意味します。例えば、WRITE は装置独立命令です。

SEQ 装置

装置 SEQ は独立装置タイプです。図 146 は、RPG ファイル名 FILEY と順次装置のシステム・ファイル記述の関連を示します。プログラムが実行される時には、実際の入出力装置が FILEY の記述に指定されます。例えば、装置が PRINTER となる場合もあります。

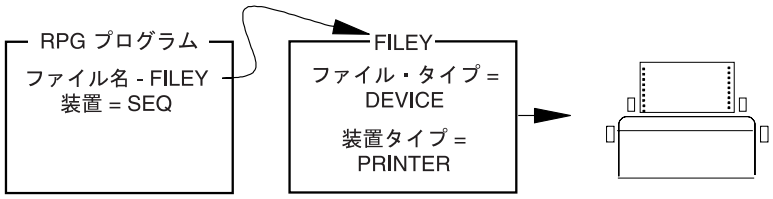


図 146. ファイル名と表示装置ファイル記述との関連付け

ファイル名およびファイル・タイプは RPG プログラムでコーディングされますが、多くの場合には、プログラムで使用されるファイルのタイプまたは入出力装置をプログラムを変更せずに変更することができます。この方法については、335 ページの『ファイル入出力の一時変更および指定変更』を参照してください。

ファイルの名前指定

iSeries システムでは、ファイルはメンバーから構成されています。これらのファイルはライブラリーに編成されます。ファイルの命名規則は、**ライブラリー名/ファイル名**です。

ILE RPG プログラムでは、ファイル名はファイル仕様書の 7〜16 桁目で識別されます。ファイル名は 1 〜 10 文字の長さとすることができ、固有としなければなりません。

プログラム内では、ファイル名をライブラリーで修飾しません。実行時にシステムは、ファイルを見つけるためにジョブと関連したライブラリー・リストを検索します。名前またはメンバーを変更するか、あるいは特定のライブラリーを指定したい場合には、ファイル一時変更コマンドを使用することができます。ファイル一時変更の詳細については、335 ページの『ファイル入出力の一時変更および指定変更』を参照してください。

ファイル記述のタイプ

プログラムが使用することになるファイル記述を識別する時には、ファイルがプログラム記述ファイルであるのか、それとも外部記述ファイルであるのかを示さなければなりません。

- **プログラム記述ファイル**の場合には、フィールドの記述は入力仕様書または出力仕様書 (あるいはその両方) で RPG ソース・メンバーの中にコーディングします。

  オペレーティング・システムに対するファイルの記述には、データがどこから取られるかについての情報、およびファイル中のレコードの長さについての情報が含まれます。

- **外部記述ファイル**の場合には、コンパイラーは DDS、IDDU、または SQL コマンドを使用して作成された外部ファイル記述からフィールドの記述を取り出します。したがって、入力仕様書または出力仕様書 (あるいはその両方) で RPG ソース・メンバー中にフィールド記述をコーディングする必要はありません。

  外部記述には、データベースや特定の装置などデータが取られる場所についての情報、および各フィールドの記述とその属性についての情報が含まれます。プログラムをコンパイルする前に、ファイルが存在し、ライブラリー・リストからアクセス可能でなければなりません。

外部記述ファイルには次の利点があります。

- プログラムのコーディングが少なくなります。同じファイルが多くのプログラムで使用される場合には、フィールドをオペレーティング・システムに対して 1 回定義して、そのすべてのプログラムで使用することができます。この実行により、外部記述ファイルを使用する RPG プログラムの入力仕様書または出力仕様書のコーディングの必要性が減少します。
- ファイルのレコード様式が変更された時の保守の作業が少なくなります。ファイルのレコード様式を変更し、そのファイルを使用しているプログラムをその後でコンパイルし直すことによって、そのプログラム内のコーディングを変更せずに、プログラムを何回も更新することができます。
- 同じファイルを使用しているプログラムでは、レコード様式およびフィールド名を共通して使用しているので、文書化が改善されます。
- 信頼性が改善されます。レベル検査が指定された場合には、RPG プログラムは外部記述に変更があったかどうかをユーザーに通知します。詳細については、331 ページの『レベル検査』を参照してください。

装置 SEQ または SPECIAL に外部記述ファイル (ファイル仕様書の 22 桁目の E で識別) が指定された場合には、RPG プログラムはファイルのフィールド記述を使用しますが、オペレーティング・システムとのインターフェースは、ファイルがプログラム記述ファイルであるかのようになります。 PRINTER ファイルの用紙制御などの装置従属機能は、この情報が既に外部記述で定義されているので、外部記述ファイルで指定することはできません。

外部記述のファイルのプログラム記述としての使用

外部記述から作成されるファイルは、プログラム中でプログラム記述ファイルとして使用することができます。外部記述ファイルをプログラム記述ファイルとして使用するためには、次のようにします。

1. プログラムのファイル仕様書にファイルをプログラム記述 (22 桁目の F) として指定してください。
2. プログラムの入力仕様書または出力仕様書 (あるいはその両方) にレコード中のフィールドを記述してください。

コンパイル時は、コンパイラーは入力仕様書または出力仕様書 (あるいはその両方) 中のフィールド記述を使用します。外部記述を検索しません。

プログラムとファイルとの代表的な関係の例

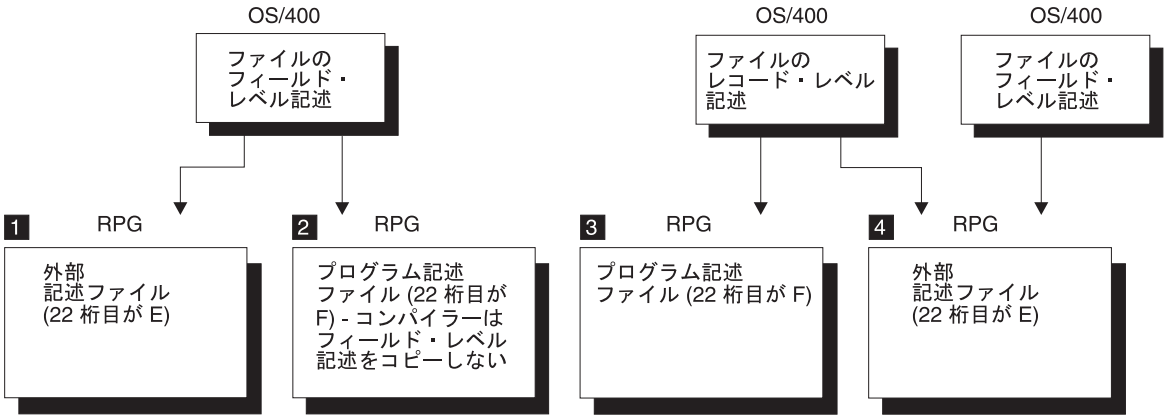


図 147. iSeries システムでの RPG プログラムとファイルの代表的な関係

1 プログラムは、オペレーティング・システムに定義されているファイルのフィールド・レベル記述を使用します。外部記述ファイルは、ファイル仕様書の 22 桁目の E によって識別されます。コンパイル時に、外部フィールド・レベル記述がコンパイラーによってコピーされます。

2 外部記述ファイル (すなわち、フィールド・レベル外部記述のあるファイル) は、プログラム内でプログラム記述ファイルとして使用されます。プログラム記述ファイルは、ファイル仕様書の 22 桁目の F によって識別されます。この項目によって、外部フィールド・レベル記述にコピーしないようにコンパイラーに通知します。このファイルはコンパイル時に存在する必要はありません。

3 ファイルは、オペレーティング・システムに対して、レコード・レベルでのみ記述されます。レコードのフィールドはプログラムの中で記述されます。したがって、ファイル仕様書の 22 桁目は F を含まなければなりません。このファイルはコンパイル時に存在する必要はありません。

4 ファイル名はコンパイル時に指定 (すなわち、RPG ソース・メンバー内でコーディング) することができ、別のファイル名を実行時に指定することができます。ファイル仕様書の 22 桁目の E は、ファイルの外部記述がコンパイル時にコピーされることを指示します。実行時には、プログラムによって別のファイルをアクセスするように、ファイル一時変更コマンドを使用することができます。実行時にファイルを一時変更するためには、両方のファイルのレコード名が間違いなく同じでなければなりません。RPG プログラムは READ 命令などの入出力命令でレコード様式名を使用しますが、その命令ではこのレコード様式名が必要なレコード・タイプを指定します。詳細については、335 ページの『ファイル入出力の一時変更および指定変更』を参照してください。

外部記述ファイルの定義

DDS を使用して OS/400 システムに対してファイルを記述することができます。ファイル中の各レコード・タイプは、固有のレコード様式名によって識別されます。

ファイル仕様書の 22 桁目の E の指定は、外部記述ファイルであることを指示します。E の指定は、プログラムがコンパイルされる時に、ファイルの外部記述がシステムから検索されることをコンパイラーに指示します。

この外部記述の情報には次のものがあります。

- ファイル・タイプなどのファイル情報、アクセス方法 (キーまたは相対レコード番号による) などのファイル属性
- レコード様式名およびフィールド記述 (名前、位置、および属性) を含むレコード様式記述

コンパイラーが外部記述から検索する情報は、ソース・メンバーのコンパイル時に CRTRPGMOD または CRTBNDRPG コマンドに OPTION(*EXPDDS) が指定されている限り、コンパイラー・リストに印刷されます (これらの両方のコマンドのデフォルト値は OPTION(*EXPDDS) です)。

次の項では、ファイル仕様書を使用してレコード様式を名前変更または無視する方法、および入力仕様書と出力仕様書を使用して外部記述を変更する方法について説明します。外部記述ファイルの場合には入力および出力仕様書は任意指定であることを忘れないでください。

レコード様式の名前変更

外部記述ファイルに指定できる多くの機能 (CHAIN 命令など) は、ファイル名またはレコード様式名のいずれかに作用します。したがって、プログラム中のファイル名およびレコード様式名はそれぞれ固有の記号名でなければなりません。

レコード様式名を変更するためには、図 148 に示すように外部記述ファイルのファイル仕様書に RENAME キーワードを使用してください。形式は RENAME(*古い名前*:*新しい名前*) です。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++++++++++
FITMMSTL   IP   E           K DISK    RENAME(ITEMFORMAT:MSTITM)
 *
```

図 148. 外部記述ファイル中のレコード様式名に対する RENAME キーワード

RENAME キーワードは一般的に、プログラムに同じレコード様式名をもつ 2 つのファイルが含まれている場合に、使用されます。図 148 では、外部記述ファイル ITMMSTL のレコード様式 ITEMFORMAT がこのプログラムで使用するために MSTITM に名前が変更されています。

フィールド名の変更

ファイルのファイル仕様書で PREFIX キーワードを使用して、外部記述ファイルのすべてのフィールドの名前を変更することができます。既存のフィールド名に接頭部を追加するか、あるいは一連の文字で既存のフィールド名の一部を置き換えることができます。形式は PREFIX(*接頭部*: {*置き換える文字数*}) です。 327 ページの図 149 には PREFIX の使用例が示されています。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++++++++++
 * レコード様式の各名前に接頭語 MST を追加します。
FITMMSTL   IP   E           K DISK    PREFIX(MST)
 *
 * レコード様式の各名前の接頭語 YTD を YE に変更します。
FSALESMSTR IP   E           K DISK    PREFIX(YE:3)
```

図 149. 外部記述ファイルのレコード様式名に対する PREFIX キーワード

レコード様式の無視

外部記述ファイルのレコード様式がプログラムで使用されない場合には、IGNORE キーワードを使用して、レコード様式がファイル中に存在していないかのようにプログラムを実行させることができます。論理ファイルの場合には、これは、その様式と関連したすべてのデータがプログラムに対してアクセス困難であることを意味します。 328 ページの図 150 に示されているとおり、外部記述ファイルのファイル仕様書に IGNORE キーワードを使用してください。

ファイルは複数のレコード様式をもっていなければなりませんが、そのすべてを無視することはできません。少なくとも 1 つのレコード様式は残っていなければなりません。この要件を除けば、1 ファイルについてレコード様式はいくつでも無視できます。

レコード様式が無視されると、その様式は別のキーワード (SFILE、RENAME、または INCLUDE) または別の IGNORE に指定することはできません。

無視されたレコード様式名は相互参照表に印刷されますが、無視されたものとしてフラグが付けられます。

外部記述ファイルのレコード様式が無視されることを指示するためには、ファイル仕様書のキーワード・フィールドにキーワードとパラメーター IGNORE(*レコード様式名*) を指定してください。

代わりの方法として、INCLUDE キーワードを使用して、プログラムで使用されるレコード様式名だけを組み込むことができます。ファイルに含まれている他のすべてのレコード様式は除外されます。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++++++++++
 *
 *  ファイル ITMMSTL は、次のレコード様式を含んでいると想定します
 *  (EMPLNO、NAME、ADDR、TEL、WAGE)。EMPLNO および NAME レコードだけが
 *  存在するかのようにプログラムを実行させるには、以下の 2 つの方式の
 *  いずれかを使用することができます。
 *
FITMMSTL   UF   E           K DISK    IGNORE(ADDR:TEL:WAGE)
 *
 *  あるいは :
 *
FITMMSTL   UF   E           K DISK    INCLUDE(EMPLNO:NAME)
 *
```

図 150. 外部記述ファイル中のレコード様式に対する IGNORE キーワード

入力仕様書を使用した外部記述の変更

入力仕様書を使用して、入力ファイルの外部記述中の一定の情報を一時変更するか、あるいは外部記述に RPG 機能を追加することができます。入力仕様書では次のことを行うことができます。

- 329 ページの図 151 に示されているようにレコード様式にレコード識別標識を割り当てる。
- 329 ページの図 151 に示されているようにフィールド名を変更する。
- 329 ページの図 151 に示されているようにフィールドに制御レベル標識を割り当てる。
- 329 ページの図 152 に示されているようにレコードの突き合わせ処理のために突き合わせフィールド値をフィールドに割り当てる。
- 329 ページの図 152 に示されているようにフィールド標識を割り当てる。

入力仕様書を使用して外部記述ファイル中のフィールドの位置を一時変更することはできません。外部記述ファイル中のフィールドは、データ記述仕様にリストされた順序でレコードに入れられます。また、外部記述ファイルの場合には、用紙制御などの装置従属機能は、RPG プログラムでは有効ではありません。

注: PREFIX キーワードがファイルに指定されている場合でも、入力仕様書で明示的にフィールド名を変更することができます。コンパイラーは、ユーザーのプログラムで最初に*使用される* 名前を認識 (および必要と) します。例えば、フィールドを標識と関連付けるために入力仕様書に接頭部付きの名前を指定し、接頭部付きでない名前を参照するフィールドの名前を変更しようとした場合には、エラーになります。逆に、最初にフィールドを接頭部付きの名前以外のものに名前変更し、仕様書に接頭部付きの名前を使用した場合には、エラーになります。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
IRcdname+++....In.....................................................*
IMSTRITEM      01 1
I..............Ext-field+..................Field+++++++++L1M1..PlMnZr......
I              ITEMNUMB  2                 ITEM          L1  3
 *
IMSTRWHSE      02
I              ITEMNUMB                    ITEM          L1
 *
```

図 151. 外部記述への RPG 機能の一時変更および追加

1 外部記述ファイル中のレコードにレコード識別標識を割り当てるためには、入力仕様書の 7 ～ 16 桁目にレコード様式名を指定し、21 ～ 22 桁目に有効なレコード識別標識を割り当ててください。外部記述ファイルでの入力仕様書の代表的な使用法は、レコード識別標識の割り当てです。

この例では、レコード識別標識 01 がレコード MSTRITEM に割り当てられ、標識 02 がレコード MSTRWHSE に割り当てられています。

2 外部記述レコード中のフィールドを名前変更するためには、フィールド記述行の 21 ～ 30 桁目にフィールドの外部名を左寄せで指定してください。49 ～ 62 桁目には、プログラムで使用される名前を指定してください。

この例では、両方のレコードのフィールド ITEMNUMB がこのプログラムで ITEM に名前変更されます。

3 外部記述レコード中のフィールドに制御レベル標識を割り当てるためには、49 ～ 62 桁目にフィールドの名前を指定し、63 ～ 64 桁目に制御レベル標識を指定してください。

この例では、レコード MSTRITEM と MSTRWHSE の両方でフィールド ITEM が L1 制御フィールドとなるように指定されています。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC.............................
IMSTREC        01 1
I..............Ext-field+..................Field+++++++++L1M1..PlMnZr......
I                                          CUSTNO        M1  1
 *
IWKREC       02
I                                          CUSTNO        M1
I                                          BALDUE              98  2
 *
```

図 152. 外部記述への RPG 機能の追加

1 外部記述レコード中のフィールドに突き合わせ値を割り当てるためには、レコード識別行の 7 ～ 16 桁目にレコード様式名を指定してください。フィールド記述行では、49 ～ 62 桁目にフィールドの名前を指定し、65 ～ 66 桁目に突き合わせレベル値を割り当ててください。

この例では、MSTREC と WKREC の両方のレコードの CUSTNO フィールドに突き合わせレベル値 M1 が割り当てられています。

2 外部記述レコード中のフィールドにフィールド標識を割り当てるためには、

レコード識別行の 7 ～ 16 桁目にレコード様式名を指定してください。フィールド記述行では、49 ～ 62 桁目にフィールドの名前を指定し、69 ～ 74 桁目に標識を指定してください。

この例では、レコード WKREC 中のフィールド BALDUE をプログラムで読み取った時点で、ゼロか否かテストします。フィールドの値がゼロの場合には、標識 98 がオンに設定されます。

出力仕様書の使用

外部記述ファイルの場合には、出力仕様書はオプションです。RPG は、外部記述ファイルの出力仕様書を必要とせずに外部レコード様式記述を使用して出力レコードを記述する、WRITE や UPDATE などのファイル命令コードをサポートします。

出力仕様書を使用して、データを書き出す時点を制御するか、あるいは書き出す選択フィールドを指定することができます。外部記述ファイルのフィールド記述行の有効な項目は、出力標識 (21 ～ 29 桁目)、フィールド名 (30 ～ 43 桁目)、および後で消去 (45 桁目) です。外部記述ファイルに書き出されるフィールドの編集語および編集コードは、ファイルの DDS で指定します。外部記述ファイルの場合には、ページ・オーバーフロー・ルーチン (18 桁目) またはスペース/スキップ (40 ～ 51 桁目) などの装置依存の従属機能は RPG プログラムでは有効ではありません。オーバーフロー標識は外部記述ファイルには正しくありません。DDS の中で編集を指定する方法の説明については、Web サイト http://www.ibm.com/eserver/iseries/infocenter で **iSeries 400 Information Center** の中の『データベースおよびファイル・システム』カテゴリーの『*DB2 UDB for iSeries*』の節を参照してください。

外部記述ファイルに出力仕様書を使用する場合には、7 ～ 16 桁目にファイル名の代わりにレコード様式名を指定します。

外部記述ファイル中のすべてのフィールドが出力レコードに入れられる場合には、フィールド記述行の 30 ～ 43 桁目に *ALL を指定してください。*ALL を指定した場合には、そのレコードに他のフィールド記述行を指定することはできません。

出力レコードに一定のフィールドだけを入れたい場合には、30 ～ 43 桁目にフィールド名を入力してください。これらの桁に指定するフィールド名は、入力仕様書でフィールドの名前が変更されない限り外部レコード記述で定義されたフィールド名でなければなりません。 331 ページの図 153 を参照してください。

外部記述ファイルの出力仕様書の使用に関する次の考慮事項に注意してください。

- 更新レコードの出力では、出力フィールド仕様に指定されたフィールド、および出力標識によって指定された条件に一致するフィールドだけが、再度書き出される出力レコードに入れられます。出力仕様書に指定されていないフィールドは、読み取られた値を使用して再書き出しされます。この手法は、すべてのフィールドを更新する UPDATE 操作コードとは対照的に、有効な制御の方法を提供します。
- 新しいレコードの作成では、出力フィールド仕様書に指定されたフィールドはレコードに入れられます。出力フィールド仕様書に指定されていないフィールド、または出力標識で指定された条件を満たしていないフィールドは、デフォルト値

として書き出されます。これは外部記述に指定されたデータ形式によって異なります (例えば、文字フィールドの場合はブランクで、数字フィールドの場合はゼロです)。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
OFilename++DF..N01N02N03Excnam++++B++A++Sb+Sa+.........................*
OITMREC    D    20
O.............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat++
O                      *ALL  1
 *
OSLSREC    D    30
O                      SLSNAM  2
O                      COMRAT
O                   15 BONUS
 *
```

図 153. 外部記述ファイルの出力仕様書

1 更新ファイルの場合、レコードのすべてのフィールドは、レコードのすべてのフィールドに対する、プログラムの現在の値を使用して外部記述レコード ITMREC に書き出されます。

新しいレコードの作成の場合、レコードのすべてのフィールドは、プログラム中のレコードのそのフィールドに対する現在の値を使用して外部記述レコード ITMREC に書き出されます。

2 レコードを更新するために、標識 30 がオンの時に、フィールド SLSNAM および COMRAT が外部記述レコード SLSREC に書き出されます。フィールド BONUS は標識 30 および 15 がオンの場合に、SLSREC レコードに書き出されます。レコード中のその他のフィールドはすべて、読み取られた値を用いて書き出されます。

新しいレコードを作成するために、標識 30 がオンの時に、フィールド SLSNAM および COMRAT が外部記述レコード SLSREC に書き出されます。フィールド BONUS は標識 30 および 15 がオンの場合に、書き出されます。レコード中のその他のフィールドはすべて、デフォルト値として書き出されます。これはデータ・タイプによって異なります (例えば、文字フィールドの場合はブランクで、数字フィールドの場合はゼロです)。

レベル検査

HLL プログラムが実行時に受け取る外部記述ファイルの様式は、コンパイル時にプログラムにコピーされた様式と一致しなければならないので、システムには様式が同じであることを確認するためのレベル検査機能があります。

外部記述の DISK、WORKSTN、または PRINTER ファイルが使用される時には、RPG コンパイラーは常に、レベル検査に必要な情報を提供します。レベル検査機能はファイルの作成、変更、または一時変更コマンドで要求することができます。ファイル作成コマンドではデフォルト値としてレベル検査が要求されます。

ファイル指定変更コマンドを出すか、ファイルを作成する時に LVLCHK(*NO) を指定しない限り、ファイルのオープン時にレコード様式単位でレベル検査が行われま

す。レベル検査値が一致しない場合には、プログラムにエラーが通知されます。ここで RPG プログラムは、 273 ページの『第 13 章 例外の処理』で説明されているように OPEN エラーを処理します。

プログラム記述ファイルあるいは装置 SEQ または SPECIAL を使用するファイルの場合には、RPG プログラムはレベル検査を行いません。

レベル検査を指定する方法の説明については、Web サイト http://www.ibm.com/eserver/iseries/infocenter で **iSeries 400 Information Center** の中の『データベースおよびファイル・システム』カテゴリーの『*DB2 UDB for iSeries*』の節を参照してください。

プログラム記述ファイルの定義

プログラム記述ファイルとは、そのレコードおよびフィールドがファイルを使用するプログラムの中の入力 / 出力仕様書で記述されるファイルのことです。RPG プログラムでプログラム記述ファイルを使用するためには、次のことを行わなければなりません。

1. ファイル仕様書でファイルを指定します。
2. 入力ファイルの場合には、入力仕様書でレコードおよびフィールドを記述します。入力仕様書の 7 ～ 16 桁目のファイル名は、ファイル仕様書に入力された対応する名前と同じでなければなりません。

   レコード識別項目で、ファイル内のレコードの順序検査を行いたいかどうかを指示します。
3. ステップ 1 におけるのと同じファイル名を、それを必要とする演算仕様書の演算項目 2 フィールドに指定します。例えば、プログラム記述ファイルに対する WRITE 命令の場合には、結果のフィールドにデータ構造名が必要です。
4. 出力ファイルの場合には、出力仕様書のレコードおよびフィールドを記述します。さらに、出力の印刷方法も指定します。出力仕様書の 7 ～ 16 桁目のファイル名は、ファイル仕様書に入力された対応する名前と同じでなければなりません。

プログラムを実行する前に、プログラム記述ファイルはシステム上に存在していなければならず、ライブラリー・リストになければなりません。ファイルを作成するには Create File コマンドのいずれかを使用します。これらのコマンドは、Web サイト http://www.ibm.com/eserver/iseries/infocenter で **iSeries 400 Information Center** の中の『プログラミング』カテゴリーの『*CL および API*』の節を参照してください。

データ管理命令と ILE RPG 入出力命令

データ管理は、アプリケーション・プログラムによるデータの保管とアクセスを制御する、オペレーティング・システムの一部です。 333 ページの表 32 は、iSeries が提供するデータ管理命令、およびそれらに対応する ILE RPG 命令を示します。また、どの命令がどの ILE RPG 装置タイプに使用可能であるかも示します。

表 32. データ管理命令および対応する RPG 入出力命令

データ管理命令	ILE RPG 入出力命令
OPEN	OPEN
READ	
相対レコード番号による	READ、CHAIN
キーによる	
	READ、READE、CHAIN、プライマリーおよび
順次	セカンダリー・ファイル
前	READ
NEXT	READP、READPE
送信勧誘装置	READ、READE
	READ
WRITE-READ	EXFMT
WRITE	
相対レコード番号による	WRITE
キーによる	WRITE、EXCEPT、プライマリー
順次	および セカンダリー・ファイル
	WRITE、EXCEPT
FEOD	FEOD
UPDATE	
相対レコード番号による	UPDATE、プライマリーおよびセカンダリー・ファイル
キーによる	UPDATE、プライマリーおよびセカンダリー・ファイル
DELETE	
相対レコード番号による	DELETE、プライマリーおよびセカンダリー・ファイル
キーによる	DELETE、プライマリーおよびセカンダリー・ファイル
ACQUIRE	ACQ
RELEASE	REL
COMMIT	COMMIT
ROLLBACK	ROLBK
CLOSE	CLOSE、LR RETURN

第 16 章 ファイルに関する一般的な考慮事項

この章では、RPG を使用した iSeries システムでのファイル処理について次の側面から説明します。

- ファイル入出力の一時変更および指定変更
- RPG プログラムによるファイルのロック
- RPG プログラムによるレコードのロック
- オープン・データ・パスの共用
- iSeries スプーリング機能
- DDS ファイルに対する RPG プログラムでの SRTSEQ/ALTSEQ の使用

ファイル入出力の一時変更および指定変更

コンパイル時または実行時に、ファイル仕様書に指定されたパラメーターを一時変更するかあるいはファイルを指定変更するためには、OS/400 コマンドを使用することができます。ファイル指定変更によって、(コンパイル時に) プログラムに指定されたファイルを置き換えるファイルを実行時に指定することができます。

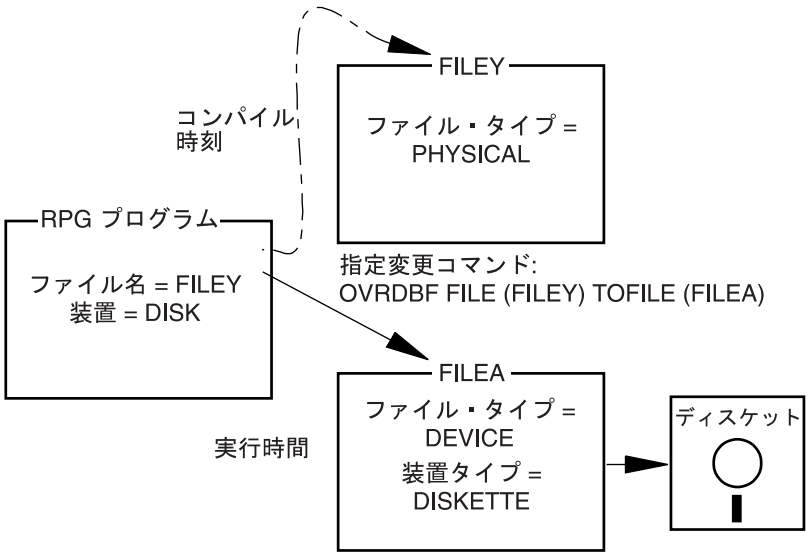


図 154. ファイル入出力の一時変更例

上記の例では、CL コマンド OVRDBF (データベース・ファイル一時変更) によって、コンパイル時に指定された装置ファイルとはまったく異なる装置ファイルを用いて、プログラムを実行することができます。

実行時にファイルを一時変更するためには、両方のファイルのレコード名が間違いなく同じでなければなりません。RPG プログラムは READ 命令などの入出力命令でレコード様式名を使用しますが、その命令ではこのレコード様式名が必要なレコード・タイプを指定します。

すべてのファイル指定変更または一時変更が有効なわけではありません。実行時に、RPG プログラム内の仕様書が処理中のファイルに対して有効であるかを確認するための検査が行われます。OS/400 システムでは、プログラムに装置の仕様書が入っていても、一部のファイルの指定変更を行うことができます。例えば、RPG 装置名が PRINTER で、プログラムが接続している実際のファイルがプリンターでない場合には、OS/400 システムは RPG の印刷スペースおよびスキップ仕様書を無視します。

OS/400 システムが使用できずに、プログラムを終了することになる、その他のファイル指定変更があります。例えば、RPG 装置名が WORKSTN で、プログラムで EXFMT 命令が指定された場合には、プログラムが接続している実際のファイルが表示装置ファイルまたは ICF ファイルでない場合に、プログラムは停止します。

ILE では、一時変更は活動化グループ・レベル、ジョブ・レベル、または呼び出しレベルまで拡大されます。活動化グループ・レベルまで拡大された一時変更は、削除または置き換えられるか、あるいはその一時変更が指定されている活動化グループが終了するまで有効となっています。ジョブ・レベルまで拡大された一時変更は、削除または置き換えられるか、あるいはその一時変更が指定されているジョブが終了するまで有効となっています。これは、一時変更が指定されている活動化グループに関係なく適用されます。呼び出しレベルまで拡大された一時変更は、削除または置き換えられるか、あるいはその一時変更が指定されているプログラムまたはプロシージャーが終了するまで有効となります。

共用ファイルのデフォルトの有効範囲は、活動化グループです。ジョブ・レベルの有効範囲の場合には、一時変更コマンドに OVRSCOPE(*JOB) を指定してください。呼び出しレベルの有効範囲の場合には、一時変更コマンドに OVRSCOPE(*CALLLVL) を指定してください。

有効なファイル指定変更およびファイル一時変更についての詳細は、Web サイト http://www.ibm.com/eserver/iseries/infocenter で **iSeries 400 Information Center** の中の『データベースおよびファイル・システム』カテゴリーの『*DB2 UDB for iSeries*』の節を参照してください。

「*ILE 概念*」にも一時変更、および活動化グループ対ジョブ・レベルの有効範囲についての説明があります。

ファイル入出力の指定変更の例

以下にコンパイル時にファイルを一時変更する例を示します。フィールド・レベル記述のない TAPE 装置の外部記述ファイルを使用したいと想定します。以下を行ってください。

1. レコード様式で各フィールドの記述が入っているレコード様式で、FMT1 という名前の物理ファイルを定義します。このレコード様式はデータ記述仕様書 (DDS) で定義します。テープ装置の場合には、外部記述ファイルの様式は 1 つだけにしてください。
2. 物理ファイル作成 CL コマンドを用いて FMT1 という名前のファイルを作成します。

3. RPG プログラムの中で QTAPE (磁気テープ装置の IBM 提供の装置ファイル名) というファイル名を指定します。これは、ファイルを外部記述ファイル (ファイル仕様書の 22 桁目の E によって示される) として指定し、36 ～ 42 桁目に装置名 SEQ を指定します。
4. 一時変更コマンド OVRDBF FILE(QTAPE) TOFILE(FMT1) をコンパイル時に使用して QTAPE ファイル名を一時変更し、FMT1 ファイル名を使用します。このコマンドによって FMT1 ファイルの外部記述をコンパイラーがコピーし、その外部記述からレコード様式が RPG コンパイラーに記述されます。
5. CRTBNDRPG コマンドまたは CRTPGM コマンドを使用して RPG プログラムを作成します。
6. 実行時にプログラムを呼び出します。ファイル FMT1 に対する一時変更は、プログラムの実行中は有効とはなっていないはずです。一時変更が有効となっている場合には、プログラムを呼び出す前に CL コマンドの DLTOVR (一時変更削除) を使用してください。

   注: テープ・ファイルのオープンに必要な情報を用意するためには、プログラムを呼び出す前に、CL コマンドの OVRTAPF を使用する必要がある場合があります。

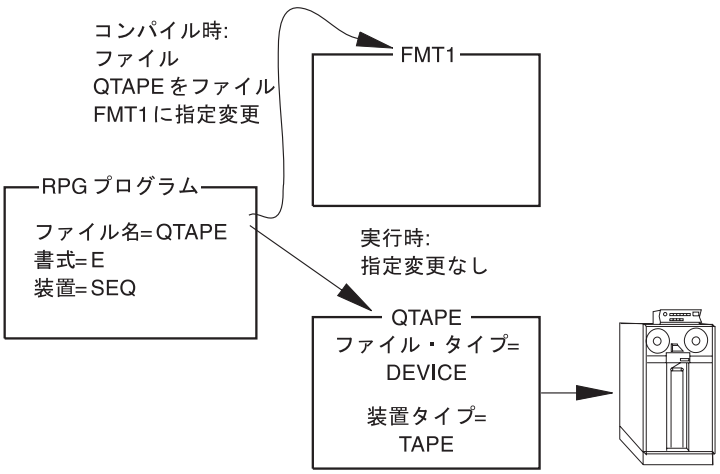


図 155. ファイル入出力の指定変更例

ファイルのロック

OS/400 システムでは、ジョブの実行中に使用するファイルをロック状態 (排他読み取り可能、更新共用、更新不可共用、または読み取り共用) に置くことができます。ジョブ内のプログラムは、ファイルのロック状態による影響を受けません。ファイルのロック状態は、別のジョブのプログラムがファイルを同時に使用しようとした時にのみ適用されます。ファイルのロック状態は、CL コマンド ALCOBJ (オブジェクト割り振り) によって割り振ることができます。リソースの割り振りとロック状態についての詳細は、http://www.ibm.com/eserver/iseries/infocenter で **iSeries 400 Information Center** の中の『データベースおよびファイル・システム』カテゴリーの『*DB2 UDB for iSeries*』の節を参照してください。

OS/400 システムではデータベース・ファイルのオープン時に、次のロック状態がデータベースに設定されます。

ファイル・タイプ	ロック状態
入力	読み取り共用
更新	更新用共用
追加	更新用共用
出力	更新用共用

読み取り共用ロック状態では、別のユーザーが読み取り共用、更新共用、更新不可共用、または排他読み取り可能の各ロック状態でファイルをオープンすることはできますが、そのユーザーはファイルの排他使用を指定することはできません。更新共用ロック状態は、別のユーザーは読み取り共用または更新共用ロック状態でこのファイルをオープンすることができます。

RPG プログラムは装置ファイルを排他読み取り許可ロック状態に置きます。別のユーザーはこのファイルを読み取り共用ロック状態でオープンすることができます。

オブジェクト割り振りコマンドを使用すると、RPG プログラムによって設定されたロック状態を変更することができます。

レコードのロック

レコードがプログラムによって読み取られる時には、入力または更新の 2 つのモードのいずれかで読み取られます。プログラムが更新用にレコードを読み取る場合には、そのレコードはロック状態になります。最初のプログラムがそのロックを解除するまで、別のプログラムが更新用に同じレコードを読み取ることはできません。プログラムが入力用にレコードを読み取る場合には、そのレコードはロック状態になりません。あるプログラムによってロックされたレコードは、別のプログラムによって入力用に読み取ることは可能です。

RPG IV プログラムでは、更新用のレコード読み取りには更新ファイルを使います。更新以外のタイプのファイルから読み取るレコードは、照会用にしか読み取ることはできません。デフォルトでは、更新ファイルから読み取られるレコードはすべて、更新用に読み取られます。更新ファイルの場合には、入力命令 CHAIN、READ、READE、READP、または READPE の 1 つを使用し、命令コード・フィールドの命令コード名の次に命令コードの拡張 (N) を指定して、レコードが入力用に読み取られるように指定することができます。

レコードが RPG IV プログラムによってロックされている時は、次のいずれかが起こるまでそのロックが続きます。

- レコードが更新される
- レコードが削除される
- 別のレコードがファイルから (照会または更新のいずれかで) 読み取られる
- ファイルに対して SETLL または SETGT 命令が実行される
- ファイルに対して UNLOCK 命令が実行される
- ファイルに対してフィールド名がない出力仕様書によって定義された出力命令が実行される

注: ファイルにレコードを追加する出力命令では、レコードのロックは解除されません。

プログラムが更新用にレコードを読み取り、そのレコードがジョブ内の別のプログラムまたは別のジョブによってロックされた場合には、その読み取り命令はレコードのアンロックされるまで待機します (共用ファイルの場合は除きます。『オープン・データ・パスの共用』を参照してください。)待機時間がファイルの WAITRCD パラメーターで指定された値を超えている場合には、例外が起こります。プログラムがこの例外を処理していない場合には、レコード・ロックのタイムアウトが起こった時にデフォルトのエラー処理プログラムが制御を受け取り、 RNQ1218 照会メッセージが出されます。このメッセージにリストされたオプションの 1 つは、タイムアウトが起こった命令の再試行です。 これにより、タイムアウトが起こった命令が再び出され、レコード・ロックのタイムアウトが起こらなかったかのようにプログラムが続行できるようになります。ファイルに INFSR の指定があり、デフォルトのエラー処理プログラムが制御を受け取る前に INFSR 内でファイルの入出力操作が実行される場合には、ファイルのカーソルが変更されている可能性があるために、再試行される操作が順次操作の場合は、予期しない結果が起こることがあることに注意してください。

注: サブプロシージャーは、照会メッセージを受け取らないので、この状況は読み取り命令のエラー標識を使用し、読み取りの後の状況 1218 について検査することによって処理する必要があります。

ロックされたレコードに変更が必要でない場合には、ファイルのカーソルを変更せずに、UNLOCK 命令を使用するか、またはフィールド名のない出力仕様書を処理することによって、そのレコードをロック状態から解除することができます。これらの出力命令は、EXCEPT 出力、明細出力、または合計出力によって処理することができます。

(コミットメント制御の下で操作する時には、これらの規則に例外があります。 詳細については、 371 ページの『コミットメント制御の使用』を参照してください。)

オープン・データ・パスの共用

オープン・データ・パスとは、ファイルのすべての入出力操作が実行されるパスのことです。通常、ファイルがオープンされるたびに、別個のオープン・データ・パスが定義されます。ファイルの作成または一時変更で SHARE(*YES) を指定した場合には、ファイルの最初のプログラムのオープン・データ・パスがそのファイルを同時にオープンする後続のプログラムでも共用されます。

現行レコードの位置は、このファイルを使用しているすべてのプログラムのオープン・データ・パスで保存されます。あるプログラムでレコードを読み取って、呼び出されるプログラムでレコードを読み取る場合には、2 回目の読み取りで検索されるレコードはオープン・データ・パスを共用しているかどうかによって異なります。オープン・データ・パスを共用している場合には、呼び出されるプログラムの現行レコードの位置は、呼び出し側プログラムの現在位置によって決まります。オープン・データ・パスを共用していない場合には、各プログラムごとに現行レコードの位置は独立したものになります。

プログラムが共用ファイルにレコード・ロックを保持してから、更新用に共用ファイルを読み取る 2 番目のプログラムを呼び出す場合には、最初のプログラムのロックは、次を行うことによって解除することができます。

- 2 番目のプログラムで更新ファイルに対して READ 命令を実行する。
- UNLOCK またはロックなしの読み取り命令を使用する。

ILE では、共用ファイルはジョブ・レベルまたは活動化グループ・レベルまで拡大されています。ジョブ・レベルまで拡大された共用ファイルは、ジョブ内のどの 活動化グループで実行中のどのプログラムとでも共用することができます。活動化グループ・レベルまで拡大された共用ファイルは、同じ活動化グループで実行中のプログラムによってのみ 共用することができます。

共用ファイルのデフォルトの有効範囲は、活動化グループです。ジョブ・レベルの有効範囲の場合には、一時変更コマンドに OVRSCOPE(*JOB) を指定してください。

ILE RPG では、共用 ODP の分野におけるいくつかの機能強化を提供します。プログラムまたはプロシージャーが読み取り命令を実行する場合には、問題のファイルに SHARE(*YES) が指定されている限り、別のプログラムまたはプロシージャーがレコードを更新することができます。さらに、複数装置ファイルを使用している時に、1 つのプログラムが装置を獲得した場合には、ODP を共用するその他のプログラムもその獲得された装置を使用することができます。更新の実行に必要なすべてのデータが呼び出し先プログラムで使用可能であるかどうかの確認は、プログラマーが行います。

オープン・データ・パスの共用により、OS/400 システムは新しいオープン・データ・パスを作成する必要がないので、パフォーマンスが向上します。しかし、オープン・データ・パスの共用によって問題が起こることもあります。例えば、次の場合にはエラーが通知されます。

- オープン・データ・パスを共用するプログラムが最初のオープンで指定したファイル操作以外のファイル操作を行おうとした場合 (例えば、最初のオープンでは出力操作のみを指定したのに、入力操作を行おうとした場合)
- 外部記述ファイルのオープン・データ・パスを共用するプログラムが最初のプログラムで無視されたレコード様式を使おうとした場合
- プログラム記述ファイルのオープン・データ・パスを共用するプログラムが最初のオープンで確立されたレコード長を超える長さを指定した場合

実行時に、あるプログラム中の複数のファイルが 1 つの共用ファイルに一時変更される時には、ファイルのオープン順序が重要です。ファイルのオープン順序を制御するためには、プログラマー制御オープンを使用するか、あるいはプログラムを呼び出す前に CL プログラムを使用する必要があります。

プログラムがプライマリーまたはセカンダリー・ファイルのオープン・データ・パスを共用する場合には、そのオープン・データ・パスを共用する別のプログラムを呼び出す前に、プログラムは、処理中のレコードの明細演算を処理しなければなりません。それを行わない場合には、先読みを使用しているか、または合計時に呼び出した場合に、プライマリー または セカンダリー・ファイルのオープン・データ・パスの共用によって、呼び出し先プログラムはファイルの間違ったレコードからデータを読み取ることがあります。

最初に共用ファイルをオープンする時には、そのファイルの後続のオープンで必要となるすべてのオープン・オプションが指定されていることを確認する必要があります。共用ファイルの後続のオープンに指定されたオープン・オプションが、共用ファイルの最初のオープンに指定されたオープン・オプションに含まれていない場合には、エラー・メッセージがプログラムに送られます。

表 33 は、ユーザーが指定できる各オープン・オプションに認められた、システムのオープン・オプションを示します。

表 33. ユーザー・オープン・オプションに使用できるシステム・オープン・オプション

RPG ユーザー・オープン・オプション	システム・オープン・オプション
INPUT	INPUT
OUTPUT	OUTPUT (プログラム作成ファイル)
UPDATE	INPUT、UPDATE、DELETE
ADD	OUTPUT (既存のファイル)

オープン・データ・パス共用、および活動化グループ対ジョブ・レベルの有効範囲についての詳細は、「*ILE 概念*」を参照してください。

スプーリング

スプーリングは、処理を待機するために、データを記憶域に入れておくシステム機能です。iSeries システムは、入出力スプーリング機能を提供します。各 iSeries ファイル記述には、実行時にファイルにスプーリングを使用するかどうかを決定するスプール属性が入っています。RPG プログラムはスプーリングを使用中かどうかは関知しません。ファイルが読み取られたり、書き出されたりする実際の物理装置は、スプール読み取りプログラムまたはスプール書き出しプログラムによって決まります。スプーリングについての詳細は、Web サイト http://www.ibm.com/eserver/iseries/infocenter で **iSeries 400 Information Center** の中の『データベースおよびファイル・システム』カテゴリーの『*DB2 UDB for iSeries*』の節を参照してください。

出力スプーリング

出力スプーリングはバッチ・ジョブまたは対話式ジョブに有効です。RPG プログラムでファイル名によって指定するファイルの記述には、次の図表に示してあるようにスプーリングの指定が含まれています。

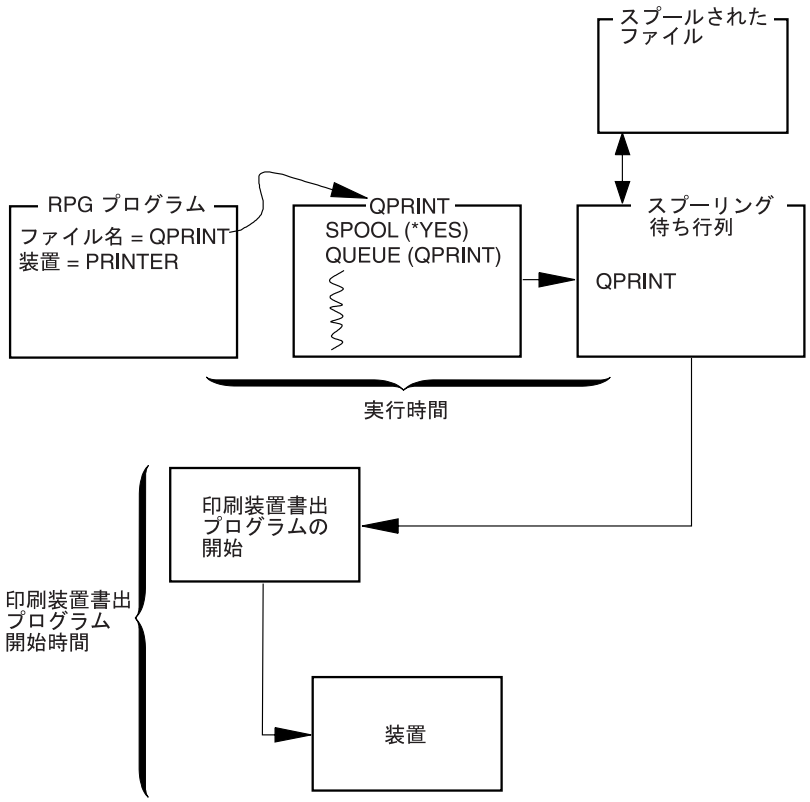


図 156. 出力スプーリング例

実行時にファイル一時変更コマンドを使用して、ファイル仕様書で指定された印刷コピー数などのスプーリング・オプションを一時変更することができます。さらに、iSeries スプーリング・サポートによって、プログラムの実行後にファイルを指定変更することができます。ディスケットなどの異なる装置に同じ印刷出力を指示することができます。

RPG プログラム対 DDS ファイルの SRTSEQ/ALTSEQ

キー付きファイルが SRTSEQ および LANGID を使用して作成される場合には、CHAIN、SETLL、SETGT、READE、および READPE の各命令を実行中にファイル中の文字キーを比較する時に、指定された SRTSEQ が使用されます。RPG プログラムまたはモジュールを作成する時には、同じまたは何かの SRTSEQ 値を指定する必要はありません。

CRTBNDRPG または CRTRPGMOD に SRTSEQ の値が指定されている時には、プログラム中のすべての文字比較操作でこの SRTSEQ が使用されることになります。この値は、キー・フィールド、他のファイルのフィールド、およびプログラム中に宣言されたフィールドを含むすべての フィールドの比較に影響します。

文字データで作業するには、ファイルの作成時に何を指定したかではなく、IFxx、COMP、および SORTA などの命令でどう実行したかに基づいて、RPG プログラムに SRTSEQ を使用するかどうかを決定してください。

第 17 章 データベース・ファイルのアクセス

適切なファイル仕様書でファイル名を装置 DISK に関連付けることによって、プログラムからデータベース・ファイルをアクセスすることができます。

ILE RPG プログラムの DISK ファイルも分散データ管理 (DDM) ファイルに関連付けられるので、リモート・システム上のファイルをデータベース・ファイルとしてアクセスすることができます。

データベース・ファイル

データベース・ファイルは、iSeries システム上のタイプ *FILE のオブジェクトです。これらは物理または論理ファイルのいずれかであり、また外部記述またはプログラム記述のいずれかであるということができます。ファイル仕様書の 36 ～ 42 桁目でファイル名を装置 DISK に関連付けることによって、データベース・ファイルをアクセスします。

データベース・ファイルは OS/400 ファイル作成コマンドによって作成することができます。データベース・ファイルを記述し作成する方法の説明については、 Web サイト http://www.ibm.com/eserver/iseries/infocenter で **iSeries 400 Information Center** の中の『データベースおよびファイル・システム』カテゴリーの『*DB2 Universal Database for AS/400*』の節を参照してください。

物理ファイルおよび論理ファイル

物理ファイルには、システム上に保管される実際のデータ、およびプログラムへのデータの表示され方またはプログラムからのデータの受け取られ方の記述が入っています。これらには、1 つのレコード様式だけが入っており、また 1 つ以上のメンバーが入っています。データベース・ファイルのレコードは、外部記述またはプログラム記述とすることができます。

物理ファイルにはキー順アクセス・パスを使用することができます。このことは、データはファイルの中の 1 つ以上の複数のキー・フィールドに基づいて順番にプログラムに与えられる、ということを意味します。

論理ファイルにはデータは入っていません。このファイルには、1 つまたは複数の物理ファイルで見付かるレコードの記述が入っています。論理ファイルは、1 つまたは複数の物理ファイルのビューまたは表現です。複数の様式が入っている論理ファイルは、**複数様式**論理ファイルとして参照されます。

プログラムが複数のレコード様式の入っている論理ファイルを処理する場合には、レコード様式による読み取りを用いて使用したい様式を設定することができます。

データ・ファイルとソース・ファイル

データ・ファイルには、実際のデータまたはデータのビューが入っています。データ・ファイルのレコードはメンバーにグループ化されます。ファイルのすべてのレコードを 1 つのメンバーに入れるか、あるいは異なるメンバーにグループ化するこ

とができます。ほとんどのデータベース・コマンドおよび命令は、そのデフォルトではデータを含むデータベース・ファイルにはメンバーを *1* つだけ もっているものとします。これは、プログラムがデータを含んでいるデータベース・ファイルをアクセスする時に、ファイルに複数のメンバーが入っていない限り、メンバー名を指定する必要がないことを意味します。ファイルに複数のメンバーが入っていて、特定のメンバー名を指定しなかった場合には、最初のメンバーが使用されます。

通常、ソース・プログラムが入っているデータベース・ファイルは、複数のメンバーから構成されています。ソース・プログラムをデータベース・ファイル内のメンバーに編成することによって、プログラムをより最適に管理することができます。**ソース・メンバー**には、プログラム・オブジェクトを作成するためにシステムが使用するソース・ステートメントが入っています。

外部記述ディスク・ファイルの使用

外部記述ファイルは、ファイル仕様書の 22 桁目の E によって識別されます。E は、コンパイラーが、プログラムのコンパイル時にシステムからファイルの外部記述を検索することを指示します。したがって、プログラムをコンパイルする前にファイルを作成する必要があります。

DISK ファイルの外部記述には以下のものが含まれます。

- レコード内のフィールドの記述が入っているレコード様式仕様書
- レコードの検索方法を記述したアクセス・パス仕様書

これらの仕様書はそのファイルの DDS および、そのファイルに対して使われる OS/400 ファイル作成コマンドからでき上がります。

レコード様式仕様書

レコード様式仕様書によって、レコード内のフィールド、およびレコード内のフィールドの位置を記述することができます。フィールドは、DDS で指定された順序でレコード内に位置指定されています。フィールド記述には一般に、フィールド名、フィールド・タイプ、およびフィールド長 (数字フィールドの場合には小数点以下の桁数を含む) が含まれます。フィールドの属性は、物理ファイルまたは論理ファイルのレコード様式に指定する代わりに、フィールド参照ファイルの中で定義することができます。

さらに、DDS キーワードを使用して次のことを行うことができます。

- 重複するキー値はファイルに使用できないことを指定する (UNIQUE)。
- レコード様式またはフィールドのテキスト記述を指定する (TEXT)。

データベース・ファイルに対して有効な DDS キーワードの全リストについては、Web サイト http://www.ibm.com/eserver/iseries/infocenter で **iSeries 400 Information Center** の中の『データベースおよびファイル・システム』カテゴリーの『*DB2 Universal Database for AS/400*』の節を参照してください。

345 ページの図 157 は、データベース・ファイルの DDS の例を示し、 346 ページの図 158 は、データベース・ファイルで使用されるフィールドの属性を定義するフィールド参照ファイルの例を示しています。フィールド参照ファイルの詳細については、上記の Web サイトを参照してください。

アクセス・パス

外部記述ファイルの記述には、レコードをファイルから取り出す方法を記述しているアクセス・パスが含まれます。レコードは、到着順 (キー順でない) アクセス・パスまたはキー順アクセス・パスに基づいて検索することができます。

到着順アクセス・パスは、レコードがファイルに保管される順序に基づいています。レコードは 1 つずつ順番にファイルに追加されていきます。

キー順アクセス・パスの場合には、ファイル内のレコードの順序は、そのファイルの DDS に定義されたキー・フィールドの内容に基づいています。例えば、図 157 に示されている DDS では、 CUST はキー・フィールドとして定義されています。キー順アクセス・パスは、レコードを追加した場合、削除した場合、あるいはキー・フィールドの内容が変更された場合には、常に更新されます。

外部記述データベース・ファイルの場合のアクセス・パスの詳細については、Web サイト http://publib.boulder.ibm.com/html/as400/infocenter.html で **iSeries 400 Information Center** の中の『データベースおよびファイル・システム』カテゴリーの『*DB2 UDB for iSeries*』の節を参照してください。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A..........T.Name++++++.Len++TDpB......Functions++++++++++++++++++++*
A** 論理     CUSMSTL    得意先マスター・ファイル
A                                    UNIQUE
A          R CUSREC                  PFILE(CUSMSTP)
A                                    TEXT('Customer Master Record')
A            CUST
A            NAME
A            ADDR
A            CITY
A            STATE
A            ZIP
A            SRHCOD
A            CUSTYP
A            ARBAL
A            ORDBAL
A            LSTAMT
A            LSTDAT
A            CRDLMT
A            SLSYR
A            SLSLYR
A          K CUST
```

図 157. データベース・ファイルのデータ記述仕様書の例

DDS のサンプルは、得意先マスター論理ファイル CUSMSTL のものです。このファイルには、1 つのレコード様式 CUSREC (得意先マスター・レコード) があります。このファイルのデータは、物理ファイル CUSMSTP に入っていますが、これはキーワード PFILE によって識別されます。UNIQUE キーワードを使用して、このファイルには重複するキーの値を使用できないことを示しています。CUST フィールドは、最後の行の 17 桁目に K があるので、このレコード様式のキー・フィールドとして識別されます。

このレコード様式のフィールドは、レコードに現れる順序でリストされます。フィールドの属性は、物理ファイル CUSMSTP から獲得されます。順に、物理ファイル

はフィールド参照ファイルを参照してフィールドの属性を獲得します。フィールド参照ファイルは 図 158 に示してあります。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A.........T.Name++++++RLen++TDpB......Functions++++++++++++++++++++++*
A**FLDRED    DSTREF     配送アプリケーションのフィールド参照
A          R DSTREF                    TEXT('Distribution Field Ref')
A* 参照として使用される共通フィールド
A            BASDAT         6  0       EDTCDE(Y) 1
A                                       TEXT('Base Date Field')
A* 得意先マスター・ファイルで使用されるフィールド
A            CUST           5          CHECK(MF) 2
A                                       COLHDG('Customer' 'Number')
A            NAME          20          COLHDG('Customer Name')
A            ADDR      R               REFFLD(NAME) 3
A                                       COLHDG('Customer Address')
A            CITY      R               REFFLD(NAME) 3
A                                       COLHDG('Customer City')
A            STATE          2          CHECK(MF) 2
A                                       COLHDG('State')
A            SRHCOD         6          CHECK(MF) 2
A                                       COLHDG('Search' 'Code')
A                                       TEXT('Customer Number Search +
A                                       Code')
A            ZIP            5  0       CHECK(MF) 2
A                                       COLHDG('Zip' 'Code')
A            CUSTYP         1  0       RANGE(1 5) 4
A                                       COLHDG('Cust' 'Type')
A                                       TEXT('Customer Type 1=Gov 2=Sch+
A                                       3=Bus 4=Pvt 5=Oth')
A            ARBAL          8  2       COLHDG('Accts Rec' 'Balance') 5
A                                       EDTCDE(J) 6
A            ORDBAL    R               REFFLD(ARBAL)
A                                       COLHDG('A/R Amt in' 'Order +
A                                       File')
A            LSTAMT    R               REFFLD(ARBAL)
A                                       COLHDG('Last' 'Amount' 'Paid')
A                                       TEXT('Last Amount Paid in A/R')
A            LSTDAT    R               REFFLD(BASDAT)
A                                       COLHDG('Last' 'Date' 'Paid')
A                                       TEXT('Last Date Paid in A/R')
A            CRDLMT    R               REFFLD(ARBAL)
A                                       COLHDG('Credit' 'Limit')
A                                       TEXT('Customer Credit Limit')
A            SLSYR     R+   2          REFFLD(ARBAL)
A                                       COLHDG('Sales' 'This' 'Year')
A                                       TEXT('Customer Sales This Year')
A            SLSLYR    R+   2          REFFLD(ARBAL)
A                                       COLHDG('Sales' 'Last' 'Year')
A                                       TEXT('Customer Sales Last Year') 7
```

図 158. フィールド参照ファイルの例

このフィールド参照ファイルの例は、 345 ページの図 157 に示されている CUSMSTL (得意先マスター論理) ファイルによって使用されるフィールドの定義を示しています。フィールド参照ファイルには通常、他のファイルによって使用されるフィールドの定義が入っています。次のテキストは、このフィールド参照ファイルの一部の項目について説明しています。

1 BASDAT フィールドは、キーワード EDTCDE(Y) によって示してあるように Y 編集コードによって編集されます。このフィールドを ILE RPG プログラムの外部記述出力ファイルで使う場合は、使用される編集コードはこの

フィールド参照ファイルで指定されたものになります。これは ILE RPG プログラムで一時変更はされません。このフィールドを ILE RPG プログラム記述の出力ファイルで使用する場合には、編集コードは出力仕様書のフィールドに指定しなければなりません。

2 CHECK(MF) 項目は、フィールドを表示ワークステーションから入力する場合には、そのフィールドが全桁入力フィールドとなることを示します。全桁入力とは、表示ワークステーションからフィールドのすべての桁を入力しなければならないことを意味します。

3 ADDR および CITY フィールドは、REFFLD キーワードで示されているように NAME フィールドに指定したのと同じ属性を共用します。

4 CUSTYP フィールドに指定した RANGE キーワードによって、表示ワークステーションからこのフィールドに入力できる有効な番号は 1 ～ 5 だけということになります。

5 フィールドが対話式データベース・ユーティリティー (IDU) で使用されている場合には、COLHDG キーワードがそのフィールドの欄見出しを提供します。

6 ARBAL フィールドは、キーワード EDTCDE(J) によって示してあるように J 編集コードによって編集されます。

7 一部のフィールドにはテキスト記述 (TEXT キーワード) が提供されます。TEXT キーワードは文書化を目的とし、いろいろなリストに出てきます。

レコードまたはファイルに有効なキー

キー順アクセス・パスの場合には、レコード様式のキー・フィールドとして使用す
るために、 DDS に 1 つまたは複数のフィールドを定義することができます。ファイル内のすべてのレコード・タイプが同じキー・フィールドをもつ必要はありません。例えば、受注見出しレコードで ORDER フィールドをキー・フィールドとして定義し、受注明細レコードで ORDER および LINE フィールドをキー・フィールドとして定義することができます。

ファイルのキーは、そのファイル内のレコード・タイプに有効なキーによって決まります。ファイルのキーは次のようにして決まります。

- ファイルのすべてのレコード・タイプが DDS で定義された同じ属性のキー・フィールドを同数もっている場合には、*ファイルのキー* はそれらのレコード・タイプのキーのすべてのフィールドから成ります。(対応するフィールドが同じ名前をもつ必要はありません。) 例えば、ファイルが 3 つのレコード・タイプを持っていて、各レコード・タイプのキーが A、B、C のフィールドで構成されている場合、そのファイルのキーはフィールド A、B、および C で構成されます。すなわち、ファイルのキーはレコードのキーと同じになります。
- ファイルのすべてのレコード・タイプが同じキー・フィールドをもっていない場合には、ファイルのキーはすべてのレコード・タイプに*共通* のキー・フィールドから成ります。例えば、ファイルに 3 つのレコード・タイプがあり、キー・フィールドが次のとおり定義されていたとします。
  - REC1 にはキー・フィールド A があります。
  - REC2 にはキー・フィールド A および B があります。
  - REC3 にはキー・フィールド A、B、および C があります。

この場合のファイルのキーはフィールド A、すなわちすべてのレコード・タイプに共通のキー・フィールドとなります。
- すべてのレコード・タイプに共通のキー・フィールドがない場合には、そのファイルにキーはありません。

ILE RPG プログラムでは、あるファイル命令コードに検索引き数を指定して、処理したいレコードを識別することができます。ILE RPG プログラムはこの検索引き数とファイルまたはレコードのキーを比較し、検索引き数と一致したキーのレコードに対して指定命令を処理します。

有効な検索引き数

ファイル名やレコード名を指定する ILE RPG 命令である CHAIN、DELETE、READE、READPE、SETGT、および SETLL に、検索引き数を指定することができます。

ファイル名に対する命令の場合、検索引き数に指定できるフィールドの最大数は、ファイルのキーとして有効なキー・フィールドの合計数です。例えば、ファイルのすべてのレコード・タイプに同じキー・フィールドが入っていない場合には、キー・リスト (KLIST) を使用して、ファイルのすべてのレコード・タイプに共通のフィールドの数だけからなる検索引き数を指定することができます。ファイルに 3 つのレコード・タイプがあり、キー・フィールドが次のとおり定義されていたとします。

- REC1 にはキー・フィールド A があります。
- REC2 にはキー・フィールド A および B があります。
- REC3 にはキー・フィールド A、B、および C があります。

すべてのレコード・タイプに共通のキー・フィールドはフィールド A のみであるため、検索引き数はフィールド A と同じ属性だけをもつ単一フィールドとしかなれません。検索引き数は、浮動小数点フィールド、可変長フィールド、または null 可能フィールドを含むことはできません。

レコード名に対する命令の場合には、検索引き数に指定できるフィールドの最大数は、そのレコード・タイプのキーとして有効なキー・フィールドの合計数と同じです。

検索引き数が 1 つ以上のフィールドから成り立つ場合は、表意定数 KLIST、および自由形式演算では式のリスト (小括弧で囲む) または %KDS を指定することができます。単一のフィールドからなる検索引き数の場合には、上記に加えて、リテラルまたは変数名も指定できます。

ヌル値キーを処理するには、以下のいずれかを行います。
- KLIST を使用して検索引き数をコーディングします。この場合、KFLD 命令コードの演算項目 2 にヌル標識を指定できます。
- ヌル可能フィールドを検索引き数としてリスト内に (括弧で囲んで) コーディングします。
- ヌル可能フィールドを %KDS で指定されたデータ構造内にコーディングします。

後の 2 つのケースでは、検索引き数に対する %NULLIND() の現在の値が検索に使用されます。

検索引き数の各フィールドの属性は、ファイルまたはレコード・キー内の対応するフィールドの属性と同じでなければなりません。属性には長さ、データ・タイプ、および小数部分の桁数があります。属性は、コンパイラー・リストのキー・フィールド情報データ・テーブルにリストされます。518 ページの『キー・フィールド情報』の例を参照してください。自由形式演算の入出力操作で使用されるリストまたは %KDS の検索引き数で必要になるのは、型の一致のみです。長さおよび形式は、ファイル内で定義されているキーと異なっても構いません。

これらすべてのファイル命令 (CHAIN、DELETE、READE、READPE、SETGT、および SETLL) の中では、ファイルまたはレコードに有効なフィールドの合計数よりも少ないフィールドで検索引き数を指定することもできます。このような検索引き数は部分キーと呼びます。

部分キーの参照

部分キーを指定する場合は、 KFLD 指定の数が少ない KLIST を使用することができます。自由形式演算では、キーの数を示す 2 つ目のパラメーターがある %KDS、または必要なキーをすべて持つ式のリストも使用できます。例えば、ファイルにキーが 3 つあるのに、そのうちの 2 つしか指定する必要がない場合は、次のいずれの方法でも部分キーを指定することができます。

```
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D keys            DS                  LIKEREC(rec : *KEY)
CL0N01Factor1+++++++Opcode&ExtFactor2+++++++Result++++++++Len++D+HiLoEq
C     klist2        KLIST
C                   KFLD                    k1
C                   KFLD                    k2
  /free
       CHAIN klist2 rec;               // KLIST with two KFLD entries
       CHAIN %KDS(keys : 2) rec;       // %KDS with two keys
       CHAIN (name : %char(id_no)) rec; // a list of two expressions
```

部分キーを参照する検索引き数の指定規則は次のとおりです。

- 検索引き数は、ファイルまたはレコードのキーの左端 (高位) のフィールドに対応するフィールドから構成されます。
- 部分キーを参照する検索引き数のキー・リストから省略できるのは右端のフィールドのみです。例えば、ファイルまたはレコードの合計キーがキー・フィールド A、B、および C から構成されている場合には、部分キーを参照する有効な検索引き数はフィールド A、およびフィールド A と B になります。
- 検索引き数内の各フィールドは、ファイルまたはレコード内の対応するキー・フィールドと属性が同じでなければなりません。自由形式演算の入出力操作で使用されるリストまたは %KDS の検索引き数で必要になるのは、型の一致のみです。長さおよび形式は、ファイル内で定義されているキーと異なっても構いません。属性には長さ、データ型、小数部分の桁数、および形式 (例えば、パックまたはゾーン) があります。
- 検索引き数でキー・フィールドの一部分を参照することはできません。

検索引き数で部分キーを参照した場合には、ファイルは検索引き数を満たす最初のレコードに位置付けられるか、あるいは検索引き数を満たす最初のレコードが検索されるレコードです。例えば、SETGT および SETLL 命令では、その命令および検

索引き数を満たすアクセス・パス上の最初のレコードにファイルが位置付けられます。CHAIN 命令では、検索引き数を満たすアクセス・パス上の最初のレコードが検索されます。DELETE 命令では、検索引き数を満たすアクセス・パス上の最初のレコードが削除されます。READE 命令では、アクセス・パス上のそのレコード (指定したタイプのレコード) のキーの一部が検索引き数を満たす場合に、その次のレコードが検索されます。READPE 命令では、アクセス・パス上のそのレコード (指定したタイプのレコード) のキーの一部が検索引き数を満たす場合に、その前のレコードが検索されます。上に述べた命令コードについて詳しくは、「*WebSphere Development Studio: ILE RPG 解説書*」を参照してください。

レコードのブロック化および非ブロック化

デフォルトでは、RPG コンパイラーは次の条件が一致した時に、入力レコードを非ブロック化し、出力レコードをブロック化して、 SEQ または DISK ファイルにおける実行時のパフォーマンスを向上します。

1. ファイルがプログラム記述であるか、あるいは外部記述の場合には、1 つだけのレコード様式をもっている。
2. ファイル仕様書にキーワード RECNO が使用されていない。

   注: RECNO を使う場合、ILE RPG コンパイラーはレコードのブロック化を許可しません。ただし、ファイルが入力ファイルで RECNO が使われると、高速順次アクセスが設定される場合、データ管理は依然、レコードをブロック化することがあります。この場合、更新されたレコードは直ちには見られなくなります。
3. 次の 1 つが真である。
   a. ファイルが出力ファイルである。
   b. ファイルが入出力共用ファイルの場合には、配列またはテーブル・ファイルである。
   c. ファイルが入力専用ファイルである。これは、レコード・アドレス・ファイルでないか、またはレコード・アドレス・ファイルによって処理されないということです。OPEN、CLOSE FEOD、および READ ファイル命令だけを使用します (換言すると、ファイル命令の READE、READPE、SETGT、SETLL、および CHAIN は許可されません)。

RPG コンパイラーは、上の条件を満たすすべての SEQ または DISK ファイルのレコードをブロック化および非ブロック化するための、オブジェクト・プログラム・コードを生成します。特定の OS/400 システム制約事項により、ブロック化および非ブロック化が妨げられることがあります。その場合には、パフォーマンスは改良されません。

ファイルのファイル仕様書にキーワード BLOCK(*YES) を指定することによってレコードのブロック化を明示的に要求することができます。デフォルトのレコードのブロック化とユーザーが要求したレコードのブロック化との相違点は、入力ファイルに BLOCK(*YES) が指定されている時には、入力ファイルで命令 SETLL、SETGT、および CHAIN を使用することができ (上の 3c に示されている条件を参照)、ブロック化が行われることです。BLOCK キーワードが指定されておらず、これらの命令を使用した場合には、レコードのブロック化は行われません。

また、ファイル仕様書にキーワード BLOCK(*NO) を指定することによってデフォルトのレコードのブロック化を防止することができます。BLOCK(*NO) を指定した場合には、コンパイラーによっても、データ管理によってもレコードのブロック化は行われません。キーワード BLOCK が指定されていない場合には、デフォルトのブロック化が上で説明したとおり行われます。

ファイル情報データ構造の入出力および装置固有のフィードバックは、レコードが RPG コンパイラーによってブロック化および非ブロック化されるファイルの各読み取りまたは書き出し後に (ブロック読み取りでの RRN およびキー情報を除く)、更新されません。フィードバック域は、レコード・ブロックが転送されるたびに更新されます。(ファイル情報データ構造について詳しくは、「*WebSphere Development Studio: ILE RPG 解説書*」を参照してください。)

ファイルのブロック化および非ブロック化を防止することによって、有効な更新フィードバック情報を獲得することができます。ブロック化を防止するためには、次の方法の 1 つを使用してください。

- ファイル仕様書に BLOCK(*NO) を指定します。
- 実行時に、SEQONLY(*NO) を指定した CL コマンド OVRDBF (データベース・ファイル一時変更) を使用します。

プログラム記述ディスク・ファイルの使用

プログラム記述ファイルはファイル仕様書の 22 桁目の F によって識別されますが、索引付きファイル、順次ファイル、またはレコード・アドレス・ファイルとして記述することができます。

索引付きファイル

索引付きファイルは、そのアクセス・パスがキーの値によって作成されるプログラム記述 DISK ファイルです。データ記述仕様書を使用して、索引付きファイルのアクセス・パスを作成しなければなりません。

索引付きファイルは、ファイル仕様書の 35 桁目の I によって識別されます。

キー・フィールドは、索引付きファイル中のレコードを指定します。ファイル仕様書では、キー・フィールドの長さを 29 ～ 33 桁目、キー・フィールドの形式を 34 桁目、キー・フィールドの開始位置を KEYLOC キーワードで指定します。

索引付きファイルはキーによる順次処理、限界内順次処理、またはキーによるランダム処理ができます。

有効な検索引き数

プログラム記述ファイルの場合には、検索引き数は単一フィールドでなければなりません。CHAIN および DELETE 命令の場合には、検索引き数は、索引付きファイルのファイル仕様書で定義したキー・フィールドと同じ長さでなければなりません。その他のファイル命令の場合には、検索引き数は部分フィールドとすることができます。

DDS は、使用するフィールドをキー・フィールドとして指定します。ファイル仕様書の KEYLOC キーワードは、最初のキー・フィールドの開始桁を指定します。ファイル仕様書の 29 ～ 33 桁目の項目は、DDS に定義されるキーの長さを指定しなければなりません。

図 159 および 353 ページの図 160 は、DDS を使って索引付きファイルのアクセス・パスをどのように記述するかを示しています。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A.........T.Name++++++.Len++TDpB......Functions++++++++++++++++++++*
A          R FORMATA                   PFILE(ORDDTLP)
A                                      TEXT('Access Path for Indexed +
A                                      File')
A            FLDA          14
A            ORDER          5  0
A            FLDB         101
A          K ORDER
A*
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++++++++++
FORDDTLL   IP   F  118     3PIDISK    KEYLOC(15)
F*
```

図 159. DDS およびそれに対応するファイル仕様書の RPG IV 例外 / エラー処理の詳細な流れ

プログラム記述索引付きファイルのアクセス・パスを作成するには、データ記述仕様書を使用しなければなりません。

論理ファイル ORDDTLL のレコード様式 FORMATA の DDS には、 5 桁の長さのフィールド ORDER がキー・フィールドとして定義され、パックされた形式になっています。キー・フィールドとしての ORDER の定義は、このファイルのキー順アクセスを確立します。その他の 2 つのフィールド FLDA と FLDB は、このレコードの残りの桁が文字フィールドとして記述されています。

プログラム記述入力ファイル ORDDTLL は、ファイル仕様書に索引付きファイルとして記述されています。29 ～ 33 桁目には、DDS で定義したキー・フィールドに必要なレコード内の桁数 (3 桁) を指定しなければなりません。KEYLOC キーワードは、レコード内のキー・フィールドの開始桁として 15 桁目を指定しています。ファイルが 22 桁目の F によってプログラム記述として定義されているので、ILE RPG コンパイラーはコンパイル時にファイルの外部フィールド・レベル記述を取り出しません。したがって、入力仕様書にレコード中のフィールドを記述しなければなりません。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A.........T.Name++++++.Len++TDpB......Functions++++++++++++++++++++*
A          R FORMAT                    PFILE(ORDDTLP)
A                                      TEXT('Access Path for Indexed +
A                                      File')
A            FLDA          14
A            ORDER          5
A            ITEM           5
A            FLDB          96
A          K ORDER
A          K ITEM
```

図 160. (1/2) データ記述仕様書を使用した索引付きファイルのアクセス・パス (複合キー) の定義

この例では、データ記述仕様書が論理ファイル ORDDTLL のレコード様式 FORMAT の 2 つのキー・フィールドを定義します。プログラム記述索引付きファイルの複合キーとして 2 つのフィールドを使用する場合には、そのキー・フィールドはレコード内で隣接していなければなりません。

ファイル仕様書では、キー・フィールドの長さが 29 ～ 33 桁目に 10 (ORDER および ITEM フィールドに必要な結合桁数) として指定されています。キー・フィールドの開始桁は、キーワード KEYLOC (44 桁目から) を使用して 15 として記述されています。開始桁は、最初のキー・フィールドの最初の桁を指定しなければなりません。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++++++++++
FORDDTLL   IP   F  120    10AIDISK    KEYLOC(15)
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
DKEY         DS
D K1                        1      5
D K2                        6     10
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   MOVE      ORDER         K1
C                   MOVE      ITEM          K2
C     KEY           CHAIN     ORDDTLL                          99
```

図 161. (2/2) データ記述仕様書を使用して定義する索引付きファイルのアクセス・パス (複合キー)

DDS で複合キーを指定した場合には、ファイルとの CHAIN をするためにプログラム中で検索引き数を作成しなければなりません。(プログラム記述ファイルの場合には、KLIST を使用することができません。) 1 つの方法は、DDS で定義したキー・フィールドと等しいサブフィールドをもったデータ構造 (定義仕様書を使用して) を作成することです。次に演算において、サブフィールドをキー・フィールドの値と等しく設定して、データ構造名を CHAIN 命令の検索引き数として使用します。

この例では、MOVE 命令はサブフィールド K1 および K2 をそれぞれ ORDER および ITEM の値と同じに設定します。次にデータ構造名 (KEY) が CHAIN 命令の検索引き数として使用されます。

順次ファイル

順次ファイルは、ファイルの中のレコードの順序が、レコードがファイルに入れられた順序 (すなわち、到着順) に基づいているファイルです。例えば、ファイル内に 10 番目に入れられたレコードは 10 番目のレコード位置を占めています。

順次ファイルは、相対レコード番号によるランダム処理、連続処理、またはレコード・アドレス・ファイルによって処理することができます。SETLL または SETGT 命令コードを使用して、ファイルに限界を設定することができます。

レコード・アドレス・ファイル

レコード・アドレス・ファイルを使用して他のファイルを処理することができます。レコード・アドレス・ファイルには、(1) ファイルの限界内順次処理で使用する限界値レコード、または (2) 相対レコード番号によるファイルの処理で使用する相対レコード番号が入っています。レコード・アドレス・ファイル自体の処理は順次処理でなければなりません。

レコード・アドレス・ファイルは、ファイル仕様書の 18 桁目の R で識別されます。レコード・アドレス・ファイルに相対レコード番号があれば、35 桁目に T がなければなりません。レコード・アドレス・ファイルによって処理されるファイルの名前はファイル仕様書に指定しなければなりません。このファイルはキーワード RAFDATA(*ファイル名*) を使用して識別します。

限界値レコード

限界内順次処理の場合には、レコード・アドレス・ファイルには限界値レコードが入ります。限界値レコードには読み取るファイルのレコードの最低のキーおよび最高のキーが含まれます。

レコード・アドレス・ファイル中の限界値レコードの形式は次のとおりです。

- 低いキーがレコードの 1 桁目で始まり、高いキーが低いキーの直後に続きます。2 つのキーの間にブランクを入れてはなりません。
- レコード・アドレス・ファイルの各レコードには 1 つの限界セットしか入れることはできません。レコード長は、レコード・キーの長さの 2 倍かそれより大きくなければなりません。
- 限界値レコード中の低いキーおよび高いキーの長さは同じでなければなりません。キーの長さは処理するファイルのキー・フィールドの長さと等しくなければなりません。
- レコード・キー・フィールドに等しい長さのブランク項目があると、ILE RPG コンパイラーはレコード・アドレス・ファイルの次のレコードを読みます。

相対レコード番号

相対レコード番号処理の場合には、レコード・アドレス・ファイルに相対レコード番号が入っています。処理中のファイルから検索された各レコードはレコード・アドレス・ファイルの相対レコード番号に基づいています。相対レコード番号が入っているレコード・アドレス・ファイルは、限界値範囲内処理に使用することができません。レコード・アドレス・ファイルの各相対レコード番号は、各フィールドに相対レコード番号が入っている複数バイトの 2 進数フィールドです。

レコード・アドレス・ファイルの長さは、ファイルのソースによって、4、3、またはブランクとして指定することができます。iSeries 環境からのレコード・アドレス・ファイルを使う時は、各フィールドの長さは 4 バイトなので、レコード・アドレス・ファイルの長さは 4 に指定します。システム/36 環境用™ で作られたレコード・アドレス・ファイルを使う時は、各フィールドの長さは 3 バイトなので、レコード・アドレス・ファイルは 3 と指定します。レコード・アドレス・ファイルの長さをブランクとして指定した場合には、コンパイラーが、実行時にプライマリー・レコードの長さを検査し、レコード・アドレス・ファイルの長さを 3 バイトとして取り扱うかまたは 4 バイトとして取り扱うかを決定します。

マイナス 1 (-1 または 16 進数の FFFFFFFF) の相対レコード番号の値によって、相対レコード・アドレス・ファイル・レコードの使用を停止します。レコード・アドレス・ファイルのすべてのレコードが処理された時に、ファイルの終わりになります。

ディスク・ファイルの処理方式

ディスク・ファイルの処理方式には次の方式があります。

- 連続処理
- キーによる順次処理
- キーによるランダム処理
- 限界内順次処理
- 相対レコード番号処理

表 34 には、各種のファイル・タイプおよび処理方式に対するファイル仕様書の 28 桁目、34 桁目、および 35 桁目に指定できる項目を示してあります。その後で各処理方式について説明します。

表 34. DISK ファイルの処理方式

処理方式	**限界処理 (28 桁目)**	**レコード・アドレス・タイプ (34 桁目)**	**ファイル編成 (35 桁目)**
外部記述ファイル			
キーあり			
順次処理	ブランク	K	ブランク
ランダム処理	ブランク	K	ブランク
限界内順次処理 (レコード・アドレス・ファイルによる)	L	K	ブランク
キーなし			
ランダム処理/連続処理	ブランク	ブランク	ブランク

表 34. DISK ファイルの処理方式 (続き)

処理方式	限界 処理 (28 桁目)	レコード・ アドレス・ タイプ (34 桁目)	ファイル 編成 (35 桁目)
プログラム記述ファイル			
キーあり (索引付きファイル)			
順次処理	ブランク	A、D、G、P、T、Z、または F	I
ランダム処理	ブランク	A、D、G、P、T、Z、または F	I
限界内順次処理 (レコード・アドレス・ファイルによる)	L	A、D、G、P、T、Z、または F	I
キーなし			
	ブランク	ブランク	ブランク
ランダム処理/連続処理	ブランク	ブランク	ブランク
レコード・アドレス・ファイルによる	ブランク	ブランク	T
レコード・アドレス・ファイルとして (相対レコード番号) レコード・アドレス限界値ファイルとして	ブランク	A、D、G、P、T、Z、F、またはブランク	ブランク

連続処理

連続処理中は、レコードはファイルにある順序で読み取られます。

ランダム関数 (例えば SETLL、SETGT、CHAIN、ADD など) を使わない出力および入力ファイルでは、ILE RPG コンパイラーはデフォルトの値を使うか、SEQONLY(*YES) が CL コマンド OVRDBF に指定されている場合と同じように (データベース・ファイル一時変更) 作動します (ILE RPG コンパイラーは更新ファイルに対しては、SEQONLY(*YES) が指定された場合と同じようには作動しません)。SEQONLY(*YES) によって、複数レコードを内部データ管理バッファーに入れることができます。その後、レコードは入力により一度に 1 つずつ ILE RPG コンパイラーに渡されます。

同一ジョブ内または活動化グループで、2 つの論理ファイルが同じ物理ファイルを使用し、1 つのファイルが連続して処理され、もう 1 つのファイルが更新用にランダムに処理される場合には、プログラムに渡されるバッファーに既に入っているレコードは更新することができます。この場合には、レコードが連続ファイルから処理される時に、レコードには更新済みデータは反映されません。この問題を防止するためには、 CL コマンド OVRDBF を使用してオプション SEQONLY(*NO) を指定し、連続処理ファイルでは複数レコードを転送しないように指示します。

順次のみの処理についての詳細は、
http://publib.boulder.ibm.com/html/as400/infocenter.html で **iSeries 400 Information Center** の中の『データベースおよびファイル・システム』カテゴリーの『*DB2 UDB for iSeries*』の節を参照してください。

キーによる順次処理

キーによる順次処理方式では、レコードはファイルからキー順に読み取られます。

キーによる順次処理方式はプライマリー、セカンダリー、または全手順ファイルとして使用されるキー付きファイルで有効になります。

出力ファイルおよび、ランダム関数 (例えば、SETLL、SETGT、CHAIN、ADD など) を使わず、かつ唯一のレコード様式を持つ入力ファイルについては、ILE RPG コンパイラーはデフォルトの値を取るか、SEQONLY(*YES) が CL コマンド OVRDBF に指定されている場合と同じように作動します (ILE RPG コンパイラーは更新ファイルに対しては、SEQONLY(*YES) が指定された場合と同じようには作動しません)。SEQONLY(*YES) によって、複数レコードを内部データ管理バッファーに入れることができます。その後、レコードは入力により一度に 1 つずつ ILE RPG コンパイラーに渡されます。

同一ジョブ内で、2 つのファイルが同じ物理ファイルを使用し、1 つのファイルが連続して処理され、もう 1 つのファイルがランダム更新用に処理される場合、プログラムに渡されるバッファーに既に入っているレコードは、更新することができます。この場合には、レコードが順次ファイルから処理される時に、レコードには更新済みデータは反映されません。この問題を防止するためには、 CL コマンド OVRDBF を使用してオプション SEQONLY(*NO) を指定し、順次処理ファイルでは複数レコードを転送しないように指示します。

順次のみの処理についての詳細は、
http://publib.boulder.ibm.com/html/as400/infocenter.html で **iSeries 400 Information Center** の中の『データベースおよびファイル・システム』カテゴリーの『*DB2 UDB for iSeries*』の節を参照してください。

キーによる順次処理の例

次の 3 つの例は、キーによる順次処理方式のデータ処理の異なる使用法を示したものです。

データ記述仕様書 (DDS): 358 ページの図 162 および 358 ページの図 163 は、例で使われる物理ファイルのデータ記述仕様書 (DDS) を示します。 358 ページの図 164 は、最初の 3 つの例で使われる論理ファイルの DDS を示します。

```
A*****************************************************************
A* 説明:  これは物理ファイル EMPMST の DDS です。                  *
A*        これには 1 つのレコード様式 EMPREC が入っています。      *
A*        このファイルには、会社の各社員ごとに 1 レコードが       *
A*        入っています。                                          *
A*****************************************************************
A*
A          R EMPREC
A            ENUM           5  0       TEXT('EMPLOYEE NUMBER')
A            ENAME         20          TEXT('EMPLOYEE NAME')
A            ETYPE          1          TEXT('EMPLOYEE TYPE')
A            EDEPT          3  0       TEXT('EMPLOYEE DEPARTMENT')
A            ENHRS          3  1       TEXT('EMPLOYEE NORMAL WEEK HOURS')
A          K ENUM
```

図 162. データベース・ファイル EMPMST (物理ファイル) の DDS

```
A*****************************************************************
A* 説明:  これは、物理ファイル TRWEEK の DDS です。                *
A*        これには 1 つのレコード様式 RCWEEK が入っています。      *
A*        このファイルには、勤務時間報告書作成システムに対して    *
A*        作成されるすべての週間項目が入っています。              *
A*****************************************************************
A*
A          R RCWEEK
A            ENUM           5  0       TEXT('EMPLOYEE NUMBER')
A            WEEKNO         2  0       TEXT('WEEK NUMBER OF CURRENT YEAR')
A            EHWRK          4  1       TEXT('EMPLOYEE HOURS WORKED')
A          K ENUM
A          K WEEKNO
```

図 163. データベース・ファイル TRWEEK (物理ファイル) の DDS

```
A*****************************************************************
A* 関連ファイル:  EMPMST    (物理ファイル)                       *
A*                TRWEEK    (物理ファイル)                       *
A*        説明:   これは論理ファイル EMPL1 の DDS です。         *
A*                これには、2 つのレコード様式                   *
A*                EMPREC と RCWEEK が入っています。              *
A*****************************************************************
A          R EMPREC                    PFILE(EMPMST)
A          K ENUM
A*
A          R RCWEEK                    PFILE(TRWEEK)
A          K ENUM
A          K WEEKNO
```

図 164. データベース・ファイル EMPL1 (論理ファイル) の DDS

プログラム例 1 (1 次ファイルを使用するキーによる順次処理): この例では、社員マスター・レコード (EMPREC) および週間勤務時間レコード (RCWEEK) が同じ論理ファイル EMPL1 に含まれています。EMPL1 ファイルはプライマリー入力ファイルとして定義され、キーによる順次で読み取られます。このファイルのデータ記述仕様書で、EMPREC レコードのキーは ENUM (社員番号) フィールドとして定義され、RCWEEK レコードのキーは ENUM フィールド、プラス WEEKNO (週番号) フィールドとして、すなわち複合キーとして定義されます。

```
     ****************************************************************
     *  プログラム名:  YTDRPT1                                      *
     *  関連ファイル:  EMPL1    (論理ファイル)                      *
     *                 PRINT    (プリンター・ファイル)              *
     *          説明:  このプログラムはキー順方式を使用             *
     *                 するレコードの処理例です。                   *
     *                 このプログラムは各社員の情報および週間勤務   *
     *                 時間を印刷します。                           *
     ****************************************************************
    FPRINT     O    F   80        PRINTER
    FEMPL1     IP   E           K DISK
     *  各レコードにレコード識別標識を割り当てます。これらのレコード識別標識を
     *  使用して、異なるレコード・タイプの処理を
     *  制御します。
    IEMPREC        01
    I*
    IRCWEEK        02
    I*

     *  EMPL1 ファイルでは、キーによって有効な社員番号が順次読み取り
     *  されるため、RCWEEK レコードの ENUM は最後に検索された
     *  EMPREC の ENUM と同じでなければなりません。これは、EMPREC
     *  レコードの ENUM をフィールド EMPNO に保管し、これを
     *  RCWEEK レコードから読み取られた ENUM と比較することによって
     *  チェックする必要があり、そのチェックがここで行なわれます。
     *  ENUM が有効な場合に *IN12 がオンに設定されます。*IN12 は
     *  RCWEEK レコードの印刷の制御に使用されます。

    C                   SETOFF                                         12
    C   01              MOVE      ENUM          EMPNO             5 0
    C*
    C                   IF        (*IN02='1') AND (ENUM=EMPNO)
    C                   SETON                                          12
    C                   ENDIF

OPRINT     H    1P                    2  6
     O                                             40 'EMPLOYEE WEEKLY WORKING '
     O                                             52 'HOURS REPORT'
     O          H    01                     1
     O                                             12 'EMPLOYEE: '
     O                        ENAME                32
     O          H    01                     1
     O                                             12 'SERIAL #: '
     O                        ENUM                 17
     O                                             27 'DEPT: '
     O                        EDEPT                30
     O                                             40 'TYPE: '
     O                        ETYPE                41
     O          H    01                     1
     O                                             20 'WEEK #'
     O                                             50 'HOURS WORKED'
     O          D    12                     1
     O                        WEEKNO               18
     O                        EHWRK          3     45
```

図 165. キーによる順次処理、例 1

プログラム例 2 (READ を使用したキーによる順次処理): この例は先の例と同じですが、異なる点は、EMPL1 ファイルが全手順ファイルとして定義されていて、ファイルの読み取りが READ 命令コードによって行われるということです。

```
     ****************************************************************
     *  プログラム名:  YTDRPT2                                      *
     *  関連ファイル:  EMPL1    (論理ファイル)                      *
     *                 PRINT    (プリンター・ファイル)              *
     *          説明:  このプログラムはキー順方式を使用             *
     *                 使用するレコードの処理例を示しています。     *
     *                 このプログラムは各社員の情報および週間勤務   *
     *                 時間を印刷します。                           *
     ****************************************************************
    FPRINT     O    F   80        PRINTER
    FEMPL1     IF   E           K DISK
     *  2 つのレコード (EMPREC と RCWEEK) は同じファイルに入っており、
     *  レコード識別標識は各レコードに割り当てられています。
     *  レコード識別標識は異なるレコード・タイプの処理の制御に使用されます。
     *  全手順ファイルの場合には、制御レベルまたは突き合わせフィールドを
     *  指定することはできません。
    IEMPREC        01
    I*
    IRCWEEK        02
    I*

     *  READ 命令コードは、EMPL1 ファイルからレコードを読み取ります。
     *  ファイルの終わり標識は 58 ～ 59 桁目に指定します。ファイルの終わり標識 99
     *  は READ 命令によってオンに設定され、プログラムは
     *  EOFEND タグに分岐して、ファイル終わりルーチンを処理
     *  します。

    C                   SETOFF                                      12
    C                   READ      EMPL1                                 99
    C   99              GOTO      EOFEND
    C*
    C   01              MOVE      ENUM          EMPNO             5 0
    C*
    C                   IF        (*IN02='1') AND (ENUM=EMPNO)
    C                   SETON                                       12
    C                   ENDIF

     *  EMPL1 は全手順ファイルとして定義されているため、最終レコードを
     *  処理した後でプログラムを終了するために標識 *INLR をオンに
     *  設定する必要があります。

    C     EOFEND        TAG
    C   99              SETON                                       LR
```

図 166. キーによる順次処理、例 2 (1/2)

```
OPRINT     H    1P                    2  6
O                                          40 'EMPLOYEE WEEKLY WORKING '
O                                          52 'HOURS REPORT'
O          H    01                    1
O                                          12 'EMPLOYEE: '
O                      ENAME               32
O          H    01                    1
O                                          12 'SERIAL #: '
O                      ENUM                17
O                                          27 'DEPT: '
O                      EDEPT               30
O                                          40 'TYPE: '
O                      ETYPE               41
O          H    01                    1
O                                          20 'WEEK #'
O                                          50 'HOURS WORKED'
O          D    12                    1
O                      WEEKNO              18
O                      EHWRK         3     45
```

図 166. キーによる順次処理、例 2 (2/2)

プログラム例 3 (レコード突き合わせ手法): この例では、TRWEEK ファイルがセカンダリー入力ファイルとして定義されています。EMPREC および RCWEEK レコードは、突き合わせレコードとして処理され、両方のレコードの ENUM フィールドには突き合わせレベル値 M1 に割り当てられています。異なるレコード・タイプの処理を制御するために、レコード識別標識 01 と 02 が割り当てられています。

```
*********************************************************************
*  プログラム名:  YTDRPT5                                           *
*  関連ファイル:  EMPMST   (物理ファイル)                           *
*                 TRWEEK   (物理ファイル)                           *
*                 PRINT    (プリンター・ファイル)                   *
*          説明:  このプログラムはキー順方式を使用                  *
*                 するレコードの処理例です。                        *
*                 このプログラムは各社員の情報および週間勤務       *
*                 および超過時間を印刷                              *
*                 します。                                          *
*********************************************************************
FPRINT     O    F   80        PRINTER
FEMPMST    IP   E           K DISK
FTRWEEK    IS   E           K DISK
IEMPREC         01
I                                        ENUM            M1
IRCWEEK         02
I                                        ENUM            M1
```

図 167. キーによる順次処理、例 3 (1/2)

```
C   01              Z-ADD     0             TOTHRS            5 1
C   01              Z-ADD     0             TOTOVT            5 1
C   01              SETOFF                                       12
C*
C   MR              IF        (*IN02='1')
C                   ADD       EHWRK         TOTHRS
C     EHWRK         SUB       ENHRS         OVTHRS            4 111
C   11              ADD       OVTHRS        TOTOVT
C                   SETON                                        12
C                   ENDIF
OPRINT     H    1P                      2  6
O                                                50 'YTD PAYROLL SUMMARY'
O          D    01                      1
O                                                12 'EMPLOYEE: '
O                        ENAME                   32
O          D    01                      1
O                                                12 'SERIAL #: '
O                        ENUM                    17
O                                                27 'DEPT: '
O                        EDEPT                   30
O                                                40 'TYPE: '
O                        ETYPE                   41
O          D    02 MR                   1
O                                                 8 'WEEK #'
O                        WEEKNO                  10
O                                                32 'HOURS WORKED = '
O                        EHWRK         3         38
 *  これらの 2 明細出力行は、*IN01 がオンで、一致するレコードが
 *  見付からない (つまりその社員レコードの RCWEEK レコードが見付からない)
 *  場合に処理されます。この場合には、明らかに
 *  合計フィールド (TOTHRS と TOTOVT) はゼロと等しくなります。
O          D    01NMR                   1
O                                                70 'YTD HOURS WORKED = '
O                        TOTHRS        3         78
O          D    01NMR                    1
O                                                70 'YTD OVERTIME HOURS = '
O                        TOTHRS        3         78

 *  これら 2 行の合計出力行は、明細演算を実行する前に処理されます。
 *  したがって、指定された標識がオンの場合は、最後に
 *  検索されたレコードの社員の合計フィールド (TOTHRS と TOTOVT)
 *  が印刷されます。

O           T    01 12                   1
O          OR    LR 12
O                                                70 'YTD HOURS WORKED = '
O                        TOTHRS        3         78
O           T    01 12                    1
O          OR    LR 12
O                                                70 'YTD OVERTIME HOURS = '
O                        TOTOVT        3         78
```

図 167. キーによる順次処理、例 3 (2/2)

キーによるランダム処理

キーによるランダム処理方式では、読み取るレコードのキーを識別する検索引き数を、CHAIN 命令の演算仕様書の演算項目 1 に指定します。 364 ページの図 169 は、キーによってランダムに処理される外部記述 DISK ファイルの例を示します。指定したレコードは明細演算時または合計演算時にファイルから読み取られます。

キーによるランダム処理の処理方式は、入力ファイルまたは更新ファイルとして指定されている全手順ファイルに有効です。

外部記述ファイルについては、ファイル仕様書の 34 桁目に K を入れ、ファイルがキーについて作成されるアクセス・パスにしたがって処理されるということを示さなければなりません。

ファイルのデータ記述仕様書 (DDS) にはキー値が入っているフィールド (キー・フィールド) を指定します。ファイル仕様書の 35 桁目はブランクでなければなりません。

プログラム記述ファイルは、索引付きファイル (35 桁目に I) として指定されていなければならず、また、ファイル仕様書の 34 桁目は A、D、G、P、T、または Z を含む必要があります。キー・フィールドの長さはファイル仕様書の 29 - 33 桁で識別され、キー・フィールドの開始位置は KEYLOC キーワードに指定されます。データ記述仕様書は、プログラム記述入力ファイルのアクセス・パスを作成するために使用しなければなりません。(351 ページの『索引付きファイル』を参照してください。)

キーによるランダム処理の例

以下は、キーによるランダム処理のデータ処理方式の使用法を示す例です。358 ページの図 162 および図 168 は、EMSTUPD (364 ページの図 169) によって使われる物理ファイルのデータ記述仕様書 (DDS) を示します。

```
A*********************************************************************
A*  関連プログラム: EMSTUPD                                          *
A*  説明:  これは物理ファイル CHANGE の DDS です。                   *
A*         これには 1 つのレコード様式 CHGREC が入っています。       *
A*         このファイルには、EMPMST ファイルの更新に使用する         *
A*         新しいデータが入っています。                              *
A*********************************************************************
A*
A          R CHGREC
A            ENUM           5  0       TEXT('EMPLOYEE NUMBER')
A            NNAME         20          TEXT('NEW NAME')
A            NTYPE          1          TEXT('NEW TYPE')
A            NDEPT          3  0       TEXT('NEW DEPARTMENT')
A            NNHRS          3  1       TEXT('NEW NORMAL WEEK HOURS')
A          K ENUM
```

図 168. データベース・ファイル CHANGE (物理ファイル) の DDS

プログラム例: この例では、EMPMST ファイルが全手順更新ファイルとして定義されています。更新ファイル CHANGE はキーによって処理されます。外部記述ファイル (EMPMST と CHANGE) のそれぞれの DDS が、ENUM フィールドをキー・フィールドとして指定します。読み取り / 更新処理は、演算仕様書に指定された命令によってすべて制御されます。

```
     *********************************************************************
     *  プログラム名:  EMSTUPD                                           *
     *  関連ファイル:  EMPMST   (物理ファイル)                           *
     *                 CHANGE   (物理ファイル)                           *
     *          説明:  このプログラムはキーによるランダム処理方式を      *
     *                 使用するレコード処理例を示しています。CHAIN       *
     *                 命令コードが使用されます。                        *
     *                 物理ファイル CHANGE には、                        *
     *                 EMPMST ファイルに行なわれた変更がすべて入って     *
     *                 います。レコード様式名は CHGREC です。            *
     *                 CHGREC にはブランクのままのフィールドもあり       *
     *                 ますが、その場合にはそのフィールドには変更は      *
     *                 行なわれていません。                              *
     *********************************************************************
    FCHANGE    IP   E           K DISK
    FEMPMST    UF   E           K DISK
     *  各レコードがプライマリー入力ファイルから読み取られるたびに、
     *  社員番号 (ENUM) が検索引き数として使用され、EMPMST
     *  ファイルの対応するレコードに連鎖されます。
     *  *IN03 は、対応するレコードが見付からなかった場合にオンに設定
     *  されます。これは正しくない ENUM が CHGREC レコードに入れられた時に
     *  起こります。
    C     ENUM          CHAIN     EMPREC                         03
    C   03              GOTO      NEXT
    C     NNAME         IFNE      *BLANK
    C                   MOVE      NNAME         ENAME
    C                   ENDIF
    C     NTYPE         IFNE      *BLANK
    C                   MOVE      NTYPE         ETYPE
    C                   ENDIF
    C     NDEPT         IFNE      *ZERO
    C                   MOVE      NDEPT         EDEPT
    C                   ENDIF
    C     NNHRS         IFNE      *ZERO
    C                   MOVE      NNHRS         ENHRS
    C                   ENDIF
    C                   UPDATE    EMPREC
    C*
    C     NEXT          TAG
```

図 169. 外部記述ファイルのキーによるランダム処理

限界内順次処理

レコード・アドレス・ファイルによる限界内順次処理は、ファイル仕様書の 28 桁目の L で指定され、キー順アクセスが行われるファイルに有効です。

限界内順次処理は、プライマリー、セカンダリー、または全手順ファイルとして指定した入力または更新ファイルに対して指定することができます。ファイルは外部記述または、(索引付き) プログラム記述です。ファイルのキーは昇順でなければなりません。

レコード・アドレス・ファイルを使用してファイルを限界内で順次処理するためには、プログラムは次のものを読み取ります。

- レコード・アドレス・ファイルの限界値レコード。
- 限界値レコードの低いレコード・キーより大きいか等しいキー、および高いレコード・キーより小さいか等しいキーを用いて限界内処理をされるファイルのレコ

ード。レコード・アドレス・ファイルによって提供される 2 つの限界値レコードが等しい場合には、指定されたキーを持ったレコードだけが検索されます。

プログラムは、レコード・アドレス・ファイルが終わりになるまでこの手順を繰り返します。

限界内順次処理の例

366 ページの図 170 は、限界内で順次処理される索引付きファイルの例を示したものです。 367 ページの図 172 は、プログラム記述ファイルの代わりに外部記述ファイルを使った、同じ例を示します。

358 ページの図 162 には、プログラム ESWLIM1 (366 ページの図 170) および ESWLIM2 (367 ページの図 172) によって使用される物理ファイルのデータ記述仕様書 (DDS) を示してあります。

プログラム例 1 (限界内順次処理): EMPMST は、レコード・アドレス・ファイル LIMITS によって限界内順次 (28 桁目の L) に処理されます。レコード・アドレス・ファイルからの限界の各セットは、処理する EMPMST ファイル中のレコードの低および高の社員番号から構成されています。社員番号キー・フィールド (ENUM) の長さが 5 桁なので、限界の各セットは 5 桁のキーから構成されます。(ENUM はパックされた形式なので、5 桁ではなく、3 桁が必要であることに注意してください。)

```
     *********************************************************************
     *  プログラム名:  ESWLIM1                                           *
     *  関連ファイル:  EMPMST   (物理ファイル)                           *
     *                 LIMITS   (物理ファイル)                           *
     *                 PRINT    (プリンター・ファイル)                   *
     *         説明 :  このプログラムは、索引付きファイルの限界内       *
     *                 順次処理を示します。                              *
     *                 このプログラムは、社員番号がファイル LIMITS       *
     *                 で指定された限界内にある社員の情報を              *
     *                 印刷します。                                      *
     *********************************************************************
    FLIMITS    IR   F    6     3  DISK    RAFDATA(EMPMST)
    FEMPMST    IP   F   28L    3PIDISK    KEYLOC(1)
    FPRINT     O    F   80        PRINTER
     *  入力仕様書を使用して、プログラム記述ファイル EMPMST のレコードを
     *  記述する必要があります。
    IEMPMST    NS  01
    I                              P    1    3 0ENUM
    I                                   4   23  ENAME
    I                                  24   24  ETYPE
    I                              P   25   26 0EDEPT

     *  EMPMST は、限界値の各セット内で処理され、対応するレコードが印刷
     *  されます。レコード・アドレス・ファイル LIMITS がファイルの終わりに
     *  達すると、EMPMST ファイルの処理は完了します。

    OPRINT     H    1P                    1
    O                                          12 'SERIAL #'
    O                                          22 'NAME'
    O                                          45 'DEPT'
    O                                          56 'TYPE'
    O          D    01                    1
    O                         ENUM             10
    O                         ENAME            35
    O                         EDEPT            45
    O                         ETYPE            55
```

図 170. 外部記述ファイルの、限界内順次処理

プログラム例 2 (限界内順次処理): 図 171 には、プログラム ESWLIM2 (367 ページの図 172) によって使用されるレコード・アドレス限界値ファイルのデータ記述仕様書 (DDS) を示してあります。

```
    A*********************************************************************
    A*   関連プログラム:  ESWLIM                                         *
    A*             説明:  これは物理ファイル LIMITS の DDS です。        *
    A*                    これには、1 つのレコード様式 LIMIT が          *
    A*                    入っています。                                 *
    A*********************************************************************
    A
    A          R LIMIT
    A            LOW           5  0
    A            HIGH          5  0
```

図 171. レコード・アドレス・ファイル LIMITS (物理ファイル) の DDS

このプログラムは、前のプログラムと同じジョブを実行します。唯一の違いは、この物理ファイル EMPMST はプログラム記述ファイルの代わりに、外部記述ファイルとして定義されていることです。

```
     ****************************************************************
     *  プログラム名:  ESWLIM2                                      *
     *  関連ファイル:  EMPMST   (物理ファイル)                      *
     *                 LIMITS   (物理ファイル)                      *
     *                 PRINT    (プリンター・ファイル)              *
     *         説明 :  このプログラムは、索引付きファイルの限界内  *
     *                 限界内順次処理を                             *
     *                 示します。                                   *
     *                 このプログラムは、社員番号がファイル LIMITS  *
     *                 で指定された限界内にある社員の情報を         *
     *                 印刷します。                                 *
     ****************************************************************
    FLIMITS    IR   F    6     3  DISK    RAFDATA(EMPMST)
    FEMPMST    IP   E     L     K DISK
    FPRINT     O    F   80        PRINTER
     *  外部記述ファイルでは入力仕様書はオプションです。
     *  ここで、レコード様式 EMPREC のレコード識別標識 *IN01 を
     *  定義し、このレコードの処理を
     *  制御します。
    IEMPREC        01

    OPRINT     H    1P                    1
    O                                         12 'SERIAL #'
    O                                         22 'NAME'
    O                                         45 'DEPT'
    O                                         56 'TYPE'
    O          D    01                    1
    O                       ENUM               10
    O                       ENAME              35
    O                       EDEPT              45
    O                       ETYPE              55
    O*
```

図 172. プログラム記述ファイルの限界内順次処理

相対レコード番号による処理

相対レコード番号による入力または更新のランダム処理ができるのは、全手順ファイルだけです。所要のレコードは、CHAIN 命令コードによってアクセスされます。

相対レコード番号によって、ファイルの先頭からの相対的なレコードの位置が識別されます。例えば、1 番目、5 番目、および 7 番目のレコードの相対レコード番号は、それぞれ 1、5、および 7 となります。

外部記述ファイルの場合には、入力または更新の相対レコード番号による処理は、ファイル仕様書の 34 桁目をブランクにし、さらに CHAIN 命令コードを使用することによって決められます。相対レコード番号による出力処理は、34 桁目のブランクおよびそのファイルのファイル仕様書行の RECNO キーワードの使用によって判別されます。

新しいレコードをこのファイルに追加する位置を指定する相対レコード番号が入っている数字フィールドを指定するためには、ファイル仕様書で RECNO キーワードを使用してください。RECNO フィールドは、小数点以下の桁数のない数値として定義しなければなりません。このフィールドの長さは、ファイルの最大レコード番

号が十分に入る長さでなければなりません。ファイルに新しいレコードを入れる場合には、出力仕様書または WRITE 命令を使用して RECNO フィールドを指定しなければなりません。

相対レコード番号によってレコードをファイルに更新または追加する時には、レコードは既にメンバーに入っていなければなりません。更新の場合には、その位置は有効な既存のレコードでなければなりません。新しいレコードの場合には、その位置が削除済みレコードの位置でなければなりません。

CL コマンド INZPFM を使用して相対レコード番号による使用のためにレコードを初期設定することができます。すべての取り出し命令、またはファイル位置の再指定命令 (例えば、SETLL、CHAIN、READ) については、現行の相対レコード番号は RECNO フィールドに入れられます。

有効なファイル命令

表 35 には、キーによって処理される DISK ファイルに有効なファイル命令コードが、また 369 ページの表 36 にはキーによらない方式によって処理される DISK ファイルに有効なファイル命令コードがそれぞれ示されています。これらの図に示された命令は、外部記述 DISK ファイルおよびプログラム記述 DISK ファイルに有効です。

プログラムの実行前に、ファイルを別のファイルに一時変更することができます。特に、プログラム内の順次ファイルを外部記述のキー付きファイルに指定変更することができます。 (ファイルは順次ファイルとして処理されます。) キー・フィールドが合致している場合には、プログラム中のキー付きファイルを別のキー付きファイルに一時変更することもできます。例えば、ファイルの一時変更は、プログラム内で指定されたキー・フィールドより短いキー・フィールドをもっている必要はありません。

注: データベース・レコードを削除した時には、物理レコードは削除済みとして印が付けられます。削除済みレコードは、物理ファイル・メンバー初期設定 (INZPFM) コマンドを使用して削除済みレコードのあるファイルを初期設定した場合に、そのファイルで発生することがあります。レコードが削除されると、そのレコードを読み取ることはできません。しかし、相対レコード番号を使用してそのレコードに位置指定し、その内容に一時変更することはできます。

表 35. キーによる処理方式で有効なファイル命令 (キーによるランダム処理、キーによる順次処理、限界内順次処理)

ファイル記述 仕様書位置					演算仕様書の桁
17	18	20	28[1]	34[2]	26-35
I	P/S			K/A/P/G/ D/T/Z/F	CLOSE、FEOD、FORCE
I	P/S	A		K/A/P/G/ D/T/Z/F	WRITE、CLOSE、FEOD、FORCE
I	P/S		L	K/A/P/G/ D/T/Z/F	CLOSE、FEOD、FORCE

表 35. キーによる処理方式で有効なファイル命令 (キーによるランダム処理、キーによる順次処理、限界内順次処理) (続き)

ファイル記述仕様書位置					演算仕様書の桁
U	P/S			K/A/P/G/D/T/Z/F	UPDATE、DELETE、CLOSE、FEOD、FORCE
U	P/S	A		K/A/P/G/D/T/Z/F	UPDATE、DELETE、WRITE、CLOSE、FEOD、FORCE
U	P/S		L	K/A/P/G/D/T/Z/F	UPDATE、DELETE、CLOSE、FEOD、FORCE
I	F			K/A/P/G/D/T/Z/F	READ、READE、READPE、READP、SETLL、SETGT、CHAIN、OPEN、CLOSE、FEOD
I	F	A		K/A/P/G/D/T/Z/F	WRITE、READ、READPE、READE、READP、SETLL、SETGT、CHAIN、OPEN、CLOSE、FEOD
I	F		L	K/A/P/G/D/T/Z/F	READ、OPEN、CLOSE、FEOD
U	F			K/A/P/G/D/T/Z/F	READ、READE、READPE、READP、SETLL、SETGT、CHAIN、UPDATE、DELETE、OPEN、CLOSE、FEOD
U	F	A		K/A/P/G/D/T/Z/F	WRITE、UPDATE、DELETE、READ、READE、READPE、READP、SETLL、SETGT、CHAIN、OPEN、CLOSE、FEOD
U	F		L	K/A/P/G/D/T/Z/F	READ、UPDATE、DELETE、OPEN、CLOSE、FEOD
O	ブランク	A		K/A/P/G/D/T/Z/F	WRITE (新規レコードのファイルへの追加)、OPEN、CLOSE、FEOD
O	ブランク			K/A/P/G/D/T/Z/F	WRITE (新規ファイルの初期ロード)[3]、OPEN、CLOSE、FEOD

注:

1. 入力または更新ファイルに対して、レコード・アドレス・ファイルによる限界内順次処理を指定するには、`L` を 28 桁目に指定しなければなりません。
2. 外部記述ファイルでは 34 桁目に `K` を、プログラム記述ファイルでは 34 桁目に `A`、`P`、`G`、`D`、`T`、`Z`、 または `F` を、また 35 桁目に `I` が必要です。
3. レコードの新規ファイルへの初期ロードについては、20 桁目に `A` は必要ありません。20 桁目に `A` を指定した場合には、出力仕様書に `ADD` を指定しなければなりません。このファイルは OS/400 CREATE FILE コマンドで既に作られている必要があります。

表 36. キーによらない処理方式の有効なファイル命令 (順次処理、相対レコード番号によるランダム処理、および 連続処理)

ファイル記述仕様書位置					演算仕様書の桁
17	18	20	34	44-80	26-35
I	P/S		ブランク		CLOSE、FEOD、FORCE

表 36. キーによらない処理方式の有効なファイル命令 (順次処理、相対レコード番号によるランダム処理、および 連続処理) (続き)

ファイル記述 仕様書位置					演算仕様書の桁
I	P/S		ブランク	RECNO	CLOSE、FEOD、FORCE
U	P/S		ブランク		UPDATE、DELETE、CLOSE、FEOD、FORCE
U	P/S		ブランク	RECNO	UPDATE、DELETE、CLOSE、FEOD、FORCE
I	F		ブランク		READ、READP、SETLL、SETGT、CHAIN、OPEN、CLOSE、FEOD
I	F		ブランク	RECNO	READ、READP、SETLL、SETGT、
U	F		ブランク		READ、READP、SETLL、SETGT、CHAIN、UPDATE、DELETE、OPEN、CLOSE、FEOD
U	F		ブランク	RECNO	READ、READP、SETLL、SETGT、CHAIN、UPDATE、DELETE、OPEN、CLOSE、FEOD
U	F	A	ブランク	RECNO	WRITE (削除済みレコードの重ね書き)、READ、READP、SETLL、SETGT、CHAIN、UPDATE、DELETE、OPEN、CLOSE、FEOD
I	R		A/P/G/ D/T/Z/ F/ ブランク[1]		OPEN、CLOSE、FEOD
I	R		ブランク[2]		OPEN、CLOSE、FEOD
O	ブランク	A	ブランク	RECNO	WRITE[3] (ファイルへのレコードの追加)、OPEN、CLOSE、FEOD
O	ブランク		ブランク	RECNO	WRITE[4] (新規ファイルの初期ロード)、OPEN、CLOSE、FEOD
O	ブランク		ブランク	ブランク	WRITE (ファイルの順次ロードまたは拡張)、OPEN、CLOSE、FEOD

注:

1. レコード・アドレス限界値ファイルで 34 桁目がブランクの場合には、レコード・アドレス・ファイル中のキーの形式は処理中のキーの形式と同じになります。
2. 相対レコード番号が入っているレコード・アドレス・ファイルでは、35 桁目に `T` が必要になります。
3. 相対レコード番号の入っている `RECNO` フィールドは `WRITE` 命令の前に設定するか、`ADD` が出力仕様書に指定されている場合は、設定する必要があります。
4. 20 桁目の `A` は、レコードを新規ファイルに初期ロードするには必要ありませんが、 20 桁目に `A` を指定したときは、出力仕様書に `ADD` を指定しなければなりません。このファイルは OS/400 ファイル作成コマンドの 1 つで、作成済みである必要があります。

コミットメント制御の使用

ここでは、コミットメント制御を使用してファイル命令をグループとして処理する方法について説明します。コミットメント制御を使用した場合には、ファイル命令の次の 2 つの結果のいずれかを確認します。

- すべてのファイル命令が正常に行われる (コミット命令)
- ファイル命令はすべて影響しない (ロールバック命令)

このように、命令のグループを 1 つの単位として処理することができます。

コミットメント制御を使用するためには、以下のことを行います。

- iSeries システムでは、
  1. コミットメント制御使用のための準備をします。 CL コマンドの CRTJRN (ジャーナル作成)、CRTJRNRCV (ジャーナル・レシーバー作成)、および STRJRNPF (物理ファイル・ジャーナル開始) を使用してください。
  2. コミットメント制御の開始と終了を iSeries システムに通知します。これには CL コマンド STRCMTCTL (コミットメント制御の開始) および ENDCMTCTL (コミットメント制御の終了) を使います。これらのコマンドについての詳細は、Web サイト http://www.ibm.com/eserver/iseries/infocenter で **iSeries 400 Information Center** の中の『*プログラミング*』カテゴリーの『*CL および API*』の節を参照してください。
- RPG プログラムでは、
  1. コミットメント制御の下に置きたいファイルのファイル仕様書にコミットメント制御 (COMMIT) を指定してください。
  2. COMMIT (コミット) 命令コードを使用して 1 群の変更をコミットメント制御下にあるファイルに適用するか、あるいは ROLBK (ロールバック) 命令コードを使用してコミットメント制御下にあるファイルに対する 1 群の保留中の変更を除去してください。システムによるロールバック機能の実行方法の情報については、「*バックアップおよび回復の手引き*」を参照してください。

注: コミットメント制御はデータベース・ファイルに対してだけ適用されます。

コミットメント制御の開始および終了

CL コマンド STRCMTCTL は、コミットメント制御を開始したいことをシステムに伝えます。

LCKLVL(ロック・レベル) パラメーターによって、レコードがコミットメント制御下でロックされるレベルを選択することができます。ロック・レベルについて詳しくは、372 ページの『コミットメント制御のロック』および「*CL プログラミング*」を参照してください。

コミットメント制御下にあるファイルを処理するかどうかの判断を実行時に行うには、コミットメント制御を条件付きにすることができます。詳細については、375 ページの『条件付きコミットメント制御の指定』を参照してください。

COMMIT 命令コードを使用して 1 群の変更を完了した時に、グループの終わりを識別するためのラベルを指定することができます。ジョブの異常終了の場合には、この識別ラベルは、1 群の変更を正常に完了した最後のグループを認識できるよう

に、ファイル、メッセージ待ち行列、またはデータ域に書き出されます。このファイル、メッセージ待ち行列、またはデータ域を STRCMTCTL コマンドで指定します。

コミットメント制御のために指定したファイルを処理するプログラムを呼び出す前に、STRCMTCTL コマンドを出してください。STRCMTCTL コマンドを出す前に、コミットメント制御のために指定したファイルをオープンするプログラムを呼び出した場合には、ファイルは正常にオープンされません。

CL コマンド ENDCMTCTL は、活動化グループまたはジョブがコミットメント制御によるファイルの処理を完了したことをシステムに通知します。STRCMTCTL コマンドおよび ENDCMTCTL コマンドについての詳細は、Web サイト http://publib.boulder.ibm.com/html/as400/infocenter.html で **iSeries 400 Information Center** の中の『*プログラミング*』カテゴリーの『*CL および API*』の節を参照してください。

コミットメント制御のロック

STRCMTCTL コマンドでは、ロックのレベルを LCKLVL(*ALL)、LCKLVL(*CHG)、または LCKLVL(*CS) のいずれかに指定します。プログラムがコミットメント制御下で操作され、コミットメント制御下のファイルのレコードで入力または出力操作を処理した時には、レコードは次のとおりコミットメント制御によってロックされます。

- ユーザー・プログラムはレコードにアクセスすることができます。
- 活動化グループまたはジョブにあって、このファイルをコミットメント制御の下に置いている別のプログラムは、レコードを読み取ることができます。ファイルが共用ファイルの場合には、2 番目のプログラムもレコードを更新することができます。
- 活動化グループまたはジョブ中のこのファイルをコミットメント制御の下に置いていない別のプログラムは、レコードを読み取ったり、更新したりすることはできません。
- LCKLVL(*CHG) を指定した場合には、別の活動化グループまたはジョブにあり、このファイルをコミットメント制御の下に置いている別のプログラムはレコードを読み取ることができますが、LCKLVL(*ALL) を指定した場合には、レコードを読み取ることができません。どちらのロック・レベルの場合でも、2 番目のプログラムはレコードを更新することはできません。
- このファイルをコミットメント制御の下に置いておらず、またこの活動化グループまたはジョブにない別のプログラムは、レコードを読み取ることはできますが、更新することはできません。
- コミットメント制御ロックは、通常のロックとは異なり、指定された LCKLVL によって変わり、COMMIT 命令および ROLBK 命令によってのみ解除できます。

COMMIT 命令および ROLBK 命令は、レコード上のロックを解除します。UNLOCK 命令は、コミットメント制御を使用してロックされたレコードを解除しません。ロック・レベルについての詳細は、Web サイト http://publib.boulder.ibm.com/html/as400/infocenter.html で **iSeries 400 Information Center** の中の『*プログラミング*』カテゴリーの『*CL および API*』の節を参照してください。

COMMIT 命令または ROLBK 命令が必要となる前にコミットメント制御によってロックできる項目数は、制限される場合があります。詳しくは、「バックアップおよび*回復の手引き*」を参照してください。

注: SETLL および SETGT 命令は、(更新ではなく) 読み取り命令でコミットメント制御のためにレコードがロックされた場合と同様にレコードをロックします。

コミットメント制御の有効範囲

コミットメント制御が STRCMTCTL コマンドを使用して開始された時には、システムは**コミットメント定義**を作成します。コミットメント定義には、そのジョブ内のコミットメント制御下で変更中の資源に関係する情報が入っています。各コミットメント定義は、STRCMTCTL コマンドを出したジョブにだけ知らされ、ENDCMTCTL コマンドを出した時に終了します。

コミットメント定義の有効範囲は、ジョブ内でそのコミットメント定義を使用するプログラムを示します。コミットメント定義は、活動化グループ・レベルまたはジョブ・レベルまで拡大することができます。

コミットメント定義のデフォルトの有効範囲は、STRCMTCTL コマンドを出したプログラムの活動化グループ、すなわち活動化グループ・レベルです。その活動化グループ内で実行するプログラムだけが、そのコミットメント定義を使用します。OPM は、*DFTACTGRP コミットメント定義を使います。ILE プログラムは、関連する活動化グループを使います。

STRCMTCTL コマンドのコミットメントの有効範囲 (CMTSCOPE) パラメーターにコミットメント定義の有効範囲を指定します。ILE 内のコミットメント制御有効範囲の詳細は、「*ILE 概念*」の「データ管理の有効範囲」を参照してください。

コミットメント制御用のファイルの指定

DISK ファイルがコミットメント制御の下で実行されることを示すためには、ファイル仕様書のキーワード・フィールドにキーワード COMMIT を指定してください。

プログラムがファイルのコミットメント制御を指定した時、この仕様書は、このファイルについてこのプログラムによって行われる入出力操作に対してだけ適用されます。コミットメント制御は、入出力操作以外の操作には適用されません。入出力操作を実行しているプログラムでコミットメント制御が指定されていないファイルには適用されません。

複数のプログラムから 1 つのファイルを共用ファイルとしてアクセスする場合には、コミットメント制御の下にファイルをおくようにすべてのプログラムで指定するか、あるいはすべてのプログラムで指定しないかのいずれかでなければなりません。

COMMIT 命令の使用

COMMIT 命令は、コミットメント制御下にあるファイルに対する一群の変更を完了したことをシステムに通知します。ROLBK 命令は、コミットメント制御下にある

ファイルに対する現行グループの変更を除去します。これらの命令コードの指定方法および各命令コードの機能については、「*WebSphere Development Studio: ILE RPG 解説書*」を参照してください。

システム障害の場合には、システムは暗黙に ROLBK 命令を出します。COMMIT 命令コードの演算項目 1 に指定したラベル、および STRCMTCTL コマンドに指定した通知オブジェクトを使用して、正常に完了している最後の変更グループの識別を検査することができます。

活動化グループまたはジョブの終わり、あるいは ENDCMTCTL コマンドを出した時には、OS/400 システムは暗黙の ROLBK を出し、これによって、最後に ROLBK または COMMIT 命令が出された後の変更が除去されます。すべてのファイル命令が確実に実行されるようにするには、コミットメント制御の下で実行している活動化グループまたはジョブの終了前に、COMMIT 命令を出してください。

OPEN 命令はファイルに対してなされる入出力操作を許可し、CLOSE 命令はファイルに対してなされる入出力操作を停止します。しかし、OPEN および CLOSE 命令は、COMMIT および ROLBK 命令に影響しません。COMMIT または ROLBK 命令は、ファイルがクローズされた後でもファイルに影響します。例えば、プログラムに次のステップを組み込むことができます。

1. COMMIT を出す (既にコミットメント制御の下でオープンされているファイルに対して)。
2. コミットメント制御のために指定されているファイルをオープンする。
3. このファイルに対していくつかの入力および出力操作を実行する。
4. ファイルをクローズする。
5. ROLBK を出す。

ステップ 3 で行われた変更は、ステップ 4 でファイルがクローズされていたとしても、ステップ 5 の ROLBK 命令によってロールバックされます。ROLBK 命令は同じ活動化グループまたはジョブ内の別のプログラムから出すことができます。

プログラムがすべてのファイルをコミットメント制御の下で処理しなければならないということはありませんし、また、そうすることによって、かえってパフォーマンスが落ちてしまう場合もあります。COMMIT および ROLBK 命令は、コミットメント制御下にないファイルには影響を与えません。

注: 複数装置がアプリケーション・プログラムに接続されていて、このプログラムが使用するファイルに対してコミットメント制御が有効となっている場合には、 COMMIT および ROLBK 命令は、装置によってではなく、ファイルに基づいて作業を続行します。データベースは部分的に完了した COMMIT ブロックで更新されるか、あるいは他のユーザーが完了した変更が削除されることもあります。これが確実に起こらないようにすることは、ユーザーの責任です。

コミットメント制御の使用例

この例は、プログラムをコミットメント制御の下で機能させるために必要な仕様および CL コマンドを説明したものです。

コミットメント制御の使用を準備するためには、次の CL コマンドを出してください。

1. CRTJRNRCV JRNRCV(RECEIVER)

   このコマンドはジャーナル・レシーバー RECEIVER を作成します。

2. CRTJRN JRN(JOURNAL) JRNRCV(RECEIVER)

   このコマンドはジャーナル JOURNAL を作成し、ジャーナル・レシーバー RECEIVER を接続します。

3. STRJRNPF FILE(MASTER TRANS) JRN(JOURNAL)

   このコマンドはファイル MASTER およびファイル TRANS のジャーナル項目をジャーナル JOURNAL に入れます。

プログラムでは、ファイル MASTER およびファイル TRANS に COMMIT を指定します。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++
FMASTER    UF   E      K      DISK    COMMIT
FTRANS     UF   E      K      DISK    COMMIT
F*
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   :
C                   :
 *
 *  一連の命令が正常に完了した場合には COMMIT 命令を使用し、
 *  正常に完了しなかった場合は変更を
 *  ロールバックします。
 *
C                   UPDATE    MAST_REC                          90
C                   UPDATE    TRAN_REC                          91
C                   IF        *IN90 OR *IN91
C                   ROLBK
C                   ELSE
C                   COMMIT
C                   ENDIF
```

図 173. コミットメント制御の使用例

プログラム (REVISE という名前) をコミットメント制御の下で機能させるためには、次のコマンドを出します。

1. STRCMTCTL LCKLVL(*ALL)

   このコマンドは、最高レベルのロックでコミットメント制御を開始します。

2. CALL REVISE

   このコマンドはプログラム REVISE を呼び出します。

3. ENDCMTCTL

   このコマンドはコミットメント制御を終了し、暗黙ロールバック操作を行います。

条件付きコミットメント制御の指定

コミットメント制御下にあるファイルのオープンの判断を実行時に行うように、プログラムを書くことができます。条件付きコミットメント制御を実行することによって、同じプログラムの 2 バージョン、すなわちコミットメント制御下で操作するものと、そうでないものの 2 つを作成し、保存する必要がなくなります。

COMMIT キーワードには、条件付きコミットメント制御を指定できる任意指定パラメーターがあります。問題のファイルのファイル仕様書のキーワードの項に COMMIT キーワードを入力します。ILE RPG コンパイラーは暗黙に、パラメーターとして指定されたものと同じ名前の、1 バイトの文字フィールドを定義します。パラメーターが '1' に設定されている場合には、ファイルはコミットメント制御の下で実行されます。

COMMIT キーワード・パラメーターは、ファイルのオープンより前に設定しなければなりません。プログラムを呼び出した時に値を渡すか、あるいはプログラム中で明示的に '1' に設定することによって、パラメーターを設定することができます。

共用オープンの場合には、問題のファイルが既にオープンされている場合には、COMMIT キーワード・パラメーターは、たとえ '1' に設定されていても効力を持ちません。

図 174 は、条件付きコミットメント制御を示す例です。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++
FMASTER    UF   E      K      DISK    COMMIT(COMITFLAG)
FTRANS     UF   E      K      DISK    COMMIT(COMITFLAG)

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *  COMITFLAG = '1' の場合にはファイルはコミットメント制御の下で
 *  オープンされます。
 *  それ以外の場合はコミットメント制御の下ではオープンされません。
C     *ENTRY        PLIST
C                   PARM                    COMITFLAG
C                   :
C                   :
 *
 *  一連の命令が正常に完了した場合には COMMIT 命令を使用し、
 *  正常に完了しなかった場合は変更をロールバックします。
 *  ファイルがコミットメント制御用にオープンされている
 *  (例 COMITFLAG = '1') 場合は、COMIT または ROLBK だけを出します。
 *
C                   UPDATE    MAST_REC                              90
C                   UPDATE    TRAN_REC                              91
C                   IF        COMITFLAG = '1'
C                   IF        *IN90 OR *IN91
C                   ROLBK
C                   ELSE
C                   COMMIT
C                   ENDIF
C                   ENDIF
C*
```

図 174. 条件付きコミットメント制御の使用例

プログラム・サイクルでのコミットメント制御

コミットメント制御は、入出力がユーザーの制御の下で行われる全手順ファイルを対象にしています。入出力が RPG プログラム・サイクルの制御の下で行われる、プライマリーおよび セカンダリー・ファイルでは、コミットメント制御を使用しないでください。この推奨事項をあげる理由の一部は次のとおりです。

- プログラム内の最終合計出力には COMMIT 命令を出すことはできません。
- ロックされたレコード状態からの回復をプログラミングすることは、サイクル内では困難です。
- レベル標識は ROLBK 命令でリセットされません。
- ROLBK 命令の後では、突き合わせレコードの処理で順序エラーが起こることがあります。

DDM ファイル

ILE RPG プログラムは、**分散データ管理機能** (DDM) を介して、リモート・システム上のファイルにアクセスします。DDM によって、あるシステムのアプリケーション・プログラムは、リモート・システムに保管されているファイルをデータベース・ファイルとして使用することができます。DDM ファイルをサポートするために ILE RPG プログラム内で特別なステートメントは必要ありません。

DDM ファイルは、ローカル (ソース) システム上でユーザーまたはプログラムによって作成されます。このファイル (オブジェクト・タイプが *FILE) は、リモート (ターゲット) システム上に保有されているファイルを指定します。DDM ファイルによって、ローカル・システムがリモート・システムを見つけたり、およびソース・ファイルのデータをアクセスするのに関する詳細に必要な情報が提供されます。DDM の使用および DDM ファイルの作成についての詳細は、http://publib.boulder.ibm.com/html/as400/infocenter.html で **iSeries 400 Information Center** の中の『データベースおよびファイル・システム』カテゴリーの『*DB2 UDB for iSeries*』の節を参照してください。

V3R1 より前の DDM ファイルの使用

バージョン 3 リリース 1.0 より前の DDM ファイルを使用している場合には、READE または READPE 命令、SETLL についての EQ 標識、またはレコード・アドレス・ファイルによる限界内順次処理の際には、データ管理レベルでのキーの比較は行われません。 READE または READPE 命令、SETLL についての EQ 標識、あるいは限界内順次処理のときには、代わりに *HEX 照合順序を使用してキーの比較が行われます。

ファイルに与えられたキーと 2 つ以上の検索引き数を突き合わせる DDS 機能を使うと、この結果は予想と異なることがあります。例えば、ABSVAL が数字キーに使われると、-1 と 1 が値が 1 のファイルのキーに対する検索引き数になります。16 進数の照合順序を使うと、-1 の検索引き数は 1 の値の実際のキーにはなりません。 キー比較を異なるものにする DDS 機能のいくつかは以下のとおりです。

- ファイルに指定された ALTSEQ
- キー・フィールドでの ABSVAL、ZONE、UNSIGNED、または DIGIT キーワード
- 可変長、日付、時刻、またはタイム・スタンプのキー・フィールド
- ファイルの SRTSEQ が *HEX でない
- ALWNULL(*USRCTL) が作成コマンドに指定され、レコードまたは検索引き数のキーに null がある (これは外部記述ファイルにのみ適用されます)

さらに、数値フィールドの符号がシステムの要求するものと異なる場合にも、キー比較で異なる結果が生じます。

READE または READPE 命令、SETLL についての EQ 標識、あるいはレコード・アドレス・ファイルによる限界内順次処理の際に、V3R1 より前の DDM ファイルでデータ管理レベルでのキーの比較が行われない時には、最初の時点で通知メッセージ (RNI2002) が出されます。

注: レコードが見つからない可能性のある入出力命令 (SETLL、CHAIN、SETGT、READE、READPE) のパフォーマンスは、バージョン 3 リリース 1.0 より前と同等のものより低下します。

第 18 章 外部接続装置へのアクセス

装置ファイルを使用して RPG から、外部接続装置にアクセスすることができます。**装置ファイル**とは、プリンター、テープ装置、ディスケット装置、表示装置、および通信線によって接続された他のシステムなど、外部接続ハードウェアへのアクセスを可能にするファイルのことです。

この章では、RPG の装置名 PRINTER、SEQ、および SPECIAL を使用して、外部接続装置にアクセスする方法について説明します。表示装置および ICF 装置については、 395 ページの『第 19 章 WORKSTN ファイルの使用』を参照してください。

装置ファイルのタイプ

プログラムがシステム上の装置に対して読み取りまたは書き出しする前に、オペレーティング・システムに対して装置のハードウェア機能を識別する装置記述を、装置の構成時に作成しなければなりません。装置ファイルは、装置を使用できる方法を指定します。特定の装置ファイルを参照することによって、RPG プログラムはシステムに対して記述されている方法で装置を使用します。装置ファイルは、装置への提示用に RPG プログラムからの出力データを様式設定し、RPG プログラムへの提示用に装置からの入力データを様式設定します。

接続した外部接続装置にアクセスするためには、表 37 にリストされた装置ファイルを使用します。

表 37. AS/400 装置ファイル、関連 CL コマンド、および RPG 装置名

装置ファイル	関連する外部接続装置	CL コマンド	RPG 装置名
プリンター・ファイル	プリンターへのアクセスを提供し、印刷出力の様式を記述します。	CRTPRTF CHGPRTF OVRPRTF	PRINTER
テープ・ファイル	テープ装置に保管されているデータ・ファイルへのアクセスを提供します。	CRTTAPF CHGTAPF OVRTAPF	SEQ
ディスケット・ファイル	ディスケット装置に保管されているデータ・ファイルへのアクセスを提供します。	CRTDKTF CHGDKTF OVRDKTF	DISK
表示装置ファイル	表示装置へのアクセスを提供します。	CRTDSPF CHGDSPF OVRDSPF	WORKSTN
ICF ファイル	1 つのシステムのプログラムが同じシステムまたは別のシステムのプログラムと通信できるようにします。	CRTICFF CHGICFF OVRICFF	WORKSTN

装置ファイルには使用される装置を識別するファイル記述が含まれていますが、データは含まれていません。

プリンターのアクセス

ILE RPG プログラムの PRINTER ファイルは、AS/400 システム上のプリンター・ファイルに関連しています。

PRINTER ファイルによって出力ファイルを印刷することができます。この章では、ILE RPG プログラムでプリンター・ファイルを指定し、使用する方法について説明します。

PRINTER ファイルの指定

プログラムで PRINTER ファイルをアクセスすることを指示するためには、ファイル記述仕様書でファイルの装置名として PRINTER を指定してください。各ファイルには固有のファイル名がなければなりません。プログラム当たり最大 8 個の PRINTER ファイルが使用可能です。

PRINTER ファイルは外部記述またはプログラム記述とすることができます。外部記述の PRINTER ファイルの場合には、オーバーフロー標識 OA ～ OG と OV、フェッチ・オーバーフロー・ルーチン、スペース / スキップの項目、および PRTCTL キーワードは使用することはできません。外部記述ファイルの有効な出力仕様書項目については、「*WebSphere Development Studio: ILE RPG 解説書*」を参照してください。

外部記述 PRINTER ファイルの場合には、DDS キーワード INDARA を指定することができます。プログラム記述の PRINTER ファイルにこのキーワードを使用しようとした場合には、実行時エラーとなります。

CL コマンド CRTPRTF (印刷ファイルの作成) を使ってプリンター・ファイルを作ることができ、または IBM 提供のファイル名を使用することもできます。

CRTPRTF コマンドについての詳細は、Web サイト http://www.ibm.com/eserver/iseries/infocenter で **iSeries 400 Information Center** の中の『プログラミング』カテゴリーの『*CL および API*』の節を参照してください。

IBM 提供のファイル名および外部記述プリンター・ファイルの DDS についての詳細は、上記の Web サイトで **iSeries 400 Information Center** の中の 『データベースおよびファイル・システム』カテゴリーの『*DB2 UDB for iSeries*』の節を参照してください。

PRINTER ファイルに有効なファイル命令コードは、WRITE、OPEN、CLOSE、および FEOD です。これらの命令コードの完全な説明は、「*WebSphere Development Studio: ILE RPG 解説書*」を参照してください。

フェッチ・オーバーフローの処理

PRINTER ファイルを使用する時の重要な考慮事項は、フェッチ・オーバーフローです。外部記述 PRINTER ファイルの場合には、フェッチ・オーバーフローの処理はユーザーの責任で行ってください。次の 1 つを実行します。

- キーワード OFLIND (*オーバーフロー標識*) を使用して標識 *IN01 ～ *IN99 をオーバーフロー標識として指定してください。

- 行番号およびフェッチ・オーバーフローについて INFDS のプリンター・フィードバック・セクションを検査してください。詳しくは、「*WebSphere Development Studio: ILE RPG 解説書*」を参照してください。
- 1 ページ当たりの出力行数を数えてください。
- 出力命令を指定した演算仕様書の 73 ～ 74 桁目に標識を指定することによって、あるいはエラーの処理ができる INFSR を指定することによって、ファイルの例外 / エラーを検査してください。INFDS には、ファイルの例外 / エラーに関する詳細情報が入っています。例外およびエラー処理の詳細については、273 ページの『第 13 章 例外の処理』を参照してください。

プログラム記述または外部記述ファイルの場合には、ファイル仕様書にキーワード OFLIND (*オーバーフロー標識*) を使用して標識 *IN01 ～ *IN99 を指定することができます。この標識は、行がオーバーフロー行に印刷されるか、あるいは行送り操作またはスキップ操作中にオーバーフロー行に達したか、これを超えた時に、オンに設定されます。標識を使用して、オーバーフロー条件への応答を条件付けることができます。この標識は、オーバーフロー標識 (*INOA - *INOG、*INOV) が実行するような、RPG オーバーフロー論理を条件付けしません。標識をオフに設定することはユーザーの責任です。

プログラム記述と外部記述の両方のファイルの場合には、行番号およびページ番号は、ファイルの INFDS のプリンター・フィードバック・セクションで使用可能です。この情報をアクセスするためには、ファイル仕様書に INFDS キーワードを指定してください。この仕様書では、367 ～ 368 桁目に行番号を定義し、データ構造の 369 ～ 372 桁目にページ番号を定義してください。行番号とページ番号の両方のフィールドとも、小数部を持たない 2 進数として定義しなければなりません。INFDS はプリンター・ファイルに対するすべての出力操作の後に更新されるので、これらのフィールドを使用して、プログラム中に行カウントの論理をもたずに現在の行およびページ番号を判別することができます。

注: プリンター・ファイルをディスクなどの別の装置に一時変更した場合には、INFDS のプリンター・フィードバック・セクションは更新されず、行カウントの論理は正しくなくなります。

プログラム記述 PRINTER ファイルの場合には、以下の項で説明するオーバーフロー標識およびフェッチ・オーバーフロー・ルーチン論理が適用されます。

プログラム記述ファイルでのオーバーフロー標識の使用

オーバーフロー標識 (OA ～ OG、OV) は、ページの最後の行を印刷、または超えた時にオンに設定されます。オーバーフロー標識を使用して、次のページに印刷する行を指定することができます。オーバーフロー標識は、プログラム記述 PRINTER ファイルの場合にだけ指定することができ、主に見出し行の印刷を条件付けるために使用されます。オーバーフロー標識は、出力仕様書に OFLIND キーワードを使用して指定され、演算仕様書 (9 ～ 11 桁目) および出力仕様書 (21 ～ 29 桁目) の命令を条件付けるために使用することができます。オーバーフロー標識が指定されていない場合には、コンパイラーが最初の未使用のオーバーフロー標識を PRINTER ファイルに割り当てます。オーバーフロー標識は、演算仕様書の結果の標識 (71 ～ 76 桁目) として指定することもできます。

コンパイラーは、ページでオーバーフロー状態が最初に起こった時点でのみオーバーフロー標識をオンに設定します。以下の 1 つが起こった場合には必ず、オーバーフロー条件ということになります。

- オーバーフロー行を超えて行が印刷された場合
- 行送り操作によってオーバーフロー行を超えた場合
- スキップ操作によってオーバーフロー行を超えた場合

383 ページの表 38 には、ファイル仕様書および出力仕様書で、オーバーフロー標識がある場合またはない場合の結果が示してあります。

以下の考慮事項は出力仕様書で使用するオーバーフロー標識に適用されます。

- 行送り操作によってオーバーフロー行を超えると、オーバーフロー標識がオンに設定されます。
- 同じページの中で別の行にオーバーフロー行を超えてスキップすると、オーバーフロー標識がオンに設定されます。
- オーバーフロー行を超えて新しいページの任意の行にスキップしたとしても、指定したオーバーフロー行を超えてスキップ先を指定したのでなければ、オーバーフロー標識はオンに設定されません。
- 新しいページへのスキップをオーバーフロー標識によって条件付けられていない行で指定した場合には、新しいページに用紙が進んだ後にオーバーフロー標識がオフに設定されます。
- 新しい行へスキップするように指定した場合にプリンターがその行に現在あるとすると、スキップは行われません。またオーバーフロー行を超えていなければ、オーバーフロー標識はオフに設定されます。
- 出力印刷レコードに OR 記入行の指定がある場合には、前の行のスペースおよびスキップの記入項目が使用されます。それらが前の行と異なる場合には、スペースを入力して ON 行の項目をスキップします。
- 各ページに 1 つだけの制御グループから情報を入れるように、制御レベルの標識をオーバーフロー標識と一緒に使用することができます。384 ページの図 176 を参照してください。
- オーバーフロー行の条件付けの場合には、オーバーフロー標識は AND 関係または OR 関係のいずれにも指定することができます。AND 関係の場合には、オーバーフロー標識は、オーバーフロー行と見なされるその行の主要な仕様書行に指定しなければなりません。OR 関係の場合には、オーバーフロー標識は主要な仕様書行または OR 記入行に指定することができます。1 つのオーバーフロー標識しか 1 グループの出力標識と関連付けることができません。OR 関係の場合には、オーバーフロー標識を指定した仕様書行上の条件付け標識だけがオーバーフロー行の条件付けに使用されます。
- オーバーフロー標識を AND 記入行で使用した場合には、この行はオーバーフロー行とはなりません。この場合オーバーフロー標識は、他のすべての出力標識と同じように扱われます。
- オーバーフロー標識がレコード識別標識との AND 関係で使用されている時には、レコード・タイプはオーバーフローが起こった時に読み取られたものでない可能性があるので、矛盾した結果となることがよくあります。したがって、レコード識別標識がオンにはならず、オーバーフローとレコード識別の両方の標識によって条件付けられたすべての行が印刷されません。

- オーバーフロー標識によって例外行 (17 桁目に E) および例外レコード内のフィールドの条件付けができます。

表 38. オーバーフロー標識がある場合またはない場合の結果

ファイル仕様書の桁 44 ～ 80	出力仕様書の桁 21 ～ 29	処置
記入なし	記入なし	未使用の最初のオーバーフロー標識がオーバーフロー時の次ページへのスキップの条件付けに使用されます。
記入なし	記入	コンパイル時のエラー。オーバーフロー標識が出力仕様書から脱落しています。未使用の最初のオーバーフロー標識がオーバーフロー時の次ページへのスキップの条件付けに使用されます。
OFLIND(*標識*)	記入なし	連続印刷中。オーバーフローは認識されません。
OFLIND(*標識*)	記入	通常のオーバーフロー処理。

すべてのページへの見出しの印刷例

図 175 には、すべてのページへの見出しの印刷に必要なコーディングの例が示されています。1 ページ目、すべてのオーバーフロー・ページ、および制御フィールドの変更 (L2 がオン) のために開始される各改ページ。最初の行によって、オーバーフローが起こった (OA がオンで L2 がオンでない) 時にのみ、見出しを新しいページの最上部 (06 にスキップ) に印刷することができます。

2 行目では、新制御グループの始め (L2 がオン) でのみ新しいページに見出しを印刷することができます。この方法では、L2 と OA の両方がオンになることによって発生する重複見出しは起こりません。2 行目によって、制御フィールドがレコードで指定されている場合には、最初のレコードは常に制御の切れ目となる (L2 をオンにする) ので、最初のレコードが読み取られた後に、見出しを 1 ページ目に印刷することができます。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
OFilename++DF..N01N02N03Excnam++++B++A++Sb+Sa+.............................
OPRINT     H    OANL2                  3  6
O..............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat++
O          OR   L2
O                                          8 'DATE'
O                                         18 'ACCOUNT'
O                                         28 'N A M E'
O                                         46 'BALANCE'
O*
```

図 175. すべてのページへの見出しの印刷

すべてのページへのフィールドの印刷例

384 ページの図 176 には、すべてのページに一定のフィールドを印刷するために必要なコーディングを示してあります。この場合には、オーバーフロー条件または制御レベルの変更 (L2) で 06 へのスキップが行われます。NL2 標識は、同じサイクルで行の印刷またはスキップが 2 回行われないようにするものです。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
OFilename++DF..N01N02N03Excnam++++B++A++Sb+Sa+.............................
OPRINT     D    OANL2                  3  6
O         OR    L2
O..............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat++
O                       ACCT                8
O*
```

図 176. すべてのページへのフィールドの印刷

プログラム記述ファイルでのフェッチ・オーバーフロー・ルーチンの使用

残りの明細、合計、例外、およびオーバーフロー標識によって条件付けられた見出し行を印刷するために十分なスペースがページ上に残っていない場合には、フェッチ・オーバーフロー・ルーチンを呼び出すことができます。このルーチンがオーバーフローを起こします。オーバーフロー・ルーチンをいつ取り出すかを決めるために、考えられるオーバーフローのすべての状況を研究してください。行およびスペースを数えることによって、各明細行、合計行、および例外行でオーバーフローが起こるとどうなるか予測することができます。

フェッチ・オーバーフロー・ルーチンを使用すると、基本 ILE RPG オーバーフロー論理を変更して、ミシン線を超えて印刷が行われるのを防ぎ、ページを可能な限り無駄なく使用することができます。通常のプログラム・サイクルでは、オーバーフロー標識がオンになっているか、合計出力の直後にコンパイラーによって 1 回だけ検査されます。フェッチ・オーバーフロー機能を指定した場合には、フェッチ・オーバーフローが指定されている各行ごとに、コンパイラーによってオーバーフローが検査されます。

385 ページの図 177 は、フェッチ・オーバーフロー・ルーチンがオンに設定された場合とオフに設定された場合のオーバーフロー印刷の通常の処理を示したものです。

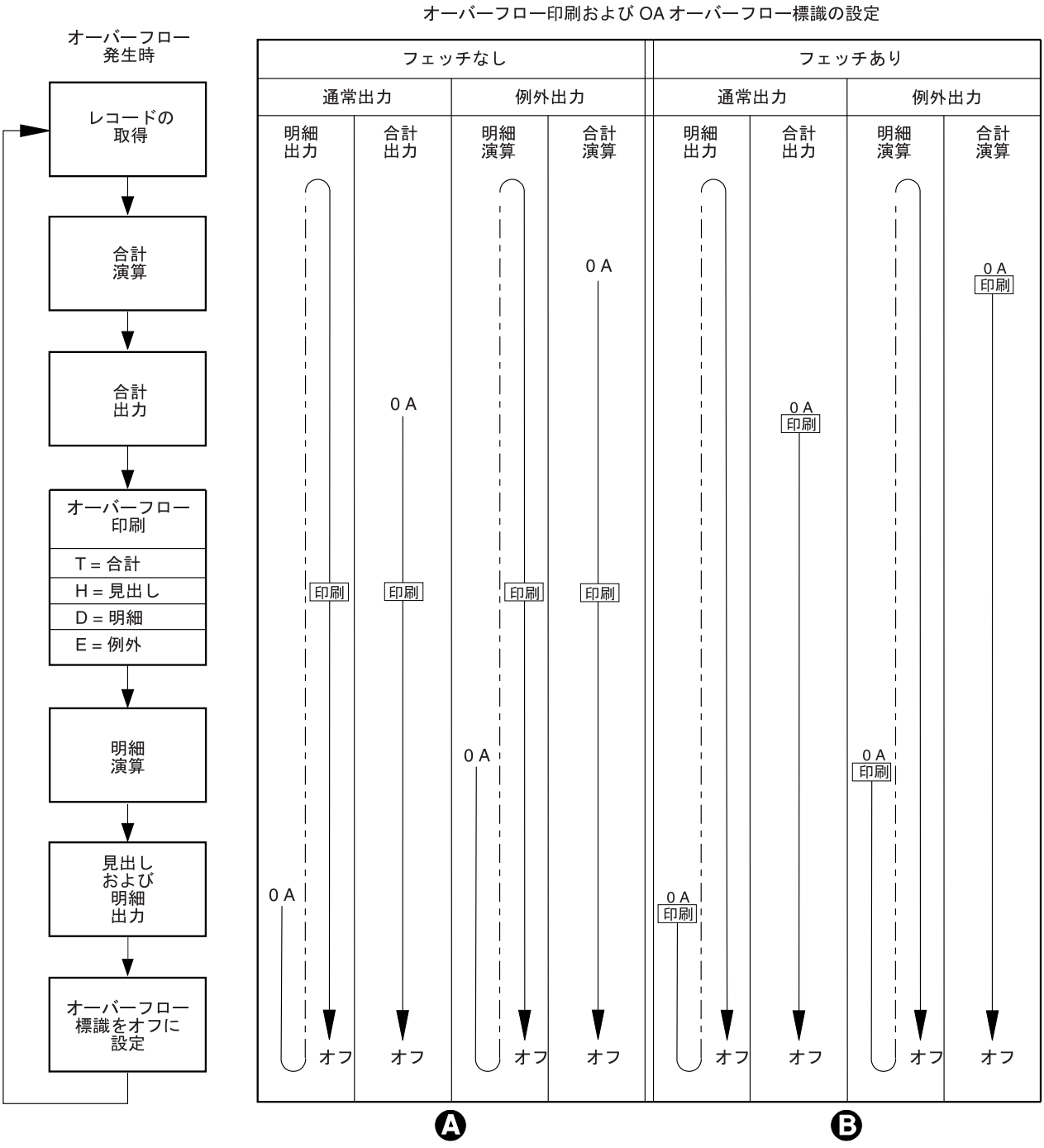


図 177. オーバーフロー印刷: オーバーフロー標識の設定

A フェッチ・オーバーフローを指定しない時は、すべての出力の後にオーバーフロー行が印刷されます。オーバーフローがいつ起こるかに関係なく (OA がオン)、オーバーフロー標識 OA はオーバーフローの出力中、オンのままで、見出しと明細出力時の後でオフになります。

B フェッチ・オーバーフローを指定した時は、オーバーフロー標識 OA がオンの場合、フェッチ・オーバーフローを指定した出力行の前にオーバーフロー行が書き出されます。OA がオンに設定された時は、見出しおよび明細出力が終った後でもオンのままです。オーバーフローが既に感知されてオーバ

ーフロー行が書かれたことがある場合を除いて、オーバーフロー行がオーバーフロー時に、一瞬の間に書かれることはありません。

フェッチ・オーバーフローの指定

フェッチ・オーバーフローは、PRINTER ファイルの詳細行、合計行、または例外行のどの出力仕様書にしても、18 桁目に F を指定します。フェッチ・オーバーフロー・ルーチンは、用紙を自動的に次のページへ進めることはしません。

出力中、出力行の条件標識は、その行が書き出されるべきものかどうかを判別するかめにテストされます。その行が書き出されるべきもので 18 桁目に F が指定されていると、コンパイラーはオーバーフロー標識がオンかどうかを判別するためにテストします。オーバーフロー標識がオンのときは、オーバーフロー・ルーチンが取り出され、以下の処理がなされます。

1. 取り出しが指定されたファイルのオーバーフロー行のみが、出力のために検査される。
2. オーバーフロー標識によって条件の設定された、すべての合計行が書き出される。
3. プリンターが現在処理中の行番号よりも小さい行番号にスキップする指定が、オーバーフロー標識によって条件設定された行に指定されると、用紙が新しいページに進む。
4. オーバーフロー標識によって条件設定された見出し、詳細、および例外行が書き出される。
5. オーバーフロー・ルーチンを取り出した行が書き出される。
6. このプログラム・サイクルに書き出すために残っていた詳細、および合計行があればすべて書き出される。

各 OR 行の 18 桁目は、オーバーフロー・ルーチンが OR 関係で各レコードに使われるものである場合は、F でなければなりません。フェッチ・オーバーフローは、オーバーフロー標識が同じ仕様書行の 21 から 29 桁目で指定されていると使われることはありません。その場合は、オーバーフロー・ルーチンは取り出されません。

フェッチ・オーバーフローの指定例

387 ページの図 178 は、取り出しフローの使い方を示します。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
OFilename++DF..N01N02N03Excnam++++B++A++Sb+Sa+.............................
OPRINTER   H    OA                     3 05
O.............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat++
O                                          15 'EMPLOYEE TOTAL'
O          TF   L1                     1
O                         EMPLTOT          25
O          T    L1                     1
O                         EMPLTOT          35
O          T    L1                     1
O                         EMPLTOT          45
O          TF   L1                     1
O                         EMPLTOT          55
O          T    L1                     1
O                         EMPLTOT          65
O          T    L1                     1
O                         EMPLTOT          75
O          T    L1                     1
O*
```

図 178. フェッチ・オーバーフローの使い方

18 桁目に F がコーディングされている合計行はオーバーフロー・ルーチンを取り出すことができます。これらの行の印刷の前にオーバーフローが検知された場合にのみ、これを実行します。フェッチ・オーバーフローが処理される前に、オーバーフロー標識がオンになっているかどうかを判別するための検査が行われます。オンになっている場合には、オーバーフロー・ルーチンが取り出され、オーバーフロー標識によって条件付けられた見出し行が印刷されて、合計演算が処理されます。

プログラム記述ファイル中の用紙制御情報の変更

PRTCTL (プリンター制御) キーワードによって、用紙制御情報を変更し、プログラム内でプログラム記述 PRINTER ファイル用の現在行の値をアクセスすることができます。PRINTER ファイル用のファイル仕様書でキーワード PRTCTL(*データ構造名*) を指定します。

ユーザーのソースで、OPM 定義のデータ構造と ILE データ構造の 2 つのタイプの PRTCTL データ構造を指定することができます。デフォルトとして 表 39 に示される、ILE データ構造レイアウトを使います。 OPM 定義のデータ構造レイアウトを使用するためには、PRTCTL(*データ構造名*:*COMPAT) を指定します。OPM PRTCTL データ構造レイアウトは 388 ページの表 40 に示されています。

ILE PRTCTL データ構造は、定義仕様書で定義しなければなりません。最小 15 バイトが必要であり、少なくとも次の順序で指定された次の 5 つのサブフィールドが入っていなければなりません。

表 39. ILE PRTCTL データ構造のレイアウト

桁	サブフィールドの内容
1 ～ 3	印刷前スペースの値が入っている 3 桁の文字フィールド。 (有効な値はブランクまたは 0 ～ 255)
4 ～ 6	印刷後スペースの値が入っている 3 桁の文字フィールド。 (有効な値はブランクまたは 0 ～ 255)
7 ～ 9	印刷前スキップの値が入っている 3 桁の文字フィールド。 (有効な値はブランクまたは 0 ～ 255)

表 39. ILE PRTCTL データ構造のレイアウト (続き)

桁	サブフィールドの内容
10 ～ 12	印刷後スキップの値が入っている 3 桁の文字フィールド。 (有効な値はブランクまたは 0 ～ 255)
13 ～ 15	現在の行カウント値が入っている小数部分のない 3 桁の数値フィールド

OPM PRTCTL データ構造は、定義仕様書で定義し、少なくとも次の 5 つのサブフィールドを以下の順序で入れなければなりません。

表 40. OPM PRTCTL データ構造のレイアウト

桁	サブフィールドの内容
1	印刷前スペースの値が入っている 1 桁の文字フィールド。 (有効な値はブランクまたは 0 ～ 3)
2	印刷後スペースの値が入っている 1 桁の文字フィールド。 (有効な値はブランクまたは 0 ～ 3)
3 ～ 4	印刷前スキップの値が入っている 2 桁の文字フィールド。 (有効な値はブランク、1 ～ 99、100 ～ 109 を表す A0 ～ A9、または 110 ～ 112 を表す B0 ～ B2)
5 ～ 6	印刷後スキップの値が入っている 2 桁の文字フィールド。 (有効な値はブランク、1 ～ 99、100 ～ 109 を表す A0 ～ A9、または 110 ～ 112 を表す B0 ～ B2)
7 ～ 9	現在の行カウント値が入っている小数部分のない 3 桁の数値フィールド。

ILE PRTCTL データ構造の最初の 4 つのサブフィールドに入っている値は出力仕様書の 40 ～ 51 桁目 (スペースとスキップの指定) に許される値と同じです。出力仕様書のスペース項目およびスキップ項目 (40 ～ 51 桁目) がブランクでサブフィールド 1 ～ 4 もブランクの場合は、デフォルトで印刷後 1 行送りになります。PRTCTL キーワードが指定された場合には、40 から 51 桁目にブランクがある出力レコードについてだけ使用されます。PRINTER ファイルのスペースおよびスキップ値 (サブフィールド 1 ～ 4) は、プログラムの実行時に PRTCTL データ構造のこれらのサブフィールドの値を変更することにより制御することができます。

サブフィールド 5 には現在行カウント値が入っています。コンパイラーは最初の出力行が印刷された後までサブフィールド 5 を初期設定しません。その後で、コンパイラーはファイルへの各出力操作の後でサブフィールド 5 を変更します。

用紙制御情報の変更例

389 ページの図 179 は、PRTCTL キーワードを使用して用紙制御情報を変更するために必要なコーディングの例を示したものです。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++++++++++
FPRINT     O    F  132        PRINTER PRTCTL(LINE)

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
DLINE             DS
D SpBefore                1      3
D SpAfter                 4      6
D SkBefore                7      9
D SkAfter                10     12
D CurLine                13     15  0

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   EXCEPT
C   01CurLine       COMP      10                                      49
C   01
CAN 49              MOVE      '3'           SpAfter

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
OFilename++DF..N01N02N03Excnam++++B++A++Sb+Sa+.........................
OPRINT     E    01
O.............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat++
O                       DATA               25
```

図 179. PRTCTL オプションの例

ファイル仕様書では、PRINT ファイルに PRTCTL キーワードを指定します。関連したデータ構造の名前は LINE です。

LINE データ構造は、PRTCTL データ構造に事前定義されたサブフィールドだけをもつ入力仕様書で定義されます。1 ～ 12 桁目の最初の 4 つのサブフィールドは、出力仕様書の 40 ～ 51 桁目で一般に指定されるスペースとスキップ情報を提供するために使用されます。PRTCTL キーワードによってプログラム内でこれらの仕様書を変更することができます。

この例では、SpAfter サブフィールドの値は、CurLine (現在の行カウント値) サブフィールドの値が 10 になった時に 3 に変更されます。 (標識 01 はレコード識別標識としてオンに設定されたとします。)

テープ装置のアクセス

テープ・ファイルへ書き出す時には常に SEQ 装置仕様書を使用します。テープ・ファイルへ可変長レコードを書き出すためには、CL コマンド CRTTAPF または OVRTAPF の RCDBLKFMT パラメーターを使用します。RCDBLKFMT パラメーターを使用する時には、テープへ書き出される各レコードの長さは次によって判別されます。

- 出力仕様書でレコードに指定された最高の終了位置。または、
- 終了位置を指定しなかった場合には、コンパイラーがフィールドの長さからレコードの長さを計算します。

テープからの可変長レコードは、順次編成ファイルからのレコードの読み取りと同様に読み取ります。ファイル仕様書に指定するレコードの長さはファイル中の最大のレコードに合ったものにしてください。

表示装置へのアクセス

プログラムとワークステーションなどの表示装置との間で情報を交換するために、表示装置ファイルを使用します。表示装置ファイルは、表示装置で表示される情報の形式を定義し、表示装置とシステムとの間での情報の処理方法を定義するために使用されます。

WORKSTN ファイルの使用法については、 395 ページの『第 19 章 WORKSTN ファイルの使用』を参照してください。

順次ファイルの使用

ILE RPG プログラムの順次ファイルは AS/400 システム上の次のような、順次編成ファイルに関連付けられます。

- データベース・ファイル
- ディスケット・ファイル
- プリンター・ファイル
- テープ・ファイル

ファイル仕様書の SEQ ファイルのファイル名は、AS/400 ファイルを示します。AS/400 ファイルのファイル記述は、テープ、プリンター、またはディスケットなどの実際の入出力装置を指定します。

プログラムの実行時に、OVRDBF、OVRDKTF、OVRTAPF などのような CL 一時変更コマンドを使用して実際の入出力装置を指定することができます。

順次ファイルの指定

ファイル仕様書の 36 ～ 42 桁目に入力された順次 (SEQ) 装置仕様書は、入力または出力が順次編成ファイルと関連していることを示しています。 391 ページの図 180 を参照してください。プログラムの実行中にファイルと関連付けられる実際の装置は、OS/400 一時変更コマンドによるか、またはファイル名で指示されるファイル記述によって指定することができます。プログラム内で SEQ が指定された場合には、スペース/スキップなどの装置依存の機能、または CHAIN は指定することができません。

次の図には、SEQ ファイルに使用できる命令コードを示しています。

表 41. 順次ファイルに有効なファイル命令コード

ファイル仕様書の桁		演算仕様書の桁
17	**18**	**26 ～ 35**
I	P/S	CLOSE、FEOD
I	F	READ、OPEN、CLOSE、FEOD
O		WRITE、OPEN、CLOSE、FEOD

注: 順次ファイルには印刷制御仕様書は使用できません。

順次ファイルの指定例

図 180 は、ILE RPG ソース・メンバーに SEQ ファイルを指定する方法の例を示します。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+ ...*
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++++++++++++
FTIMECDS   IP   E             DISK
FPAYOTIME  O    F  132        SEQ
 *
```

図 180. SEQ 装置

SEQ 装置が PAYOTIME ファイルに指定されています。プログラムを実行する時には、OS/400 一時変更コマンドを使用して、プログラムの実行中にファイルと関連付けられる実際の装置 (プリンター、テープ、またはディスケットなど) を指定することができます。例えば、あるプログラムの実行用にディスケットを指定する一方、別のプログラムの実行にプリンターを指定することができます。ファイル名によって指示されるファイル記述で実際の装置を指定することができ、この場合には、一時変更コマンドを使用する必要はありません。

SPECIAL ファイルの使用

RPG 装置名 SPECIAL (ファイル仕様書の 36 ～ 42 桁目) によって、ILE RPG 命令で直接サポートされない入出力装置を指定することができます。ファイルの入出力命令は、ユーザー作成ルーチンによって制御されます。ユーザー作成ルーチンの名前は、キーワード PGMNAME('プログラム名') を使用してファイル仕様書で指定しなければなりません。

ILE RPG はこのユーザー作成ルーチンを呼び出してファイルをオープンし、レコードの読み取りおよび書き出しを行い、ファイルをクローズします。ILE RPG は、ユーザー作成ルーチンが使用するパラメーター・リストも作成します。このパラメーター・リストには次のものが入っています。

- オプション・コード・パラメーター (オプション)
- 戻り状況パラメーター (状況)
- エラー検出パラメーター (エラー)
- レコード域パラメーター (区域)

このパラメーター・リストは、ILE RPG コンパイラーおよびユーザー作成ルーチンによってアクセスされますが、SPECIAL ファイルのあるプログラムでアクセスすることはできません。

以下では、この RPG 作成パラメーター・リスト中のパラメーターについて説明します。

オプション

オプション・パラメーターは 1 桁の文字フィールドで、ユーザー作成ルーチンで実行する処置を示します。SPECIAL ファイルに対して処理中の命令 (OPEN、CLOSE、FEOD、READ、WRITE、DELETE、UPDATE) によって、ILE RPG からユーザー作成ルーチンに次の値の 1 つが渡されます。

渡される値

	説明
O	ファイルをオープンする。
C	ファイルをクローズする。
F	ファイルを強制終了する。
R	レコードを読み取って区域パラメーターで定義された区域に入れる。
W	区域パラメーターで定義された区域に、ILE RPG プログラムがレコードを入れている。このレコードは書き出される。
D	レコードを削除する。
U	最後に読み取られたレコードを更新したレコード。

状況 状況パラメーターは 1 桁の文字フィールドで、制御が ILE RPG プログラムに戻される時のユーザー作成ルーチンの状況を示します。ユーザー作成ルーチンから ILE RPG プログラムに制御が戻る時、状況には次の戻り値の 1 つがなければなりません。

戻り値	説明
0	通常の戻り。要求された処置が処理された。
1	入力ファイルがファイルの終わりであり、レコードが戻されない。ファイルが出力ファイルの場合には、この戻り値はエラーである。
2	要求された処置が処理されなかった。エラー状態が存在する。

エラー エラー・パラメーターは 5 桁のゾーン数値フィールドで、小数部はありません。ユーザー作成ルーチンでエラーが検出された場合には、エラー・パラメーターにそのエラーの種類を表す印または値が入ります。状況パラメーターに 2 が入っている場合には、INFDS 中の位置 *RECORD の最初の 5 桁にこの値が入ります。

区域 区域パラメーターは文字フィールドで、その長さは SPECIAL ファイルと関連したレコード長と同じです。このフィールドは ILE RPG プログラムとのレコードの授受のために使われます。

RPG 作成のパラメーター・リストに追加のパラメーターを追加することができます。SPECIAL ファイル用のファイル仕様書でキーワード PLIST(パラメーター・リスト名) を指定します。 393 ページの図 181 を参照してください。次に演算仕様書の中の PLIST 命令を使用して追加するパラメーターを定義します。

SPECIAL ファイルのファイル仕様書のキーワード PGMNAME によって指定されるユーザー作成ルーチンは、RPG 作成のパラメーターとユーザー指定のパラメーターの両方が入っている入り口パラメーター・リストを含んでいなければなりません。

1 次ファイルとして SPECIAL ファイルを指定した場合には、最初のプライマリーを読み取る前にユーザー指定のパラメーターを初期設定しなければなりません。PARM パラメーターの演算項目 2 の指定で、あるいはパラメーターとしてコンパイル時配列または配列要素の仕様書によって、これらのパラメーターを初期設定することができます。

表 42 は、SPECIAL ファイルに有効なファイル命令コードを示します。

表 42. SPECIAL ファイルに有効なファイル命令

ファイル仕様書の桁		演算仕様書の桁
17	**18**	**26 ～ 35**
I	P/S	CLOSE、FEOD
C	P/S	WRITE、CLOSE、FEOD
U	P/S	UPDATE、DELETE、CLOSE、FEOD
O		WRITE、OPEN、CLOSE、FEOD
I	F	READ、OPEN、CLOSE、FEOD
C	F	READ、WRITE、OPEN、CLOSE、FEOD
U	F	READ、UPDATE、DELETE、OPEN、CLOSE、FEOD

SPECIAL ファイルの使用例

図 181 には、プログラム内での RPG 装置名 SPECIAL の使用方法を示してあります。この例では、ファイル EXCPTN に見付かったファイル記述が装置 SPECIAL と関連しています。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++++++++++++
FEXCPTN    O    F   20        SPECIAL PGMNAME('USERIO')
F                                    PLIST(SPCL)
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
DName++++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++++++++++++++
D OUTBUF          DS
D  FLD                    1     20

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C     SPCL          PLIST
C                   PARM                    FLD1

C                   MOVEL     'HELLO'       FLD
C                   MOVE      '1'           FLD1             1
C                   WRITE     EXCPTN        OUTBUF
C                   MOVE      '2'           FLD1             1
C                   WRITE     EXCPTN        OUTBUF
C                   SETON                                      LR
```

図 181. SPECIAL 装置

394 ページの図 182 には、ユーザー作成プログラム USERIO を示してあります。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
DName++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++++++++++++
D ERROR           S              5S 0
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....

 *----------------------------------------------------------------*
 * 最初の 4 つのパラメーターは、ILE RPG 作成のパラメーター・      *
 * リスト。残りはプログラマー定義の PLIST によって定義されます。  *
 *----------------------------------------------------------------*
C     *ENTRY        PLIST
C                   PARM                    OPTION            1
C                   PARM                    STATUS            1
C                   PARM                    ERROR             5 0
C                   PARM                    AREA             20
C                   PARM                    FLD1              1

 *----------------------------------------------------------------*
 * ユーザー作成プログラムは渡されたオプションに従って            *
 * ファイル入出力を実行します。                                   *
 *----------------------------------------------------------------*
C                   SELECT
C                   WHEN      OPTION = 'O'
C*  perform OPEN operation
C                   WHEN      OPTION = 'W'
C*  perform WRITE operation
C                   WHEN      OPTION = 'C'
C*  perform CLOSE operation
C                   ENDSL
C                   RETURN
```

図 182. ユーザー作成プログラム USERIO

SPECIAL 装置の入出力操作は、ユーザー作成プログラム USERIO によって制御されます。プログラマー定義の PLIST(SPCL) に指定されるパラメーターは、SPECIAL 装置の RPG 作成のパラメーター・リストの終わりに追加されます。プログラマー指定のパラメーターは、ユーザーの ILE RPG プログラムおよびユーザー作成のルーチン USERIO によってアクセスすることができますが、RPG 作成のパラメーター・リストは、内部 ILE RPG 論理およびユーザー作成のルーチンでしかアクセスすることはできません。

第 19 章 WORKSTN ファイルの使用

iSeries サーバーの対話式アプリケーションは一般に次のものとの通信を必要とします。

- 1 つまたは複数のワークステーション・ユーザー (表示装置ファイルを介する)
- リモート・システム上の 1 つまたは複数のプログラム (ICF ファイルを介する)
- リモート・システム上の 1 つまたは複数の装置 (ICF ファイルを介する)

表示装置ファイルは、iSeries システム上の DSPF の属性をもつタイプ *FILE のオブジェクトです。表示装置ファイルを使用して、表示端末のユーザーと対話式に通信します。データベース・ファイルと類似して、表示装置ファイルは外部記述またはプログラム記述とすることができます。

ICF ファイルは、iSeries システム上の ICFF の属性をもつタイプ *FILE のオブジェクトです。ICF ファイルを使用して、リモート・システム (iSeries または iSeries 以外) 上の他のアプリケーション・プログラムと通信 (データの送信およびデータの受信) します。ICF ファイルには、システム間のデータの送信および受信に必要な通信形式が入っています。リモート・システム上の他のアプリケーション・プログラムと通信 (データの送信およびデータの受信) できる ICF ファイルを使用するプログラムを書き出すことができます。

RPG プログラム内のファイルが WORKSTN 装置名で識別されていれば、そのプログラムはワークステーション・ユーザーと対話式に通信することもできれば、システム間通信機能 (ICF) を使用して他のプログラムと通信することもできます。この章では、次のものの使用法について説明します。

- システム間通信機能 (ICF)
- 外部記述 WORKSTN ファイル
- プログラム記述 WORKSTN ファイル
- 複数装置ファイル

システム間通信機能

ICF を使用するためには、ICF 装置ファイルを参照するプログラム中で WORKSTN ファイルを定義します。システム提供のファイル QICDMF または OS/400 コマンド CRTICFF を使用して作成されたファイルを使用してください。

ICF のためのコーディングは、ICF をプログラムの中でファイルとして使用することによって行います。ICF は、表示装置ファイルと類似しており、システム間でデータを送信および受信するために必要な通信形式が入っています。

ICF について詳しくは、「*ICF Programming*」を参照してください。

外部記述 WORKSTN ファイルの使用

RPG WORKSTN ファイルには、外部記述表示装置ファイルまたは ICF 装置ファイルを使用することができます。これにはファイル情報、および書き出されるレコードのフィールドの記述が入っています。最も一般に使用される外部記述 WORKSTN ファイルは表示装置ファイルです。(表示装置ファイルを記述し作成する方法の説明については、Web サイト http://www.ibm.com/eserver/iseries/infocenter で **iSeries 400 Information Center** の中の『データベースおよびファイル・システム』カテゴリーの『*DB2 UDB for iSeries*』の節を参照してください。)

フィールド記述 (フィールド名、属性など) のほかに、次のことを行うために表示装置ファイルの DDS が使用されます。

- 各フィールドおよび固定情報について行番号と桁番号を指定することによって、画面上におけるレコードの配置形式を設定する。
- フィールドの下線付けや強調表示、反転イメージ、明滅カーソルなどのアテンション機能を指定する。
- 表示ワークステーションから入力されたデータを妥当性検査するように指定する。妥当性検査機能にはデータが必要なフィールドの検出、全桁入力フィールドの検出、誤りデータ・タイプの検出、特定の範囲についてのデータの検出、データが有効な入力であるかどうかの検査、およびモジュール 10 または 11 の検査数字検査の処理があります。
- 新しいデータが表示される時にフィールドを消去するか、オーバーレイするか、または保存するかを判別するなどの画面管理機能を制御する。
- 標識 01 ～ 99 をコマンド・アテンション・キーまたはコマンド機能キーと関連付ける。機能キーをコマンド機能キー (CF) として記述した場合には、応答標識とデータ・レコード (画面で変更が入力された) の両方ともプログラムに戻されます。機能キーをコマンド・アテンション・キー (CA) として記述した場合には、応答標識がプログラムに戻されますが、データ・レコードは変更されないままです。したがって、入力専用文字フィールドはブランクであり、入力専用数字フィールドはゼロが埋め込まれます。ただし、これらのフィールドがそれ以外の値で初期設定されている場合は除きます。
- 編集コード (EDTCDE) または編集語 (EDTWRD) キーワードをフィールドに割り当てて、フィールド値の表示法を指定する。
- サブファイルを指定する。

表示装置レコード様式は次の 3 種類のフィールドで構成されます。

- *入力フィールド*。入力フィールドは、プログラムがレコードを読み取る時に、装置からプログラムに渡されます。入力フィールドはデフォルト値を用いて初期設定することができます。デフォルト値が変更されない場合には、デフォルト値がプログラムに渡されます。初期設定されていない入力フィールドは、ブランクとして表示され、ワークステーション・ユーザーはそこにデータを入力することができます。
- *出力フィールド*。出力フィールドは、プログラムがレコードを表示装置に書き出す時に、プログラムから装置に渡されます。出力フィールドは、プログラムまたはレコード様式によって装置ファイルに指定することができます。
- *出力 / 入力 (入出力共用) フィールド*。出力 / 入力フィールドは変更可能な出力フィールドです。変更される場合には、入力フィールドとなります。出力 / 入力

フィールドは、プログラムが表示装置にレコードを書き出す時にプログラムから渡され、プログラムが表示装置からレコードを読み取る時にプログラムに渡されます。出力 / 入力フィールドは、ユーザーがプログラムから表示装置に書き出されるデータを変更または更新する時に使用されます。

WORKSTN ファイルの DDS にキーワード INDARA を指定した場合には、 RPG プログラムは標識を入出力バッファーではなく、独立標識域内の WORKSTN ファイルに渡します。

外部記述表示装置ファイルの詳細な説明、および有効な DDS キーワードのリストについては、Web サイト http://www.ibm.com/eserver/iseries/infocenter で **iSeries 400 Information Center** の中の『データベースおよびファイル・システム』カテゴリーの『*DB2 UDB for iSeries*』の節を参照してください。

図 183 に、表示装置ファイルの DDS の例が示してあります。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
AAN01N02N03T.Name++++++RLen++TDpBLinPosFunctions++++++++++++++++++++*
A** 品目マスター照会
A                                       REF(DSTREF) 1
A          R PROMPT                     TEXT('Item Prompt Format')
A  73N61                                OVERLAY 2
A                                       CA03(98 'End of Program') 3
A                                    1  2'Item Inquiry'
A                                    3  2'Item Number'
A            ITEM      R        I  3 15PUTRETAIN 4
A  61                                   ERRMSG('Invalid Item Number' 61) 5
A          R RESPONSE                   TEXT('Response Format')
A                                       OVERLAY 2
A                                       LOCK 6
A                                    5  2'Description'
A            DESCRP    R            5 15
A                                    5 37'Price'
A            PRICE     R            5 44
A                                    7  2'Warehouse Location' 7
A            WHSLOC    R            7 22
A                                    9  2'On Hand'
A            ONHAND    R            9 10
A                                    9 19'Allocated' 8
A            ALLOC     R            9 30
A                                    9 40'Available'
A            AVAIL     R            9 51
A*
```

図 183. 表示装置ファイルのデータ記述仕様書の例

この表示装置ファイルは、2 つのレコード様式 PROMPT および RESPONSE で構成されています。

1 このファイルのフィールドの属性は、DSTREF フィールド参照ファイルに定義してあります。

2 OVERLAY キーワードを使用して、両方のレコード様式が同じ画面で使用できるようにします。

3 機能キー 3 を標識 98 と関連付けてありますが、この標識はプログラマーがプログラムを終了させるために使用します。

4 PUTRETAIN キーワードによって、ITEM フィールドに入力された値を表示

装置に保存することができます。さらに、ITEM フィールドは 38 桁目の I で入力フィールドとして定義されます。ITEM は、これらのレコード様式の中では唯一の入力フィールドです。レコード中の他のフィールドはすべて、それぞれの 38 桁目がブランクになっているので、出力フィールドです。

5 ERRMSG キーワードは、このレコード様式を使用するプログラムで標識 61 がオンに設定された場合に表示されるエラー・メッセージを指定します。

6 LOCK キーワードによって、RESPONSE レコード様式が最初に表示された時に、ワークステーション・ユーザーはキーボードを使用できなくなります。

7 'Description' (品名)、'Price' (価格)、'Warehouse Location' (倉庫内棚番) などの固定情報は、プログラムで書き出されるフィールドを記述しています。

8 行および桁を記入することによって、フィールドまたは固定情報が画面に書き出される場所を示します。

表示装置ファイルでの機能キー標識の指定

機能キー標識 KA ～ KN および KP ～ KY は、関連する機能キーが DDS に指定してあれば、表示装置 WORKSTN ファイルを含むプログラムで有効となります。

機能キー標識と機能キーの対応関係は次のとおりです。すなわち、機能キー標識 KA は機能キー 1 と対応し、KB は機能キー 2 ...、KX は機能キー 23、KY は機能キー 24 と対応します。

機能キーは、DDS で CFxx (コマンド機能) または CAxx (コマンド・アテンション) キーワードを用いて指定します。例えば、キーワード CF01 によって機能キー 1 を使用することができます。機能キー 1 を押すと、RPG プログラム中で機能キー標識 KA がオンに設定されます。機能キーを CF01 (99) として指定した場合には、RPG プログラム中で機能キー標識 KA と標識 99 の両方がオンに設定されます。ワークステーション・ユーザーが DDS で指定されていない機能キーを押した場合には、OS/400 システムは誤りのキーが押されたことをユーザーに通知します。

ワークステーション・ユーザーが指定機能キーを押した場合には、レコードからフィールドが取り出される時 (フィールド転送論理)、RPG プログラムの関連する機能キー標識がオンに設定され、その他の機能キー標識はすべてオフに設定されます。機能キーを押さなければ、フィールドの転送時にすべての機能キー標識がオフに設定されます。ユーザーが実行キーを押すと、機能キー標識はオフに設定されます。

表示装置ファイルでのコマンド・キーの指定

表示装置ファイルの DDS の中で、キーワード HELP、ROLLUP、ROLLDOWN、PRINT、CLEAR、および HOME を使用して、ヘルプ、次ページ、前ページ、印刷、消去、およびホームの各コマンド・キーを指定することができます。

該当するキーワードが DDS で指定されたレコード様式に対する READ または EXFMT 命令をコンパイラーが処理する時は常に、RPG プログラムによってコマンド・キーが処理されます。コマンド・キーが有効となっている時にコマンド・キーが押されると、OS/400 システムは RPG プログラムに制御を戻します。選択された

コマンドの応答標識が DDS で指定されている場合には、その標識がオンに設定され、レコード様式およびファイルに対して有効となっている他のすべての応答標識がオフに設定されます。

コマンド・キーの応答標識が DDS に指定されていない場合には、次のことが行われます。

- *PGM が指定されていない印刷キーの場合には、印刷機能が処理されます。
- 次ページまたは前ページ・キーをサブファイルで使用した場合には、表示中のサブファイルがそのサブファイルの範囲内で上方または下方に移動します。サブファイルの始めまたは終わりを超えて画面送りをしようとすると、実行時エラーとなります。
- *PGM を使用して指定された印刷キー、サブファイルなしで使用された次ページまたは前ページ・キー、および消去、ヘルプ、ホームの各キーの場合には、*STATUS 値 1121 ～ 1126 の 1 つがそれぞれ設定されて、処理が続行されます。

外部記述 WORKSTN ファイルの処理

外部記述 WORKSTN ファイルが処理されると、OS/400 システムは、プログラムからのデータをそのファイルに指定された形式に変換して、そのデータを表示します。データがプログラムに渡されると、そのデータはプログラムが使用する形式に変換されます。

OS/400 システムは、装置の入出力命令の処理用にその装置の制御情報を提供します。装置から入力レコードが要求されると、OS/400 システムはその要求を出して、プログラムにデータを渡す前にデータから装置制御情報を除去します。さらに、OS/400 システムは、レコード中に変更されたフィールドがある場合には、どのフィールドが変更されたかを示す標識をプログラムに渡すことができます。

プログラムが出力命令を要求する時は、出力レコードを OS/400 システムに渡します。OS/400 システムは、そのレコードの表示に必要な装置制御情報を提供します。また、レコード様式に固定情報が指定されていれば、システムはレコードの表示時にそれも追加します。

レコードがプログラムに渡される時には、フィールドは DDS に指定されている順に配置されます。フィールドが表示される順序は、DDS でフィールドに割り当てられた表示位置 (行番号および桁) に基づきます。したがって、DDS でのフィールドの指定順序と画面でのフィールドの表示順序が同じである必要はありません。

WORKSTN ファイルの処理の詳細については、 406 ページの『有効な WORKSTN ファイル命令』を参照してください。

サブファイルの使用

表示装置ファイルの DDS にサブファイルを指定することによって、画面上で同じタイプの複数レコードを処理することができるようになります。(400 ページの図 184 を参照してください。) サブファイルは、表示装置ファイルから読み取ったり、あるいは書き出したりするレコードのグループです。例えば、プログラムはデータベース・ファイルからレコードを読み取って、出力レコードのサブファイルを作成します。サブファイル全体が書き出された時に、プログラムは 1 回の書き出し

操作でサブファイル全体を表示装置に送ります。ワークステーション・ユーザーは、サブファイル中のデータを変更したり、サブファイルに追加のデータを入力したりすることができます。その後で、プログラムはサブファイル全体を表示装置からプログラムに読み取って、サブファイル中の各レコードを個別に処理します。

サブファイルに含めたいレコードは、ファイルの DDS で指定します。サブファイルに含めることができるレコード数も、DDS で指定しなければなりません。1 つのファイルには、複数のサブファイルを含むことができ、最大 12 個のサブファイルを同時に活動状態にすることができます。同時に 2 つのサブファイルを表示することができます。

サブファイルの DDS は、サブファイル・レコード様式とサブファイル制御レコード様式の 2 つのレコード様式から構成されています。サブファイル・レコード様式には、サブファイル制御レコード様式の制御の下で、表示装置ファイルとの間で転送されるフィールド情報が入っています。サブファイル制御レコード様式により、サブファイルの物理的な読み取り、書き出し、または制御操作が実行されます。402 ページの図 185 は、サブファイル・レコード様式の DDS の例を、 403 ページの図 186 は、サブファイル制御レコード様式の DDS の例をそれぞれ示しています。

サブファイル・キーワードを使用する方法の説明については、Web サイト http://www.ibm.com/eserver/iseries/infocenter で **iSeries 400 Information Center** の中の『データベースおよびファイル・システム』カテゴリーの『*DB2 UDB for iSeries*』の節を参照してください。

```
Customer Name Search

Search Code _______

Number  Name                  Address               City                  State

XXXX    XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
XXXX    XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
XXXX    XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
XXXX    XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
XXXX    XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
XXXX    XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
XXXX    XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
XXXX    XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
XXXX    XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
XXXX    XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
XXXX    XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
XXXX    XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
XXXX    XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
XXXX    XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
XXXX    XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
XXXX    XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
XXXX    XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
XXXX    XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
```

図 184. サブファイルの画面

RPG プログラム中で表示装置ファイルにサブファイルを使用するためには、WORKSTN ファイルのファイル仕様書で SFILE キーワードを指定しなければなりません。 SFILE キーワードの形式は SFILE(*レコード様式名*:*RECNO フィールド名*) です。WORKSTN ファイルは、外部記述ファイル (22 桁目が E) でなければなりません。

SFILE キーワードには、サブファイル・レコード様式 (制御レコード様式ではない) の名前と、サブファイルの処理に使用する相対レコード番号を含むフィールドの名前を指定しなければなりません。

RPG プログラムでは、相対レコード番号の処理は SFILE 定義の一部として定義されます。SFILE 定義は、サブファイル用の ADD を含む全手順更新ファイルを暗黙指定します。したがって、サブファイルに有効なファイル操作は、メイン WORKSTN ファイルの定義に依存しません。すなわち、WORKSTN ファイルを 1 次ファイルまたは全手順ファイルとして定義することができます。

プログラムとサブファイル間でデータを転送信するためには、サブファイル・レコード様式のある CHAIN、READC、UPDATE、または WRITE 命令コードを使用してください。プログラムと表示装置間でデータを転送信するか、またはサブファイル制御操作を処理するためには、サブファイル制御レコード様式のある READ、WRITE、または EXFMT 命令コードを使用してください。

サブファイル処理は、相対レコード番号処理の規則に従います。RPG プログラムは、READC 命令によって検索されたレコードの相対レコード番号を SFILE キーワードの 2 番目の位置に指定されたフィールドに入れます。このフィールドを使用して、サブファイルへの WRITE 命令または ADD を使用する出力操作の場合に RPG プログラムが使用するレコード番号も指定します。SFILE キーワードに指定される RECNO フィールド名は小数点以下の桁数のない数字として定義しなければなりません。フィールドはファイルの最大レコード番号を入れるのに十分な桁数をもっていなければなりません。(SFLSIZ キーワードについては、Web サイト http://www.ibm.com/eserver/iseries/infocenter で **iSeries 400 Information Center** の中の『データベースおよびファイル・システム』カテゴリーの『*DB2 UDB for iSeries*』の節を参照してください。) WRITE 命令コードおよび出力仕様書上の ADD 仕様書には、ファイル仕様書の SFILE キーワードの 2 番目の位置に相対レコード番号を指定することが必要です。

WORKSTN ファイルに関連したサブファイルがある場合には、そのファイル名を参照するすべての暗黙の入力操作および明示の演算命令がメイン WORKSTN ファイルに対して処理されます。サブファイルとして指定していないレコード様式名を指定する操作は、すべてのメイン WORKSTN ファイルで処理されます。

指定された機能キーを非サブファイル・レコードの読み取り中に押した場合には、サブファイル・レコードの後続の読み取りにより、対応する機能キー標識が再びオフに設定されます。読み取りと読み取りの間に機能キー標識がオフに設定された場合でも、そうなります。これは、WORKSTN ファイルから非サブファイル・レコードが読み取られるまで続きます。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
AAN01N02N03T.Name++++++RLen++TDpBLinPosFunctions++++++++++++++++++++*
A** 顧客名検索
A                                      REF(DSTREF) 1
A          R SUBFIL                    SFL 2
A                                      TEXT('Subfile Record')
A            CUST      R           7  3
A            NAME      R           7 10
A            ADDR      R           7 32 3
A            CITY      R           7 54
A            STATE     R           7 77
A*
```

図 185. サブファイル・レコード様式のデータ記述仕様書

サブファイル・レコード様式のデータ記述仕様書 (DDS) は、サブファイルのレコードについて記述します。

1 このレコード様式のフィールドの属性は、REF キーワードによって指定さたフィールド参照ファイル DSTREF に入っています。

2 SFL キーワードにより、このレコード様式はサブファイルとして識別されます。

3 行と桁の指定によって、画面上におけるフィールドの位置が定義されます。

サブファイルの使用

サブファイルの代表的な使用法をいくつか次に示します。

- 表示のみ。ワークステーション・ユーザーが画面を検討します。
- 選択を伴う表示。画面上の項目の 1 つについてユーザーは、詳細情報を要求します。
- 変更。ユーザーは 1 つ以上のレコードを変更します。
- 入力のみで、妥当性検査を行わない。サブファイルがデータ入力機能に使用されます。
- 入力のみで、妥当性検査を行う。サブファイルがデータ入力機能に使用されますが、レコードは検査されます。
- 作業の組み合わせ。サブファイルを、変更がある画面、および新しいレコードの入力がある画面として使用することができます。

次の図は、サブファイル制御レコード様式のデータ記述仕様書の例を示しています。RPG プログラムでのサブファイルの使用例については、 426 ページの『郵便番号による検索』を参照してください。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
AAN01N02N03T.Name++++++RLen++TDpBLinPosFunctions++++++++++++++++++++*
A          R FILCTL                    SFLCTL(SUBFIL)
A N70                                  SFLCLR
A  70                                  SFLDSPCTL
A  71                                  SFLDSP
A                                      SFLSIZ(15)
A                                      SFLPAG(15)
A                                      TEXT('Subfile Control Record')
A                                      OVERLAY
A  71                                  ROLLUP(97 'Continue Search')
A                                      CA01(98 'End of Program')
A                                      HELP(99 'Help Key')
A                                   1  2'Customer Name Search'
A                                   3  2'Search Code'
A            SRHCOD   R         I  3 14PUTRETAIN
A                                   5  2'Number'
A                                   5 10'Name'
A                                   5 32'Address'
A                                   5 54'City'
A                                   5 76'State'
A*
```

図 186. サブファイル制御レコード様式のデータ記述仕様書

サブファイル制御レコード様式は、サブファイルの属性、検索入力フィールド、固定情報、および機能キーを定義します。使用できるキーワードは次のとおりです。

- SFLCTL は、関連するサブファイル (SUBFIL) を指定します。
- SFLCLR は、サブファイルを消去する時点 (標識 70 がオフの時) を指示します。
- SFLDSPCTL は、サブファイル制御レコードを表示する時点 (標識 70 がオンの時) を指示します。
- SFLDSP は、サブファイルを表示する時点 (標識 71 がオンの時) を指示します。
- SFLSIZ は、サブファイルに組み込むレコードの合計数 (15) を指示します。
- SFLPAG は、1 ページ当たりのレコードの合計数 (15) を指示します。
- ROLLUP は、ユーザーが次ページ・キーを押した時にプログラムの中で標識 97 がオンに設定されることを示します。
- HELP により、ユーザーがヘルプ・キーを押すと、有効な機能キーについて説明したメッセージが表示されます。
- PUTRETAIN により、SRHCOD フィールドに入力した値を画面に保存することができます。

サブファイル制御レコード様式は、これらの制御情報のほかに、サブファイル・レコード様式の欄見出しとして使用される固定情報も定義します。

プログラム記述 WORKSTN ファイルの使用

プログラム記述 WORKSTN ファイルは、出力仕様書で様式名を指定してもしなくても使用することができます。様式名を指定した場合には、その様式名は、データ記述仕様書のレコード様式の名前を参照します。このレコード様式では、次のことを記述します。

- RPG プログラムから送られてきたデータ・ストリームを画面上でどのように様式設定するか
- どんなデータが送られて来るか
- どんな ICF 機能を実行するか

様式名を使用する場合には、入力および出力仕様書を使用して、入力および出力レコードを記述しなければなりません。

プログラム記述 WORKSTN ファイルのファイル仕様書に PASS(*NOIND) を指定することができます。PASS(*NOIND) キーワードは、RPG プログラムがそれ以上標識をデータ管理に渡さないこと (出力の場合)、またはそれ以上標識を受け取らないこと (入力の場合) を示します。標識を渡すのはユーザーの責任なので、入力または出力レコードにフィールドとして (*INxx、*IN、または *IN(x) の形式で) 記述してください。これらはデータ記述仕様書 (DDS) で必要な順序で指定しなければなりません。DDS リスト出力を使用してこの順序を判別することができます。

様式名のあるプログラム記述 WORKSTN ファイルの使用

次の仕様書は、プログラム記述 WORKSTN ファイルに様式名を使用する場合に適用されます。

出力仕様書

出力仕様書上で、7 ～ 16 桁目に WORKSTN ファイル名を指定しなければなりません。DDS レコード様式の名前である様式名は、次のフィールド記述行の 53 ～ 80 桁目にリテラルまたは名前付き固定情報として指定されます。様式名を指定した行の 47 ～ 51 桁目に K1 ～ K10 を指定 (右寄せ) しなければなりません。K は、この項目を終了位置でなく長さとして指定し、数字は様式名の長さを示します。例えば、様式名が CUSPMT の場合には、47 ～ 51 桁目に K6 を指定します。(K の後に先行ゼロを使用することができます。) 様式名を条件付けることはできません (21 ～ 29 桁目の標識は正しくありません)。

出力レコードの出力フィールドは、DDS で定義したのと同じ順序で位置指定されていなければなりませんが、フィールド名は同じにする必要はありません。フィールドの終了位置の指定は、画面上のフィールドの位置ではなく、RPG プログラムからデータ管理に渡される出力レコードの終了位置を指示しています。

出力時に標識を渡すためには、次の 1 つを行ってください。

- WORKSTN ファイルの DDS にキーワード INDARA を指定します。ファイル仕様書で PASS(*NOIND) キーワードを使用しないでください。また、出力仕様書に標識を指定しないでください。プログラムとファイルは独立標識域を使用して標識を渡します。
- ファイル仕様書で PASS(*NOIND) キーワードを指定します。*INxx の形式のフィールドとして出力仕様書に標識を指定します。標識フィールドは、出力レコードの他のフィールドより先行していなければならず、これらは WORKSTN ファイルの DDS で指定された順序で現れなければなりません。この順序は DDS リストから決定することができます。

入力仕様書

入力仕様書では、RPG プログラムが表示装置または ICF 装置から受け取るレコードについて記述します。WORKSTN ファイル名は 7 ～ 16 桁目に指定しなければなりません。入力レコードの入力フィールドの配置は、DDS で定義したのと同じ順序でなければなりませんが、フィールド名は同じにする必要はありません。フィールドの位置の指定は、入力レコードのフィールドの位置を指示しています。

入力時に標識を受け取るためには、次の 1 つを行ってください。

- WORKSTN ファイルの DDS にキーワード INDARA を指定します。ファイル仕様書で PASS(*NOIND) キーワードを使用しないでください。また、入力仕様書に標識を指定しないでください。プログラムとファイルは独立標識域を使用して標識を渡します。
- ファイル仕様書で PASS(*NOIND) キーワードを指定します。*INxx の形式のフィールドとして入力仕様書に標識を指定します。これらは WORKSTN ファイルの DDS で指定した順序で入力レコードになければなりません。この順序は DDS リストから決定することができます。

レコード識別標識をファイルの各レコードに割り当てて、WORKSTN ファイルから読み取ったレコードを識別するようにしてください。デフォルト値をもつ潜在フィールドをレコード識別コードとして DDS で指定することができます。

演算仕様書

命令コード READ は、入出力共用、全手順ファイルとして定義されたプログラム記述 WORKSTN ファイルに対して有効です。 406 ページの表 43 を参照してください。ファイル名はこの命令の演算項目 2 に指定しなければなりません。様式は、入力命令を実行する前に装置に存在していなければなりません。この要件は、出力レコードを 1P で条件付けるか、または別のプログラム (例えば、 CL プログラム) で最初の様式を装置に書き出すことによって、満たすことができます。 EXFMT 命令は、プログラム記述 WORKSTN ファイルには有効ではありません。 EXCEPT 命令を使用して WORKSTN ファイルに書き出すこともできます。

追加の考慮事項

プログラム記述 WORKSTN ファイルに様式名を使用する時には、さらに次の点を考慮してください。

- 出力仕様書の 53 ～ 80 桁目に指定した名前は、ファイルの作成に使用された DDS の中のレコード様式の名前と見なされます。
- 出力レコードに Kn 指定がある場合には、そのファイルの他の出力レコードにもこれを使用しなければなりません。ファイルへの出力レコードのすべてに Kn 指定が使用されていない場合には、実行時エラーが起こります。

様式名のないプログラム記述 WORKSTN ファイルの使用

レコード様式名を使用しない場合には、プログラム記述表示装置ファイルは、 1 つのフィールドに 1 つのレコード様式記述を記します。レコード内のフィールドはファイルを使用するプログラムの中で記述しなければなりません。

表示装置ファイル作成コマンドを使用して表示装置ファイルを作成する場合には、ファイルには以下の属性が組み込まれます。

- 可変レコード長を指定できるので、実際のレコード長は使用するプログラムの中で指定しなければなりません。(使用できるレコードの最大長は画面サイズから 1 を引いたものです。)
- プログラムとの間で標識の受け渡しは行われません。
- 機能キー標識は定義されません。
- レコードは使用できる最初の行の 2 桁目から画面に書き出されます。

入力ファイル

入力ファイルの場合には、入力レコードは OS/400 装置サポートによって単一の入力フィールドとして扱われますが、ファイルがオープンされる時ブランクに初期化されます。カーソルはフィールドの先頭、すなわち画面上で 2 桁目に位置付けられます。

出力ファイル

出力ファイルの場合には、OS/400 装置サポートは出力レコードを画面に送られる文字ストリングとして扱います。各出力レコードはファイルの次の順序のレコードとして書き出されます。すなわち、各レコードは前に表示されたレコードの上に重ねて表示されます。

入出力共用ファイル

入出力共用ファイルの場合には、レコードは OS/400 装置サポートによって単一のフィールドとして扱われ、画面上に表示されて出力レコードおよび入力レコードの両方になります。装置サポートによって入力レコードはブランクに初期化され、カーソルは 2 桁目に置かれます。

プログラム記述表示装置ファイルについての詳細は、Web サイト http://www.ibm.com/eserver/iseries/infocenter で **iSeries 400 Information Center** の中の『データベースおよびファイル・システム』カテゴリーの『*DB2 UDB for iSeries*』の節を参照してください。

有効な WORKSTN ファイル命令

表 43 に、WORKSTN ファイルに有効なファイル命令コードが示してあります。

表 43. WORKSTN ファイルに有効なファイル命令コード

ファイル仕様書の桁		演算仕様書の桁
17	**18**	**26 ～ 35**
I	P/S	CLOSE、ACQ、REL、NEXT、POST、FORCE
I	P/S	WRITE[1]、CLOSE、ACQ、REL、NEXT、POST、FORCE
I	F	READ、OPEN、CLOSE、ACQ、REL、NEXT、POST
C	F	READ、WRITE[1]、EXFMT[2]、OPEN、CLOSE、ACQ、REL、NEXT、POST、UPDATE[3]、CHAIN[3]、READC[3]
O	ブランク	WRITE[1]、OPEN、CLOSE、ACQ、REL、POST

表 43. WORKSTN ファイルに有効なファイル命令コード (続き)

ファイル仕様書の桁	演算仕様書の桁
注: 1. WRITE 命令は様式名を用いたプログラム記述ファイルには無効です。 2. EXFMT 命令を用いると、ファイルは外部記述 (ファイル仕様書の 19 桁目に E) でなければなりません。 3. サブファイル・レコード様式については、UPDATE、CHAIN、および READC 命令が有効です。	

以下に、WORKSTN ファイルの処理に使用される時の EXFMT、READ、および WRITE 命令コードについて詳しく説明します。

EXFMT 命令

EXFMT 命令は WRITE と READ を組み合わせたものであって、この 2 つの命令が同じレコード様式に対して、続けて実行されます (この命令はデータ管理の WRITE-READ 命令に対応します)。ファイル仕様書に WORKSTN ファイルを、外部記述データ (22 桁目に E) を使用する全手順 (18 桁目に F) 入出力共用ファイル (17 桁目に C) として定義すれば、EXFMT (様式の実行) 命令コードを画面への書き出しおよび画面からの読み取りに使用することができます。

READ 命令

READ 命令は、全手順入出力共用ファイル、あるいは外部記述データまたはプログラム記述データを使用する全手順入力ファイルに対して有効です。READ 命令は画面からレコードを取り出します。しかし、様式は、入力命令を実行する前に装置に存在していなければなりません。この要件は、出力レコードを 1P で条件付けるか、別のプログラムから最初の様式を装置に書き出すか、または読み取りがレコード様式名による場合は、DDS のレコード記述にキーワード INZRCD を使用することによって、表示装置で満たすことができます。

WRITE 命令

WRITE 命令は新しいレコードを表示装置に書き出し、入出力共用ファイルまたは出力ファイルに対して有効です。出力仕様書および EXCEPT 命令を使用して WORKSTN ファイルに書き出すこともできます。これらの各命令コードについて完全な説明は、「*WebSphere Development Studio: ILE RPG 解説書*」を参照してください。

複数装置ファイル

ファイル仕様書に指定されたキーワード DEVID、SAVEIND、MAXDEV(*FILE)、または SAVEDS のうち、少なくとも 1 つを持つ任意の RPG WORKSTN ファイルが複数装置ファイルです。複数装置ファイルを介して、ユーザーのプログラムは複数の装置にアクセスすることができます。

RPG プログラムは、プログラム装置を介してアクセスしますが、プログラム装置とは実際の装置への操作を指示する記号メカニズムのことです。ファイルを作成する場合には (DDS およびファイル作成コマンドなどのコマンドを使用して)、プログラ

ム装置と関連した装置、ファイルに要求元プログラム装置があるかどうか、ファイル名別 READ 命令に応答するための装置の送信勧誘に使用されるレコード様式、およびこの READ 命令が応答を待機する時間などを考慮します。複数装置ファイルを作成する場合のオプションおよび要件については、Web サイト http://www.ibm.com/eserver/iseries/infocenter で **iSeries 400 Information Center** の中の『データベースおよびファイル・システム』カテゴリーの『*DB2 UDB for iSeries*』の節にある表示装置ファイルの章を参照してください。ICF ファイルに関する情報は、「*ICF Programming*」でも参照できます。

複数装置ファイルについては、次の命令コードで特別の使用法があります。

- OPEN 命令は、ファイルをオープンするほかに、ファイルの作成時に指定された装置を暗黙のうちに獲得します。
- ACQ (獲得) 命令は、複数装置ファイル用にその他の装置を獲得します。
- REL (解放) 命令は、ファイルから装置を解放します。
- DDS キーワード INVITE で使用された時の WRITE 命令は、後続の送信勧誘プログラム装置からの読み取り操作に応答するようにプログラム装置を勧誘します。「*ICF Programming*」にある、プログラム装置の案内に関する節を参照してください。
- READ 命令は、送信勧誘プログラム装置からの読み取り操作、または 1 プログラム装置からの読み取り操作のいずれかを処理します。NEXT 命令が有効となっていない場合には、プログラム・サイクル読み取りまたはファイル名別 READ 命令は、応答するように勧誘されている装置からの入力を待機します (送信勧誘プログラム装置からの読み取り)。他の入出力操作 (NEXT 命令の後のファイル名別 READ、および様式名別 READ を含む) は、特別なフィールドで指示されたプログラム装置を使用して、1 プログラム装置からの読み取り操作を処理します。(このフィールドは、ファイル仕様書行の DEVID キーワードで指定されます。)

  この装置は最後の入力操作で使用した装置、ユーザーが指定した装置、または要求元プログラム装置となります。「*ICF Programming*」の、プログラム装置の案内、および各プログラム装置についての節を参照してください。
- NEXT 命令は、次のファイル名別 READ 命令またはプログラム・サイクル読み取り操作でどの装置を使用するかを指定します。
- POST 命令は、情報を INFDS 情報データ構造に入れます。情報は特定の装置またはファイルに関するものです。(POST 命令を使用するのは、複数装置ファイルの場合だけではありません。)

RPG 命令コードについて詳しくは、「*WebSphere Development Studio: ILE RPG 解説書*」を参照してください。

ファイル仕様書で、いくつかのキーワードを指定して、複数装置ファイルの処理を制御することができます。

- MAXDEV キーワードは、ファイルが単一装置ファイルであるか複数装置ファイルであるかを示します。

  処理するファイルの定義から装置の最大数が取られる複数装置ファイルを処理するためには、MAXDEV(*FILE) を指定してください。ただ 1 つの装置を処理するためには、MAXDEV(*ONLY) を指定してください。

- DEVID キーワードにより、入出力操作が向けられるプログラム装置の名前を指定することができます。

  1 プログラム装置からの読み取りまたは WRITE 命令が出された時には、この命令に使用される装置は DEVID キーワードに対するパラメーターとして指定された装置です。このフィールドはブランクに初期設定され、最後の正常な入力操作が行われた装置の名前で更新されます。ある値をフィールドに転送して、明示的に設定することもできます。ACQ 命令コードは、このフィールドの値に影響しません。DEVID キーワードが指定されていない場合には、最後の正常な入力操作が実行された時の装置に対して入力操作が実行されます。読み取り命令が装置からまだ正常に実行されていない場合には、ブランクの装置名が使用されます。

  1 プログラム装置からの読み取りまたは WRITE 命令がブランクの装置名で出された時には、RPG コンパイラーはプログラムの要求元装置の装置名を暗黙に使用します。RPG プログラムを対話式に呼び出して、これらの命令のいずれかの実行対象としたい ICF 装置を獲得する場合には、命令を実行する前に ICF 装置の装置名を DEVID キーワードに指定されたフィールド名に明示的に転送しなければなりません。これを行わない場合には、使用される装置名はブランク (この場合には、対話式要求元装置名が使用されます)、または最後の正常な入力操作からの装置名となります。ICF 装置に対して入出力操作を実行した場合には、別の装置で入力操作が正常に完了するまでは、値を変更する必要はありません。
- SAVEDS キーワードは、ファイルに獲得された各装置ごとに保管および復元されるデータ構造を示します。SAVEIND キーワードは、ファイルに獲得された各装置ごとに保管および復元される標識のセットを示します。入力操作の前に、標識およびデータ構造の現在のセットが保管されます。入力操作の後に、RPG コンパイラーはこの操作と関連した装置の標識およびデータ構造を復元します。これは、入力操作の前に使用可能であった標識およびデータ構造のセットと異なる場合があります。
- INFDS キーワードは、WORKSTN ファイルのファイル情報データ構造を指定します。RPG *STATUS フィールドおよび入出力操作に対するメジャー / マイナー戻りコードは、このデータ構造を介してアクセスすることができます。ICF が使用されている時には特に、複数装置ファイルへの入出力操作中に起こったエラーを検出するために、両方のフィールドが有用です。

  注: これらの制御オプションを指定する時には、DEVID、SAVEIND、または SAVEDS オプションよりも前に MAXDEV オプションをコーディングしなければなりません。

第 20 章 対話式アプリケーションの例

この章では、一般的ないくつかのワークステーションのアプリケーションと、それらの ILE RPG コーディングについて説明します。

この章で提示されるアプリケーション・プログラムは、4 つのモジュールから構成されています。各モジュールで、WORKSTN ファイルの一般的な使用方法について説明します。最初のモジュール (CUSMAIN) では、プログラムのメイン・メニューを提供します。ユーザーの選択に基づいて、該当するモジュールから、要求された機能を提供するプロシージャーを呼び出します。

各モジュールは、WORKSTN ファイルを使用して、画面上の入力および表示情報についてプロンプトを出します。また、メイン・モジュール CUSMAIN を除く各モジュールは、マスター・データベース・ファイルのビュー を表示する論理ファイルも使用します。このビューは、モジュールがその処理に必要とするマスター・ファイルのフィールドのみで構成されます。

注: CUSMAIN 以外の各モジュールは、それぞれ独立したプログラムとしてコンパイルすることができます。すなわち、各モジュールをそれぞれ独立したプログラムとして使用することができます。

表 44. 対話式アプリケーション例の各モジュールの説明

モジュール	説明
412 ページの『メイン・メニュー照会』	WORKSTN ファイルを使用して、メニュー選択項目を表示し、入力を受け入れる、基本的なメニュー照会プログラムの例。
415 ページの『ファイルの維持』	マスター・ファイルの得意先レコードの更新、削除、追加、および表示を可能にする保守プログラムの例。
426 ページの『郵便番号による検索』	WORKSTN サブファイル処理を使用して、指定された郵便番号について一致したすべてのレコードを表示するプログラムの例。
434 ページの『名前による検索と照会』	WORKSTN サブファイル処理を使用して、指定された得意先名について一致したすべてのレコードを表示し、次にユーザーがサブファイルからレコードを選択して完全な得意先情報を表示できるようにする、プログラムの例。

データベース物理ファイル

412 ページの図 187 は、得意先マスター・ファイルのデータ記述仕様書 (DDS) を示しています。このファイルには、各得意先の重要情報 (名前、住所、勘定残高、および得意先番号など) が入っています。得意先情報を必要とするすべてのモジュールは、このデータベース・ファイル (あるいはその論理ビュー) を使用します。

```
A*****************************************************************
A*    ファイル名:  CUSMST                                        *
A*関連プログラム:  CUSMNT, SCHZIP, SCHNAM                        *
A*  関連ファイル:  CUSMSTL1, CUSMSTL2, CUSMSTL3 (論理ファイル)   *
A*          説明:  これは物理ファイル CUSMST です。これには      *
A*                 CUSREC というレコード様式が 1 つあります。    *
A*****************************************************************
A* CUSTOMER MASTER FILE -- CUSMST
A          R CUSREC
A            CUST           5  0       TEXT('CUSTOMER NUMBER')
A            NAME          20          TEXT('CUSTOMER NAME')
A            ADDR1         20          TEXT('CUSTOMER ADDRESS')
A            ADDR2         20          TEXT('CUSTOMER ADDRESS')
A            CITY          20          TEXT('CUSTOMER CITY')
A            STATE          2          TEXT('CUSTOMER STATE')
A            ZIP            5  0       TEXT('CUSTOMER ZIP CODE')
A            ARBAL         10  2       TEXT('ACCOUNTS RECEIVABLE BALANCE')
```

図 187. マスター・データベース・ファイル CUSMST (物理ファイル) の DDS

メイン・メニュー照会

この項では、WORKSTN ファイルを使用してメニュー選択項目の表示と入力の受け入れを行う、簡単な照会プログラムについて説明します。

MAINMENU: 表示装置ファイルの DDS

MAINMENU 表示装置ファイルの DDS は、ファイル・レベル項目を指定し、1 つのレコード様式 HDRSCN を記述します。ファイル・レベル項目は、画面サイズ (DSPSIZ)、デフォルトの入力値 (CHGINPDFT)、ページ印刷キー (PRINT)、および独立標識域 (INDARA) を定義します。

HDRSCN レコード様式には、画面を識別する固定情報 'CUSTOMER MAIN INQUIRY' が入っています。また、画面上に現在の時刻と日付を表示するキーワード TIME および DATE も入っています。CA キーワードは、使用可能な機能キーを定義し、それらの機能キーを RPG プログラム中の標識と関連付けています。

```
A*****************************************************************
A*    ファイル名:  MAINMENU                                      *
A*関連プログラム:  CUSMAIN                                       *
A*          説明:  これは表示装置ファイル MAINMENU です。これ    *
A*                 には 1 つのレコード様式 HDRSCN があります。    *
A*****************************************************************
A                                      DSPSIZ(24 80 *DS3)
A                                      CHGINPDFT(CS)
A                                      PRINT(QSYSPRT)
A                                      INDARA
A          R HDRSCN
A                                      CA03(03 'END OF INQUIRY')
A                                      CA05(05 'MAINTENANCE MODE')
A                                      CA06(06 'SEARCH BY ZIP MODE')
A                                      CA07(07 'SEARCH BY NAME MODE')
A                                   2  4TIME
A                                      DSPATR(HI)
A                                   2 28'CUSTOMER MAIN INQUIRY'
A                                      DSPATR(HI)
A                                      DSPATR(RI)
A                                   2 70DATE
A                                      EDTCDE(Y)
A                                      DSPATR(HI)
A                                   6  5'Press one of the following'
A                                   6 32'PF keys.'
A                                   8 22'F3 End Job'
A                                   9 22'F5 Maintain Customer File'
A                                  10 22'F6 Search Customer by Zip Code'
A                                  11 22'F7 Search Customer by Name'
```

図 188. 表示装置ファイル MAINMENU の DDS

レコード様式は、画面の固定情報、フィールド、行番号、および桁番号を記述しているほかに、これらの項目の表示属性も定義しています。

注: 通常、フィールド属性はファイルの DDS ではなく、フィールド参照ファイルに定義されます。属性が何であるか分るように、DDS 上に属性を示してあります。

CUSMAIN: RPG ソース

```
//*****************************************************************
// プログラム名:   CUSMAIN                                        *
// 関連ファイル:   MAINMENU (DSPF)                                *
// 関連プログラム: CUSMNT   (ILE RPG PGM)                         *
//                 SCHZIP   (ILE RPG PGM)                         *
//                 SCHNAM   (ILE RPG PGM)                         *
//         説明:   これは得意先のメイン照会プログラムです。       *
//                 このプログラムは、次の処置のいずれかを選ぶ     *
//                 ようにユーザーにプロンプトを出します。         *
//                 1.得意先レコードの維持                         *
//                     (追加、更新、削除、表示)                   *
//                 2.郵便番号による得意先レコードの検索           *
//                 3.名前による得意先レコードの検索               *
//*****************************************************************

Fmainmenu  cf   e             workstn indds(indicators)

 // プロトタイプ定義:
D CustMaintain    pr                  extproc('CUSMNT')
D SearchZip       pr                  extproc('SCHZIP')
D SearchName      pr                  extproc('SCHNAM')

 // フィールド定義:
D indicators      ds
D    exitKey                      n   overlay(indicators:3)
D    maintainKey                  n   overlay(indicators:5)
D    srchZipKey                   n   overlay(indicators:6)
D    srchCustKey                  n   overlay(indicators:7)

 /free
      // 終了キーが押されるまでループする
      dow  '1';
         // メインメニューの表示
         exfmt hdrscn;

         // 要求された処置の実行
         if exitKey;
            // プログラムの終了
            leave;

         elseif maintainKey;
            // 得意先データの維持
            CustMaintain();

         elseif srchZipKey;
            // 郵便番号に基づく得意先の検索
            SearchZip();

         elseif srchCustKey;
            // 得意先名による得意先の検索
            SearchName();
         endif;
      enddo;

      *inlr = *on;
 /end-free
```

図 189. モジュール CUSMAIN のソース

このモジュールでは、CALLB 命令コードの使用法について説明します。適当な RPG モジュール (CUSMNT、SCHZIP、または SCHNAM) は、ユーザーの選択するメニュー項目に従って CUSMAIN によって呼び出されます。

プログラム・オブジェクトを作成するために、次のことを行ってください。

1. CRTRPGMOD を使用して、各ソース・メンバー (CUSMAIN、CUSMNT、SCHZIP、および SCHNAM) のモジュールを作成する。
2. 次を入力することによってプログラムを作成する。

   ```
   CRTPGM PGM(MYPROG) MODULE(CUSMAIN CUSMNT SCHZIP SCHNAM) ENTMOD(*FIRST)
   ```

   注: *FIRST オプションは、リストの最初のモジュールである CUSMAIN がプログラム入力プロシージャーとして選択されることを指定します。
3. 次を入力することによってプログラムを呼び出す。

   ```
   CALL MYPROG
   ```

図 190 に示すように『メイン・メニュー』が現れます。

```
22:30:05              CUSTOMER MAIN INQUIRY                  9/30/94


 Press one of the following PF keys.

                  F3 End Job
                  F5 Maintain Customer File
                  F6 Search Customer by Zip Code
                  F7 Search Customer by Name
```

図 190. 得意先メイン照会プロンプト画面

ファイルの維持

この項では、WORKSTN ファイルを使用する維持プログラムについて説明します。このプログラムにより、得意先マスター・ファイルのレコードの追加、削除、更新、および表示を行うことができます。

CUSMSTL1: 論理ファイルの DDS

```
     A*****************************************************************
     A*    ファイル名:  CUSMSTL1                                      *
     A*関連プログラム:  CUSMNT                                       *
     A*  関連ファイル:  CUSMST   (物理ファイル)                       *
     A*          説明:  これは論理ファイル CUSMSTL1 です。これには    *
     A*                 1 つのレコード様式 CMLREC1 があります。       *
     A*                 これは得意先マスター・ファイル (CUSMST) の    *
     A*                 得意先番号 (CUST) による論理ビューです。      *
     A*****************************************************************
     A          R CMLREC1                   PFILE(CUSMST)
     A            CUST
     A            NAME
     A            ADDR1
     A            ADDR2
     A            CITY
     A            STATE
     A            ZIP
     A          K CUST
```

図 191. 論理ファイル CUSMSTL1 の DDS

このプログラムが使用するデータベース・ファイルの DDS は、1 つのレコード様式 CMLREC1 を記述しています。このレコード様式の各フィールドが記述され、CUST フィールドがこのレコード様式のキー・フィールドとして識別されます。

MNTMENU: 表示装置ファイルの DDS

```
A*********************************************************************
A*    ファイル名:  MNTMENU                                           *
A*関連プログラム:  CUSMNT                                            *
A*  関連ファイル:  CUSMSTL1   (論理ファイル)                         *
A*          説明:  これは表示装置ファイル MNTMENU です。              *
A*                 レコード様式があります                            *
A*********************************************************************
A                                     REF(CUSMSTL1)
A                                     CHGINPDFT(CS)
A                                     PRINT(QSYSPRT)
A                                     INDARA
A          R HDRSCN
A                                     TEXT('PROMPT FOR CUST NUMBER')
A                                     CA03(03 'END MAINTENANCE')
A                                     CF05(05 'ADD MODE')
A                                     CF06(06 'UPDATE MODE')
A                                     CF07(07 'DELETE MODE')
A                                     CF08(08 'DISPLAY MODE')
A            MODE           8A  O  1  4DSPATR(HI)
A                                    1 13'MODE'
A                                      DSPATR(HI)
A                                    2  4TIME
A                                      DSPATR(HI)
A                                    2 28'CUSTOMER FILE MAINTENANCE'
A                                      DSPATR(HI RI)
A                                    2 70DATE
A                                      EDTCDE(Y)
A                                      DSPATR(HI)
A            CUST      R     Y  I 10 25DSPATR(CS)
A                                     CHECK(RZ)
A  51                                 ERRMSG('CUSTOMER ALREADY ON +
A                                     FILE' 51)
A  52                                 ERRMSG('CUSTOMER NOT ON FILE' +
A                                     52)
A                                   10 33'<--Enter Customer Number'
A                                      DSPATR(HI)
A                                   23  4'F3 End Job'
A                                   23 21'F5 Add'
A                                   23 34'F6 Update'
A                                   23 50'F7 Delete'
A                                   23 66'F8 Display'
```

図 192. 表示装置ファイル MNTMENU の DDS (1/2)

```
A          R CSTINQ
A                                      TEXT('DISPLAY CUST INFO')
A                                      CA12(12 'PREVIOUS SCREEN')
A            MODE           8A  O  1  4DSPATR(HI)
A                                    1 13'MODE'
A                                       DSPATR(HI)
A                                    2  4TIME
A                                       DSPATR(HI)
A                                    2 28'CUSTOMER FILE MAINTENANCE'
A                                       DSPATR(HI)
A                                       DSPATR(RI)
A                                    2 70DATE
A                                       EDTCDE(Y)
A                                       DSPATR(HI)
A                                    4 14'Customer:'
A                                       DSPATR(HI)
A                                       DSPATR(UL)
A            CUST      R        O  4 25DSPATR(HI)
A            NAME      R        B  6 25DSPATR(CS)
A  04                                   DSPATR(PR)
A            ADDR1     R        B  7 25DSPATR(CS)
A  04                                   DSPATR(PR)
A            ADDR2     R        B  8 25DSPATR(CS)
A  04                                   DSPATR(PR)
A            CITY      R        B  9 25DSPATR(CS)
A  04                                   DSPATR(PR)
A            STATE     R        B 10 25DSPATR(CS)
A  04                                   DSPATR(PR)
A            ZIP       R        B 10 40DSPATR(CS)
A                                       EDTCDE(Z)
A  04                                   DSPATR(PR)
A                                      23  2'F12 Cancel'
A            MODE1          8   O 23 13
A          R CSTBLD                     TEXT('ADD CUST RECORD')
A                                       CA12(12 'PREVIOUS SCREEN')
A            MODE           8   O  1  4DSPATR(HI)
A                                    1 13'MODE'     DSPATR(HI)
A                                    2  4TIME
A                                       DSPATR(HI)
A                                    2 28'CUSTOMER FILE MAINTENANCE'
A                                       DSPATR(HI RI)
A                                    2 70DATE
A                                       EDTCDE(Y)
A                                       DSPATR(HI)
A                                    4 14'Customer:' DSPATR(HI UL)
A            CUST      R        O  4 25DSPATR(HI)
A                                    6 20'Name'     DSPATR(HI)
A            NAME      R        I  6 25
A                                    7 17'Address'  DSPATR(HI)
A            ADDR1     R        I  7 25
A                                    8 17'Address'  DSPATR(HI)
A            ADDR2     R        I  8 25
A                                    9 20'City'     DSPATR(HI)
A            CITY      R        I  9 25
A                                   10 19'State'    DSPATR(HI)
A            STATE     R        I 10 25
A                                   10 36'Zip'      DSPATR(HI)
A            ZIP       R     Y  I 10 40
A                                   23  2'F12 Cancel Addition'
```

図 192. 表示装置ファイル MNTMENU の DDS (2/2)

MNTMENU 表示装置ファイルの DDS には、 HDRSCN、CSTINQ、および CSTBLD という 3 つのレコード様式が入っています。HDRSCN レコードは、得意先番号および処理方式のプロンプトを出します。CSTINQ レコードは、更新、削

除、および表示モードのために使用されます。フィールドは出力 / 入力 (38 桁目に B) として定義されています。これらのフィールドは、表示または削除モードが選択された時に保護されます (DSPATR(PR))。 CSTBLD レコードは、新しいレコードの入力フィールド (38 桁目に I) のみを提供します。

HDRSCN レコード様式には、固定情報 'Customer File Maintenance' が入っています。 ERRMSG キーワードは、エラーが起こった場合に表示するメッセージを定義しています。 CA キーワードは、使用可能な機能キーを定義し、それらの機能キーを RPG プログラム中の標識と関連付けています。

CUSMNT: RPG ソース

```
//*****************************************************************
//  プログラム名:  CUSMNT                                         *
//  関連ファイル:  CUSMSTL1 (LF)                                  *
//                 MNTMENU  (DSPF)                                *
//          説明:  このプログラムは、WORKSTN ファイルを使用した  *
//                 得意先マスター・ファイル維持プログラムです。   *
//                 このプログラムでユーザーは、得意先レコードを  *
//                 追加、更新、削除、表示できます。               *
//                 PF3 を使用してプログラムを終了します。         *
//*****************************************************************

Fcusmstl1  uf a e           k disk
Fmntmenu   cf   e             workstn indds(indicators)

 // フィールド定義:

D indicators      ds
D   exitKey                      n   overlay(indicators:3)
D   disableInput                 n   overlay(indicators:4)
D   addKey                       n   overlay(indicators:5)
D   updateKey                    n   overlay(indicators:6)
D   deleteKey                    n   overlay(indicators:7)
D   displayKey                   n   overlay(indicators:8)
D   prevKey                      n   overlay(indicators:12)
D   custExists                   n   overlay(indicators:51)
D   custNotFound                 n   overlay(indicators:52)

 // キー・リスト定義

C     CSTKEY        KLIST
C                   KFLD                    CUST
```

図 193. モジュール CUSMNT のソース (1/5)

```
     //*****************************************************************
     //   メインライン                                                  *
     //*****************************************************************

     /free

      mode = 'DISPLAY';
      exfmt hdrscn;

      // 終了キーが押されるまでループする
      dow not exitKey;
         exsr SetMaintenanceMode;

         if cust <> 0;
            if mode = 'ADD';
               exsr AddSub;
            elseif mode = 'UPDATE';
               exsr UpdateSub;
            elseif mode = 'DELETE';
               exsr DeleteSub;
            elseif mode = 'DISPLAY';
               exsr InquirySub;
            endif;
         endif;

         exfmt hdrscn;
         custExists   = *off;   // エラー・メッセージをオフにする
         CustNotFound = *off;
      enddo;

      *inlr = *on;
```

図 193. モジュール CUSMNT のソース (2/5)

```
//*****************************************************************
//    サブルーチン - AddSub                                        *
//    目的         - 新しい得意先をファイルに追加する              *
//*****************************************************************
begsr AddSub;

   // 得意先番号は既にファイルにあるか ?
   chain CstKey cmlrec1;
   if %found(cusmstl1);
      // 既に使用されている得意先番号の場合
      custExists = *on;
      leavesr;
   endif;

   // 新しい得意先データの初期化
   custExists   = *off;   // エラー・メッセージをオフにする
   CustNotFound = *off;
   name = *blank;
   addr1 = *blank;
   addr2 = *blank;
   city = *blank;
   state = *blank;
   zip = 0;

   // この得意先レコードの更新データを入れるためのプロンプトを出す
   exfmt cstbld;

   // よければ、得意先を得意先ファイルに追加する
   if not *in12;
      write cmlrec1;
   endif;
endsr;  // サブルーチン AddSub の終わり



//*****************************************************************
//    サブルーチン - UpdateSub                                     *
//    目的         - 得意先マスター・レコードを更新する            *
//*****************************************************************
begsr UpdateSub;

   // 得意先番号の探索
   chain cstkey cmlrec1;
   if not %found(cusmstl1);
      // 得意先がファイルにない
      custNotFound = *on;
      leavesr;
   endif;

   // この得意先の情報を表示する
   disableInput = *off;
   exfmt cstinq;
   if not prevKey;
      // ファイルの中の情報を更新する
      update cmlrec1;
   else;
      // 更新したくない場合でも、少なくとも
      // そのレコードのアンロックは行なう
      unlock cusmstl1;
   endif;
endsr;  // サブルーチン UpdateSub の終わり
```

図 193. モジュール CUSMNT のソース (3/5)

```
     //*****************************************************************
     //    サブルーチン - DeleteSub                                     *
     //    目的         - 得意先マスター・レコードを削除する            *
     //*****************************************************************
     begsr DeleteSub;

        // 得意先番号の探索
        chain cstkey cmlrec1;
        if not %found(cusmstl1);
           // 得意先がファイルにない
           custNotFound = *on;
           leavesr;
        endif;

        // この得意先の情報を表示する
        disableInput = *on;
        exfmt cstinq;
        if not prevKey;
           // 得意先レコードの削除
           delete cmlrec1;
        else;
           // 削除したくない場合でも、少なくとも
           // そのレコードのアンロックは行なう
           unlock cusmstl1;
        endif;
     endsr;  // サブルーチン DeleteSub の終わり



     //*****************************************************************
     //    サブルーチン - InquirySub                                    *
     //    目的         - 得意先マスター・レコードを表示する            *
     //*****************************************************************
     begsr InquirySub;

        // 得意先番号の探索
        chain(n) cstkey cmlrec1;  // レコードをロックしない
        if not %found(cusmstl1);
           // 得意先がファイルにない
           custNotFound = *on;
           leavesr;
        endif;

        // この得意先の情報を表示する
        disableInput = *on;
        exfmt cstinq;
     endsr;  // サブルーチン InquirySub; の終わり
```

図 193. モジュール CUSMNT のソース (4/5)

```
      //*****************************************************************
      //    サブルーチン - SetMaintenanceMode                            *
      //    目的         - 維持モードの設定                              *
      //*****************************************************************
      begsr SetMaintenanceMode;
         if addKey;
            mode = 'ADD';
         elseif updateKey;
            mode = 'UPDATE';
         elseif deleteKey;
            mode = 'DELETE';
         elseif displayKey;
            mode = 'DISPLAY';
         endif;
      endsr;  // サブルーチン SetMaintenanceMode の終わり

     /end-free
```

図 193. モジュール CUSMNT のソース (5/5)

このプログラムは、得意先マスター・ファイルの追加、変更、および削除の保守を行います。このプログラムは照会に使用することもできます。

プログラムは最初は、デフォルト (表示) の処理方式を設定し、得意先保守プロンプト画面を表示します。ワークステーション・ユーザーは、F3 キーを押して (標識 03 がオンになる) ジョブの終了を要求することができます。そうでない場合には、得意先情報を処理するために、ユーザーは得意先番号を入力して、実行キーを押します。ユーザーは、F5 (追加)、F6 (更新)、F7 (削除)、または F8 (表示) を押して処理方式を変更することができます。

新しいレコードをファイルに追加するためには、プログラムは得意先番号を検索引き数として使用して、マスター・ファイルに連鎖します。ファイルにレコードが存在していない場合には、プログラムは CSTBLD 画面を表示して、ユーザーが新しい得意先レコードを入力できるようにします。レコードが既にファイルに存在している場合には、エラー・メッセージが表示されます。ユーザーは、F12 キーを押して (標識 12 をオンに設定する)、追加操作を取り消してレコードを解放することができます。そうでない場合には、追加操作を続行するために、ユーザーは、入力フィールドに新しい得意先レコードの情報を入力し、新しいレコードをマスター・ファイルに書き出します。

既存のレコードを更新、削除、または表示するためには、プログラムは得意先番号を検索引き数として使用し、マスター・ファイルに連鎖します。その得意先のレコードがファイルに存在している場合には、プログラムは得意先照会画面 CSTINQ を表示します。レコードがファイルに存在していない場合には、エラー・メッセージが表示されます。処理方式が表示または削除の場合には、入力フィールドは変更できないように保護されています。そうでない場合には、得意先レコードを続行するために、ユーザーは、得意先レコード入力フィールドに新しい情報を入力することができます。ユーザーは、F12 キーを押して (標識 12 をオンに設定する)、更新操作または削除操作を取り消して、レコードを解放することができます。表示モードでは、実行キーを押すと、自動的にレコードが解放されます。

424 ページの図 194 では、ワークステーション・ユーザーがプロンプトへの応答として得意先番号 00007 を入力して、その得意先レコードを表示します。

```
DISPLAY MODE
22:30:21              CUSTOMER FILE MAINTENANCE                9/30/94




                    00007   <--Enter Customer Number




F3 End Job       F5 Add       F6 Update       F7 Delete       F8 Display
```

図 194. 'CUSTOMER FILE MAINTENANCE' (得意先ファイル維持) 表示モード・プロンプト画面

得意先番号 00007 の得意先レコードはマスター・ファイルに存在しているので、図 195 に示すようにデータが表示されます。

```
DISPLAY MODE
22:31:06              CUSTOMER FILE MAINTENANCE                9/30/94
          Customer:  00007

                     Mikhail Yuri
                     1001 Bay Street
                     Suite 1702
                     Livonia
                     MI             11201









F12 Cancel DISPLAY
```

図 195. 'CUSTOMER FILE MAINTENANCE' (得意先ファイル維持) 表示モード画面

ワークステーション・ユーザーは、 425 ページの図 196 に示すように、追加プロンプトに対して新しい得意先番号を入力して応答します。

```
ADD MODE
22:31:43                CUSTOMER FILE MAINTENANCE                  9/30/94






                     00012   <--Enter Customer Number















F3 End Job       F5 Add       F6 Update       F7 Delete       F8 Display
```

図 196. 'CUSTOMER FILE MAINTENANCE' (得意先ファイル維持) 追加モード・プロンプト画面

図 197 では、新しい得意先が得意先マスター・ファイルに追加されます。

```
ADD MODE
22:32:04                CUSTOMER FILE MAINTENANCE                     9/30/94
          Customer:  00012

             Name JUDAH GOULD
          Address 2074 BATHURST AVENUE
          Address
             City YORKTOWN
            State NY         Zip 70068












F12 Cancel Addition
```

図 197. 'CUSTOMER FILE MAINTENANCE' (得意先ファイル維持) 追加モード・プロンプト画面

ワークステーション・ユーザーは、 426 ページの図 198 に示すように、削除プロンプトに対して得意先番号を入力して応答します。

```
DELETE MODE
22:32:55              CUSTOMER FILE MAINTENANCE                9/30/94




                    00011   <--Enter Customer Number




F3 End Job       F5 Add       F6 Update       F7 Delete       F8 Display
```

図 198. 'CUSTOMER FILE MAINTENANCE' (得意先ファイル維持) 削除モード・プロンプト画面

ワークステーション・ユーザーは、図 199 に示すように、更新プロンプトに対して得意先番号を入力して応答します。

```
UPDATE MODE
22:33:17              CUSTOMER FILE MAINTENANCE                9/30/94




                    00010   <--Enter Customer Number




F3 End Job       F5 Add       F6 Update       F7 Delete       F8 Display
```

図 199. 'CUSTOMER FILE MAINTENANCE' (得意先ファイル維持) 更新モード・プロンプト画面

郵便番号による検索

この項では、WORKSTN サブファイル処理 (表示のみ) について説明します。サブファイルは、指定された郵便番号について一致したすべてのレコードを表示するために使用されます。

CUSMSTL2: 論理ファイルの DDS

```
A*****************************************************************
A*    ファイル名:  CUSMSTL2                                      *
A*関連プログラム:  SCHZIP                                       *
A*  関連ファイル:  CUSMST   (物理ファイル)                       *
A*          説明:  これは論理ファイル CUSMSTL2 です。これには    *
A*                 1 つのレコード様式 CMLREC2 があります。       *
A*                 これは得意先マスター・ファイル (CUSMST) の    *
A*                 得意先郵便番号 (ZIP) による論理ビューです。    *
A*****************************************************************
A          R CMLREC2                   PFILE(CUSMST)
A            ZIP
A            NAME
A            ARBAL
A          K ZIP
```

図 200. 論理ファイル CUSMSTL2 の DDS

このプログラムが使用するデータベース・ファイルの DDS は、1 つのレコード様式 CMLREC2 を記述しています。郵便番号によるキー順の論理ファイル CUSMSTL2 は、PFILE キーワードによって示されているように、物理ファイル CUSMST を基礎にしています。論理ファイルによって作成されたレコード様式には、論理ファイル DDS に指定されたこれらのフィールドだけが含まれます。その他のフィールドはすべて除外されます。

SZIPMENU: 表示装置ファイルの DDS

```
A*****************************************************************
A*    ファイル名:  SZIPMENU                                      *
A*関連プログラム:  SCHZIP                                       *
A*  関連ファイル:  CUSMSTL2   (論理ファイル)                     *
A*          説明:  これは、表示装置ファイル SZIPMENU です。      *
A*                 レコード様式があります                        *
A*****************************************************************
A                                      REF(CUSMSTL2)
A                                      CHGINPDFT(CS)
A                                      PRINT(QSYSPRT)
A                                      INDARA
A                                      CA03(03 'END OF JOB')
A          R HEAD
A                                      OVERLAY
A                                    2  4TIME
A                                      DSPATR(HI)
A                                    2 28'CUSTOMER SEARCH BY ZIP'
A                                      DSPATR(HI RI)
A                                    2 70DATE
A                                      EDTCDE(Y)
A                                      DSPATR(HI)
A          R FOOT1
A                                   23  6'ENTER - Continue'
A                                      DSPATR(HI)
A                                   23 29'F3 - End Job'
A                                      DSPATR(HI)
A          R FOOT2
A                                   23  6'ENTER - Continue'
A                                      DSPATR(HI)
A                                   23 29'F3 - End Job'
A                                      DSPATR(HI)
A                                   23 47'F4 - RESTART ZIP CODE'
A                                      DSPATR(HI)
A          R PROMPT
A                                      OVERLAY
A                                    4  4'Enter Zip Code'
A                                      DSPATR(HI)
A            ZIP       R     Y  I  4 19DSPATR(CS)
A                                      CHECK(RZ)
A  61                                  ERRMSG('ZIP CODE NOT FOUND' +
A                                      61)
A          R SUBFILE                   SFL
A            NAME      R           9  4
A            ARBAL     R           9 27EDTCDE(J)
A          R SUBCTL                    SFLCTL(SUBFILE)
A  55                                  SFLCLR
A  55                                  SFLCLR
A N55                                  SFLDSPCTL
A N55                                  SFLDSP
A                                      SFLSIZ(13)
A                                      SFLPAG(13)
A                                      ROLLUP(95 'ROLL UP')
A                                      OVERLAY
A                                      CA04(04 'RESTART ZIP CDE')
A                                    4  4'Zip Code'
A            ZIP       R        O  4 14DSPATR(HI)
A                                    7  4'Customer Name'
A                                      DSPATR(HI UL)
A                                    7 27'A/R Balance'
A                                      DSPATR(HI UL)
```

図 201. 表示装置ファイル SZIPMENU の DDS

SZIPMENU 表示装置ファイルの DDS には、HEAD、FOOT1、FOOT2、PROMPT、SUBFILE、および SUBCTL という 6 つのレコード様式があります。

PROMPT レコード様式は、ユーザーに郵便番号を入力するように要求します。郵便番号がファイルに見付からない場合には、エラー・メッセージが表示されます。ユーザーは、F3 キーを押して (標識 03 をオンに設定する)、プログラムを終了することができます。

サブファイル・レコード様式は、サブファイル制御レコード様式 SUBCTL の直前に定義しなければなりません。サブファイル・レコード様式はキーワード SFL を用いて定義しますが、これはレコード内の各フィールドを記述し、最初のレコードが画面に表示される位置 (ここでは 9 行目) を指定します。

サブファイル制御レコード様式には、次の固有なキーワードが含まれています。

- SFLCTL は、この様式が制御レコード様式であることを示すとともに、関連するサブファイル・レコード様式の名前を指定します。
- SFLCLR は、いつサブファイルから既存のレコードを消去するか (標識 55 がオンの場合) を記述します。このキーワードは追加の画面に必要です。
- SFLDSPCTL は、どのような場合にサブファイル制御レコード様式を表示するか (標識 55 がオフの場合) を示します。
- SFLDSP は、どのような場合にサブファイルを表示するか (標識 55 がオフの場合) を示します。
- SFLSIZ は、サブファイルの合計サイズを指定します。この例ではサブファイル・サイズは 13 レコードで、9 ～ 21 行目に表示されています。
- SFLPAG は、1 ページ当たりのレコード数を定義します。この例では、ページ・サイズはサブファイル・サイズと同じです。
- ROLLUP は、次ページ機能が使用されると、プログラムで標識 95 がオンに設定されることを示します。

OVERLAY キーワードは、このサブファイル制御レコード様式をオーバーレイ様式として定義します。このレコード様式は、OS/400 システムが最初に画面を消去しなくても書き出すことができます。F4 は、同じ郵便番号の検索を繰り返す場合に有効です。 (この F4 の使用によって前ページの形式が使用可能です。)

SCHZIP: RPG ソース

```
//*****************************************************************
// プログラム名:  SCHZIP                                          *
// 関連ファイル:  CUSMSTL2 (論理ファイル)                         *
//                SZIPMENU (WORKSTN ファイル)                     *
//         説明:  このプログラムは、WORKSTN サブファイル処理を    *
//                使用した得意先マスター検索プログラムです。      *
//                このプログラムは、ユーザーに郵便番号の          *
//                プロンプトを出し、郵便番号による得意先          *
//                マスター・レコードを表示します。                *
//                次ページ・キーを使用して、別のページを見ること *
//                ができ、PF3 でプログラムが終了します。          *
//*****************************************************************
Fcusmstl2  if   e           k disk
Fszipmenu  cf   e             workstn sfile(subfile:recnum)
F                                     indds(indicators)


 // フィールド定義:
D recnum          s               5p 0
D recordFound     s                n

D indicators      ds
D    exitKey                       n   overlay(indicators:3)
D    restartKey                    n   overlay(indicators:4)
D    sflClear                      n   overlay(indicators:55)
D    zipNotFound                   n   overlay(indicators:61)
D    rollupKey                     n   overlay(indicators:95)

 // キー・リスト定義
C     cstkey        klist
C                   kfld                    zip
```

図 202. モジュール SCHZIP のソース (1/3)

```
//*********************************************************************
//    メインライン                                                     *
//*********************************************************************

 /free

  // 初期メニューの書き出し
  write foot1;
  write head;
  exfmt prompt;

  // PF03 押されるまでループする
  dow not exitKey;
     setll cstkey cmlrec2;
     recordFound = %equal(cusmstl2);
     if recordFound;
        exsr ProcessSubfile;
     endif;

     // サブファイル表示で PF03 が押された場合は、ループを終了
     if exitKey;
        leave;
     endif;

     // PF04 が押された場合は、
     // 同じ郵便番号で検索を再実行する。
     if restartKey;
        iter;
     endif;

     // 新規郵便番号のプロンプトを出す。
     if not recordFound;
        // 郵便番号が検出されなかった場合は、
        // 再びヘッダーとフッターを書かない。
        write foot1;
        write head;
     endif;
     zipNotFound = not recordFound;
     exfmt prompt;
  enddo;

  *inlr = *on;
```

図 202. モジュール SCHZIP のソース (2/3)

```
     //*****************************************************************
     //  サブルーチン - ProcessSubfile                                  *
     //  目的     - サブファイルを処理し、それを表示する                *
     //*****************************************************************
     begsr ProcessSubfile;

        // ロールアップ・キーが押されるまでループする
        dou not rollupKey;
           // サブファイルに追加する情報は他にあるか ?
           if not %eof(cusmstl2);
              // サブファイルを消去し、得意先データで充てんする
              exsr ClearSubfile;
              exsr FillSubfile;
           endif;

           // サブファイルを書き出し、応答を待つ
           write foot2;
           exfmt subctl;
        enddo;

     endsr;  // サブルーチン ProcessSubfile の終わり



     //*****************************************************************
     //   サブルーチン - FillSubfile                                    *
     //   目的      - 指定した郵便番号に一致する得意先レコードで        *
     //               サブファイルを充てんする。                        *
     //*****************************************************************
     begsr FillSubfile;

        // 指定した郵便番号で得意先レコード全体をループする
        recnum = 0;
        dou %eof(szipmenu);
           // 指定した郵便番号で次のレコードを読み取る
           reade zip cmlrec2;
           if %eof(cusmstl2);
              // レコードがなくなったら、以下を行う
              leavesr;
           endif;

           // このレコードの情報をサブファイルに追加する
           recnum = recnum + 1;
           write subfile;
        enddo;
     endsr;  // サブルーチン  FillSubfile の終わり



     //*****************************************************************
     //    サブルーチン - ClearSubfile                                  *
     //    目的       - サブファイル・レコードの消去                    *
     //*****************************************************************
     begsr ClearSubfile;

        sflClear = *on;
        write subctl;
        sflClear = *off;

     endsr;  // サブルーチン ClearSubfile の終わり

    /end-free
```

図 202. モジュール SCHZIP のソース (3/3)

ファイル仕様書は、検索するディスク・ファイルおよび使用される表示装置ファイル (SZIPMENU) を指定します。WORKSTN ファイルの SFILE キーワードは、サブファイルとして使用されるレコード様式 (SUBFILE) を指定します。指定された相対レコード番号フィールド (RECNUM) は、サブファイル内のどのレコードをアクセスするかを制御します。

プログラムは、PROMPT レコード様式を表示し、ワークステーション・ユーザーの応答を待機します。F3 は、プログラムの終了を制御する標識 03 をオンに設定します。郵便番号 (ZIP) は、SETLL 命令によって CUSMSTL2 ファイルを位置付けるために使用されます。 SETLL 命令では、ファイル名 CUSMSTL2 ではなく、レコード様式名 CMLREC2 が使用されることに注意してください。レコードが見付からない場合には、エラー・メッセージが表示されます。

SFLPRC サブルーチンは、サブファイルの処理 (消去、充てん、および表示) を扱います。サブファイルは、サブルーチン SFLCLR の中での追加の要求のために準備されます。標識 55 がオンの場合には、画面上で処置は行われませんが、サブファイル・レコードの主記憶域は消去されます。SFLFIL ルーチンはサブファイルをレコードで充てんします。レコードは、CUSMSTL2 ファイルから読み取られます。郵便番号が同じである場合には、レコード・カウント (RECNUM) が増え、レコードはそのサブファイルに書き出されます。このサブルーチンは、サブファイルがいっぱいになるか (WRITE 命令の標識 21)、または CUSMSTL2 ファイルでファイルの終わりが起こる (READE 命令の標識 71) まで、反復されます。サブファイルがいっぱいになるか、ファイルの終わりが検出されると、サブファイルが EXFMT 命令によってサブファイル制御レコード様式で、画面に表示されます。ユーザーは、画面を検討して次のことを決定します。

- F3 キーを押してプログラムを終了する。
- F4 キーを押してそのサブファイルを再開する。PROMPT レコード様式は表示されずに、サブファイルが同じ郵便番号から再び表示されます。
- 次ページ・キーを押して別のページを充てんする。CUSMSTL2 ファイルでファイルの終わりが検出された場合には、現在のページが再び表示され、そうでない場合には、サブファイルが消去されて、次のページが表示されます。
- 実行キーを押して別の郵便番号から続行する。PROMPT レコード様式が表示されます。ユーザーは、郵便番号を入力するか、またはプログラムを終了することができます。

434 ページの図 203 では、ユーザーはプロンプトに応答して郵便番号を入力します。

```
22:34:38               CUSTOMER SEARCH BY ZIP                   9/30/94

Enter Zip Code 11201

  ENTER - Continue       F3 - End Job
```

図 203. 'CUSTOMER SEARCH BY ZIP' (郵便番号による得意先検索) プロンプト画面

図 204 に示されているようにサブファイルが画面に書き出されます。

```
22:34:45               CUSTOMER SEARCH BY ZIP                   9/30/94

Zip Code  11201

Customer Name          A/R Balance

Rick Coupland                300.00
Mikhail Yuri                 150.00
Karyn Sanders                  5.00

  ENTER - Continue       F3 - End Job      F4 - RESTART ZIP CODE
```

図 204. 'CUSTOMER SEARCH BY ZIP' (郵便番号による得意先検索) 画面

名前による検索と照会

この項では、WORKSTN サブファイル処理 (選択を伴う表示) について説明します。指定された得意先名について一致するすべてのレコードを表示するためにサブファイルが使用され、その後で、ユーザーはサブファイルから選択を行うことができ、例えば得意先に関する追加情報を表示することができます。

CUSMSTL3: 論理ファイルの DDS

```
A*****************************************************************
A*    ファイル名:  CUSMSTL3                                      *
A*関連プログラム:  SCHNAM                                       *
A* 関連ファイル :  CUSMST                                       *
A*          説明:  これは論理ファイル CUSMSTL3 です。これには、  *
A*                 CUSREC というレコード様式が 1 つあります。    *
A*                 これは得意先マスター・ファイル (CUSMST) の    *
A*                 得意先名 (NAME) による論理ビューです。        *
A*****************************************************************
A          R CUSREC                    PFILE(CUSMST)
A          K NAME
A*
A*****************************************************************
A*  注 : 物理ファイル (CUSMST) のレコード様式は、同じレコード    *
A*       様式名なので、フィールドのリストはこの DDS ファイルでは *
A*       必要ありません。                                        *
A*****************************************************************
```

図 205. 論理ファイル CUSMSTL3 の DDS

このプログラムで使用されるデータベース・ファイルの DDS は、CUSREC という名前の 1 つのレコード様式を定義しており、NAME フィールドをキー・フィールドとして指定しています。

SNAMMENU: 表示装置ファイルの DDS

```
A*****************************************************************
A*    ファイル名:  SNAMMENU                                      *
A*関連プログラム:  SCHNAM                                       *
A* 関連ファイル :  CUSMSTL3   (論理ファイル)                     *
A*          説明:  これは表示ファイル SNAMMENU です。7 つの      *
A*                 レコード様式があります                        *
A*****************************************************************
A                                      REF(CUSMSTL3)
A                                      CHGINPDFT(CS)
A                                      PRINT(QSYSPRT)
A                                      INDARA
A                                      CA03(03 'END OF JOB')
A          R HEAD
A                                      OVERLAY
A                                    2  4TIME
A                                       DSPATR(HI)
A                                    2 25'CUSTOMER SEARCH & INQUIRY BY NAME'
A                                       DSPATR(HI UL)
A                                    2 70DATE
A                                       EDTCDE(Y)
A                                       DSPATR(HI)
A          R FOOT1
A                                   23  6'ENTER - Continue'
A                                       DSPATR(HI)
A                                   23 29'F3 - End Job'
A                                       DSPATR(HI)
A          R FOOT2
A                                   23  6'ENTER - Continue'
A                                       DSPATR(HI)
A                                   23 29'F3 - End Job'
A                                       DSPATR(HI)
A                                   23 47'F4 - Restart Name'
A                                       DSPATR(HI)
A          R PROMPT
A                                       OVERLAY
A                                    5  4'Enter Search Name'
```

図 206. 表示装置ファイル SNAMMENU の DDS (1/2)

```
     A                                      DSPATR(HI)
     A            SRCNAM    R        I  5 23REFFLD(NAME CUSMSTL3)
     A                                      DSPATR(CS)
     A          R SUBFILE                   SFL
     A                                      CHANGE(99 'FIELD CHANGED')
     A            SEL            1A  B  9  8DSPATR(CS)
     A                                      VALUES(' ' 'X')
     A            ZIP       R        O  9 54
     A            CUST      R        O  9 43
     A            NAME      R        O  9 17
     A          R SUBCTL                    SFLCTL(SUBFILE)
     A                                      SFLSIZ(0013)
     A                                      SFLPAG(0013)
     A  55                                  SFLCLR
     A N55                                  SFLDSPCTL
     A N55                                  SFLDSP
     A                                      ROLLUP(95 'ROLL UP')
     A                                      OVERLAY
     A                                      CF04(04 'RESTART SEARCH NAME')
     A                                   5  4'Search Name'
     A            SRCNAM    R        O  5 17REFFLD(NAME CUSMSTL3)
     A                                      DSPATR(HI)
     A                                   7  6'Select'
     A                                      DSPATR(HI)
     A                                   8  6' "X"       Customer Name '
     A                                      DSPATR(HI)
     A                                      DSPATR(UL)
     A                                   8 42' Number     Zip Code   '
     A                                      DSPATR(HI)
     A                                      DSPATR(UL)
     A          R CUSDSP
     A                                      OVERLAY
     A                                   6 25'Customer'
     A            CUST           5S 0O  6 35DSPATR(HI)
     A                                   8 25'Name'
     A            NAME          20A  O  8 35DSPATR(HI)
     A                                  10 25'Address'
     A            ADDR1         20A  O 10 35DSPATR(HI)
     A            ADDR2         20A  O 11 35DSPATR(HI)
     A                                  13 25'City'
     A            CITY          20A  O 13 35DSPATR(HI)
     A                                  15 25'State'
     A            STATE          2A  O 15 35DSPATR(HI)
     A                                  15 41'Zip Code'
     A            ZIP            5S 0O 15 50DSPATR(HI)
     A                                  17 25'A/R Balance'
     A            ARBAL         10Y 2O 17 42DSPATR(HI)
     A                                      EDTCDE(J)
```

図 206. 表示装置ファイル SNAMMENU の DDS (2/2)

SNAMMENU 表示装置ファイルの DDS には、HEAD、FOOT1、FOOT2、PROMPT、SUBFILE、SUBCTL、および CUSDSP という 7 つのレコード様式があります。

PROMPT レコード様式は、ユーザーに郵便番号と検索名を入力するように要求します。この項目を入力しない場合には、表示はファイルの始めから開始されます。ユーザーは、F3 キーを押して (標識 03 をオンに設定する)、プログラムを終了することができます。

サブファイル・レコード様式は、サブファイル制御レコード様式 SUBCTL の直前に定義しなければなりません。キーワード SFL によって定義されるサブファイル・

レコード様式は、レコードの各フィールドを記述し、最初のレコードが画面に表示される位置 (ここでは 9 行目) を指定します。

サブファイル制御レコード様式 SUBCTL には、次の固有なキーワードが含まれています。

- SFLCTL は、この様式が制御レコード様式であることを示すとともに、関連するサブファイル・レコード様式の名前を指定します。
- SFLCLR は、いつサブファイルから既存のレコードを消去するか (標識 55 がオンの場合) を記述します。このキーワードは追加の画面に必要です。
- SFLDSPCTL は、どのような場合にサブファイル制御レコード様式を表示するか (標識 55 がオフの場合) を示します。
- SFLDSP は、どのような場合にサブファイルを表示するか (標識 55 がオフの場合) を示します。
- SFLSIZ は、サブファイルの合計サイズを指定します。この例ではサブファイル・サイズは 13 レコードで、9 ～ 21 行目に表示されています。
- SFLPAG は、1 ページ当たりのレコード数を定義します。この例では、ページ・サイズはサブファイル・サイズと同じです。
- ROLLUP は、次ページ機能が使用されると、プログラムで標識 95 がオンに設定されることを示します。

OVERLAY キーワードは、このサブファイル制御レコード様式をオーバーレイ様式として定義します。このレコード様式は、OS/400 システムが最初に画面を消去しなくても書き出すことができます。F4 は、同じ名前で検索を繰り返す場合に有効です。 (この F4 の使用によって前ページの形式が使用可能です。)

CUSDSP レコード様式は、選択された得意先の情報を表示します。

SCHNAM: RPG ソース

```
//*****************************************************************
// プログラム名:   SCHNAM                                         *
// 関連ファイル:  CUSMSTL3 (論理ファイル)                         *
//                SNAMMENU (WORKSTN ファイル)                     *
//         説明:  このプログラムは、WORKSTN サブファイル処理を    *
//                使用した得意先マスター検索プログラムです。      *
//                このプログラムは、得意先名でユーザーに          *
//                プロンプトを出し、setll 命令での cusmstl3 の    *
//                位置付けにそれを使用します。さらにサブファイル *
//                を使ってレコードを表示します。                  *
//                他のページを充てんするために、ロールアップ・    *
//                キーを押します。得意先詳細を表示するには、「X」*
//                をその得意先のところに入れ、Enter を押します。 *
//                プログラムを終了するために PF3 を押します。     *
//*****************************************************************

Fcusmstl3  if   e           k disk
Fsnammenu  cf   e             workstn sfile(subfile:recnum)
F                                     indds(indicators)

 // フィールド定義:
D recnum          s              5p 0

D indicators      ds
D    exitKey                      n   overlay(indicators:3)
D    restartKey                   n   overlay(indicators:4)
D    sflClear                     n   overlay(indicators:55)
D    rollupKey                    n   overlay(indicators:95)

 // キー・リスト定義
C     cstkey        klist
C                   kfld                    srcnam
C     zipkey        klist
C                   kfld                    name
```

図 207. モジュール SCHNAM のソース (1/4)

```
//*****************************************************************
//   メインライン                                                  *
//*****************************************************************

/free

 write foot1;
 write head;
 exfmt prompt;

 // 終了キーが押されるまでループする
 dow not exitKey;
    setll cstkey cusrec;
    exsr ProcessSubfile;
    exsr DisplayCustomerDetail;

    // 終了キーがサブファイル表示で押された場合は、ループを出る
    if exitKey;
       leave;
    endif;

    // 再始動キーがサブファイル表示で押された場合は、ループを繰り返す
    if restartKey;
       iter;
    endif;

    write foot1;
    write head;
    exfmt prompt;

 enddo;

 *inlr = *on;



//*****************************************************************
//  サブルーチン - ProcessSubfile                                  *
//  目的     - サブファイルを処理し、表示する                      *
//*****************************************************************
begsr ProcessSubfile;

    // ロールアップ・キーが押されるまでループする
    dou not rollupKey;
       // サブファイルに追加する情報は他にあるか ?
       if not %eof(cusmstl3);
          // サブファイルを消去し、得意先データで充てんする
          exsr ClearSubfile;
          exsr FillSubfile;
       endif;

       // サブファイルを書き出し、応答を待つ
       write foot2;
       exfmt subctl;
    enddo;

 endsr;  // サブルーチン ProcessSubfile の終わり
```

図 207. モジュール SCHNAM のソース (2/4)

```
//*********************************************************************
//   サブルーチン - FillSubfile                                       *
//   目的       - サブファイルを充てんする                            *
//*********************************************************************
begsr FillSubfile;

   // 指定した郵便番号で得意先レコード全体をループする
   recnum = 0;
   dou %eof(snammenu);
      // 指定した郵便番号で次のレコードを読み取る
      read cusrec;
      if %eof(cusmstl3);
         // レコードがなくなったら、以下を行う
         leavesr;
      endif;

      // このレコードの情報をサブファイルに追加する
      recnum = recnum + 1;
      sel = *blank;
      write subfile;
   enddo;

endsr;  // サブルーチン  FillSubfile の終わり



//*********************************************************************
//    サブルーチン - ClearSubfile                                     *
//    目的       - サブファイル・レコードの消去                       *
//*********************************************************************
begsr ClearSubfile;

   sflClear = *on;
   write subctl;
   sflClear = *off;

endsr;  // サブルーチン ClearSubfile の終わり
```

図 207. モジュール SCHNAM のソース (3/4)

```
      //*****************************************************************
      //  サブルーチン - DisplayCustomerDetail                          *
      //  目的       - 指定した得意先レコードの表示                     *
      //*****************************************************************
      begsr DisplayCustomerDetail;

         // サブファイルの変更されたレコード全体をループする
         readc subfile;
         dow not %eof(snammenu);
            // 要求した得意先レコードの表示を再始動する
            restartKey = *on;

            // 得意先レコードを検索し、表示する
            chain zipkey cusrec;
            exfmt cusdsp;

            // 終了キーが押される場合は、ループを終了する
            if exitKey;
               leave;
            endif;

            readc subfile;
         enddo;


       endsr;  // サブルーチン ChangeSubfile の終了

      /end-free
```

図 207. モジュール SCHNAM のソース (4/4)

ファイル仕様書は、検索するディスク・ファイルおよび使用される表示装置ファイル (SNAMMENU) を指定します。WORKSTN ファイルの SFILE キーワードは、サブファイルとして使用されるレコード様式 (SUBFILE) を指定します。相対レコード番号フィールド (RECNUM) は、サブファイル内のどのレコードをアクセスするかを指定します。

プログラムは、PROMPT レコード様式を表示し、ワークステーション・ユーザーの応答を待機します。F3 は、プログラムの終了を制御する標識 03 をオンに設定します。 SETLL 命令によって CUSMSTL3 ファイルを位置付けるためのキーとして、名前 (NAME) が使用されます。SETLL 命令では、ファイル名 CUSMSTL3 ではなく、レコード様式名 CUSREC が使用されることに注意してください。

SFLPRC サブルーチンは、サブファイルの処理 (消去、充てん、および表示) を扱います。サブファイルは、サブルーチン SFLCLR の中での追加の要求のために準備されます。標識 55 がオンの場合には、画面上で処置は行われませんが、サブファイル・レコードの主記憶域は消去されます。SFLFIL ルーチンはサブファイルをレコードで充てんします。CUSMSTL3 ファイルからレコードが読み取られ、レコード・カウント (RECNUM) が増え、そのレコードがサブファイルに書き出されます。このサブルーチンは、サブファイルがいっぱいになるか (WRITE 命令の標識 21)、または CUSMSTL3 ファイルでファイルの終わりが起こる (READ 命令の標識 71) まで、反復されます。サブファイルがいっぱいになるか、ファイルの終わりが検出されると、サブファイルが EXFMT 命令によってサブファイル制御レコード様式で、画面に表示されます。ユーザーは、画面を検討して次のことを決定します。

- F3 キーを押してプログラムを終了する。

- F4 キーを押してそのサブファイルを再開する。PROMPT レコード様式は表示されずに、サブファイルが同じ名前から再び表示されます。
- 次ページ・キーを押して別のページを充てんする。CUSMSTL3 ファイルでファイルの終わりが検出されたら、現在のページが再び表示され、そうでない場合には、サブファイルが消去されて、次のページが表示されます。
- X を入力してから実行キーを押して得意先の明細を表示する。その後、ユーザーは、実行キーを押して PROMPT 画面に戻るか、F4 キーを押して再びサブファイルを表示するか、あるいは F3 キーを押してプログラムを終了させることができます。

図 208 では、ユーザーは、初期化プロンプトに対して得意先名を入力して応答します。

```
22:35:26            CUSTOMER SEARCH & INQUIRY BY NAME            9/30/94

Enter Search Name  JUDAH GOULD









   ENTER - Continue       F3 - End Job
```

図 208. 'CUSTOMER SEARCH & INQUIRY BY NAME' (名前による得意先検索および照会) プロンプト画面

444 ページの図 209 に示すように、ユーザーは X を入力して、さらに情報を要求します。

```
22:35:43            CUSTOMER SEARCH & INQUIRY BY NAME            9/30/94

Search Name  JUDAH GOULD

  Select
   "X"       Customer Name            Number     Zip Code
    X        JUDAH GOULD              00012      70068
             JUDAH GOULD              00209      31088

   ENTER - Continue       F3 - End Job      F4 - Restart Name
```

図 209. 'CUSTOMER SEARCH & INQUIRY BY NAME' (名前による得意先検索および照会) 情報画面

選択した得意先の詳細は、 図 210 に示されています。この時点で、ユーザーは、適切な機能キーを選択して照会を続行または終了します。

```
23:39:48            CUSTOMER SEARCH & INQUIRY BY NAME            9/30/94


                    Customer  00012

                    Name      JUDAH GOULD

                    Address   2074 BATHURST AVENUE

                    City      YORKTOWN

                    State     NY    Zip Code 70068

                    A/R Balance               .00



   ENTER - Continue       F3 - End Job      F4 - Restart Name
```

図 210. 'CUSTOMER SEARCH & INQUIRY BY NAME' (名前による得意先検索および照会) 詳細情報画面

第 5 部 付録

付録 A. OPM RPG/400 と AS/400 用 ILE RPG との動作上の相違点

次のリストは、OPM RPG/400 コンパイラーと ILE RPG における動作上の相違点を記したものです。

コンパイル

1. OPM RPG で CVTOPT(*NONE) を指定すると、RPG がサポートしないタイプまたは属性を持つ、外部記述されたすべてのフィールドは無視されます。ILE RPG で CVTOPT(*NONE) を指定すると、外部記述のすべてのフィールドは、外部記述で指定されたのと同じタイプでプログラムに取り込まれます。
2. RPG IV では、制御仕様書の DATEDIT と DECEDIT との間に従属関係はありません。
3. ILE RPG の作成コマンド (CRTBNDRPG および CRTRPGMOD) に関しては次のとおりです。
   - CRTRPGPGM コマンドの IGNDECERR パラメーターは、ILE RPG の作成コマンドでは FIXNBR パラメーターに置き換えられました。IGNDECDTA は、10 進データ・エラーを無視して、次の機械命令から続行します。場合によっては、このために、フィールドが、誤った値や予期せぬ値で更新されることがあります。FIXNBR は、データが使用される前に予測可能な方法でそのデータを訂正します。
   - 数値オーバーフローを許容するかどうかを制御する新しいパラメーター TRUNCNBR があります。
   - RPG IV には、報告書簡易作成機能あるいはコマンドはありません。
   - コンパイラーに MI リストを要求することはできません。
4. コンパイラー・リストでは、デフォルトの OPTION(*NOSRCSTMT) が指定されていると、行番号は 1 で始まり、ソースまたは生成仕様書の 1 行ごとに 1 ずつ増えます。OPTION(*SRCSTMT) が指定されていると、行番号の代わりに順序番号が示されます。ソース ID は数値です。すなわち、/COPY メンバーまたは拡張 DDS に AA000100 を超える行番号はありません。
5. RPG IV では、/TITLE などのコンパイラー指示ステートメントはすべてコンパイル時データより*前に* なければなりません。RPG IV は、/TITLE 指示ステートメントを見つけると、これをデータと見なします。(RPG III は、/TITLE 指定がソースのどこにあっても、これをコンパイラー指示ステートメントと見なします。)

   変換援助プログラムは、コンパイル時データの中で /TITLE 指定を見つけると、これを除去します。
6. ILE RPG は、データ構造におけるフィールドの重なりを検出することにより厳格です。オーバーラップ・オペランドがかかわるいくつかの計算命令では、ILE RPG がメッセージを出し、OPM コンパイラーは出しません。

7. ILE RPG において NOT という語は変数名として使用することができません。NOT は式の中の演算子として使用される特殊語です。
8. コンパイル時に、ソースはメイン・ソース・ファイルの CCSID を使用して読み取られますが、OPM RPG の場合には、ソースはジョブの CCSID を使用して読み取られます。

実行

1. FREE 命令は、RPG IV ではサポートされていません。
2. OPM のもとでは現れないある種の MCH メッセージがジョブ・ログに現れることがあります。例えば、MCH1202 などがそうです。これらのメッセージが現れても、プログラムの動作に変更があるわけではありません。
3. バインド不能 API QMHSNDPM を使用してプログラムからメッセージを送り出す場合には、スタック中のプログラム入力プロシージャーの存在を可能にするために、スタック・オフセット・パラメーターに 1 を加算することが必要になる場合があります。これは、ILE プロシージャーがユーザー入り口プロシージャーである場合、および呼び出しメッセージ待ち行列に特殊値 '*' とスタック・オフセットに 0 より大きい値を使用した場合だけです。
4. ILE RPG は、例外がなく終了するプログラムまたはプロシージャーに対する呼び出しについて、0 でも 1 でもない戻りコードは解釈しません。
5. ILE RPG プログラムについてのキャンセル・ハンドラーは制御を受け取ると、システム戻りコードを 2 に設定します。OPM RPG プログラムについてのキャンセル・ハンドラーは、システム戻りコードの設定値を変更しません。
6. 再帰が検出された時には、OPM RPG/400 は照会メッセージ RPG8888 を表示します。ILE RPG シグナルはメッセージ RNX8888 を出しません。この条件に対する照会メッセージは表示されません。これはメイン・プロシージャーにだけ適用されることに注意してください。サブプロシージャーに対して再帰は使用可能です。
7. ゾーン 10 進数またはパック 10 進数サブフィールドの初期設定中に 10 進数データ・エラーが起こった場合には、リセット値 (これらの値を使用して RESET 命令のあるサブフィールドを復元します) は有効でないことがあります。例えば、サブフィールドが初期設定されていないか、あるいは別のタイプの別の初期設定されたサブフィールドに重ね書きしている可能性もあります。OPM RPG/400 で、そのサブフィールドに RESET 命令が試みられた場合には、 10 進数データ・エラーが起こります。ただし、ILE RPG の同じサブフィールドに対する RESET は正常に完了しますが、この RESET 後、このサブフィールドは同じ無効な値になっています。結果として、この値を使おうとする試みは 10 進数データ・エラーを受け取ることになります。
8. ILE RPG では、プログラム状況データ構造 (PSDS) の位置 254 ～ 263 には、起点となるジョブのユーザー名が入れられます。OPM RPG では、これらの位置は、現行ユーザー・プロファイルを反映します。ILE RPG 内の現行ユーザー・プロファイルは、位置 358 ～ 367 に入っています。

デバッグおよび例外処理

1. DEBUG 命令は、RPG IV ではサポートされていません。

2. ILE ソース・デバッガーを使用している時には、停止点の設定に、 RPG タグ、サブルーチン名、または *GETIN や *DETC などのサイクル内の点を使用することはできません。
3. 機能チェックは、OPM RPG と ILE RPG の両方によって通常ジョブ・ログに残ります。しかし ILE RPG において、エラー標識、'E' 拡張、または *PSSR エラー・ルーチンがコーディングしてあると、機能チェックは現れません。

   標識、'E' 拡張、*PSSR などがあると機能チェックができませんので、機能チェックを削除するようなコードがあれば取り除いてください。
4. 80 バイトより後に PSDS を埋め込む情報を入手するのには時間を要するので、LR がオンによる呼び出しパフォーマンスは、 PSDS を指定しないことによって、あるいは PSDS を 80 バイトより長くしないことによって大きく改善されます。 PSDS がコーディングされていないか、またはプログラムが開始した日付および時刻を入れるのに短すぎる場合には、これらの 2 つの値は定様式ダンプでは使用可能ではありません。他のすべての PSDS の値は、PSDS の長さに関係なく使用可能です。
5. ILE RPG 照会メッセージの接頭部は RNQ であるため、デフォルト応答リストを使用する場合、既存の RPG 項目とほぼ同じ RNQ 項目を追加しなければなりません。
6. OPM で、CL プログラムが RPG プログラムの後に MONMSG を呼び出す場合、この RPG プログラムが通知メッセージまたは状況メッセージを受け取っても、CL MONMSG はこの通知メッセージも状況メッセージも処理しません。ILE CL から ILE RPG を呼び出す場合に、両方が同一活動化グループ内にあるときは、ILE CL MONMSG はこの通知または状況メッセージを処理し、 RPG プロシージャーは RPG エラー・メッセージを出さずにただちに停止します。詳細については、 307 ページの『ILE CL が通知および状況メッセージを監視する際の問題』を参照してください。
7. ILE ソース・デバッガーを使って変数を表示する時、次のような場合は結果に信頼が置けません。
   - ILE RPG プログラムが外部記述ファイルを使っている場合、および
   - 変数がデータベース・ファイルに定義されていても ILE RPG プログラムで参照されない場合。
8. ユーザーの RPG III プログラムにパラメーター不一致の問題がある場合 (例えば、長さ 20 のパラメーターを予想しているプログラムに対して長さ 10 のパラメーターを渡し、呼び出し先プログラムが 20 バイトすべてを変更するような場合)、ユーザーのプログラムは記憶域破壊の問題に遭遇することになります。破壊される記憶域がそのプログラムの実行にとって重要でない場合には、この問題は必ずしもエラーになるとは限りません。

   このプログラムが RPG IV に変換される場合、記憶域のレイアウトは異なる可能性があり、破壊された記憶域がプログラムによって使用されます。この結果、予期しない例外、例えば SETLL などのファイル操作の際の例外 MCH3601 などが発生する場合があります。ユーザーのアプリケーションとは関係がないと思われる疑わしいエラーに遭遇した場合は、すべての呼び出し命令のパラメーターをチェックして、パラメーターがすべて正しい長さであることを確認する必要があります。

9. OPM では、プログラマーがプログラムに対して *USE 権限を持っている場合に定様式ダンプを実行できます。 ILE では、定様式ダンプを実行するには、プログラムまたはサービス・プログラムに対して *CHANGE 権限を持っている必要があります。

入出力

1. ILE RPG では、更新用にオープンされ、SHARE(*YES) で作成または一時変更されたファイルからレコードを読み取った後、同じファイルを更新用にオープンしている別のプログラムで、このロックされたレコードを更新することができます。
2. MOVE または SETON 命令を使用して MR 標識を変更することはできません。(RPG III では MR に対して SETON を使用できないだけです。)
3. ファイル仕様書のファイル・タイプ項目は、演算仕様書になければならない入出力命令のタイプをもはや指示しません。

   例えば、RPG III では、更新ファイルとしてファイルを定義した場合には、後からプログラムで UPDAT 命令を指定しなければなりません。RPG IV では、これはもはや必要ではありません。ただし、ファイル定義については、プログラムにある入出力命令と一貫性がなければなりません。したがって、ソースに UPDATE 命令がある場合には、ファイルは更新ファイルとして定義しなければなりません。
4. ILE RPG では、ファイル仕様書に COMMIT キーワードが指定されていても、レコードのブロック化が可能です。
5. RPG IV では、更新用にオープンされたファイルが削除可能としてもオープンされます。このファイルを削除可能にするために、DELETE 命令は不要です。
6. RPG IV では、複数装置ファイルに使用される装置の数として実際の数をコーディングする必要はありません。ファイル仕様書に MAXDEV(*FILE) を指定した場合には、SAVEDS および SAVEIND 用に作成される記憶域の数は、ユーザーのファイルが扱える装置の数に基づいて決められます。(RPG IV ファイル仕様書の SAVEDS、SAVEIND、および MAXDEV キーワードは、RPG III ファイル仕様書の継続行の SAVDS、IND、および NUM オプションとそれぞれ対応します。)

   ILE RPG では、プログラムが獲得できるプログラム装置の合計数を、装置ファイルで定義された装置の最大数と異なって指定することはできません。OPM RPG/400 では NUM オプションによってこれが可能でした。
7. ILE RPG では、ACQ および REL 命令コードを単一装置ファイルに対して使用することができます。
8. ILE RPG では、ブロック化された読み取りを実行する時、各入力命令ごとに、INFDS のデータベース固有フィードバック・セクションにある相対レコード番号およびキー・フィールドが更新されます。
9. OPM RPG/400 で参照制約エラーが起こると、状況コードが ″01299″ (入出力エラー) に設定されます。 ILE RPG では、起こった参照制約エラーのタイプに応じて、状況コードが ″01022″、″01222″、または ″01299″ に設定されます。
   - 参照制約エラーのためにデータ管理がレコードを割り振ることができない場合には、CPF502E 通知メッセージが出されます。ILE RPG は状況コードを ″01222″ に、OPM RPG/400 は状況コードを ″01299″ に設定します。

エラー標識、'E' 拡張、または INFSR エラー処理サブルーチンがなければ、ILE RPG は RNQ1222 照会メッセージを出し、OPM RPG/400 は RPG1299 照会メッセージを出します。この 2 つのメッセージの主な違いは、RNQ1222 の場合には操作を再試行できるということです。

- データ管理に CPF503A、CPF502D、または CPF502F 通知メッセージを出させるような参照制約エラーをデータ管理が検出した場合には、ILE RPG は状況コードを ″01022″ に設定し、OPM RPG/400 用 は状況コードを ″01299″ に設定します。

  エラー標識、'E' 拡張、または INFSR エラー処理サブルーチンがなければ、ILE RPG は RNQ1022 照会メッセージを出し、OPM RPG は RPG1299 照会メッセージを出します。
- データ管理にエスケープ・メッセージを出させる参照制約エラーをデータ管理が検出した場合には、そのすべては、OPM と ILE RPG の両方が状況コードを ″01299″ に設定する原因となります。

10. ILE RPG では、 INFDS のデータベース固有フィードバック・セクションは、入出力操作の結果にかかわらず更新されます。OPM RPG/400 では、このフィードバック・セクションは、レコードが見付からない状態が検出されると更新されません。
11. ILE RPG は、OPM RPG/400 と比べてデータ管理エラー処理に頼る点がより多くあります。このことは、場合によっては、ユーザーが OPM RPG/400 プログラムではないが ILE RPG プログラムのジョブ・ログではある種のエラー・メッセージが見つかるということを意味します。エラー処理でユーザーの目に止まると思われる相違点のいくつかを次に示します。
    - 前の入力命令でロックされていないデータベース・ファイルのレコードに対して UPDATE を実行すると、ILE RPG も OPM RPG/400 も状況コードを ″01211″ に設定します。ILE RPG は、データ管理が CPF501B 通知メッセージを出してそれをジョブ・ログに入れた時にこの状況を検出します。
    - WORKSTN ファイルを処理し、獲得または定義されていない装置に対して入出力を行おうとすると、ILE と OPM RPG の両方とも状況を ″01281″ に設定します。ILE RPG は、データ管理が CPF5068 エスケープ・メッセージを出して、それをジョブ・ログに入れた時に、この状況を検出します。
12. データベース・ファイルに対して READE、REDPE (ILE では READPE)、SETLL を行う時、またはレコード・アドレス・ファイルにより限界内順次処理を行う時には、OPM RPG/400 は *HEX 照合順序を使用してキー比較を行います。ファイルに与えられたキーと 2 つ以上の検索引き数を突き合わせる DDS 機能を使うと、この結果は予想と異なることがあります。

    例えば、ABSVAL が数字キーに使われると、ファイルの中で 1 の値を持つキーに対する検索引き数として -1 と 1 が使われてしまいます。16 進の照合順序を使うと、-1 の検索引き数は 1 の値を持つ実際のキーには突き合わされません。

    ILE RPG は、V3R1 より前の DDM ファイルの場合にのみ、*HEX 照合順序を使用してキーの比較を行います。詳細については、 377 ページの『V3R1 より前の DDM ファイルの使用』を参照してください。

13. ILE RPG では、実行前の配列およびテーブルに対して指定する TO ファイルおよび FROM ファイルが異なっていても構いません。 OPM RPG では、両方のファイル名が同じでなければならず、もし異なっている場合には、診断メッセージ QRG3038 が出されます。
14. RAF 制御ファイルの変換が指定された時には、ILE RPG を使用した結果は、変換テーブルによっては OPM RPG/400 と異なることがあります。これは異なる操作順序によるものです。OPM RPG/400 では、順序は、レコードの検索、変換、および比較であり、ILE RPG では、順序は、レコードの変換、比較、および検索です。

文字フィールドの DBCS データ

1. OPM RPG/400 では、制御仕様書の 57 桁目 (透過性検査) により、RPG/400 コンパイラーが DBCS 文字を調べるために、文字リテラルおよび固定情報を走査するかどうかを指定することができます。コンパイラーが透過リテラルを走査するように指定し、シフトアウトに続くアポストロフィで始まる文字リテラルが透過性検査を通らなかった場合には、そのリテラルが透過的でないリテラルとして再解析されます。

   ILE RPG では、コンパイラーが文字リテラルに対して透過性検査を実行するかどうかを指定するオプションは制御仕様書にはありません。文字リテラルにシフトアウト制御文字が入っている場合には、文字リテラル内のシフトアウト文字の位置に関係なく、シフトアウト文字は DBCS データの始めを示します。コンパイラーは次のことを検査します。

   - シフトアウトに対応するシフトインがある (すなわち、シフトアウト制御文字とシフトイン制御文字が対応している)。
   - シフトインとシフトアウトの間のバイト数は偶数 (最小は 2)。
   - DBCS データに組み込みシフトアウトがない。

   上記の条件が満たされなければ、コンパイラーは診断メッセージを出し、リテラルは再解析されません。その結果、OPM RPG コンパイラーによって行われる透過性検査に通らない文字リテラルが OPM RPG プログラムにあれば、ILE RPG ではこのようなプログラムはコンパイル・エラーを受け取ります。
2. OPM RPG/400 では、文字リテラルの中のシフトアウト制御文字とシフトイン制御文字で囲まれた中に連続した 2 個のアポストロフィがある場合に、その文字リテラルが透過リテラルでなければ、連続した 2 個のアポストロフィは 1 個のアポストロフィと見なされます。以下のような場合、文字リテラルは透過リテラルはありません。

   - 文字リテラルが、シフトアウトに続くアポストロフィで始まっていない。
   - 文字リテラルが、コンパイラーの行う透過性検査に通らない。
   - ユーザーが、コンパイラーによる透過性検査の実行を指定していない。

   ILE RPG では、文字リテラルの中のシフトアウト制御文字とシフトイン制御文字で囲まれた中に連続した 2 個のアポストロフィがある場合に、アポストロフィは 1 個のアポストロフィとは見なされません。文字リテラルの中の 1 対のアポストロフィは、シフトアウト制御文字とシフトイン制御文字で囲まれた中にない場合にだけ、 1 個のアポストロフィと見なされます。

3. ILE RPG では、シフトアウト文字に対してリテラルの検査を避けたい (すなわち、シフトアウト文字をシフトアウト文字として解釈したくない) 場合には、リテラル全体を 16 進リテラルとして指定する必要があります。例えば、リテラル 'AoB' がある場合 (ここで、'o' はシフトアウト制御文字を表す) には、このリテラルを X'C10EC2' としてコーディングしなければなりません。

付録 B. RPG III から RPG IV への変換援助プログラムの使用

RPG IV ソースの仕様レイアウトは、システム/38™ 環境 RPG III および OPM RPG/400 のレイアウトとはかなり異なります。例えば、仕様書の項目の桁が変更されており、使用可能な仕様書のタイプも変更されています。RPG IV の仕様書レイアウトは、それ以前のレイアウトと互換性がありません。RPG IV の機能を利用するためには、アプリケーション・プログラム中の RPG III および RPG OS/400 用ソース・メンバーを RPG IV ソース形式に変換しなければなりません。

注: 変換できるソース・メンバーの有効なタイプは、RPG、RPT、RPG38、RPT38、SQLRPG、およびブランクです。この変換援助プログラムは、RPG36、RPT36、およびその他の非 RPG ソース・メンバー・タイプの変換をサポートしていません。

すぐに始めたいという場合には、 458 ページの『ソースの変換』に進み、概略ステップに従ってください。

変換の概要

CL コマンドの RPG ソースの変換 (CVTRPGSRC) を使用して変換援助プログラムを呼び出すことにより、ソース・プログラムを RPG IV ソース形式に変換します。変換援助プログラムは、次のものを変換します。

- 単一のメンバー
- ソース物理ファイルのすべてのメンバー
- 同一ファイル中の共通するメンバー名接頭部をもつすべてのメンバー

変換の問題が起きるのを最小限に抑えるために、 /COPY メンバーを任意に変換後のソース・コードに含めることができます。またコーディングを読みやすくするために、仕様書テンプレートを変換済みの任意のソース・コードに含めることもできます。

変換援助プログラムは、各ソース・メンバーを行単位で変換します。ユーザーがコマンドにログ・ファイルを指定している場合には、各メンバーの変換後に、変換の状況によってログ・ファイルを更新します。また、変換エラー、/COPY ステートメント、CALL 命令、および変換状況などの情報を含む変換報告書を作成することもできます。

変換援助プログラムは、ユーザーのソース・コードにコンパイル・エラーがないものと見なします。この場合には、ユーザーのソース・コードの大部分は正常に変換されます。場合によっては、手操作で変換しなければならない少量のコーディングがあることもあります。このような事例の一部は、変換援助プログラムによって識別されます。その他は、ユーザーが変換後のソースをコンパイルするまで検出されません。変換援助プログラムが識別できるものを調べるためには、未変換メンバーを入力として使用して変換援助プログラムを実行し、出力メンバーではなく、変換報告書を指定することができます。変換できないコーディングのタイプについては、 475 ページの『変換上の問題の分析解決』を参照してください。

ファイルに関する考慮事項

変換援助プログラムはファイル・メンバーに対して機能します。この項では、変換援助プログラムの使用時に、考慮しなければならないファイルのいろいろな局面での情報を提供します。

ソース・メンバー・タイプ

表 45 は各種のソース・メンバー・タイプをリストしたものであり、各メンバー・タイプが変換可能であるかどうかを示し、かつ出力ソース・メンバー・タイプを示します。

表 45. ソース・メンバー・タイプとその変換状況

ソース・メンバー・タイプ	変換?	変換後のメンバー・タイプ
RPG	はい	RPGLE
RPG38	はい	RPGLE
RPT	はい	RPGLE
RPT38	はい	RPGLE
'ブランク'	はい	RPGLE
RPG36	いいえ	N/A
RPT36	いいえ	N/A
SQLRPG	はい	SQLRPGLE
その他のタイプ	いいえ	N/A

ソース・メンバー・タイプが 'ブランク' である場合には、変換援助プログラムはメンバー・タイプが RPG であると見なします。報告書簡易作成ソース・メンバーのソース・メンバー・タイプがブランクである場合には、変換前にそのメンバーに適切なソース・メンバー・タイプ (RPT または RPT38) を割り当てる必要があります。そうした場合には、変換援助プログラムは、適切に変換できるように報告書簡易作成機能ソース・メンバーを自動的に展開します。ILE RPG は報告書簡易作成機能ソース・メンバーをサポートしていないので、この展開が必要です。

報告書簡易作成ソース・メンバーの変換の詳細については、 467 ページの『報告書簡易作成機能ソース・メンバーの変換』を参照してください。

ファイルのレコード長

変換ソース物理ファイルの推奨されるレコード長は 112 桁です。このレコード長は 457 ページの図 211 に示したような、新しい RPG IV 構造を考慮に入れています。推奨されるレコード長の 112 桁はまた、コンパイラー・リストの 1 行に収まる情報の最大量と対応しています。

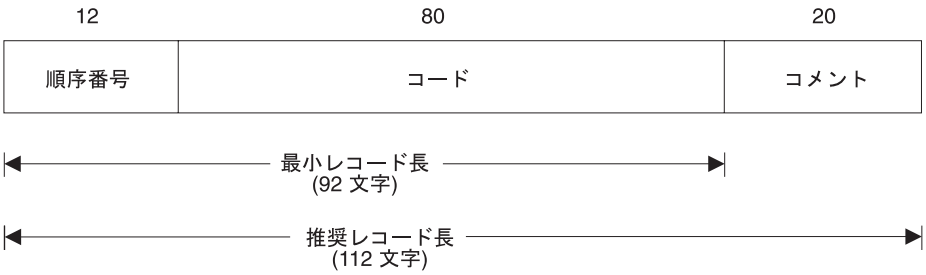


図 211. RPG IV レコード長の説明

変換後のソース・ファイルのレコード長が 92 桁より小さい場合には、エラー・メッセージが出され、変換が停止します。これは、レコード長がソース・コードに許容される 80 桁を入れるのに十分でないので、コードの一部が失われる可能性があるためです。

ファイル名およびメンバー名

変換前のメンバーと変換後の出力のメンバーは、これらが異なるファイルまたはライブラリーに入っている場合にのみ、同じ名前をもつことができます。

変換後のソース・メンバーの名前は、1 つのメンバーを変換するか複数のメンバーを変換するかによって、異なります。1 つのメンバーを変換する場合には、デフォルトで、変換前のメンバーと同じ名前が変換後のソース・メンバーに与えられます。もちろん、出力メンバーに別の名前を指定することができます。ファイル中のすべてのソース・メンバーを変換するか、あるいは総称名を使用してそれらのグループを変換する場合には、メンバーには変換前のソース・メンバーと同じ名前が自動的に与えられます。

変換後の出力のファイル、ライブラリー、またはメンバー名の指定は任意指定であることに注意してください。これらの名前を 1 つも指定しない場合には、変換後の出力はファイル QRPGLESRC に入れられ、変換前のメンバー名と同じメンバー名をもつことになります。(ライブラリー・リストはファイル QRPGLESRC に対して検索されます。)

ログ・ファイル

変換援助プログラムは、ログ・ファイルを使用して、各ソース・メンバー変換の状況に関する監査証跡を提供します。ログ・ファイルをブラウズすることによって、以前の変換の状況を判別することができます。ユーザー作成のプログラムを使ってログ・ファイルにアクセスし、例えばプログラムのコンパイルやバインドのような、その先の処理をすることができます。

ログ・ファイルを更新する指定を行うと、そのレコード様式はライブラリー QRPGLE 内にある IBM 提供の ″モデル″ データベース・ファイル QARNCVTLG の様式と一致していなければなりません。 475 ページの図 218 はこのファイルの DDS を示します。次の CRTDUPOBJ コマンドを使用して、このモデルのコピーをユーザー自身のライブラリー (ここでは MYLIB として参照されている) に作成してください。ユーザーのログ・ファイルに QRNCVTLG という名前を指定することもできます。これは変換援助プログラムのデフォルトのログ・ファイル名です。

```
CRTDUPOBJ OBJ(QARNCVTLG) FROMLIB(QRPGLE) OBJTYPE(*FILE)
          TOLIB(MYLIB) NEWOBJ(QRNCVTLG)
```

変換援助プログラムがアクセスするログ・ファイルに対して、オブジェクト管理、操作、および追加の各権限が必要です。

ログ・ファイルの使用については、 474 ページの『ログ・ファイルの使用』を参照してください。

変換援助プログラム・ツールの要件

変換援助プログラムを使用するためには、次の権限が必要です。

- CVTRPGSRC コマンドに対する *USE 権限
- ソース・ファイルとソース・メンバーが入っているライブラリーに対する *USE 権限
- ソース・ファイルと変換後のソース・メンバーが入る新しいライブラリーに対する *CHANGE 権限
- 変換援助プログラムが使用するログ・ファイルに対するオブジェクト管理、操作、および追加の各権限

オブジェクト権限の要件の他に、追加の記憶域要件がある場合もあります。変換後の各ソースは、変換前のプログラムのサイズより、平均して 25% より大きくなります。変換援助プログラムを使用するためには、変換後のソース・ファイルを保管できる十分な記憶域が必要です。

変換援助プログラムが行わないこと

- 変換援助プログラムは、RPG IV 形式を RPG III または RPG/400 形式に戻す変換をサポートしていません。
- RPG IV コンパイラーは、*コンパイル時における* RPG III または RPG/400 ソース・メンバーから RPG IV ソース形式への自動変換をサポートしていません。
- 変換援助プログラムは、RPG II ソース・プログラムから RPG IV ソース形式への変換をサポートしていません。ただし、最初に **RPG II - RPG III 変換援助プログラム**を使用し、その後で RPG III - RPG IV 変換援助プログラムを使用することができます。
- 変換援助プログラムは、必要なところ (例えば、条件付け標識の数など) を除き、ソース・コードをリエンジニアリングしません。
- 変換援助プログラムはファイルを作成しません。変換援助プログラムを実行する前に、ログ・ファイルと出力ファイルが存在していなければなりません。

ソースの変換

この項では、ソース・プログラムを RPG IV 形式へ変換する方法を説明します。変換援助プログラムを開始するコマンド CVTRPGSRC とその使用方法について説明します。

ユーザーのソース・コードを RPG IV 形式に変換するためには、次の概略ステップに従ってください。

1. 制御仕様書としてデータ域を使用する場合には、RPG IV 形式の新しいデータ域を作成しなければなりません。詳しくは、「*WebSphere Development Studio: ILE RPG 解説書*」の制御仕様書に関する章を参照してください。

2. 必要な場合には、ログ・ファイルを作成する。

   LOGFILE(*NONE) を指定しない限り、変換援助プログラムがアクセスするログ・ファイルがなければなりません。このログ・ファイルがない場合には、CRTDUPOBJ コマンドを使用してこれを作成することができます。詳細については、457 ページの『ログ・ファイル』および 474 ページの『ログ・ファイルの使用』を参照してください。

3. 変換後のソース・メンバーを入れるファイルを作成する。

   変換援助プログラムはどのようなファイルも作成しません。CVTRPGSRC コマンドを実行する前に、変換後のソースを入れる出力ファイルを作成しなければなりません。出力ファイルの推奨される名前およびレコード長はそれぞれ、QRPGLESRC および 112 桁です。追加のファイル情報については、456 ページの『ファイルに関する考慮事項』を参照してください。

4. CVTRPGSRC コマンドを使用してユーザーのソースを変換する。

   変換するファイルおよびメンバーの名前を入力する必要があります。デフォルトの値を受け入れた場合には、ファイル QRPGLESRC に変換後のメンバーが入れられます。メンバーの名前は、変換前のソース・メンバーの名前と対応しています。/COPY メンバーは、変換後のソース・メンバーのタイプが RPT または RPT38 でない場合はソース・メンバーには展開されません。変換報告書が生成されます。

   詳細については、『CVTRPGSRC コマンド』を参照してください。

5. エラーについて、ログ・ファイルまたはエラー報告書を検査する。詳細については、471 ページの『変換の分析』を参照してください。

6. エラーがある場合には、それらを訂正してステップ 4 に進む。

7. エラーがない場合には、ユーザーのプログラムを作成する。ILE RPG を作成する方法についての詳細は、65 ページの『第 6 章 CRTBNDRPG コマンドによるプログラムの作成』を参照してください。

8. 変換後のソース・メンバーにまだコンパイル上の問題がある場合には、ユーザーの最初のソース・メンバーに /COPY コンパイラー指示ステートメントが入っていることがその最も有力な原因であると考えられます。この状況を訂正するためには、次の 2 つの方法があります。

   a. コピー・メンバーを変換後のソース・メンバーに展開するために、EXPCPY(*YES) を指定してソース・メンバーを変換し直す。

   b. コンパイラー・リストを手引きとして、残っているエラーを手操作で訂正する。

   詳しくは、475 ページの『変換上の問題の分析解決』を参照してください。

9. 変換後のソース・メンバーのコンパイルが正常に完了した場合には、実行用に戻す前にプログラムを再テストする。

CVTRPGSRC コマンド

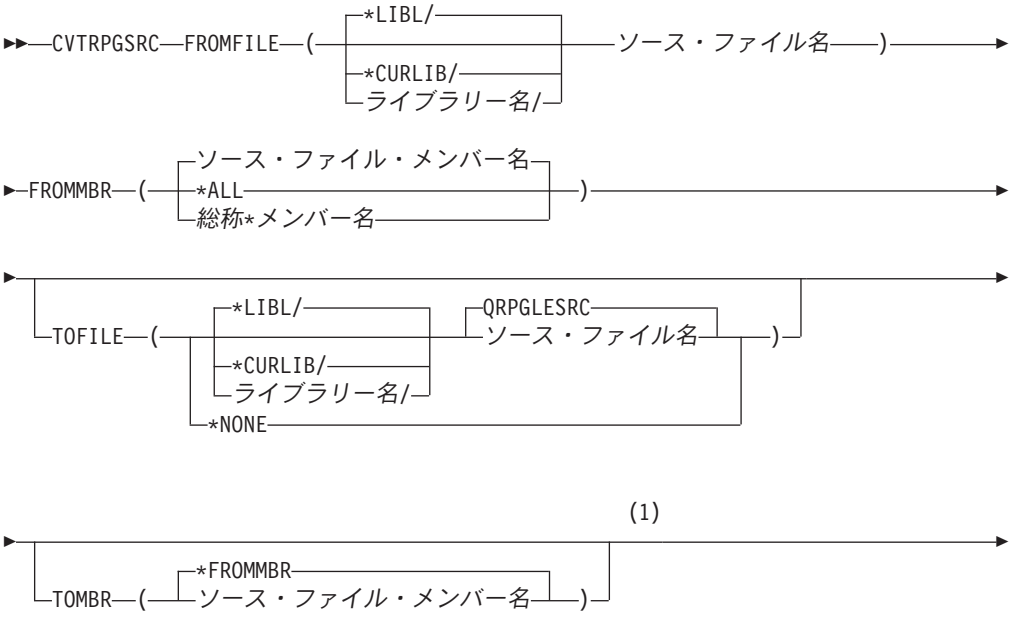
RPG III または RPG/400 ソースを新しい RPG IV 形式に変換するには、CVTRPGSRC コマンドを使って変換援助プログラムを開始します。460 ページの表 46 は、このコマンドのパラメーターをその機能に基づいて示したものです。

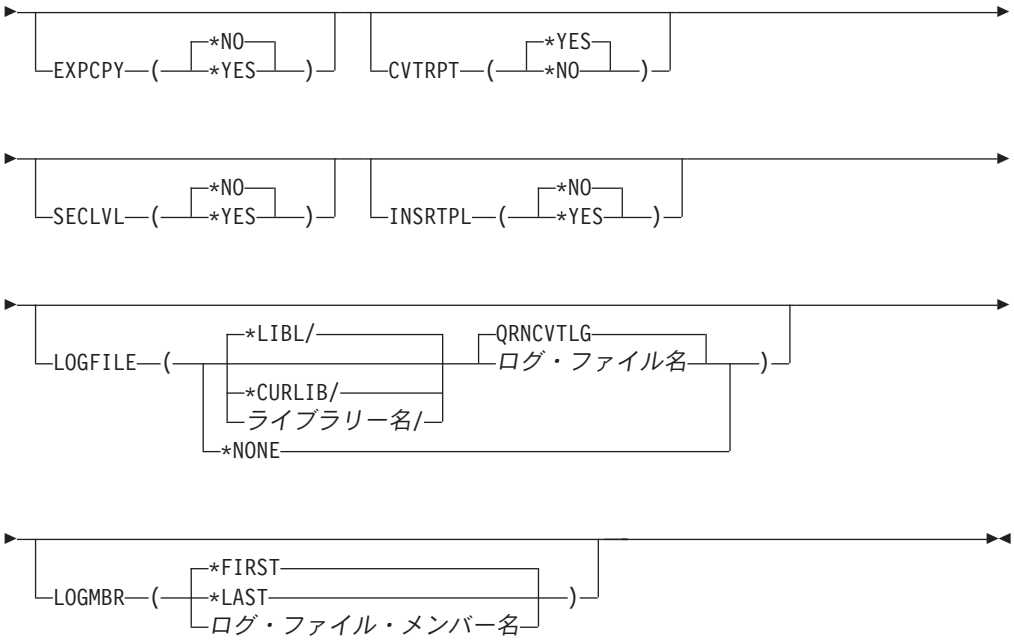
表 46. 機能別にまとめた CVTRPGSRC のパラメーターとそのデフォルト値

プログラム識別	
FROMFILE	変換する RPG ソースのライブラリーとファイル名を指定します。
FROMMBR	変換するソース・メンバーを指定します。
TOFILE(*LIBL/QRPGLESRC)	変換後の出力のライブラリーとファイル名を指定します。
TOMBR(*FROMMBR)	変換後のソースのファイル・メンバー名を指定します。
変換処理	
TOMBR	*NONE を指定すると、ファイル・メンバーは保管されません。
EXPCPY(*NO)	/COPY ステートメントが変換後の出力に組み込まれるかどうかを決めます。
INSRTPL(*NO)	仕様書テンプレートを変換後の出力に組み込むかどうかを示します。
変換フィードバック	
CVTRPT(*YES)	変換報告書を作成するかどうかを決めます。
SECLVL(*NO)	第 2 レベル・メッセージ・テキストを組み込むかどうかを決めます。
LOGFILE(*LIBL/QRNCVTLG)	監査報告書用のログ・ファイルを指定します。
LOGMBR(*FIRST)	監査報告書に使用するログ・ファイルのメンバーを指定します。

CVTRPGSRC コマンドの構文を次に示します。

ジョブ: B,I　プログラム: B,I　REXX: B,I　EXEC





注:

1 この点より前のパラメーターは、すべて定位置形式によって指定することができます。

パラメーターとその使用できる値は、構文図の後に続きます。プロンプトの必要がある場合には、CVTRPGSRC を入力して F4 キーを押してください。CVTRPGSRC 画面が表示され、パラメーターがリストされ、デフォルト値が提供されます。表示上のパラメーターの説明については、そのパラメーターにカーソルを置いて F1 キーを押してください。任意のパラメーターで F1 キーを押してから F2 キーを押すことによって、すべてのパラメーターの全般ヘルプが使用可能です。

FROMFILE

変換される RPG III または RPG ソース・コードを含むソース・ファイルの名前、そのおよびソース・ファイルが保管されるライブラリーを指定します。これは必須パラメーターであり、デフォルトのファイル名はありません。

ソース・ファイル名

変換するソース・メンバー (複数の場合もある) が入っているソース・ファイルの名前を入力します。

***LIBL**

システムは、ライブラリー・リストを検索して、ソース・ファイルが保管されているライブラリーを見付けます。

***CURLIB**

ソース・ファイルを検索するために現行ライブラリー・リストが使用されます。現行ライブラリーを指定していない場合には、ライブラリー QGPL が使用されます。

ライブラリー名
ソース・ファイルが保管されているライブラリーの名前を入力します。

FROMMBR
変換するメンバー (複数の場合もある) の名前を指定します。これは必須パラメーターであり、デフォルトのメンバー名はありません。

変換するソース・メンバーの有効なソース・メンバー・タイプは、RPG、RPT、RPG38、RPT38、SQLRPG、およびブランクです。RPG ソース変換コマンドは、ソース・メンバー・タイプ RPG36、RPT36、およびその他の非 RPG ソース・メンバー・タイプ (例えば、CLP や TXT) をサポートしません。

ソース・ファイル・メンバー名
変換するソース・メンバーの名前を入力します。

**ALL*
このコマンドは、指定されたソース・ファイル中のすべてのメンバーを変換します。

*総称*メンバー名*
名前に同じ接頭部をもつメンバーの総称名に続けて '*' (アスタリスク) を入力します。このコマンドは、指定されたソース・ファイル中の総称名をもつすべてのメンバーを変換します。例えば、FROMMBR(PR*) を指定すると、'PR' で始まる名前をもつすべてのメンバーが変換されます。

(総称名の詳細については、CL プログラマーの手引きを参照してください。)

TOFILE
変換後のソース・メンバーを入れるソース・ファイルの名前および変換後のソース・ファイルを保管するライブラリーを指定します。変換後のソース・ファイルは存在していなければならず、レコード長は 112 桁 (順序番号と日付の 12 桁、コードの 80 桁、およびコメントの 20 桁) でなければなりません。

QRPGLESRC
デフォルトのソース・ファイル QRPGLESRC には、変換後のソース・メンバー (複数の場合もある) が入ります。

***NONE**
変換後のメンバーは生成されません。TOMBR パラメーター値は無視されます。CVTRPT(*YES) も指定しなければなりません。指定しないと、変換はただちに終了します。

この機能によって、変換後のソース・メンバーを作成しないで、起こる可能性のある問題を見つけることができます。

ソース・ファイル名
変換後のソース・メンバー (複数の場合もある) が入れる変換後のソース・ファイルの名前を入力します。

TOFILE ライブラリー名が FROMFILE ライブラリーと同じである場合には、 TOFILE ソース・ファイル名は FROMFILE ファイル名と異なるものでなければなりません。

***LIBL**
システムは、ライブラリー・リストを検索して、変換後のソース・ファイルを保管するライブラリーを見付けます。

***CURLIB**
変換後のソース・ファイルを検索するために現行ライブラリー・リストが使用されます。現行ライブラリーを指定していない場合には、ライブラリー QGPL が使用されます。

ライブラリー名
変換後のソース・ファイルを保管するライブラリーの名前を入力します。

TOMBR
変換後のソース・ファイル中の変換後のソース・メンバー (複数の場合もある) の名前を指定します。FROMMBR パラメーターに指定された値が *ALL または 総称* の場合には、TOMBR は *FROMMBR と等しくなければなりません。

***FROMMBR**
FROMMBR パラメーターに指定されたメンバー名は、変換後のソース・メンバー名として使用されます。 FROMMBR(*ALL) を指定した場合には、FROMFILE 中のすべてのソース・メンバーが変換されます。変換後のソース・メンバーは、元のソース・メンバーの名前と同じ名前をもちます。FROMMBR パラメーターに総称名を指定した場合には、その名前に同じ接頭部をもつ指定されたすべてのソース・メンバーが変換されます。変換後のソース・メンバーは、元の総称ソース・メンバーの名前と同じ名前をもちます。

ソース・ファイル・メンバー名
変換後のソース・メンバーの名前を入力します。メンバーが存在していない場合には、これが作成されます。

EXPCPY
/COPY メンバー (複数の場合もある) を変換後のソース・メンバーに展開するかどうかをします。EXPCPY(*YES) を指定する必要があるのは、/COPY メンバーに関係する変換の問題がある場合だけです。

注: メンバーのタイプが RPT または RPT38 の場合には、報告書簡易作成プログラムは常に /COPY メンバーを展開するために、 EXPCPY(*YES) または EXPCPY(*NO) は影響しません。

***NO**
/COPY ファイル・メンバー (複数の場合もある) を変換後のソース・メンバーに展開しません。

***YES**
/COPY ファイル・メンバー (複数の場合もある) を変換後のソース・メンバーに展開します。

CVTRPT
変換報告書を印刷するかどうかを指定します。

***YES**
変換報告書が印刷されます。

***NO**
変換報告書は印刷されません。

SECLVL
変換報告書のメッセージ要約セクションに第 2 レベル・テキストを印刷するかどうかを指定します。

***NO**
第 2 レベル・テキストは変換報告書に印刷されません。

***YES**
第 2 レベル・テキストは変換報告書に印刷されます。

INSRTPL
ILE RPG 仕様書テンプレート (H-、F-、D-、I-、C-、O- 仕様書テンプレート) を変換後のソース・メンバーに挿入するかどうかを指定します。デフォルト値は *NO です。

***NO**
仕様書テンプレートを変換後のソース・メンバーに挿入しません。

***YES**
仕様書テンプレートを変換後のソース・メンバーに挿入します。各仕様書テンプレートは、該当する仕様書セクションの始めに挿入されます。

LOGFILE
変換情報を追跡するために使用されるログ・ファイルの名前を指定します。*NONE が指定されない限り、ログ・ファイルが存在していなければなりません。ファイルは既に存在していなければならず、これは物理データ・ファイルでなければなりません。CPYF コマンドの「FROM ファイル」に QRNCVTLG、ライブラリーに QRPGLE、「TO ファイル」に QARNCVTLG を指定して、ユーザー・ライブラリーにログ・ファイルを作成します。

QRNCVTLG
デフォルトのログ・ファイル QRNCVTLG は、変換情報を入れるために使用されます。

***NONE**
変換情報はログ・ファイルに書き出されません。

ログ・ファイル名
変換情報を追跡するために使用されるログ・ファイルの名前を入力します。

***LIBL**
システムは、ライブラリー・リストを検索して、ログ・ファイルが保管されているライブラリーを見付けます。

ライブラリー名
ログ・ファイルが保管されているライブラリーの名前を入力します。

LOGMBR
変換情報を追跡するために使用されるログ・ファイル・メンバーの名前を指定します。指定されたログ・ファイル・メンバーの既存のデータに新しい情報が追加されます。

ログ・ファイルにメンバーが入っていない場合には、ログ・ファイルと同じ名前をもつメンバーが作成されます。

***FIRST**
コマンドは、指定されたログ・ファイルの最初のメンバーを使用します。

***LAST**
コマンドは、指定されたログ・ファイルの最後のメンバーを使用します。

ログ・ファイル・メンバー名
変換情報を追跡するために使用されるログ・ファイル・メンバーの名前を入力します。

デフォルトの値を使用したメンバーの変換

CVTRPGSRC コマンドで提供されているデフォルト値を利用することができます。ただ単に次のように入力してください。

```
CVTRPGSRC FROMFILE(file name) FROMMBR(member name)
```

この結果として、指定されたソース・メンバーが変換されます。出力は、このファイルを含むライブラリー・リスト中のライブラリーのファイル QRPGLESRC に入れられます。/COPY メンバーは展開されず、仕様書テンプレートが挿入されず、変換報告書は作成されます。ログ・ファイル QRNCVTLG が更新されます。

注: ファイル QRPGLESRC および QRNCVTLG は既に存在していなければなりません。

1 ファイル中のすべてのメンバーの変換

CVTRPGSRC コマンドに FROMMBR(*ALL) および TOMBR(*FROMMBR) を指定することによって、ソース物理ファイル中のすべてのメンバーを変換することができます。変換援助プログラムは、指定されたファイル中のすべてのメンバーを変換するものと見なします。1 つのメンバーが正常に変換できない場合でも、変換処理はまだ続行されます。

例えば、ファイル QRPGSRC 中のすべてのソース・メンバーを変換して、変換後のメンバーをファイル QRPGLESRC に入れたい場合には、次のように入力します。

```
CVTRPGSRC  FROMFILE(OLDRPG/QRPGSRC)
           FROMMBR(*ALL)
           TOFILE(NEWRPG/QRPGLESRC)
           TOMBR(*FROMMBR)
```

このコマンドは、ライブラリー OLDRPG に入っているソース物理ファイル QRPGSRC 中のすべてのソース・メンバーを変換します。新しいメンバーは、ライブラリー NEWRPG のソース物理ファイル QRPGLESRC に作成されます。

同じファイルにすべてのソース (DDS ソース、RPG ソースなど) を保存したい場合には、FROMMBR(*ALL) を指定することによって RPG ソース・メンバーを 1 回のステップで変換することができます。変換援助プログラムは、有効な RPG タイプのメンバーだけを変換します (456 ページの表 45 を参照してください)。

1 ファイル中のいくつかのメンバーの変換

1 つのソース物理ファイル内のいくつかのメンバーのみを変換する必要があり、それらのメンバーがメンバー名に共通する接頭部を共用している場合には、その接頭部の後に * (アスタリスク) を指定することによって、それらのメンバーを変換することができます。

例えば、PAY という接頭部をもつすべてのメンバーを変換したい場合には、次のように入力します。

```
CVTRPGSRC  FROMFILE(OLDRPG/QRPGSRC)
           FROMMBR(PAY*)
           TOFILE(NEWRPG/QRPGLESRC)
           TOMBR(*FROMMBR)
```

このコマンドは、ライブラリー OLDRPG に入っているソース物理ファイル QRPGSRC 中のすべてのソース・メンバーを変換します。新しいメンバーは、ライブラリー NEWRPG のソース物理ファイル QRPGLESRC に作成されます。

試行変換の実行

変換中に問題の起こる可能性があると思われるソース・メンバーについては、試行的に実行を行うことができます。これを行うと、変換後のソース・メンバーの変換報告書が得られ、これによってある種の変換エラーを識別することができます。

例えば、PAYROLL というソース・メンバーに対して試行変換を行うためには、次のように入力します。

```
CVTRPGSRC  FROMFILE(OLDRPG/QRPGSRC)
           FROMMBR(PAYROLL)
           TOFILE(*NONE)
```

TOMBR パラメーターは *FROMMBR として指定する必要があります。しかし、これはデフォルト値なので、デフォルト値が変更されていない限り、これを指定する必要はありません。CVTRPT パラメーターは *YES と指定します。これもデフォルト値です。そうでない場合には、変換はただちに停止します。

TOFILE(*NONE) パラメーターを使用すると、変換援助プログラムは変換後のメンバーを生成しませんが、変換報告書を作成することはできます。変換報告書の詳細については、471 ページの『変換の分析』を参照してください。

変換報告書の入手

変換援助プログラムは通常、ユーザーがコマンドを出すたびに変換報告書を作成します。スプール・ファイルの名前は、TOFILE パラメーターに指定されたファイル名と対応します。既に存在しているメンバー、またはサポートされないメンバー・タイプのメンバーを変換しようとした場合には、これらのメンバーが変換されなかったことを示すメッセージがジョブ・ログに書き込まれます。要求された場合には、ログ・ファイルも変換が行われなかったことを反映するように更新されます。しかし、これらのメンバーに関する情報は報告書に入れられません。

変換報告書には次の情報が入れられます。

- CVTRPGSRC コマンドのオプション
- 次のものを含むソース・セクション

  - 変換エラーまたは警告
  - CALL 命令
  - /COPY 指示ステートメント
- メッセージの要約
- 最終の要約

変換エラー・メッセージは、エラーの訂正方法を提供します。さらに、変換後のソースの CALL 命令および /COPY 指示ステートメントにフラグが付けられ、変換中のアプリケーション・プログラムの各種の部分を識別するのに役立ちます。一般に、アプリケーション・プログラムのすべての RPG 構成要素を同時に変換する必要があります。

変換報告書が不要な場合には、CVTRPT(*NO) を指定してください。

報告書簡易作成機能ソース・メンバーの変換

報告書簡易作成機能ソース・メンバー (タイプ RPT または RPT38) が RPG III または OPM RPG/400 ソース・プログラムで検出された場合には、変換援助プログラムが CRTRPTPGM コマンドを呼び出してこれらのソース・メンバーを展開し、それを変換します。(これは、報告書簡易作成機能が ILE RPG によってサポートされないためです。)

報告書簡易作成プログラムは、変換援助プログラムによって呼び出されるたびに、スプール・ファイルを作成します。これらのエラーは変換報告書に含まれないので、報告書簡易作成機能の展開でエラーが起こったかどうかを調べるために、このファイルを検査したい場合があります。

特に、/COPY メンバーが見付からなかったことを示すエラー・メッセージについて、報告書簡易作成機能スプール・ファイルを検査したい場合があります。変換援助プログラムは、これらのファイルが抜けているかどうかは認識できません。しかし、これらのファイルがない場合には、ソースを正常に変換することができないことがあります。

注: 変換するメンバーのソース・メンバー・タイプが RPT でも RPT38 でもないが、そのメンバーが報告書簡易作成ソース・メンバーである 場合には、変換前にそのメンバーに適切なソース・メンバー・タイプ (RPT または RPT38) を割り当ててください。そうしないと、変換エラーが起こることがあります。

報告書簡易作成機能は、/COPY メンバー内のコンパイル時データをサポートします。RPG IV はこれをサポートしません。いくつかのプログラムが使用できるように、/COPY メンバー内にコンパイル時データを保持している場合は、このコンパイル時データをユーザー・スペースに移してユーザー・スペース API を介してアクセスすることを検討してください。

組み込み SQL をもつソース・メンバーの変換

組み込み SQL を含むコードを変換する際に、SQL コードが複数行にわたって継続している場合には、以下のようになります。

- 継続行はあっても 74 桁目がブランクであれば、行は単に ILE メンバーにコピーされるだけです。

注: 74 桁目が文字ストリング内のブランク文字である場合には、これが問題になる場合があります。

- 74 桁目がブランクでない場合には、この行から /END-EXEC までのすべての SQL コードは連結され、80 桁全部を埋め込んだ ILE メンバーにコピーされます。これが起こると、次のようになります。
  - 75 桁目からのコメントは無視されます。
  - 組み込まれたコメント行 (C*) が ILE メンバーにコピーされてから、連結されたコードがコピーされます。
  - DBCS リテラルが分割されると、問題が起きる場合があります。

  こうした連結や再形式設定を行いたくない場合は、必ず 74 桁目をブランクにしておいてください。

仕様書テンプレートの挿入

RPG IV のソースの仕様は新しいので、仕様書テンプレートを変換後のソースに挿入したい場合があります。テンプレートを挿入するためには、CVTRPGSRC コマンドに INSRTPL(*YES) を指定してください。デフォルト値は INSRTPL(*NO) です。

データ・ファイルからのソースの変換

変換援助プログラムはデータ・ファイルからソースを変換します。データ・ファイルは一般的に順序番号をもっていないために、変換後の出力を入れるためのファイルの最小レコード長は 80 桁です。(457 ページの図 211 を参照してください。)データ・ファイルの推奨されるレコード長は 100 桁です。

注: データ・ファイルに順序番号がある場合には、変換援助プログラムを実行する前に、順序番号を除去してください。

ソース変換の例

この例では、サンプル RPG III ソース・メンバーを RPG IV に変換します。 469 ページの図 212 は、RPG III バージョンのソースを示しています。

```
     H                                                                  TSTPGM
     FFILE1   IF  E                    DISK                             COMM1
     FQSYSPRT O   F     132     OF    LPRINTER
     LQSYSPRT  60FL 56OL
     E                    ARR1    3   3  1              COMM2
     E                    ARR2    3   3  1
     IFORMAT1
     I              OLDNAME                         NAME
     I* DATA STRUCTURE COMMENT
     IDS1         DS
     I                                        1   3 FIELD1
     I* NAMED CONSTANT COMMENT
     I              'XYZ'                 C         CONST1               COMM3
     I                                        4   6 ARR1
     C           ARR1,3    DSPLY
     C                     READ FORMAT1                 01
     C           NAME      DSPLY
     C                     SETON                     LR
     C                     EXCPTOUTPUT
     OQSYSPRT E   01           OUTPUT
     O                         ARR2,3    10
**
123
**
456
```

図 212. TEST1 の RPG III ソース

このソースを変換するためには、次のように入力してください。

```
CVTRPGSRC  FROMFILE(MYLIB/QRPGSRC) FROMMBR(TEST1)
           TOFILE(MYLIB/QRPGLESRC) INSRTPL(*YES)
```

変換後のソース・メンバーを 470 ページの図 213 に示します。

```
 1 .....H*unctions++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++Comments+++++++++
 2      H DFTNAME(TSTPGM)
 3 .....F*ilename++IPEASFRlen+LKlen+AIDevice+.Functions++++++++++++++++++++++++++++Comments+++++++++
 4      FFILE1     IF   E             DISK                                        COMM1
 5      FQSYSPRT   O    F  132        PRINTER OFLIND(*INOF)
 6      F                                 FORMLEN(60)
 7      F                                 FORMOFL(56)
 8 .....D*ame++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++++++++++++Comments+++++++++
 9      D ARR2            S              1    DIM(3) CTDATA PERRCD(3)
10      D* DATA STRUCTURE COMMENT
11      D DS1             DS
12      D  FIELD1                 1      3
13      D  ARR1                   4      6
14      D                                 DIM(3) CTDATA PERRCD(3)              COMM2
15      D* NAMED CONSTANT COMMENT
16      D CONST1          C               CONST('XYZ')                         COMM3
17 .....I*ilename++SqNORiPos1+NCCPos2+NCCPos3+NCC.............................Comments+++++++++
18 .....I*.............Ext_field+Fmt+SPFrom+To+++DcField+++++++++L1M1FrP1MnZr......Comments+++++++++
19      IFORMAT1
20      I              OLDNAME                    NAME
21 .....C*0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....Comments+++++++++
22      C     ARR1(3)       DSPLY
23      C                   READ      FORMAT1                                01
24      C     NAME          DSPLY
25      C                   SETON                                        LR
26      C                   EXCEPT    OUTPUT
27      OQSYSPRT   E            OUTPUT           01
28      O                       ARR2(3)             10
29 **CTDATA ARR1
30 123
31 **CTDATA ARR2
32 456
```

図 213. TEST1 の変換後の (RPG IV) ソース

変換後のソースについて、次のことに注意してください。

- 新しい仕様書タイプは、H (制御)、F (ファイル)、D (定義)、I (入力)、C (演算)、および O (出力) です。これらはこの順に入力しなければなりません。

  CVTRPGSRC に INSRTPL(*YES) が指定されていたので、変換後のソースには新しいタイプ用の仕様書テンプレートが入っています。
- 制御、ファイル、および定義仕様書は、キーワード指向です。行 2、4 ～ 7、および 9 ～ 16 を参照してください。
- ILE メンバーには新規の仕様書タイプである定義があります。これはデータ構造の他に、独立型フィールド、配列、および名前付きの固定情報も定義するために使用されます。

  この例では、

  – ARR2 は、独立型配列として定義されています (9 行目)。
  – データ構造 DS1 は、2 つのサブフィールド FIELD1 および ARR1 をもつデータ構造として定義されています (11 ～ 14 行目)。
  – 固定情報 CONST1 は、固定情報として定義されています (16 行目)。

  入力 (I) 仕様書は、現在では、ファイルのレコードおよびフィールドを定義するためにのみ、使用されています。19 ～ 20 行目を参照してください。
- ファイル仕様書 (補足 E) は除外されています。配列とテーブルは、定義仕様書を使用して定義されています。

- 補足仕様書のレコード・アドレス・ファイル (RAF) は、ファイル仕様書のキーワード RAFDATA で置き換えられています。
- ファイル仕様書 (補足 L) は除外されています。この仕様書は、ファイル仕様書のキーワード FORMLEN および FORMOFL で置き換えられています。6 および 7 行目を参照してください。
- フィールドおよびファイルの 10 桁の名前が使えるよう、すべての仕様書タイプが拡張されています。
- RPG IV では、データ構造 (定義仕様書を使って定義される) は入力仕様書より前になければなりません。

  変換後のソースでは、データ構造 DS1 (11 行目) が、FORMAT1 情報 (19 行目) を含む仕様書より前に来るように移動されていることに、注意してください。
- RPG III では、名前付き固定情報をデータ構造の真ん中に置くことができます。これは RPG IV では許されません。

  変換後のソースでは、CONST1 (16 行目) は、データ構造 DS1 (11 行目) の後に来るように移動されています。
- 仕様書が移動される場合には、その前にあるコメントも一緒に移動されます。

  変換後のソースでは、CONST1 および DS1 の上にあったコメントは、それに続く仕様書と一緒に移動されました。
- RPG III では、配列をデータ構造サブフィールドとして定義するためには、配列とデータ構造サブフィールドの両方を同じ名前で定義します。この二重定義は RPG IV では許されません。代わりに、新しいキーワード構文を使用してサブフィールドを定義する時に、配列属性を指定します。

  この例では、ARR1 は OPM バージョンで 2 度定義されていますが、変換後のソースでは単一の定義にマージされています。13 ～ 14 行目を参照してください。

  RPG III 配列仕様書の組み合わせの結果として、配列定義の順序が変更されることがあります。順序が変更された配列がコンパイル時配列の場合には、配列データのロードは影響を受けることがあります。この問題を解決するため RPG IV では、** レコードにはキーワード形式を提供します。** の後には、キーワード FTRANS、ALTSEQ、または CTDATA の 1 つを入力します。キーワードが CTDATA の場合には、10 ～ 19 桁目に配列またはテーブル名を入力します。

  この例では、ARR2 の 2 つの RPG III 仕様書の組み合わせのために、配列 ARR2 は、配列 ARR1 より先行しています。変換援助プログラムは、変換後の ** レコードにキーワードおよび配列名を挿入し、これによりコンパイル時データの正しいロードは保証されます。29 と 31 行目を参照してください。
- 配列構文が変更されたことに注意してください。RPG III での表記 ARR1,3 は RPG IV では ARR1(3) となります。28 行目を参照してください。

変換の分析

変換援助プログラムは、変換結果を分析する方法を 2 つ提供します。それらは次のとおりです。

- 変換エラー報告書
- ログ・ファイル

変換報告書の使用

CVTRPGSRC コマンドで CVTRPT(*YES) パラメーターを指定した場合には、変換援助プログラムは変換報告書を生成します。スプール・ファイル名は、TOFILE パラメーターに指定するファイル名と同じです。

変換報告書は、次の 4 つの部分から成ります。

1. CVTRPGSRC コマンドのオプション
2. ソース・セクション
3. メッセージ要約
4. 最終要約

リストの最初の部分には、CVTRPGSRC で使用されたコマンド・オプションの要約が入ります。図 214 は、変換例のコマンドの要約を示します。

```
5722WDS V5R2M0 020719 RN         IBM ILE RPG                 ISERIES   02/08/15  20:41:35       PAGE      1
  コマンド . . . . . . . . . . . . :   CVTRPGSRC
    投入元 . . . . . . . . . . . . :     DAVE
 FROM ファイル . . . . . . . . . . :   QRPGSRC
    ライブラリー . . . . . . . . . :     MYLIB
 取り出しメンバー  . . . . . . . . :   REPORT
 TO ファイル . . . . . . . . . . . :   QRPGLESRC
    ライブラリー . . . . . . . . . :     MYLIB
 受け入れメンバー  . . . . . . . . :   *FROMMBR
 ログ・ファイル  . . . . . . . . . :   *NONE
    ライブラリー  . . . . . . . . . :
 ログ・メンバー  . . . . . . . . . :   *FIRST
 拡張コピー・メンバー  . . . . . . :   *NO
 印刷変換報告書  . . . . . . . . . :   *YES
 第 2 レベルのテキストの組み込み . :   *YES
 仕様テンプレートの挿入  . . . . . :   *YES
```

図 214. サンプル変換報告書のコマンド要約

ソース・セクションには、通知、または、エラー・メッセージと関連付けられた行が組み込まれています。これらの行には SEU でのブラウズを容易にするために 1 桁目にアスタリスク (*) があります。メッセージ要約には、3 つのメッセージ・タイプがすべて入っています。

特に関心があると思われる 2 つの通知メッセージは、次のものです。

- RNM0508 — フラグ /COPY ステートメント
- RNM0511 — フラグ CALL 命令

プログラムの中のすべての /COPY メンバーは、対応する ILE RPG プログラムがエラーを起こさずにコンパイルするために、変換しなければなりません。同様に、CALL で関連付けられているすべてのメンバーを同時に変換したい場合があります。報告書のこの部分を使うと、これらのメンバーの識別に役立ちます。 473 ページの図 215 は、サンプル変換のソース・セクションを示しています。

```
# 5722WDS V5R2M0  020719 RN         IBM ILE RPG                                ISERIES1    02/08/15 20:41:35       PAGE       2
  FROM ファイル  . . . . . . . . . :   MYLIB/QRPGSRC(REPORT)
  TO ファイル .. . . . . . . . . . :   MYLIB/QRPGLESRC(REPORT)
  ログ・ファイル . . . . . . . . . :   *NONE
                          変    換    報    告    書
   順序  <---------------------------- ソース仕様 ----------------------------><----------- 注記 ------------------> PAGE
   番号  ....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+....9....+...10....+...11....+...12 行
  000002 C                   CALL      PROG1
*RNM0511 00 CALL 命令コードが見つかった。
  000003 C/COPY COPYCODE
*RNM0508 00 /COPY コンパイラー指示が見つかった。
  000004 C                   FREE      PROG2
*RNM0506 30 FREE 命令コードは RPG IV ではサポートされていない。

      * * * * *  ソ ー ス の 終 わ り  * * * * *
```

図 215. 変換報告書のサンプル・ソース・セクション

リストのメッセージ要約は、出されたいろいろなメッセージを示します。 SECLVL(*YES) を指定すると、メッセージ要約に第 2 レベルのメッセージが現れます。図 216 は、第 2 レベルのメッセージを含む、変換例のメッセージ・セクションを示します。

```
# 5722WDS V5R2M0  020719 RN         IBM ILE RPG                                ISERIES1    02/08/15 20:41:35       PAGE       2
                        メ ッ セ ー ジ の 要 約
 MSG ID  SV 番号 メッセージ・テキスト
*RNM0508 00      1 /COPY コンパイラー指示が見つかった。
                   原因 . . . . . :  この RPG IV ソースを正しくコンパイルする
                     ためには、このソース・メンバーに入っている /COPY ソース・
                     メンバーも RPG IV に変換されていることを確認してください。
                   回復手順 . . . :  RPG IV でのコンパイルの前に、すべての
                     /COPY ソース・メンバーが変換されていることを確認してください。
                     場合によっては、/COPY コンパイラー指示を使用するソース・メンバー
                     の変換およびコンパイルを試みると、問題が起こる結果となる場合
                     があります。結果がこの状態になった場合には、/COPY メンバー
                     を変換されるソースに拡張するために、CVT RPG SRC コマンドの
                     EXPCPY パラメーターに *YES を指定してください。詳細については、
                     「ILE RPG プログラマーの手引き」を参照してください。
*RNM0511 00      1 CALL 命令コードが見つかった。
                   原因 . . . . . :  CALL 命令コードが入っている RPG 仕様が見つかり
                       ましたが、次のことが必要となる可能性があります。
                       -- 静的バインドの利点を生かすために CALL 命令コードを CALLB に
                       変更する。
                       -- 適用業務のすべてのプログラムを RPG IV に変換する。
                   回復手順 . . . :  静的バインドの利点を生かしたい場合には、CALL
                     命令コードを CALLB に変換してください。あるいは適用業務のすべての
                     プログラムを移行したい場合には、呼び出されるプログラムを RPG IV に
                     変換してください。
*RNM0506 30      1 FREE 命令コードは RPG IV ではサポートされていない。
                   原因 . . . . . :  RPG III または RPG/400 プログラムに、RPG IV で
                     サポートされていない FREE 命令コードが入っています。
                   回復手順 . . . :  FREE 命令を除去して、変換されたソースをコンパイル
                     する前に、プログラミング論理が影響を受けないように代替コードと
                     置き換えてください。
      * * * * *  メ ッ セ ー ジ の 要 約 の 終 わ り  * * * * *
```

図 216. 変換報告書のサンプル・メッセージ要約

リストの最終の要約は、メッセージおよびレコードの統計を提供します。最終状況メッセージは、ジョブ・ログにも入れられます。 474 ページの図 217 は、サンプル変換の最終の要約セクションを示しています。

```
                        最終の要約
メッセージ合計:
  通知          (00) . . . . . . . :       2
  警告          (10) . . . . . . . :       0
  重大エラー    (30+)  . . . . . . :       1
  --------------------------------  ------
  合計   . . . . . . . . . . . . . :       3
ソース合計:
  元のレコード   . . . . . . . . . . :       3
  書き込まれた変換レコード   . . . . :       4
  出された最も重大なメッセージ   . . :      30
      * * * * *  最 終 の 要 約 の 終 わ り  * * * * *
      * * * * *  変 換 の 終 わ り  * * * * *
```

図 217. 変換報告書のサンプル最終要約

ログ・ファイルの使用

ログ・ファイルを参照することによって、変換の結果を調べることができます。各変換の後に、ログ・ファイルが更新されます。次のものが追跡されます。

- ソース・メンバーとそのライブラリー名
- 変換後のソース・ファイル名とそのライブラリー名
- 見つかった最高重大度エラー

例えば、エラーが見つからなければ、変換状況は 0 に設定されます。重大エラーが見付かった場合には、状況は 30 に設定されます。

サポートされないメンバー・タイプのメンバー、または既に存在しているメンバーを変換しようとした場合には、これが重大エラー (重大度 40 以上) であるので、変換は実行されません。レコードがログ・ファイルに追加され、変換状況が 40 に設定されます。変換が行われず TO MBR が生成されなかったことを示すために、TO ファイル、TO MBR、および TO LIB が、ブランクに設定されます。

ログ・ファイルは外部記述の物理データベース・ファイルです。このファイルの ″モデル″ は、ライブラリー QRPGLE のファイル QARNCVTLG に提供されています。このモデルは、QRNCVTLG という 1 つのレコード様式をもっています。すべてのフィールド名は長さが 6 桁で、命名規則 LGxxxx に従います。ここで xxxx はフィールドを記述します。 475 ページの図 218 は、このファイルの DDS を示しています。

次の CPYF コマンドを使用して、このモデルのコピーをユーザー自身のライブラリー (ここでは MYLIB) に作成してください。ユーザーのログ・ファイルに QRNCVTLG という名前を指定することもできます。これは変換援助プログラムのデフォルトのログ・ファイル名です。

```
CPYF FROMFILE(QRPGLE/QARNCVTLG) TOFILE(MYLIB/QRNCVTLG)
     CRTFILE(*YES)
```

```
A          R QRNCVTFM
A            LGCENT          1A         COLHDG('CVT' 'CENT')
A                                       TEXT('Conversion Century: 0-20th 1-+
A                                       21st')
A            LGDATE          6A         COLHDG('CVT' 'DATE')
A                                       TEXT('Conversion Date : format is Y+
A                                       YMMDD')
A            LGTIME          6A         COLHDG('CVT' 'TIME')
A                                       TEXT('Conversion Time : format is H+
A                                       HMMSS')
A            LGSYST          8A         COLHDG('CVT' 'SYST')
A                                       TEXT('Name of the system running co+
A                                       nversion')
A            LGUSER         10A         COLHDG('CVT' 'USER')
A                                       TEXT('User Profile name of the user+
A                                       running conversion')
A            LGFRFL         10A         COLHDG('FROM' 'FILE')
A                                       TEXT('From File')
A            LGFRLB         10A         COLHDG('FROM' 'LIB')
A                                       TEXT('From Library')
A            LGFRMR         10A         COLHDG('FROM' 'MBR')
A                                       TEXT('From Member')
A            LGFRMT         10A         COLHDG('FMBR' 'TYPE')
A                                       TEXT('From Member Type')
A            LGTOFL         10A         COLHDG('TO' 'FILE')
A                                       TEXT('To File')
A            LGTOLB         10A         COLHDG('TO' 'LIB')
A                                       TEXT('To Library')
A            LGTOMR         10A         COLHDG('TO' 'MBR')
A                                       TEXT('To Member')
A            LGTOMT         10A         COLHDG('TMBR' 'TYPE')
A                                       TEXT('To Member Type')
A            LGLGFL         10A         COLHDG('LOG' 'FILE')
A                                       TEXT('Log File')
A            LGLGLB         10A         COLHDG('LOG' 'LIB')
A                                       TEXT('Log Library')
A            LGLGMR         10A         COLHDG('LOG' 'MBR')
A                                       TEXT('Log Member')
A            LGCEXP          1A         COLHDG('CPY' 'EXP')
A                                       TEXT('Copy Member Expanded: Y=Yes, +
A                                       N=No')
A            LGERRL          1A         COLHDG('CVT' 'RPT')
A                                       TEXT('Conversion Report Printed: Y=+
A                                       Yes, N=No')
A            LGSECL          1A         COLHDG('SEC' 'LVL')
A                                       TEXT('Second Level Text Printed: Y=+
A                                       Yes, N=No')
A            LGINSR          1A         COLHDG('INSR' 'TPL')
A                                       TEXT('Template Inserted: Y=Yes, N=N+
A                                       o')
A            LGSTAT          2A         COLHDG('CVT' 'STAT')
A                                       TEXT('Conversion Status')
A            LGMRDS         50A         COLHDG('MBR' 'DESC')
A                                       TEXT('Member Description')
```

図 218. ライブラリー QRPGLE のモデル・ログ・ファイル QARNCVTLG の DDS

変換上の問題の分析解決

変換問題は、次の理由の 1 つ以上によって起こることがあります。

- RPG III ソースにコンパイル・エラーがある
- RPG III 言語のある種の機能が RPG IV ではサポートされていない

- 1 つまたは複数の /COPY コンパイラー指示ステートメントが RPG III ソースに存在している
- 外部記述データ構造の使用
- OPM と ILE との実行時における動作上の相違点

これらの領域のおのおのについて、以下の各項で説明します。

既存の RPG III コードのコンパイル・エラー

変換援助プログラムは、有効な RPG III プログラム、すなわちコンパイル・エラーのないプログラムを変換しようとしているものと見なします。これ以外の場合には、変換中に予測できない結果となることがあります。プログラムにコンパイル・エラーが入っていると考えられる場合には、変換を実行する前に、最初に RPG III コンパイラーを使用してこれをコンパイルし、エラーを訂正してください。

サポートされていない RPG III 機能

RPG III 言語のいくつかの機能は、RPG IV ではサポートされていません。それらの機能のうち最も重要なのは次のものです。

- 報告書簡易作成機能
- FREE 命令コード
- DEBUG 命令コード

報告書簡易作成機能がサポートされていないので、タイプが RPT または RPT38 の場合には、変換に先立って、変換援助プログラムがこれらのプログラムを自動的に展開します (すなわち、報告書簡易作成機能を呼び出します)。

変換の前または後に、FREE または DEBUG 命令コードを等価の論理で置き換える必要があります。

CVTRPGSRC コマンドで CVTRPT(*YES) オプションを指定した場合には、これらのタイプのほとんどの問題を識別する変換報告書を受け取ります。

報告書簡易作成機能メンバーの詳細については、 467 ページの『報告書簡易作成機能ソース・メンバーの変換』を参照してください。RPG III と RPG IV の相違点について詳しくは、 447 ページの『付録 A. OPM RPG/400 と AS/400 用 ILE RPG との動作上の相違点』を参照してください。

/COPY コンパイラー指示ステートメントの使用

場合によっては、変換後の RPG IV ソースを実際にコンパイルするまでエラーが見付からないことがあります。このタイプの変換エラーは通常、/COPY コンパイラー指示ステートメントの使用に関連しています。これらのエラーは 2 つのカテゴリー、すなわち組み合わせ問題および文脈依存問題に分類されます。これらの問題が起こる理由およびその分析解決方法の説明は次のとおりです。

組み合わせ問題

RPG III 言語と RPG IV 言語との相違点のために、変換援助プログラムは、特定のソースの順序を変えなければなりません。この順序変えの例は、RPG III ソース・メンバー TEST1 の場合の 468 ページの『ソース変換の例』に示されています。デ

ータ構造 DS1 の配置を 469 ページの図 212 と 470 ページの図 213 で比較した場合に、データ構造 DS1 がレコード様式 FORMAT1 の前にくるように移動されたことが分ります。

今、RPG III メンバー TEST1 が 2 つのメンバー TEST2 および COPYDS1 に分割されているものとします。ここで、データ構造 DS1 と名前付き固定情報 CONST1 は、コピー・メンバー COPYDS1 に入っています。このコピー・メンバーはソース TEST2 に含まれます。 図 219 と 図 220 は、それぞれ TEST2 と COPYDS1 のソースを示します。

```
     H                                                                  TSTPGM
     FFILE1   IF  E                    DISK                             COMM1
     FQSYSPRT O   F     132     OF    LPRINTER
     LQSYSPRT  60FL 56OL
     E                    ARR1    3   3  1               COMM2
     E                    ARR2    3   3  1
     IFORMAT1
     I              OLDNAME                          NAME
      /COPY COPYDS1
     C           ARR1,3    DSPLY
     C                     READ FORMAT1                  01
     C           NAME      DSPLY
     C                     SETON                     LR
     C                     EXCPTOUTPUT
     OQSYSPRT E   01           OUTPUT
     O                         ARR2,3    10
**
123
**
456
```

図 219. TEST2 の RPG III ソース

```
     I* DATA STRUCTURE COMMENT
     IDS1         DS
     I                                        1   3 FIELD1
     I* NAMED CONSTANT COMMENT
     I              'XYZ'                 C         CONST1               COMM3
     I                                        4   6 ARR1
```

図 220. COPYDS1 の RPG III ソース

この場合には、変換援助プログラムはメンバー TEST2 およびコピー・メンバー COPYDS1 の両方を正しく変換します。ただし、コピー・メンバーは、コンパイル時に組み込まれる時に、/COPY コンパイラー指示ステートメントが置かれている FORMAT1 の下に挿入されます。結果として、コピー・メンバー COPYDS1 のすべてのソース行が ″ソース・レコードの順序が違っている″ というエラーを受け取ります。RPG IV では、定義仕様書は入力仕様書より前になければなりません。

/COPY メンバーの内容が不明なので、変換援助プログラムは /COPY 指示ステートメントを FORMAT1 の上に移動できなかったことに、注意してください。

この種の問題の訂正方法には次の 2 つがあります。

1. CVTRPGSRC コマンドの EXPCPY(*YES) オプションを使用して、変換後の RPG IV ソース・メンバーにすべての /COPY メンバーを組み込む。

この方法は簡単であり、時間を最も有効に利用できます。ただし、各ソース・メンバーに /COPY メンバーを含むことにより、アプリケーション・プログラムの保守容易性は減少します。

2. 変換後に ILE RPG のコンパイラー・リストおよび「*WebSphere Development Studio: ILE RPG 解説書*」にある情報を使って、手操作でコーディングを訂正する。

この種の問題の例としては、ほかに次のものがあります。

- ファイル仕様書 (補足 L) およびレコード・アドレス・ファイル

  RPG III の場合、ファイル仕様書 (補足 L) およびファイル仕様書 (補足 E) のレコード・アドレス・ファイルはファイル仕様書のキーワード (RAFDATA、FORMLEN、および FORMOFL) に変換されます。 /COPY メンバーの内容に、ファイル仕様書 (補足 L) または ファイル仕様書 (補足 E) のレコード・アドレス・ファイル、あるいはその両方が含まれているが、対応するファイル仕様書が含まれていない場合には、変換援助プログラムはキーワードの挿入位置が分りません。

- ファイル仕様書 (補足 E) の配列およびデータ構造サブフィールド

  468 ページの『ソース変換の例』で述べたように、RPG IV では、独立型配列とデータ構造サブフィールドを同じ名前で定義することはできません。したがって、例 TEST1 (470 ページの図 213) で示したように、変換援助プログラムはこの 2 つの定義を組み合わせなければなりません。しかし、配列およびデータ構造サブフィールドが同じソース・メンバーにない (すなわち、一方または両方が /COPY メンバーにある) 場合には、この組み合わせを実行することはできず、結果としてコンパイル時エラーとなります。

- 組み合わされたコンパイル時配列およびコンパイル時データ (**) レコード

  例 TEST1 (470 ページの図 213) で示したように、コンパイル時配列がデータ構造サブフィールド定義と組み合わされた場合には、配列データのロードが影響を受けることがあります。この問題を解決するために、少なくとも 1 つのコンパイル時配列が組み合わされた場合には、コンパイル時配列データが新しい **CTDATA 形式に変更されます。しかし、配列およびデータが同じソース・メンバーにない (すなわち、一方または両方が COPY メンバーにある) 場合には、**CTDATA 形式を使用するコンパイル時データ・レコードの命名は適切に進めることはできません。

文脈に依存した問題

RPG III では、基本ソース・メンバーの前後の仕様書の文脈がない /COPY メンバーでは仕様書のタイプを判別できないことがあります。この問題には次の 2 つの場合があります。

- データ構造サブフィールドまたはプログラム記述ファイル・フィールド

```
I* RPG III ソース・メンバーが、以下のフィールド FIELD1 および
I* FIELD2 を記述するソース・ステートメントしか含んでいない場合、
I* 変換援助プログラムは変換の方法を判別できません。
I* これらのステートメントは、(定義仕様書に変換される)
I* データ構造フィールドである場合も、(入力仕様書に変換される)
I* プログラム記述ファイル・フィールドである場合もあります。
I                                       1   3 FIELD1
I                                       4   6 FIELD2
```

図 221. 入力フィールドのみの RPG III /COPY ファイル

- 外部記述データ構造フィールドの名前変更または外部記述ファイル・フィールドの名前変更

```
I* RPG III ソース・メンバーが、以下のフィールド CHAR を
I* 記述するソース・ステートメントしか含んでいない場合、
I* 変換援助プログラムは変換の方法を判別できません。
I* このステートメントは、(定義仕様書に変換される)
I* 外部記述データ構造フィールドの名前変更である場合も、
I* (入力仕様書に変換される) 外部記述ファイル・フィールドの
I* 名前変更である場合もあります。
I             CHARACTER                     CHAR
```

図 222. 名前変更フィールドのある RPG III ソース

上の 2 つの例では、データ構造が仮定され、定義仕様書が作成されます。入力仕様書コードを含むコメントのブロックも作成されます。例えば、変換援助プログラムは図 221 のソースを図 223 に示されているコードに変換します。入力仕様書コードが必要な場合には、定義仕様書を削除し、対応する入力仕様書からアスタリスクを消去してください。

```
D* RPG III ソース・メンバーが、以下のフィールド FIELD1 および
D* FIELD2 を記述するソース・ステートメントしか含んでいない場合、
D* 変換援助プログラムは変換の方法を判別できません。
D* これらのステートメントは、(定義仕様書に変換される)
D* データ構造フィールドである場合も、(入力仕様書に変換される)
D* プログラム記述ファイル・フィールドである場合もあります。
D FIELD1                  1      3
D FIELD2                  4      6
I*                                  1    3  FIELD1
I*                                  4    6  FIELD2
```

図 223. 入力フィールドのみのソースを変換した後の RPG IV ソース

この種の問題を訂正するためには 2 つの方法があることを覚えておいてください。CVTRPGSRC コマンドの EXPCPY(*YES) オプションを使用するか、あるいは変換後にコードを手操作で訂正してください。

外部記述データ構造の使用

CVTRPGSRC コマンドで EXPCPY(*YES) オプションを指定しても、手操作で修正しなければならないことがある問題が、2 つあります。

- 配列と外部記述 DS サブフィールドの組み合わせ
- 外部記述 DS サブフィールドの名前変更および初期設定

これらの問題は、外部記述データ構造の使用に関係しています。

これらの問題はコンパイル時エラーを生じるので、ILE RPG コンパイラー・リストや「*WebSphere Development Studio: ILE RPG 解説書*」の情報を使ってそれらを訂正することができます。

配列と外部記述 DS サブフィールドの組み合わせ

前に述べたように、RPG IV では、独立型配列とデータ構造サブフィールドを同じ名前で定義することはできません。一般に、変換援助プログラムはこの 2 つの定義を組み合わせます。しかし、サブフィールドが外部記述データ構造にある場合には、この組み合わせは処理されず、変換後のソース・メンバーを手操作で訂正する必要があります。

例えば、図 224 のフィールド ARRAY は図 225 に 2 回組み込まれます。すなわち、独立型配列として 1 回組み込まれ、外部記述データ構造 EXTREC としてもう 1 回組み込まれます。変換時に、生成される RPG IV ソースが図 226 に示されています。ARRAY が 2 回定義されているので、このコーディングはコンパイルされません。この問題を訂正するためには、図 227 に示されているように、独立型配列を削除して、キーワードをもつサブフィールドをデータ構造 DSONE に追加してください。

```
A          R RECORD
A            CHARACTER     10
A            ARRAY         10
```

図 224. 外部データ構造の DDS

```
E                    ARRAY      10  1
IDSONE     E DSEXTREC
C           CHAR      DSPLY
C                     SETON                    LR
```

図 225. 配列付きの外部データ構造を使う RPG III ソース

```
D ARRAY           S              1    DIM(10)
D DSONE         E DS                  EXTNAME(EXTREC)
C     CHAR          DSPLY
C                   SETON                    LR
```

図 226. 配列に 2 つの定義を持つ RPG IV ソース

```
D DSONE         E DS                  EXTNAME(EXTREC)
D  ARRAY        E                     DIM(10)
C     CHAR          DSPLY
C                   SETON                    LR
```

図 227. 配列に単一の定義を持つ訂正済みの RPG IV ソース

外部記述 DS サブフィールドの名前変更と初期化

RPG III の場合には、外部記述データ構造のフィールドの名前変更と初期化の両方とも行う時には、図 228 のフィールド CHAR に示されているように、2 つのソース行を使用しなければなりません。図 229 に示されているように、変換後のソースにも 2 つのソース行が入っています。フィールドの 2 つのソース行をこれに使用すると、フィールド CHAR が 2 回定義されているので、コンパイル時エラーが起こります。このコーディングを訂正するためには、図 230 に示されているように、フィールド CHAR のキーワードを組み合わせて単一行にしなければなりません。ここでは、キー・フィールド INZ と EXTFLD が組み合わされたので、フィールド CHAR が 1 つだけ示されています。

```
IDSONE     E DSEXTREC
I              CHARACTER                       CHAR
I I            'XYZ'                           CHAR
C           CHAR      DSPLY
C                     SETON                     LR
```

図 228. 名前変更され初期化された外部サブフィールドを持つ RPG III ソース

```
D DSONE         E DS                  EXTNAME(EXTREC)
D  CHAR         E                     EXTFLD(CHARACTER)
D  CHAR         E                     INZ('XYZ')
C     CHAR          DSPLY
C                   SETON                                      LR
```

図 229. 名前変更されたサブフィールドに 2 つの定義を持つ RPG IV ソース

```
D DSONE         E DS                  EXTNAME(EXTREC)
D  CHAR         E                     EXTFLD(CHARACTER) INZ('XYZ')
C     CHAR          DSPLY
C                   SETON                                      LR
```

図 230. 単一の定義を持つ訂正済みの RPG IV ソース

実行時の相違点

データ構造内でオーバーラップする実行前の配列がある場合、実行時にこれらの配列をロードする順序は、RPG III と RPG IV では異なる場合があります。こうした順序の相違点のために、オーバーラップしている部分のデータに相違が起こることがあります。配列がロードされる順序は、ソースで見付かった順序です。この順序は、配列が変換時にサブフィールドと組み合わせられた時に変更される可能性があります。

一般に、OPM と ILE プログラムからなるアプリケーション・プログラムが、OPM のデフォルトの活動化グループと指定の活動化グループとにまたがって分割されるような状況は避けるべきです。これらの 2 つの活動化グループにまたがって分割さ

れると、OPM の動作と ILE の動作を混合することになり、結果を予測するのが困難になることがあります。詳しくは、23 ページの『第 3 章 プログラムの作成方針』 または「*ILE 概念*」を参照してください。

付録 C. 作成コマンド

この節では、次のことに関する情報を提供します。

- CL コマンドの使用
- CRTBNDRPG の構文図と説明
- CRTRPGMOD の構文図と説明

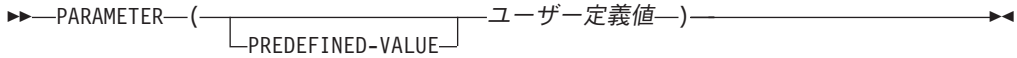
Create Program コマンドおよび Create Service Program コマンドについての詳細は、Web サイト http://www.ibm.com/eserver/iseries/infocenter で **iSeries 400 Information Center** の中の『*プログラミング*』カテゴリーの『*CL および API*』の節を参照してください。

CL コマンドの使用

制御言語 (CL) コマンド、**パラメーター**、および**キーワード**は大文字または小文字で入力することができます。構文図では、これらは大文字で示されます (例えば、PARAMETER、 PREDEFINED-VALUE)。変数は日本語で表されています (例えば、*「ユーザー定義値」など*)。変数とはユーザー定義の名前または値です。

構文図の解釈法

本書の構文図は、次の規則を使用しています。



構文図は左から右に上から下に、線に沿って読んでください。

▶▶— 記号は、構文図の始めを示します。

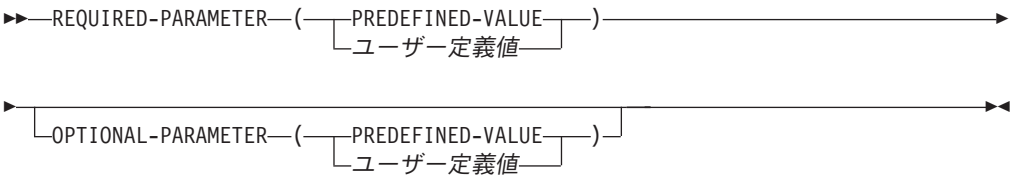
—▶◀ 記号は、構文図の終わりを示します。

—▶ 記号は、構文図が次の行に続くことを示します。

▶— 記号は、ステートメントが前の行から続いていることを示します。

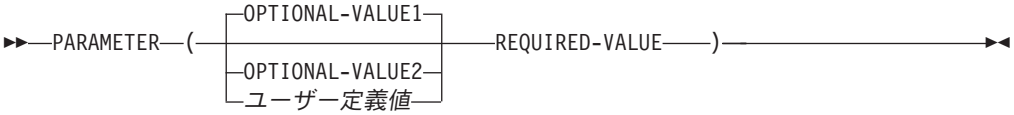
—(—)— 記号は、パラメーターまたは値を括弧で囲んで入力しなければならないことを示しています。

必須パラメーターは基準線に表示され、必ず入力しなければなりません。**任意指定パラメーター**は基準線の下に表示され、必ず入力する必要はありません。次のサンプルでは、REQUIRED-PARAMETER およびその値を入力しなければなりませんが、OPTIONAL-PARAMETER またはその値は入力の必要がありません。

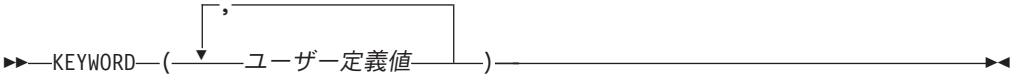


デフォルト値は基準線の上に表示され、入力の必要はありません。これらはパラメーターを指定しない場合に使用されます。次のサンプルでは、DEFAULT-VALUE、OTHER-PREDEFINED-VALUE を入力するか、あるいは何も入力しないことができます。何も入力しない場合には、DEFAULT-VALUE と見なされます。

任意指定の値はブランク行で示されます。ブランク行は最初のグループからの値 (OPTIONAL-VALUE1, OPTIONAL-VALUE2, *ユーザー定義値*) は入力する必要がないことを示しています。例えば、下の構文に基づき KEYWORD(REQUIRED-VALUE) を入力することができます。



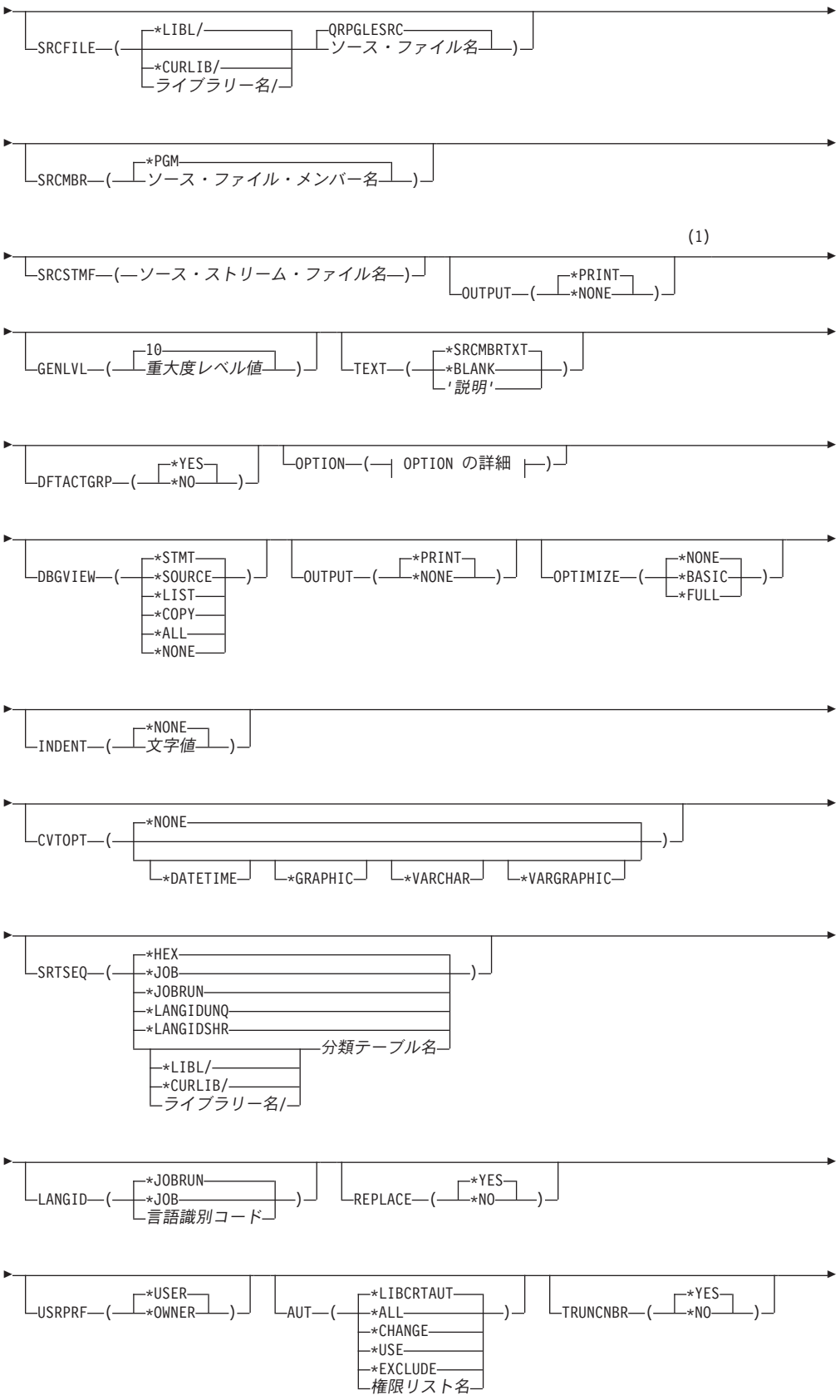
反復値は複数のパラメーターに指定することができます。次のサンプルの中のコンマ (,) は、各「*ユーザー定義値*」をコンマで区切らなければならないことを示します。

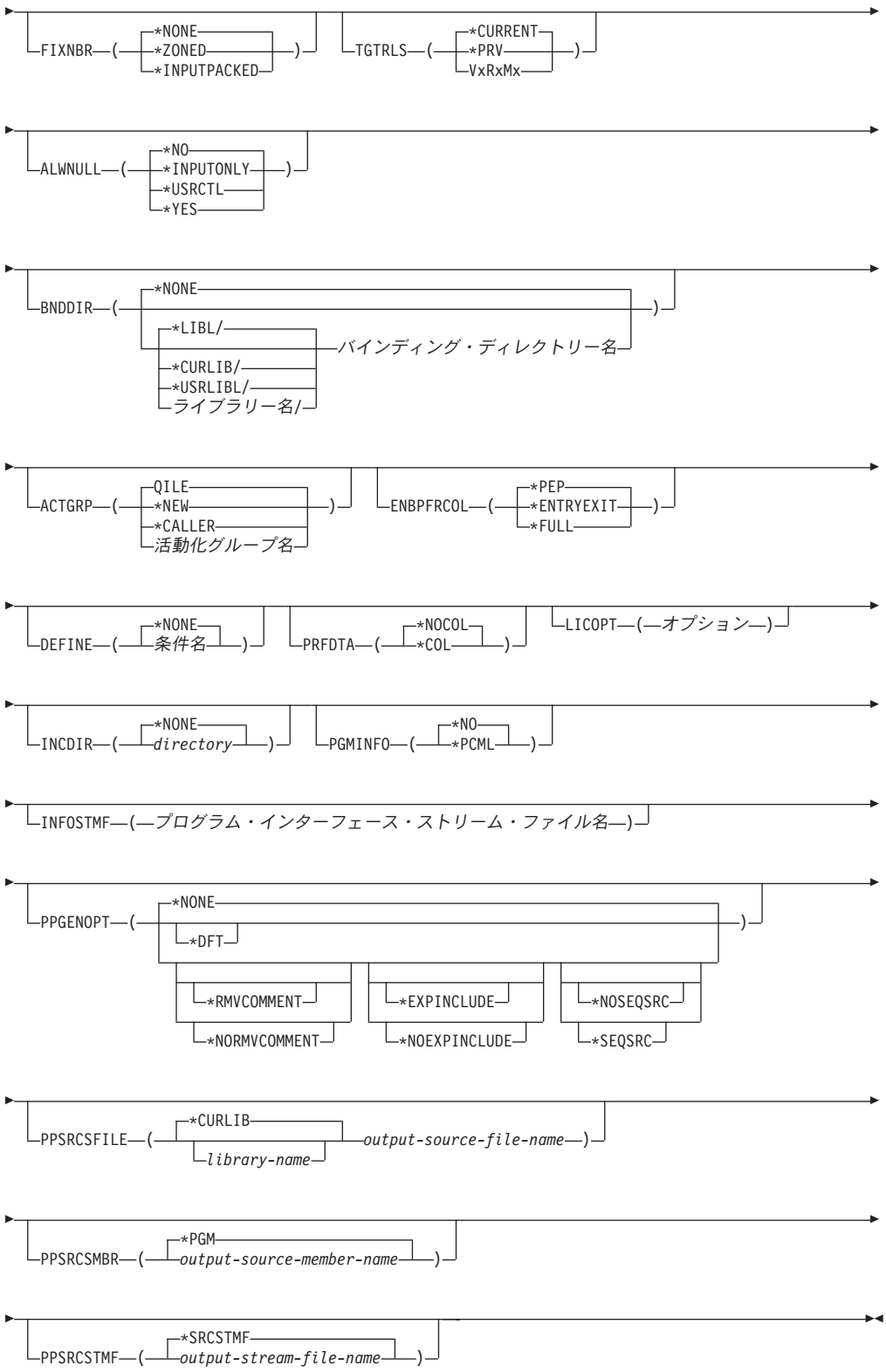


CRTBNDRPG コマンド

バインド RPG プログラムの作成 (CRTBNDRPG) コマンドは、ソース・コードから一時モジュール・オブジェクトを作成してから、プログラム・オブジェクトを作成することによって、RPG モジュールの作成 (CRTRPGMOD) コマンドおよびプログラムの作成 (CRTPGM) コマンドの結合されたタスクを実行します。プログラム・オブジェクトがいったん作成されると、CRTBNDRPG は作成した一時モジュールを削除します。CRTBNDRPG コマンドの構文図全体を下に示します。

ジョブ: B,I　プログラム: B,I　REXX: B,I　EXEC

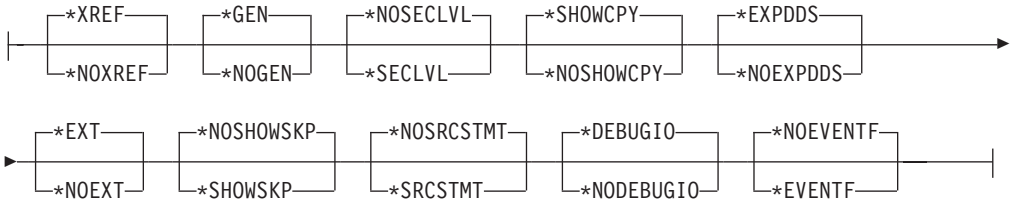

SRCFILE—(—*LIBL/—QRPGLESRC—)
*CURLIB/
ライブラリー名/
ソース・ファイル名
SRCMBR—(—*PGM—)
ソース・ファイル・メンバー名
(1)
SRCSTMF—(—ソース・ストリーム・ファイル名—)
OUTPUT—(—*PRINT—)
*NONE
GENLVL—(—10—)
重大度レベル値
TEXT—(—*SRCMBRTXT—)
*BLANK
'説明'
DFTACTGRP—(—*YES—)
*NO
OPTION—(— OPTION の詳細 —)
DBGVIEW—(—*STMT—)
*SOURCE
*LIST
*COPY
*ALL
*NONE
OUTPUT—(—*PRINT—)
*NONE
OPTIMIZE—(—*NONE—)
*BASIC
*FULL
INDENT—(—*NONE—)
文字値
CVTOPT—(—*NONE—)
*DATETIME
*GRAPHIC
*VARCHAR
*VARGRAPHIC
SRTSEQ—(—*HEX—)
*JOB
*JOBRUN
*LANGIDUNQ
*LANGIDSHR
分類テーブル名
*LIBL/
*CURLIB/
ライブラリー名/
LANGID—(—*JOBRUN—)
*JOB
言語識別コード
REPLACE—(—*YES—)
*NO
USRPRF—(—*USER—)
*OWNER
AUT—(—*LIBCRTAUT—)
*ALL
*CHANGE
*USE
*EXCLUDE
権限リスト名
TRUNCNBR—(—*YES—)
*NO



注:

1 この点より前のパラメーターは、すべて定位置形式によって指定することができます。

OPTION の詳細:



CRTBNDRPG コマンドの説明

CRTBNDRPG コマンドのパラメーター、キーワード、および変数を下にリストします。オンラインで同じ情報が使用可能です。コマンド入力行にコマンド名を入力し、 PF4 (プロンプト) キーを押してから、説明を表示したいパラメーター上で PF1 (ヘルプ) キーを押してください。

PGM

作成するプログラム・オブジェクト (*PGM) のプログラム名およびライブラリー名を指定します。プログラム名およびライブラリー名は iSeries の命名規則に適合していなければなりません。ライブラリーを指定しない場合には、作成されたプログラムは現行ライブラリーに保管されます。

***CTLSPEC**

コンパイル済みプログラムの名前は、制御仕様書の DFTNAME キーワードに指定された名前から取られます。制御仕様書にプログラム名が指定されていず、ソース・メンバーがデータベース・ファイルからのものである場合には、SRCMBR パラメーターで指定されたメンバー名がプログラム名として使用されます。ソースがデータベース・ファイルからのものでなければ、プログラム名にはデフォルトの値として RPGPGM が使用されます。

プログラム名

プログラム・オブジェクトの名前を入力します。

***CURLIB**

作成されたプログラム・オブジェクトは現行ライブラリーに保管されます。現行ライブラリーを指定していない場合には、QGPL が使用されます。

ライブラリー名

作成されたプログラム・オブジェクトを保管するライブラリーの名前を入力します。

SRCFILE

コンパイルする ILE RPG ソース・メンバーが入っているソース・ファイルの名前およびそのソース・ファイルが入っているライブラリーを指定します。推奨されるソース物理ファイルの長さは 112 桁です。順序番号と日付の 12 桁、コードの 80 桁、および注記の 20 桁です。これは、コンパイラー・リストに示されるソースの最大サイズです。

QRPGLESRC

デフォルトのソース・ファイル QRPGLESRC には、コンパイルされる ILE RPG ソース・メンバーが入ります。

ソース・ファイル名

コンパイルする ILE RPG ソース・メンバーが入っているソース・ファイルの名前を入力します。

***LIBL**
システムは、ライブラリー・リストを検索して、ソース・ファイルが保管されているライブラリーを見付けます。これはデフォルトです。

***CURLIB**
ソース・ファイルを検索するために現行ライブラリー・リストが使用されます。現行ライブラリーを指定していない場合には、QGPL が使用されます。

ライブラリー名
ソース・ファイルが保管されているライブラリーの名前を入力します。

SRCMBR
コンパイルする ILE RPG ソース・プログラムが入っているソース・ファイルのメンバーの名前を指定します。

***PGM**
PGM パラメーターによって指定された名前をソース・ファイル・メンバー名として使用します。コンパイル済みプログラム・オブジェクトはソース・ファイル・メンバーと同じ名前をもつことになります。 PGM パラメーターによってプログラム名が指定されていない場合には、コマンドは、ソース・メンバー名としてソース・ファイルに最初に作成または追加されたメンバーを使用します。

ソース・ファイル・メンバー名
ILE RPG ソース・プログラムを入れるメンバーの名前を入力します。

SRCSTMF
コンパイルする ILE RPG ソース・コードが入っているストリーム・ファイルのパス名を指定します。

パス名は、絶対名でも、相対修飾名でも構いません。絶対パス名の先頭は '/'、相対パス名の先頭は '/' 以外の文字です。

絶対修飾であれば、パス名としては完全です。相対修飾の場合、パス名にジョブの現行作業ディレクトリーを付け加えることによって、パス名が完全なものになります。

SRCMBR パラメーターおよび SRCFILE パラメーターを、 SRCSTMF パラメーターと一緒に指定することはできません。

GENLVL
プログラム・オブジェクトの作成を制御します。コンパイルで検出されたすべてのエラーが指定された生成重大度レベルより小さいか等しい場合に、プログラム・オブジェクトが作成されます。

10 10 より大きい重大度レベルのメッセージがある場合には、プログラム・オブジェクトは生成されません。

重大度レベル値
0 ～ 20 の範囲の数を入力します。重大度が 20 より大きいエラーの場合には、プログラム・オブジェクトは生成されません。

TEXT
プログラムとその機能を簡単に説明したテキストを入力することができます。このテキストはプログラム情報が表示されるたびに現れます。

***SRCMBRTXT**
ソース・メンバーのテキストが使用されます。

***BLANK**
テキストは表示されません。

'説明'
ソースの仕様の機能を簡単に説明したテキストを入力します。テキストは最大 50 桁とすることができ、アポストロフィで囲まれなければなりません。アポストロフィは、50 文字のストリングには含まれません。プロンプト画面でテキストを入力する場合には、アポストロフィは不要です。

DFTACTGRP
作成されたプログラムが常にデフォルトの活動化グループで実行されるかどうかを指定します。

***YES**
このプログラムが呼び出された場合には、常にデフォルトの活動化グループで実行されます。デフォルトの活動化グループとは、すべてのオリジナル・プログラム・モデル (OPM) プログラムが実行される活動化グループのことです。

DFTACTGRP(*YES) を指定することによって ILE RPG プログラムは、一時変更の有効範囲、オープンの有効範囲、および RCLRSC の領域で OPM プログラムと類似した動作をすることができるようになります。

プログラムが DFTACTGRP(*YES) で作成された時には、ILE 静的バインドは使用できません。これは、このプログラムの作成中に BNDDIR または ACTGRP パラメーターを使用できないことを意味します。さらに、ソースの呼び出し命令は、プロシージャーではなく、プログラムを呼び出さなければなりません。

DFTACTGRP(*YES) は、プログラム単位でアプリケーションを ILE RPG に転送する場合に役立ちます。

***NO**
プログラムは、ACTGRP パラメーターによって指定された活動化グループと関連付けられます。 *NO を指定した時には、静的バインドを使用することができます。

ACTGRP(*CALLER) を指定し、このプログラムがデフォルトの活動化グループで実行中のプログラムによって呼び出された場合には、このプログラムは、ファイルの共用、ファイルの有効範囲、および RCLRSC の領域で ILE の意味構造にしたがって動作します。

DFTACTGRP(*NO) は、例えば、名前付きの活動化グループで実行したり、サービス・プログラムにバインドするなど、ILE 概念を利用したい時に便利です。

OPTION
ソース・メンバーのコンパイル時に使用するオプションを指定します。任意のオプションまたはすべてのオプションを任意の順序で指定することができます。オプションは 1 つ以上のブランク・スペースで区切ってください。オプションを複数回指定した場合には、最後に指定されたオプションが使用されます。

***XREF**
ソース・メンバーの相互参照表 (該当する場合) を作成します。

***NOXREF**
相互参照表は作成されません。

***GEN**
コンパイラーによって戻された最高の重大度レベルが GENLVL オプションで指定された重大度を超えなかった場合に、プログラム・オブジェクトを作成します。

***NOGEN**
プログラム・オブジェクトは作成されません。

***NOSECLVL**
第 1 レベル・メッセージ・テキストの行に続けて、第 2 レベル・メッセージ・テキストを印刷しません。

***SECLVL**
メッセージ要約セクションで、第 1 レベル・メッセージ・テキストの行に続けて、第 2 レベル・メッセージ・テキストを印刷します。

***SHOWCPY**
/COPY コンパイラー指示ステートメントによって組み込まれたメンバーのソース・レコードを示します。

***NOSHOWCPY**
/COPY コンパイラー指示ステートメントによって組み込まれたメンバーのソース・レコードを示しません。

***EXPDDS**
リストの中の外部記述ファイルの拡張を示し、またキー・フィールド情報を表示します。

***NOEXPDDS**
リストの中の外部記述ファイルの拡張を示さず、またキー・フィールド情報を表示しません。

***EXT**
コンパイル時に参照された外部プロシージャーおよびフィールドのリストをリスト上に示します。

***NOEXT**
コンパイル時に参照された外部プロシージャーおよびフィールドのリストをリスト上に示しません。

***NOSHOWSKP**
リストのソース部分に無視されたステートメントを示しません。/IF、/ELSEIF または /ELSE 指示の結果、コンパイラーはステートメントを無視します。

***SHOWSKP**
コンパイラーがスキップしたかどうかに関係なく、リストのソース部分にすべてのステートメントを示します。

***NOSRCSTMT**
リスト内の行番号は、順番に割り当てられます。これらの番号は、ステート

メント番号を使用してデバッグを行うときに使用されます。行番号は、リストの左端の桁に示されます。ソース ID および SEU 順序番号は、リストの右端の 2 桁に示されます。

***SRCSTMT**
デバッグ用のステートメント番号は、以下のように SEU 順序番号とソース ID を使用して生成されます。

```
ステートメント番号 = ソース ID * 1000000 + SEU 順序番号
```

SEU 順序番号は、リストの左端の桁に示されます。ステートメント番号は、リストの右端の桁に示されます。これらの番号は、ステートメント番号を使用してデバッグを行うときに使用されます。

注: OPTION(*SRCSTMT) が指定されている場合には、ソース・ファイル内のすべての順序番号に、有効な数値が入っている必要があります。同じソース・ファイル内に重複する順序番号がある場合、デバッガーの動作は予測不能であり、診断メッセージまたは相互参照項目のステートメント番号は意味をもたないことがあります。

***DEBUGIO**
すべての入出力仕様書について停止点を生成します。

***NODEBUGIO**
入出力仕様書について停止点を生成しません。

***NOEVENTF**
連携開発環境/400 (CODE/400) で使用するイベント・ファイルを作成しません。CODE/400 は、CODE/400 エディターで統合されたエラー・フィードバックを提供するために、このファイルを使用します。イベント・ファイルは通常、CODE/400 内からモジュールまたはプログラムを作成した時に作成されます。

***EVENTF**
連携開発環境/400 (CODE/400) で使用するイベント・ファイルを作成します。イベント・ファイルは、作成されたモジュールまたはプログラム・オブジェクトが保管されるライブラリー中のファイル EVFEVENT のメンバーとして作成されます。ファイル EVFEVENT が存在していない場合には、自動的に作成されます。イベント・ファイル・メンバー名は、作成されるオブジェクトの名前と同じです。

CODE/400 は、CODE/400 エディターで統合されたエラー・フィードバックを提供するために、このファイルを使用します。イベント・ファイルは通常、CODE/400 内からモジュールまたはプログラムを作成した時に作成されます。

DBGVIEW
コンパイル済みプログラムに使用可能なデバッグ・レベル、およびソース・レベル・デバッグに使用可能なソース・ビューを指定します。

***STMT**
コンパイラー・リスト行番号またはステートメント番号を使用してプログラム・オブジェクトをデバッグできるようにします。OPTION(*NOSRCSTMT) が指定された場合、行番号は、コンパイラー・リストのソース・セクション

の左端の桁に示されます。OPTION(*SRCSTMT) が指定された場合、ステートメント番号は、コンパイラーのソース・セクションの右端の桁に示されます。

***SOURCE**
コンパイル済みプログラム・オブジェクトをデバッグするための、ソース・ビューを生成します。このビューは、ルート・ソース・メンバーが DDM ファイルの場合には使用可能ではありません。また、コンパイルした後、プログラムをデバッグする前にソース・メンバーに対して変更が行われた場合には、これらのソース・メンバーのビューは使えないことがあります。

***LIST**
コンパイル済みプログラム・オブジェクトをデバッグするための、リスト・ビューを生成します。リスト・ビューに入っている情報は、OPTION パラメーターに *SHOWCPY、*EXPDDS、および *SRCSTMT が指定されているかどうかによって異なります。

注: リスト・ビューには、字下げオプションを使用して要求した字下げは示されません。

***COPY**
コンパイル済みプログラム・オブジェクトをデバッグするためのソースおよびコピー・ビューを生成します。このオプションのソース・ビューは、*SOURCE オプションで生成されたものと同じソース・ビューです。コピー・ビューは、すべての /COPY ソース・メンバーを含むデバッグ・ビューです。これらのビューは、ルート・ソース・メンバーが DDM ファイルの場合には使用可能ではありません。また、コンパイルした後、プログラムをデバッグする前にソース・メンバーに対して変更が行われた場合には、これらのソース・メンバーのビューは使えないことがあります。

***ALL**
コンパイル済みプログラム・オブジェクトをデバッグするためのリスト、ソース、およびコピー・ビューを生成します。リスト・ビューに入っている情報は、OPTION パラメーターに *SHOWCPY、*EXPDDS、および *SRCSTMT が指定されているかどうかによって異なります。

***NONE**
コンパイル済みプログラム・オブジェクトをデバッグするためのすべてのデバッグ・オプションを使用禁止にします。

OUTPUT
コンパイラー・リストを生成するかどうかを指定します。

***PRINT**
ILE RPG プログラム・ソースおよびすべてのコンパイル時のメッセージから成る、コンパイラー・リストを作成します。このリストに含まれる情報は、OPTION パラメーターに *XREF、*SECLVL、*SHOWCPY、*EXPDDS、*EXT、*SHOWSKP、および *SRCSTMT が指定されているかどうかによって異なります。

***NONE**
コンパイラー・リストを生成しません。

OPTIMIZE
プログラムの最適化のレベル (ある場合) を指定します。

***NONE**
生成されるコードは最適化されません。これは変換時間のためには、最も速い方法です。これによって、デバッグ・モードの間に変数を表示して変更することができます。

***BASIC**
生成されるコードに対して一部の最適化が実行されます。これによって、プログラムがデバッグ・モードの間にユーザー変数を表示することができますが、変更することはできません。

***FULL**
最も効率的なコードを生成する最適化です。変換時間は最も長くなります。デバッグ・モードでユーザー変数を変更することはできませんが、表示することはできます。ただし、提示される値は現在の値ではない可能性があります。

INDENT
読みやすくするために、ソース・リスト中の構造化命令を字下げする必要があるかどうかを指定します。また、構造化命令文節にマークを付けるために使用される文字を指定します。

注: ここで要求する字下げは、DBGVIEW(*LIST) を指定した時に作成されるリスト・デバッグ・ビューには反映されません。

***NONE**
構造化命令はソース・リスト中で字下げされません。

文字値
構造化命令文節の場合に、ソース・リストが字下げされます。ステートメントおよび文節の配置は選択した文字を使用して印が付けられます。最高 2 桁の長さの任意の文字ストリングを選択することができます。ストリングにブランク文字を使用したい場合には、それを単一引用符で囲む必要があります。

注: プログラムにエラーがある場合には、字下げが予定どおりに出ないことがあります。

CVTOPT
外部記述データベース・ファイルから検索される日付、時刻、タイム・スタンプ、図形データ・タイプ、および可変長データ・タイプを ILE RPG コンパイラーが処理する方法を指定します。

***NONE**
可変長データベースのデータ・タイプを無視し、固有の日付、時刻、タイム・スタンプ、および図形データ・タイプを使用します。

***DATETIME**
日付、時刻、およびタイム・スタンプ・データ・タイプを固定長文字フィールドとして宣言することを指定します。

***GRAPHIC**
2 バイト文字セット (DBCS) 図形データ・タイプを固定長文字フィールドとして宣言することを指定します。

***VARCHAR**
可変長文字データ・タイプを固定長文字フィールドとして宣言することを指定します。

***VARGRAPHIC**
可変長文字 2 バイト文字セット (DBCS) 図形データ・タイプを固定長文字フィールドとして宣言することを指定します。

SRTSEQ
ILE RPG ソースで使用する分類順序テーブルを指定します。

***HEX**
分類順序テーブルを使用しません。

***JOB**
*PGM の作成時のジョブの SRTSEQ 値を使用します。

***JOBRUN**
*PGM の実行時のジョブの SRTSEQ 値を使用します。

***LANGIDUNQ**
固有の重み付けテーブルを使用します。この特殊値は、適切な分類順序テーブルを決めるために LANGID パラメーターと一緒に使用されます。

***LANGIDSHR**
共有の重み付けテーブルを使用します。この特殊値は、適切な分類順序テーブルを決めるために LANGID パラメーターと一緒に使用されます。

分類テーブル名
プログラムで使用する分類順序テーブルの修飾名を入力します。

***LIBL**
システムは、ライブラリー・リストを検索して、分類順序テーブルが保管されているライブラリーを見付けます。

***CURLIB**
分類順序テーブルを検索するために現行ライブラリー・リストが使用されます。現行ライブラリーを指定していない場合には、QGPL が使用されます。

ライブラリー名
分類順序テーブルが保管されているライブラリーの名前を入力します。

SRTSEQ パラメーターおよび LANGID パラメーターを使用して代替照合順序を決定したい場合は、制御仕様書に ALTSEQ(*EXT) も指定することが必要です。

LANGID
分類順序が *LANGIDUNQ および *LANGIDSHR の時に使用する、言語識別コードを指定します。 LANGID パラメーターは分類順序テーブルを選択するために、SRTSEQ パラメーターと一緒に使用されます。

***JOBRUN**
RPG プログラムの実行時のジョブと関連した LANGID 値を使用します。

***JOB**
RPG プログラムの作成時のジョブと関連した LANGID 値を使用します。

言語識別コード
指定された言語識別コードを使用します。(例えば、フランス語の FRA およびドイツ語の DEU など。)

REPLACE
同じ名前のプログラムが指定された (暗黙の) ライブラリーに既に存在している時に、新しいプログラムを作成するかどうかを指定します。CRTBNDRPG コマンドの処理中に作成される中間モジュールは、REPLACE 仕様書に対する対象とはならず、QTEMP ライブラリーに対する暗黙の REPLACE(*NO) があるものと見なされます。CRTBNDRPG コマンドが処理を完了した時に、中間モジュールは削除されます。

***YES**
指定したライブラリーに新しいプログラムが作成されます。指定したライブラリーにある同じ名前の既存のプログラムは、ライブラリー QRPLOBJ に移動されます。

***NO**
指定したライブラリーに同じ名前のプログラムが既に存在している場合には、新しいプログラムは作成されません。既存のプログラムは置き換えられず、メッセージが表示され、コンパイルは停止します。

USRPRF
作成されたプログラム・オブジェクトを実行するユーザー・プロファイルを指定します。プログラム所有者またはプログラム・ユーザーのプロファイルは、そのプログラムを実行したり、そのプログラムでどのオブジェクトが使えるか (各オブジェクトに対してそのプログラムがもつ権限を含む) を制御したりするために使用されます。このパラメーターは、そのプログラムが既に存在している場合には更新されません。この値を変更するためには、そのプログラムを削除し、新しい値を用いて再コンパイルします (あるいは、構成要素 *MODULE オブジェクトが存在している場合には、CRTPGM コマンドを呼び出す選択をすることができます)。

***USER**
プログラムのユーザーのユーザー・プロファイルのもとで、プログラムが実行されます。

***OWNER**
プログラムのユーザーと所有者の両方のユーザー・プロファイルのもとで、プログラムが実行されます。両方のユーザー・プロファイルのオブジェクト権限をまとめて使用し、プログラムの実行時にオブジェクトを検索しアクセスします。プログラム中で作成されたオブジェクトは、プログラムのユーザーによって所有されます。

AUT
オブジェクトに対して特定権限をもたないユーザー、権限リスト中にないユーザー、および属するユーザー・グループがオブジェクトに対して特定権限をもたないユーザーに与えられる権限を指定します。プログラムを作成した後、CL コマンドのオブジェクト権限認可 (GRTOBJAUT) またはオブジェクト権限取り消し (RVKOBJAUT) によって、すべてのユーザーまたは指定されたユーザーに対す

る権限を変更することができます。これらのコマンドについての詳細は、 Web サイト http://www.ibm.com/eserver/iseries/infocenter で **iSeries 400 Information Center** の中の『*プログラミング*』カテゴリーの『*CL および API*』の節を参照してください。

***LIBCRTAUT**
オブジェクトの共通認可は、目的ライブラリー (そのオブジェクトが入っているライブラリー) の CRTAUT キーワードから取られます。この値は、オブジェクトが作成される時に決められます。作成後にライブラリーの CRTAUT 値が変更された場合には、新しい値は既存のオブジェクトに影響しません。

***ALL**
所有者に限定された操作、または権限リスト管理権限により制御される操作を除く、プログラム・オブジェクトに対するすべての操作の権限を提供します。ユーザーは、プログラム・オブジェクトの存在を制御し、その機密保護を指定し、これを変更し、これに対する基本機能を実行できますが、所有権を移すことはできません。

***CHANGE**
すべてのデータ権限および、所有者に限定された操作またはオブジェクト権限およびオブジェクト管理権限によって制御される操作を除くプログラム・オブジェクトに対するすべての操作を実行する権限を提供します。ユーザーはオブジェクトを変更し、これに基本機能を実行することができます。

***USE**
オブジェクトの処理権限および読み取り権限、すなわちプログラム・オブジェクトに対する基本操作権限を提供します。ユーザーは、オブジェクトを変更することはできません。

***EXCLUDE**
ユーザーは、オブジェクトにアクセスすることはできません。

権限リスト名
ユーザーの権限リストの名前およびプログラムに追加する権限を入力します。プログラム・オブジェクトは、この権限リストによって保護され、プログラム・オブジェクトの共通認可は *AUTL に設定されます。CRTBNDRPG コマンドを出す時には、この権限リストはシステム上に存在していなければなりません。

注: AUT パラメーターを使用して、システムの機密保護要件を表します。使用可能な機密保護機能の詳細は、「*iSeries 機密保護解説書*」に説明があります。

TRUNCNBR
プログラムの実行で数値オーバーフローが起こった時に、切り捨てられた値が結果のフィールドに転送されるか、あるいはエラーが生成されるかどうかを指定します。

注: TRUNCNBR オプションは、式の中で実行される演算には適用されません (式は拡張演算項目 2 フィールドに入っています)。これらの演算に対してオーバーフローが起こった場合には、常にエラーが起こります。 さらに、

整数または符号のないフィールドに割り当てられた値が範囲外になる命令の場合には、オーバーフローは常に通知されます。

***YES**
数値オーバーフローを無視し、切り捨てられた値を結果のフィールドに転送します。

***NO**
数値オーバーフローが検出された時には、エラー・コード RNX0103 の実行時エラーが生成されます。

FIXNBR
無効な 10 進数データをコンパイラーで修正するかどうかを指定します。

***NONE**
無効な 10 進数データが実行時に使われたときは、データ・エラーを起こすように指示します。

***ZONED**
無効なゾーン 10 進数はパック・データに変換される時にコンパイラーによって修正されます。数値フィールドにあるブランクはゼロとして取り扱われます。各 10 進桁の数字の妥当性が検査されます。10 進桁の数字が正しくない場合には、ゼロに置き換えられます。符号が正しくない場合には、符号は強制的に正符号コードの 16 進数 'F' に変更されます。符号が有効な場合には、符号はそれぞれ該当する正符号の 16 進数 'F' または負符号の 16 進数 'D' に変更されます。結果のパック・データが正しくない場合には、修正されません。

***INPUTPACKED**
入力仕様書の処理中に無効なパック 10 進数に出会ったときは、内部変数をゼロに設定するように指示します。

TGTRLS
作成するオブジェクトを使おうとするオペレーティング・システムのリリース・レベルを指定します。*CURRENT と *PRV 値が与えられた例で、*目的リリース* 値を指定する時、フォーマット VxRxMx を使ってリリースを指定しています。ここで Vx はバージョン、Rx はリリース、Mx は修正レベルを表します。例えば、V2R3M0 は、バージョン 2、リリース 3、モディフィケーション・レベル 0 です。

このパラメーターに有効な値は、リリースごとに変わります。指定できる値は次のとおりです。

***CURRENT**
オブジェクトは、システムで現在稼働中のオペレーティング・システムのリリースを使用することになります。例えば、システムで V2R3M5 が稼働中の場合、*CURRENT は V2R3M5 が導入されているシステムでオブジェクトを使用することを意味します。また、導入されているオペレーティング・システム以降のリリースのシステムでもこのオブジェクトを使用することができます。

注: V2R3M5 がシステム上で稼働し、 V2R3M0 がインストールされているシステムで使用することを目的としている場合は、TGTRLS(*CURRENT) ではなく、TGTRLS(V2R3M0) を指定してください。

***PRV**
オブジェクトは、前のリリースのモディフィケーション・レベル 0 のオペレーティング・システムを使用して実行されます。例えば、システムで V2R3M5 が稼働中の場合には、*PRV は V2R2M0 が導入されているシステムでオブジェクトを使用することを意味します。また、導入されているオペレーティング・システム以降のリリースのシステムでもこのオブジェクトを使用することができます。

ターゲット・リリース
VxRxMx の形式でリリースを指定します。指定されたリリースまたはそれ以降のリリースのオペレーティング・システムが導入されているシステムでオブジェクトを使用することができます。

有効な値は、現行バージョン、リリース、およびモディフィケーション・レベルによって異なり、新しいリリースごとに変わります。このコマンドでサポートされる一番古いリリースよりももっと前のターゲット・リリース を指定すると、サポートされる一番古いリリースを知らせるエラー・メッセージが出されます。

注: 現行バージョンのコマンドは、コマンドの前のリリースでは使用できないオプションをサポートすることがあります。このコマンドを、前のリリースで使用することになっているオブジェクトを作成するのに使用すると、そのリリースに該当するコンパイラーによって処理され、サポートされていないオプションは認識されません。コンパイラーは、処理できないオプションに関して、必ずしも警告を発令しません。

ALWNULL
ILE RPG プログラムが、外部記述データベース・ファイルからの null 可能フィールドをもつレコードをどう使えるようにするかを指定します。

***NO**
ILE RPG プログラムが、外部記述ファイルからの null フィールドをもつレコードを処理しないことを指定します。null を含んでいるレコードを取り出そうとした場合には、ILE RPG プログラムはそのレコードのデータをアクセスできず、データ・マッピング・エラーが起こります。

***INPUTONLY**
ILE RPG プログラムが、外部記述の入力専用データベース・ファイルから、 null を含む null 可能フィールドをもつレコードを正常に読み取ることができることを指定します。null を含んでいるレコードを取り出そうとした場合には、データ・マッピング・エラーは起こらず、null を含んでいるフィールドにデータベースのデフォルト値が入れられます。このモジュールは次のいずれも実行不可能です。

- null 可能キー・フィールドを使用する
- null 可能フィールドを含むレコードを作成または更新する
- モジュールの実行中に null 可能フィールドが実際に null であるかどうかを判別する

- null 可能フィールドが null であるように設定する

***USRCTL**
ILE RPG モジュールが、外部記述データベース・ファイルからの null をもつレコードの読み取り、書き込み、および更新を行うことができることを指定します。null キーのあるレコードは、キー付き命令を使用すれば取り出すことができます。モジュールは、null 可能フィールドが実際に null であるかどうかを判別し、出力または更新の場合に null 可能フィールドが null であるように設定することができます。 null を含むフィールドがモジュール内で正常に使用されるように保証するのはプログラマーの責任です。

***YES**
*INPUTONLY と同じ。

BNDDIR
記号の解決に使用するバインディング・ディレクトリーのリストを指定します。

***NONE**
バインディング・ディレクトリーは指定されません。

バインディング・ディレクトリー名
記号の解決に使用するバインディング・ディレクトリーの名前を指定します。

ディレクトリー名は、次のライブラリー値の 1 つで修飾することができます。

***LIBL**
システムは、ライブラリー・リストを検索して、バインディング・ディレクトリーが保管されているライブラリーを見付けます。

***CURLIB**
ジョブの現行ライブラリーが検索されます。ジョブの現行ライブラリーとしてライブラリーを指定していない場合には、ライブラリー QGPL が使用されます。

***USRLIBL**
ジョブのライブラリー・リストのユーザー部分にあるライブラリーだけが検索されます。

ライブラリー名
検索するライブラリーの名前を指定します。

ACTGRP
このプログラムが呼び出される時に関連付けられる活動化グループを指定します。

QILE
このプログラムが呼び出された場合には、指定の活動化グループ QILE で活動化されます。

***NEW**
このプログラムが呼び出された場合には、新しい活動化グループで活動化されます。

***CALLER**
このプログラムが呼び出された場合には、呼び出し元の活動化グループで活動化されます。

活動化グループ名
このプログラムが呼び出される時に使用される活動化グループの名前を指定します。

ENBPFRCOL
パフォーマンス収集を使用可能にするかどうかを指定します。

***PEP**
プログラム入力プロシージャーの入り口および出口でのみ、パフォーマンス統計情報が収集されます。これは、プログラム内のモジュールのメイン・プロシージャーではなく、そのプログラムの実際のプログラム入力プロシージャーに適用されます。これはデフォルトです。

***NEW**
このプログラムが呼び出された場合には、新しい活動化グループで活動化されます。

***ENTRYEXIT**
プログラムのすべてのプロシージャーの入り口および出口で、パフォーマンス統計情報が収集されます。

***FULL**
すべてのプロシージャーの入り口および出口で、パフォーマンス統計情報が収集されます。また、外部プロシージャーへの呼び出しごとにその呼び出しの前後にも統計情報が収集されます。

DEFINE
コンパイルの開始前に定義する条件名を指定します。パラメーター DEFINE(条件名) を使うのと、ソース・ファイルの最初の行に /DEFINE 条件名宣言をコーディングするのとは同じことです。

***NONE**
条件名は定義されません。これはデフォルトです。

条件名
最高 32 の条件名を指定することができます。名前ごとに最高 50 桁が可能です。条件名は、コンパイルの開始時に定義されたものと見なされます。

PRFDTA
プログラムのプロファイル作成データ属性を指定します。プログラム・プロファイル作成とは拡張最適化技法で、プロシージャー内のプロシージャーとコードを統計データ (プロファイル作成データ) に基づいてリオーダーするために使われます。

***NOCOL**
このプログラムはプロファイル作成データを集めるために使うことができません。これはデフォルトです。

***COL**
プログラムはプロファイル作成データを集めるために使うことができます。

*COL が指定できるのは、モジュールの最適化レベルが *FULL である場合およびターゲット・リリース *CURRENT でコンパイルする場合だけに限られます。

LICOPT

1 つ以上のライセンス内部コード・コンパイル時オプションを指定します。このパラメーターにより個別のコンパイル時オプションを選択することができます。このパラメーターは、コンパイラー・オプションの各選択タイプの利点と欠点を理解している上級プログラマーを対象にしています。

INCDIR

コンパイラーがコピー・ファイルを検索する場合に使用する検索パスに追加する 1 つ以上のディレクトリーを指定します。

コンパイラーは、現行ディレクトリーを参照することによって、ソース・プログラム内の相対的に指定されたコピー・ファイルを解決できないと、ここで指定されたディレクトリーを検索します。コピー・ファイルが現行ディレクトリーまたは INCDIR パラメーターで指定されたディレクトリー内で見つからないと、RPGINCDIR 環境変数で指定されたディレクトリーを検索し、その次にメイン・ソース・ファイルが入っているディレクトリーを検索します。

***NONE**

ディレクトリーは指定されていません。

ディレクトリー

コピー・ファイルの検索先として最大 32 のディレクトリーを指定します。

PGMINFO

ストリーム・ファイル内にプログラム・インターフェース情報を生成するかどうかを指定します。

指定できる値は次のとおりです。

***NO**

このオプションは、プログラム・インターフェース情報を生成しないことを示すデフォルト値を指定します。

***PCML**

PCML (プログラム呼び出しマークアップ言語) をストリーム・ファイルに生成することを指定します。

PCML を生成すれば、この RPG モジュールでの Java メソッドによるプロシージャー呼び出しが簡単になり、 Java コードを削減することができます。 INFOSTMF オプションで、生成された PCML を入れるストリーム・ファイルの名前を指定します。

INFOSTMF

PGMINFO オプションで指定された、生成されたプログラム・インターフェース情報を入れるストリーム・ファイルのパス名を指定します。

パス名は、絶対名でも、相対修飾名でも構いません。絶対パス名の先頭は '/'、相対パス名の先頭は '/' 以外の文字です。

絶対修飾であれば、パス名としては完全です。相対修飾の場合、パス名にジョブの現行作業ディレクトリーを付け加えることによって、パス名が完全なものになります。

このパラメーターは、 PGMINFO パラメーターの値が *NO 以外の場合しか指
定できません。

PPGENOPT
ソース・コードのコンパイル時に使用するプリプロセッサー生成オプションを指定します。

以下のオプションを指定できます。

***NONE**
ソース・ファイルに対してコンパイラー全体を実行します。プリプロセッサーの出力をファイルにコピーしません。

***DFT**
入力ソースに対してプリプロセッサーを実行します。プリプロセッサー出力を生成するためのオプションとして *RMVCOMMENT、*EXPINCLUDE および *NOSEQSRC が使用されます。出力ソース・ファイルおよびメンバーを指定するには PPSRCFILE および PPSRCMBR を使用し、プリプロセッサー出力を含むストリーム・ファイルを指定するには PPSRCSTMF を使用します。

***RMVCOMMENT**
コメント、ブランク行、およびほとんどのディレクティブをプリプロセス中に除去します。 RPG 指定およびその指定を正しく解釈するために必要なディレクティブのみを保持します。

***NORMVCOMMENT**
プリプロセス中にコメント、ブランク行およびリスト制御ディレクティブ (/EJECT、/TITLE など) を保持します。プリプロセス中に、ソース制御ディレクティブ (/COPY、/IF など) をコメントに変換します。

***EXPINCLUDE**
/INCLUDE ディレクティブを、生成された出力ファイル内に展開します。

***NOEXPINCLUDE**
/INCLUDE ディレクティブを未変更のまま、生成された出力ファイルに出力します。

注: /COPY ディレクティブは常に展開されます。

***SEQSRC**
PPSRCFILE を指定すると、生成された出力メンバーには 000001 で始まり 000001 ずつ増加するシーケンス番号が付きます。

***NOSEQSRC**
PPSRCFILE を指定すると、生成された出力メンバーに、プリプロセッサーが読み取ったオリジナルのソースと同一のシーケンス番号が付きます。

PPSRCFILE
プリプロセッサー出力用のソース・ファイル名とライブラリーを指定します。

ソース・ファイル名
プリプロセッサー出力用のソース・ファイルの名前を指定します。

以下のライブラリー値を指定できます。

***CURLIB**
プリプロセッサー出力は現行ライブラリー内に作成されます。ジョブに現行ライブラリーがない場合、プリプロセッサー出力ファイルは QGPL ライブラリー内に作成されます。

ライブラリー名
プリプロセッサー出力用のライブラリーの名前を指定します。

PPSRCMBR
プリプロセッサー出力用のソース・ファイル・メンバーの名前を指定します。

***PGM**
PGM パラメーターで指定された名前はプリプロセッサーの出力メンバー名として使用されます。

メンバー名
プリプロセッサー出力用のメンバーの名前を指定します。

PPSRCSTMF
プリプロセッサー出力用のストリーム・ファイルのパス名を指定します。

***SRCSTMF**
SRCSTMF パラメーターで指定されたパス名はプリプロセッサーの出力パス名として使用されます。このファイルには拡張子「.i」が付きます。

'パス名'
プリプロセッサー出力ストリーム・ファイル用のパス名を指定します。

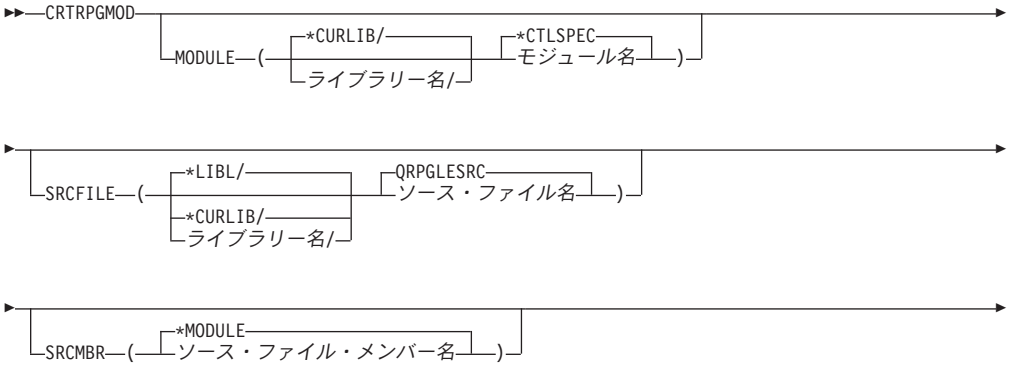
このパス名は絶対パスまたは相対パスのどちらでも構いません。絶対パス名の先頭は '/'、相対パス名の先頭は '/' 以外の文字です。

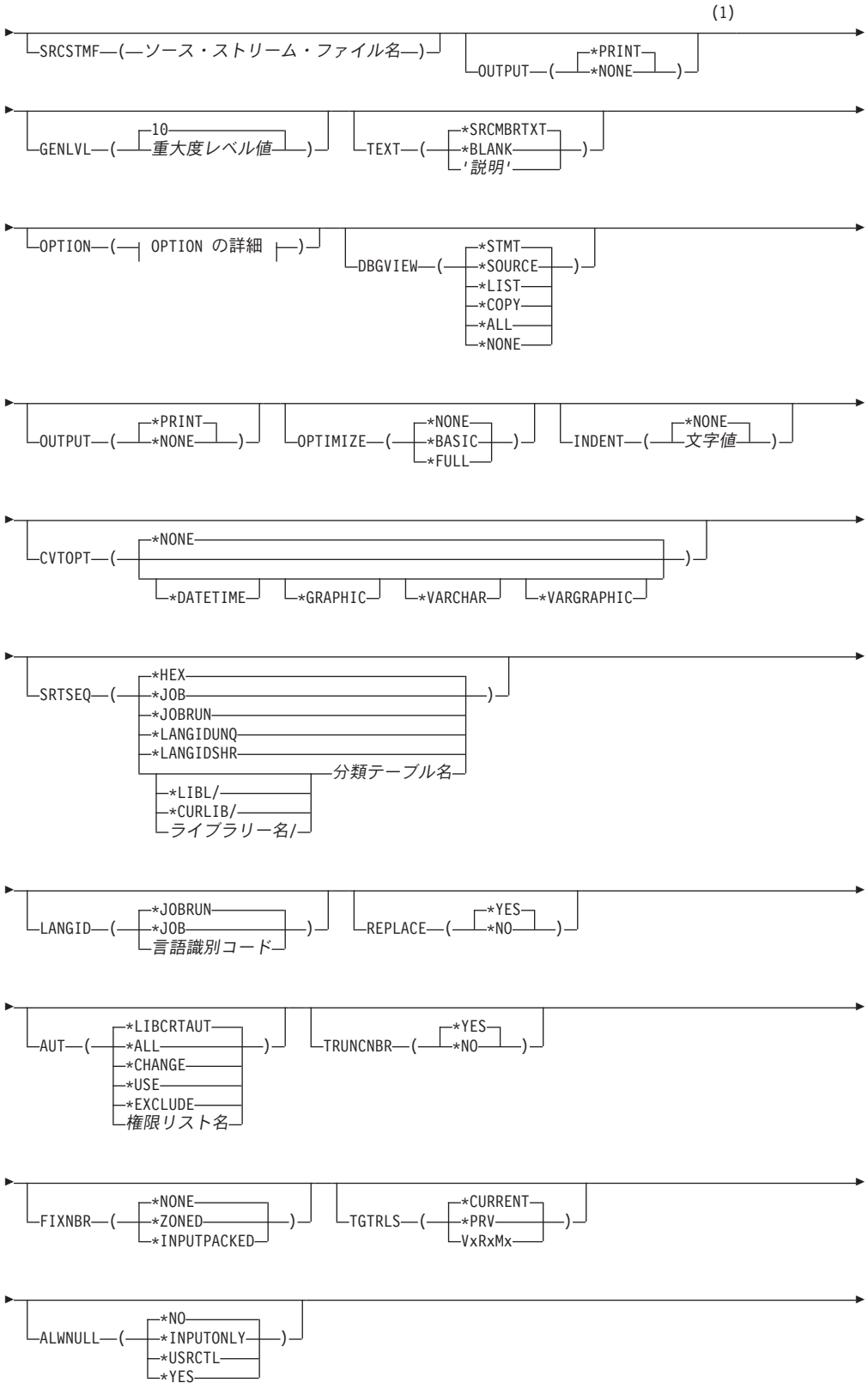
絶対修飾であれば、パス名としては完全です。相対修飾の場合、パス名にジョブの現行作業ディレクトリーを付け加えることによって、パス名が完全なものになります。

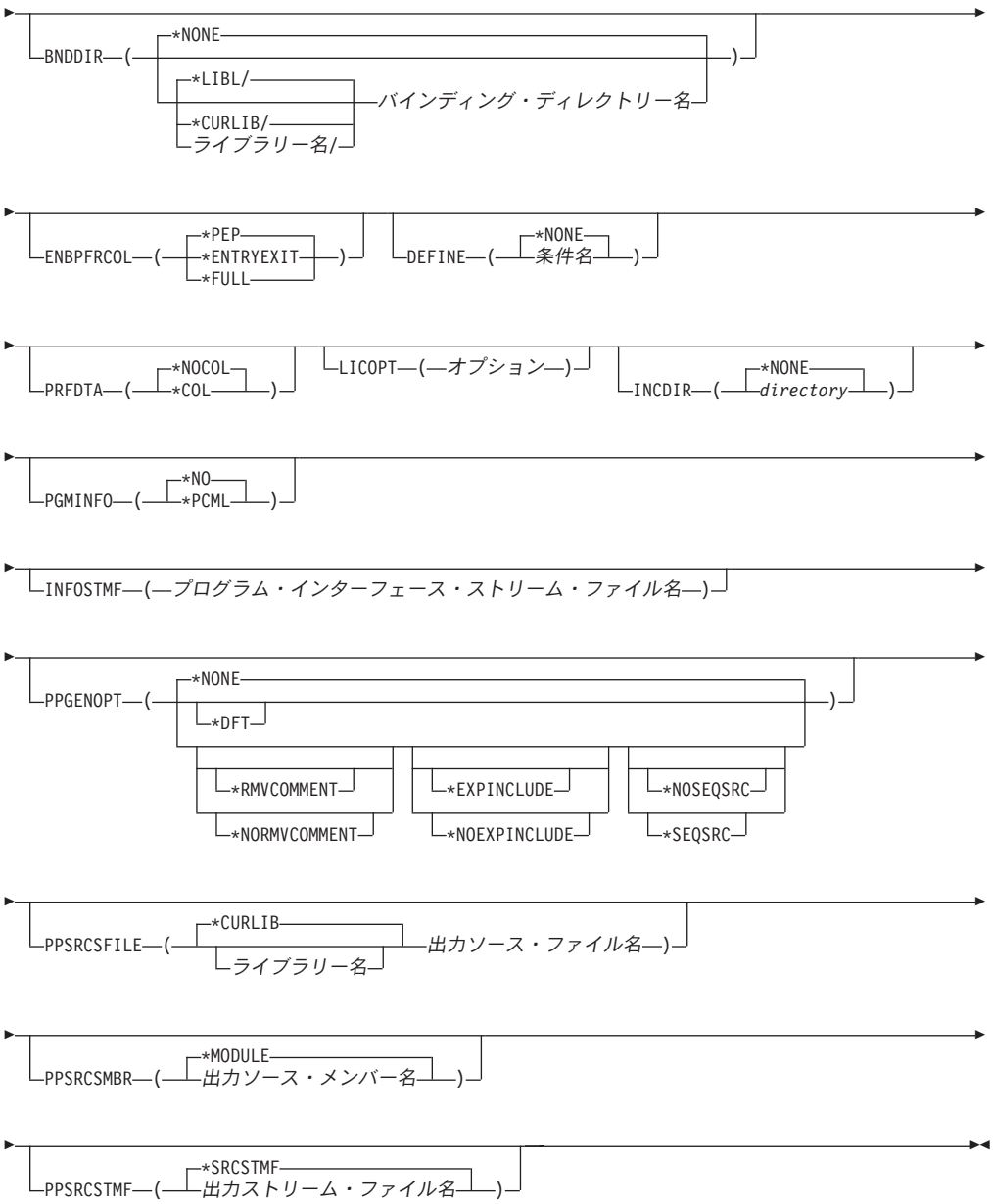
CRTRPGMOD コマンド

RPG モジュールの作成 (CRTRPGMOD) コマンドは ILE RPG ソース・コードをコンパイルして、モジュール・オブジェクト (*MODULE) を作成します。
CRTRPGMOD コマンドの構文図全体が下に示されます。

ジョブ: B,I　プログラム: B,I　REXX: B,I　EXEC



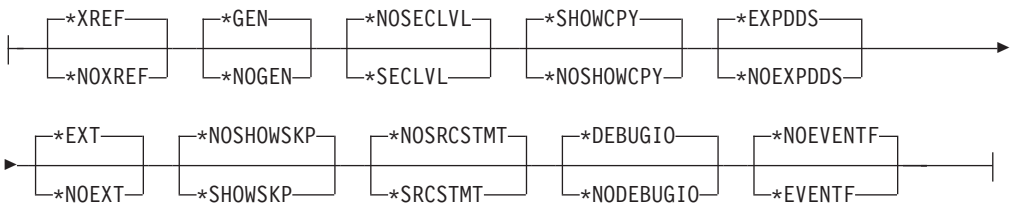

(1)
SRCSTMF—(—ソース・ストリーム・ファイル名—)
OUTPUT—(—*PRINT—*NONE—)
GENLVL—(—10—重大度レベル値—)
TEXT—(—*SRCMBRTXT—*BLANK—'説明'—)
OPTION—(— OPTION の詳細 —)
DBGVIEW—(—*STMT—*SOURCE—*LIST—*COPY—*ALL—*NONE—)
OUTPUT—(—*PRINT—*NONE—)
OPTIMIZE—(—*NONE—*BASIC—*FULL—)
INDENT—(—*NONE—文字値—)
CVTOPT—(—*NONE—*DATETIME—*GRAPHIC—*VARCHAR—*VARGRAPHIC—)
SRTSEQ—(—*HEX—*JOB—*JOBRUN—*LANGIDUNQ—*LANGIDSHR—分類テーブル名—*LIBL/—*CURLIB/—ライブラリー名/—)
LANGID—(—*JOBRUN—*JOB—言語識別コード—)
REPLACE—(—*YES—*NO—)
AUT—(—*LIBCRTAUT—*ALL—*CHANGE—*USE—*EXCLUDE—権限リスト名—)
TRUNCNBR—(—*YES—*NO—)
FIXNBR—(—*NONE—*ZONED—*INPUTPACKED—)
TGTRLS—(—*CURRENT—*PRV—VxRxMx—)
ALWNULL—(—*NO—*INPUTONLY—*USRCTL—*YES—)



注:

1 この点より前のパラメーターは、すべて定位置形式によって指定することができます。

OPTION の詳細:



CRTRPGMOD コマンドの説明

CRTRPGMOD コマンドのパラメーター、オプションおよび変数の説明については、CRTBNDRPG コマンドの対応する説明を参照してください。これらは正確に対応しますが、CRTRPGMOD ではプログラムではなくモジュールを指すことが例外です。(CRTBNDRPG の説明を見る時は、CRTRPGMOD には ACTGRP、DFTACTGRP、USRPRF のパラメーターがないことに留意してください。)

CRTRPGMOD の説明はオンラインでも使用可能です。コマンド入力行にコマンド名を入力し、 PF4 (プロンプト) キーを押してから、説明を表示したいパラメーター上で PF1 (ヘルプ) キーを押してください。

付録 D. コンパイラー・リスト

コンパイラー・リストは、RPG IV 言語の構文および意味構造に関して、コーディングが正しいかどうかを判断するための情報を提供します。リストは、モジュールのデバッグ中の手助けだけでなく、ソース編集機能を通してエラーを訂正する手助けともなるように設計されています。このセクションでは、ILE RPG コンパイラー・リストの解釈の方法について述べます。リストの使用方法については、72 ページの『コンパイラー・リストの使用』を参照してください。

コンパイラー・リストを入手するには、CRTRPGMOD コマンドまたは CRTBNDRPG コマンドのどちらかで OUTPUT(*PRINT) を指定してください。(これはデフォルトの設定です。) OUTPUT(*NONE) と指定すると、リスト出力を止めます。

表 47 は、キーワードの指定とそれに対応するコンパイラー・リスト情報を要約したものです。

表 47. コンパイラー・リストのセクション

リスト・セクション[1]	OPTION[2]	説明
プロローグ		コマンド・オプションの要約
ソース・リスト		ソースの仕様
インライン診断メッセージ		ソースの 1 行に含まれるエラー
/COPY メンバー	*SHOWCPY	/COPY メンバーのソース・レコード
スキップされたステートメント	*SHOWSKP	条件付きコンパイル指示によって除外されたソース行
外部記述ファイル	*EXPDDS	生成された仕様書
突き合わせフィールド・テーブル		突き合わせフィールドに基づいて突き合わされる長さ
追加の診断メッセージ		ソースの複数行にわたるエラー
出力バッファー中のフィールド位置		プログラム記述出力フィールドの開始桁と終了桁
/COPY メンバー・テーブル		/COPY メンバーおよびその外部名のリスト
コンパイル時データ		コンパイル・ソース・レコード
代替照合順序		ALTSEQ レコードとテーブルまたは NLSS 情報とテーブル
ファイル変換		ファイル変換レコード
配列		配列レコード
テーブル		テーブル・レコード
キー・フィールド情報	*EXPDDS	キー・フィールド属性
相互参照	*XREF	ファイルとレコードおよびフィールドと標識の参照
外部参照	*EXT	コンパイル時参照された外部プロシージャーおよびフィールドのリスト
メッセージの要約		メッセージとそれが出された回数のリスト
第 2 レベル・テキスト	*SECLVL	メッセージの第 2 レベル・テキスト
最終の要約		メッセージとソース・レコードの合計数、および最終コンパイル・メッセージ

表 47. コンパイラー・リストのセクション (続き)

リスト・セクション[1]	OPTION[2]	説明
コード生成エラー[3]		コード生成段階で起こったエラー (ある場合)
バインド・セクション[3]		CRTBNDRPG コマンドのバインド段階で起こったエラー (ある場合)

注:

1. リスト・セクションに入っている情報は、OPTION パラメーターに *SRCSTMT または *NOSRCSTMT が指定されるかどうかによって異なります。この情報がどのように変化するかについては、 514 ページの『*NOSRCSTMT ソース見出し』および 514 ページの『*SRCSTMT ソース見出し』を参照してください。*SRCSTMT は、デバッグのためにステートメント番号を生成する際にコンパイラーが SEU 順序番号およびソース ID を使用するよう要求できるようにします。*NOSRCSTMT の場合は、ステートメント番号はリストの行番号と関連付けられ、数字が順番に割り当てられます。
2. OPTION 欄には、この情報を入手するために OPTION パラメーターにどのような値を指定すればよいかを示しています。ブランクの場合には、OUTPUT(*PRINT) を指定するとその情報が常に現れることを意味します。
3. コード生成エラーとバインド・エラーを含むこれらのセクションは、エラーがある場合にだけ現れます。これらのセクションの生成を抑えるオプションはありません。

コンパイラー・リストの読み方

次のテキストには、コンパイラー・リストの各セクションについて、簡単な説明と例が示されています。セクションは、リストに現れる順に示されています。

プロローグ

プロローグ・セクションは、コマンド・パラメーターおよびその値を、 CL コマンド分析プログラムによって処理された時に要約します。*CURLIB または *LIBL が指定された場合には、実際のライブラリー名がリストされます。またプロローグに表示されるものには一時変更の値が反映されます。 509 ページの図 231 は、CRTBNDRPG コマンドを使ってコンパイルさたプログラム MYSRC のリストのプロローグ・セクションの解釈方法を図解しています。

```
#   ソースの 1 行目で指示されたタイトル 1a
#   5722WDS V5R2M0  020719 RN       IBM ILE RPG              MYLIB/MYSRC     1b     ISERIES1   02/08/15 12:58:46      PAGE     1
      コマンド . . . . . . . . . . . . . . . . . . . . . . . :   CRTBNDRPG
        投入元 . . . . . . . . . . . . . . . . . . . . . . . :     MYUSERID
      プログラム . . . . . . . . . . . . . . . . . . . . . . :   MYSRC     2
        ライブラリー . . . . . . . . . . . . . . . . . . . . :     MYLIB
      テキスト '記述' . . . . . . . . . . . . . . . . . . . . :   Text specified on the Command
      ソース・メンバー . . . . . . . . . . . . . . . . . . . :   MYSRC     3
      ソース・ファイル . . . . . . . . . . . . . . . . . . . :   QRPGLESRC     4
        ライブラリー . . . . . . . . . . . . . . . . . . . . :     MYLIB
        CCSID  . . . . . . . . . . . . . . . . . . . . . . . :     37
      テキスト '記述' . . . . . . . . . . . . . . . . . . . . :   Text specified on the Source Member
      最終変更 . . . . . . . . . . . . . . . . . . . . . . . :   98/07/27  12:50:13
      生成重大度レベル . . . . . . . . . . . . . . . . . . . :   10
      省略時の活動化グループ . . . . . . . . . . . . . . . . :   *NO
      コンパイラー・オプション . . . . . . . . . . . . . . . :   *XREF      *GEN       *SECLVL    *SHOWCPY     5
                                                                 *EXPDDS    *EXT       *SHOWSKP   *NOSRCSTMT
                                                                 *DEBUGIO   *NOEVENTF
      デバッグ・ビュー . . . . . . . . . . . . . . . . . . . :   *ALL
      出力 . . . . . . . . . . . . . . . . . . . . . . . . . :   *PRINT
      最適化レベル . . . . . . . . . . . . . . . . . . . . . :   *NONE
      ソース・リストの字下げ . . . . . . . . . . . . . . . . :   '| '   6
      タイプ変換オプション . . . . . . . . . . . . . . . . . :   *NONE
      ソート順序 . . . . . . . . . . . . . . . . . . . . . . :   *HEX
      言語識別コード . . . . . . . . . . . . . . . . . . . . :   *JOBRUN
      プログラムの置き換え . . . . . . . . . . . . . . . . . :   *YES
      ユーザー・プロファイル . . . . . . . . . . . . . . . . :   *USER
      権限 . . . . . . . . . . . . . . . . . . . . . . . . . :   *LIBCRTAUT
      数値の切り捨て . . . . . . . . . . . . . . . . . . . . :   *YES
      数値の修正 . . . . . . . . . . . . . . . . . . . . . . :   *ZONED     *INPUTPACKED
      ターゲット・リリース . . . . . . . . . . . . . . . . . :   *CURRENT
      ヌル値可能 . . . . . . . . . . . . . . . . . . . . . . :   *NO
      バインド・ディレクトリー . . . . . . . . . . . . . . . :   BNDDIRA      BNDDIRB
        ライブラリー . . . . . . . . . . . . . . . . . . . . :     CMDLIBA      CMDLIBB
      活動化グループ . . . . . . . . . . . . . . . . . . . . :   CMDACTGRP
      条件名の定義 . . . . . . . . . . . . . . . . . . . . . :   ABC   7
                                                                 DEF
      パフォーマンス収集使用可能化 . . . . . . . . . . . . . :   *PEP
      プロファイル・データ . . . . . . . . . . . . . . . . . :   *NOCOL
#     生成 PGM インターフェースの生成  . . . . . . . . . . . :   *PCML
#     プログラム・インターフェース・ストリーム・ファイル . . :   /home/mydir/MYSRC.pcml     8
#     組み込みディレクトリー . . . . . . . . . . . . . . . . :   /projects/ABC Electronics Corporation/copy files/prototypes
#                                                                :   /home/mydir      9
#
```

図 231. CRTBNDRPG のサンプル・プロローグ

1 **ページ見出し**

ページ見出し情報にはプロダクト情報行 **1b** および /TITLE 指示によって指定されたテキスト **1a** が含まれます。コンパイラー・リストのページ見出しとスペーシングのカスタマイズ方法については、73 ページの『コンパイラー・リストのカスタマイズ』を参照してください。

2 **モジュールまたはプログラム**

作成されたモジュール・オブジェクトの名前 (CRTRPGMOD を使用した場合) または作成されたプログラム・オブジェクトの名前 (CRTBNDRPG を使用した場合)。

3 **ソース・メンバー**

ソース・レコードが取り出されたソース・メンバーの名前 (一時変更コマンドを使用した場合には、これは **2** と異なることがあります)。

4 **ソース**

ソース・レコードを提供するために実際に使用されたファイルの名前。このファイルが一時変更された場合には、一時変更ソースの名前が使用されます。

5 **コンパイラー・オプション**
CRTRPGMOD コマンドまたは CRTBNDRPG コマンドでの指定にしたがって、コンパイル時に有効となっていたコンパイラー・オプション。

6 **字下げマーク**
リストのソース・セクションにある構造化命令にマークを付けるために使用される文字。

7 **条件名の定義**
ソースの読み取り前に効力をもつ条件名を指定します。

8 PCML (プログラム呼び出しマークアップ言語) の書き込み先の IFS ファイル
ルを指定します。

9 /COPY ファイルまたは /INCLUDE ファイルの検索先のディレクトリーを指
定します。

ソース・セクション

ソース・セクションは、ILE RPG ソースを含むレコードを示しています。ルート・ソース・メンバー・レコードは常に表示されます。OPTION(*EXPDDS) も指定されると、ソース・セクションは外部記述ファイルから生成されたレコードを示し、それらの行番号の横の欄に '=' の印を付けます。これらのレコードは、*NOEXPDDS が指定された場合には表示されません。 OPTION(*SHOWCPY) が指定された場合には、これもまたソースに指定された /COPY メンバーからのレコードを示し、行番号の横の欄に '+' でこれらに印を付けます。これらのレコードは、*NOSHOWCPY が指定された場合には表示されません。

またソース・セクションは、条件付きコンパイル処理を示します。/IF、/ELSEIF、/ELSE および /ENDIF 指示のあるすべての行および /IF グループが選択するソース行は印刷され、リスト行番号が指定されます。OPTION(*SHOWSKP) を指定すると、 /IF、/ELSEIF、および /ELSE 指示によって除外されたすべてのステートメントを表示し、ステートメントの横の桁に '-------' の印を付けます。リスト内の行番号は、除外された行について増分されることはありません。スキップされたステートメントはすべて、指定されたとおり正確に印刷されますが、解釈はされません。例えば、/EJECT 指示で除外されたステートメントによってページに切れ目が生じることはありません。同様に、/SPACE、/TITLE、/COPY および /EOF コンパイラー指示は、除外行で検出されても無視されます。こうしたステートメントは、デフォルトの OPTION(*NOSHOWSKP) が指定されている場合には表示されません。その代わりに、除外された行数を示すメッセージが印刷されます。

ソース・セクションは、ソースの構文エラーを識別し、適切であれば、フィールド表を含みます。

OPTION(*NOSRCSTMT) が指定された場合、行番号はリストの左側に順番に示され、コンパイルされたソース行番号を反映します。ソース ID および SEU 順序番号は、リストの右側に示され、ソース・メンバーおよびレコードをそれぞれ識別します。例えば、 511 ページの図 232 は、行 35 に /COPY ステートメントが入っているリストのセクションを示します。ルート・ソース・メンバーでは、次の行が DOWEQ 命令です。しかし、このリストでは、DOWEQ 命令は行 39 にあります。リストに示された 3 つの中間の行は /COPY ソース・メンバーからのものです。

```
行  <--------------------------------ソース仕様 ----------------------------------------------><---- 注記  ----> SRC SEQ
番号 ....1....+....2....+<-------- 26 - 35 -------->....4....+....5....+....6....+....7....+....8....+....9....+...10 ID  番号
    34 C                   MOVE                    '123'         BI_FLD1                                                  001500
    35 C/COPY MYCPY                                                                                               971104     001600
       *----------------------------------------------------------------------------------------------*
       * RPG メンバー名 . . . . . :  MYCPY                                                           *                    5
       * 外部名 . . . . . . . . . :  RPGGUIDE/QRPGLESRC(MYCPY)                                       *                    5
       * 最終変更 . . . . . . . . :  98/07/24  16:20:04                                              *                    5
       * テキスト '記述'  . . . . . :  Text on copy member                                            *                    5
       *----------------------------------------------------------------------------------------------*
    36+C     Blue(1)       DSPLY                                                                                       5000100
    37+C     Green(4)      DSPLY                                                                                       5000200
    38+C     Red(2)        DSPLY                                                                                       5000300
    39 C     *in20         doweq                   *OFF                                                             001700
```

*図 232. OPTION(*NOSRCSTMT) が指定された場合のリストのセクション例*

OPTION(*SRCSTMT) が指定された場合、順序番号は、リストの左側に示され、SEU 順序番号を反映します。ステートメント番号は、リストの右側に示されます。ステートメント番号情報は、ソース ID および SEU 順序番号情報と同じです。例えば、図 233 は、順序番号 001600 をもつ /COPY ステートメントが入っているリストのセクションを示しています。ルート・ソース・メンバーの次の行は、リスト内の次の行番号、つまり順序番号 001700 をもつ行と同じです。リストに示された中間の 3 行には、/COPY ソース・メンバーからの SEU 順序番号が割り当てられています。対応するステートメント番号は、ルート・ソース・メンバーおよび /COPY ソース・メンバーのソース ID および SEU 順序番号から生成されます。

```
SEQ    <-------------------------------- ソースの仕様 ------------------------------------  注記 -------------><  STATEMENT
番号   ....1....+....2....+<-------- 26 - 35 -------->....4....+....5....+....6....+....7....+....8....+....9....+...10 番号
001500 C                   MOVE                    '123'         BI_FLD1                                                  001500
001600 C/COPY MYCPY                                                                                               971104     001600
       *----------------------------------------------------------------------------------------------*
       * RPG メンバー名 . . . . . :  MYCPY                                                           *                    5
       * 外部名 . . . . . . . . . :  RPGGUIDE/QRPGLESRC(MYCPY)                                       *                    5
       * 最終変更 . . . . . . . . :  98/07/24  16:20:04                                              *                    5
       * テキスト '記述'  . . . . . :  Text on copy member                                            *                    5
       *----------------------------------------------------------------------------------------------*
000100+C     Blue(1)       DSPLY                                                                                       5000100
000200+C     Green(4)      DSPLY                                                                                       5000200
000300+C     Red(2)        DSPLY                                                                                       5000300
001700 C     *in20         doweq                   *OFF                                                             001700
```

*図 233. OPTION(*SRCSTMT) が指定された場合のリストのセクション例*

512 ページの図 234 は、OPTION(*NOSRCSTMT) が指定された MYSRC のソース・セクション全体を示しています。

```
# 5722WDS V5R2M0  020719 RN        IBM ILE RPG            MYLIB/MYSRC             ISERIES1   02/08/15 14:21:00        Page      2
  1a
行    <---------------------- ソース仕様 ------------------------------------><---- 注記 -----> DO  PAGE  変更   SRC 順序
番号  ....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+....9....+...10 NUM 行    日付   ID  番号
                    ソ ー ス リ ス ト
     1 H DFTACTGRP(*NO) ACTGRP('Srcactgrp') CCSID(*GRAPH:*SRC)                                                    980727     000100
     2 H OPTION(*NODEBUGIO)                                                                                      980727     000200
     3 H BNDDIR('SRCLIB1/BNDDIR1' : 'SRCLIB2/BNDDIR2' : '"ext.nam"')                                             971104     000300
     4 H ALTSEQ(*SRC)                                                                                            971104     000400
     5 H FIXNBR(*ZONED)                                                                                          980728     000500
     6 H TEXT('Text specified on the Control Specification')                                                     971104     000600
       *------------------------------------------------------------------*  2
       * 有効なコンパイラー・オプション:                                 *
       *------------------------------------------------------------------*
       *  テキスト '記述' . . . . . . . . . :                             *
       *               Text specified on the Control Specification        *
       *  生成重大度レベル . . . . . . . . . :   10                       *
       *  省略時の活動化グループ . . . . . :   *NO                        *
       *  コンパイラー・オプション . . . . :   *XREF      *GEN           *
       *                                  *SECLVL    *SHOWCPY         *
       *                                  *EXPDDS    *EXT             *
       *                                  *SHOWSKP *NOSRCSTMT         *
       *                                  *NODEBUGIO   *NOEVENTF      *
       *  最適化レベル . . . . . . . . . . :   *NONE                      *
       *  ソース・リストの字下げ . . . . . :   '| '                       *
       *  タイプ変換オプション . . . . . . :   *NONE                      *
       *  ソート順序 . . . . . . . . . . . :   *HEX                       *
       *  言語識別コード . . . . . . . . . :   *JOBRUN                    *
       *  ユーザー・プロファイル . . . . . :   *USER                      *
       *  権限 . . . . . . . . . . . . . . :   *LIBCRTAUT                 *
       *  数値の切り捨て . . . . . . . . . :   *YES                       *
       *  数値の修正 . . . . . . . . . . . :   *ZONED      *INPUTPACKED   *
       *  ヌル値可能 . . . . . . . . . . . :   *NO                        *
       *  CMD からの BINDING ディレクトリー:   BNDDIRA      BNDDIRB       *
       *    ライブラリー  . . . . . . . . :     CMDLIBA      CMDLIBB      *
       *  SRC からの BINDING ディレクトリー:   BNDDIR1      BNDDIR2       *
       *    ライブラリー . . . . . . . . . :     SRCLIB1      SRCLIB2      *
       *                                  "ext.nam"                     *
       *                                    *LIBL                       *
       *  活動化グループ . . . . . . . . . :   Srcactgrp                  *
       *  パフォーマンス収集使用可能化 . . :   *PEP                       *
       *-プロファイリング・データ . . . . :   *NOCOL                     *
       *------------------------------------------------------------------*
     7 FInFile    IF   E            DISK                                                                        971104     000700
       *-------------------------------------------------------------------------------------------------*  3
       *                            RPG 名          外部名                                               *
       * ファイル名 . . . . . . . . :  INFILE         MYLIB/INFILE                                         *
       * レコード様式 . . . . . . . :  INREC          INREC                                                *
       *-------------------------------------------------------------------------------------------------*
     8 FKEYL6     IF   E          K DISK                                                                       971104     000800
       *-------------------------------------------------------------------------------------------------*
       *                            RPG 名          外部名                                               *
       * ファイル名 . . . . . . . . :  KEYL6          MYLIB/KEYL6                                          *
       * レコード様式 . . . . . . . :  REC1           REC1                                                 *
       *                            REC2           REC2                                                 *
       *-------------------------------------------------------------------------------------------------*
     9 FOutfile   O    E            DISK                                                                        971104     000900
       *-------------------------------------------------------------------------------------------------*
       *                            RPG 名          外部名                                               *
       * ファイル名 . . . . . . . . :  OUTFILE        MYLIB/OUTFILE                                        *
       * レコード様式 . . . . . . . :  OUTREC         OUTREC                                               *
       *-------------------------------------------------------------------------------------------------*
    10 D Blue           S              4    DIM(5)CTDATA PERRCD(1)                                          971104     001000
    11 D Green          S              2    DIM(5)ALT(Blue)                                                 971104     001100
    12 D Red            S              4    DIM(2)CTDATA PERRCD(1)                                          980727     001200
    13 D DSEXT1        E DS           100    PREFIX(BI_)INZ(*EXTDFT)                                         980727     001300
    14 D FLD3          E                     INZ('111')                                                      980727     001400
```

図 234. リストのサンプル・ソース部分 (1/3)

```
      *----------------------------------------------------------------------------------------*  4              1
      * データ構造 . . . . . :  DSEXT1                                                        *                 1
      * 接頭部 . . . . . :  BI_ :    0                                                        *                 1
      * 外部様式 . . . . . :  REC1 : MYLIB/DSEXT1                                             *                 1
      * 様式テキスト . . . . . :  Record format description                                   *                 1
      *----------------------------------------------------------------------------------------*                 1
  5
   15=D BI_FLD1                      5A   EXTFLD (FLD1)                    FLD1 description                    1000001
   16=D                                   INZ (*BLANK)                                                         1000002
   17=D BI_FLD2                     10A   EXTFLD (FLD2)                    FLD2 description                    1000003
   18=D                                   INZ (*BLANK)                                                         1000004
   19=D BI_FLD3                     18A   EXTFLD (FLD3)                    FLD3 description                    1000005
   20=D                                   INZ ('111')                                                          1000006
   21=IINREC                                                                                                   2000001
      *----------------------------------------------------------------------------------------*                 2
      * RPG レコード様式 . . . . . :  INREC                                                   *                 2
      * 外部様式 . . . . . :  INREC : MYLIB/INFILE                                            *                 2
      *----------------------------------------------------------------------------------------*                 2
   22=I                           A    1   25  FLDA                                                           2000002
   23=I                           A   26   90  FLDB                                                           2000003
   24=I                13488 *VAR C   91  112  UCS2FLD                                                        2000004
   25=IREC1                                                                                                    3000001
      *----------------------------------------------------------------------------------------*                 3
      * RPG レコード様式 . . . . . :  REC1                                                    *                 3
      * 外部様式 . . . . . :  REC1 : MYLIB/KEYL6                                              *                 3
      *----------------------------------------------------------------------------------------*                 3
   26=I                      *ISO-D    1   10  FLD12                                                          3000002
   27=I                           A   11   13  FLD13                                                          3000003
   28=I                           A   14   17  FLD14                                                          3000004
   29=I                           A   18   22  FLD15                                                          3000005
   30=I                13488      C   23   32  FLDC                                                          3000006
   31=I                13488 *VAR C   33   44  FLDCV                                                         3000007
   32=I                835        G   45   54  FLDG                                                          3000008
   33=IREC2                                                                                                    4000001
      *----------------------------------------------------------------------------------------*                 4
      * RPG レコード様式 . . . . . :  REC2                                                    *                 4
      * 外部様式 . . . . . :  REC2 : MYLIB/KEYL6                                              *                 4
      *----------------------------------------------------------------------------------------*                 4
   34=I                      *ISO-D    1   10  FLD22                                                          4000002
   35=I                           A   11   13  FLD23                                                          4000003
   36=I                           A   14   17  FLD24                                                          4000004
   37=I                           A   18   22  FLD25                                                          4000005
行  <------------------------------ ソース仕様 ----------------------------------------------><---- 注記 -----> SRC SEQ
番号 ....1....+....2....+<-------- 26 - 35 -------->....4....+....5....+....6....+....7....+....8....+....9....+...10 ID  番号
   38 C                   MOVE                   '123'         BI_FLD1                                       001500
   39 C/COPY MYCPY                                                                                 971104     001600
      *----------------------------------------------------------------------------------------*  6
      * RPG メンバー名 . . . . . :  MYCPY                                                     *                 5
      * 外部名 . . . . . :  MYLIB/QRPGLESRC(MYCPY)                                            *                 5
      * 最終変更 . . . . . :  98/07/24  16:20:04                                              *                 5
      * テキスト '記述' . . . . . :  Text specified on Copy Member                             *                 5
      *----------------------------------------------------------------------------------------*
  7
   40+C     Blue(1)       DSPLY                                                                               5000100
   41+C     Green(4)      DSPLY                                                                               5000200
   42+C     Red(2)        DSPLY                                                                               5000300
  8
   43 C     *in20         doweq                  *OFF                                                         001700
   44 C                   | READ                 InRec                        ----20                          001800
   45 C                   | if                   NOT *in20                                                    001900
   46 C     FLDA          | | DSPLY                                                                         002000
   47 C                   | endif                                                                           002100
   48 C                   enddo                                                                             002200
   49 C                   write                  outrec
                                                                                 9
   50 C                   SETON                                                LR----                       002400
   47 C/DEFINE ABC                                                                                 971104     002500
   51 C/IF DEFINED(ABC)                                                                            971104     002600
   52 C                   MOVEL                  'x'           Y              10                              002700
   54 C                   MOVEL                  'x'           Z              10                              002800
   55 C/ELSE                                                                                       971104     002900
  10
------ C                   MOVEL     ' '          Y              10                                971104     003000
------ C                   MOVEL     ' '          Z              10                                971104     003100
   56 C/ENDIF                                                                                      971104     003200
```

図 234. リストのサンプル・ソース部分 (2/3)

```
行     <---------------------- ソース仕様 ----------------------------------><---- 注記 -----> DO  PAGE  変更   SRC SEQ
番号   ....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+....9....+...10   NUM 行  日付  ID  番号
    57=OOUTREC                                                                                                6000001
       *------------------------------------------------------------------------------------------*           6
       * RPG レコード様式 . . . . . :  OUTREC                                                      *           6
       * 外部様式 . . . . . . . . . :  OUTREC : MYLIB/OUTFILE                                      *           6
       *------------------------------------------------------------------------------------------*           6
    58=O                      FLDY             100A CHAR      100                                             6000002
    59=O                      FLDZ             132A CHAR       32                                             6000003
    60=O                      GRAPHFLD         156G GRPH       12 835                                         6000004
       * * * * *  ソ ー ス の 終 わ り  * * * * *
```

図 234. リストのサンプル・ソース部分 (3/3)

1a ***NOSRCSTMT ソース見出し**

上の例のソース見出しは、OPTION(*NOSRCSTMT) を指定して生成されたものです。

行番号 1 で始まり、各ソースまたは生成されたレコードで 1 ずつ増えます。ステートメント番号を使用してデバッグする時には、この番号を使用してください。

桁表示行

この行は、字下げが指定された時に調整されます。

DO NO.

構造化命令のレベルを指定します。字下げが要求された場合には、この番号は現れません。

PAGE 行

ソース・レコードの最初の 5 桁を示します。

Src ID

レコードのソース (/COPY または DDS) を指定します。/COPY メンバーの場合には、これを使用して、/COPY メンバー・テーブルから外部メンバー名を入手することができます。

順序番号 (リストの右側に示されるもの)

ソース物理ファイルのメンバーからのレコードの SEU 順序番号を示します。/COPY メンバーからのレコードまたは DDS からのレコードの増分番号を示します。

1b ***SRCSTMT ソース見出し**

OPTION(*SRCSTMT) が指定された場合には、ソース見出しは、次のように変わります。

```
       1b
SEQ    <---------------------- ソース仕様 ----------------------------------><---- 注記 -----> DO  PAGE  変更   STATEMENT
番号   ....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+....9....+...10   NUM 行   日付   番号
```

桁表示行、DO NUM、および PAGE 行は、変わりません。

順序番号 (リストの左側に示されるもの)

ソース物理ファイルのメンバーからのレコードの SEU 順序番号を示します。/COPY メンバーからのレコードまたは DDS からのレコードの増分番号を示します。

STATEMENT 番号

以下のような、ソース ID 番号および SEU 順序番号から生成されたステートメント番号を示します。

```
ステートメント番号 = ソース ID * 1000000 + ソース SEU 順序番号
```

ステートメント番号を使用してデバッグする時には、この番号を使用してください。

2 有効なコンパイラー・オプション

有効なコンパイラー・オプションを識別します。コンパイル・オプション・キーワードが制御仕様書に指定されると表示されます。

3 ファイル/レコード情報

外部記述ファイル、およびこれが含むレコードを指定します。

4 DDS 情報

フィールド情報がどの外部記述ファイルから取り出されるかを確認します。指定された場合には、接頭部値が示されます。DDS で指定された場合には、様式レコード・テキストが示されます。

5 生成された仕様書

行番号の横に '=' が示された、DDS から生成された仕様書を示します。DDS で指定された場合には、最大 50 桁のフィールド・テキストを示します。定義仕様書に INZ キーワードで指定されたとおりの初期値を示します。外部記述データ構造サブフィールドに INZ(*EXTDFT) が指定された場合には、DDS デフォルト値が表示されます。長すぎて 1 行に収まらないデフォルト値は、切り捨てられ、終わりに '...' が付けられます。

6 /COPY メンバー情報

使用される /COPY メンバーを指定します。もしあれば、メンバー・テキストを示します。メンバーへの最終変更の日付および時刻を示します。

7 /COPY メンバー・レコード

/COPY メンバーからのレコードを示しますが、これは行番号の横に '+' で示されます。

8 字下げ

構造化命令にマークを付けるように要求した時に、構造化命令がどのように現れるかを示します。

9 標識使用状況

標識が使用される時に、未使用の標識の位置を示します。

10 OPTION(*SHOWSKP) の使用

/IF 指示によって除外される 2 つのステートメントは、横に '-------' を付けて表示されます。OPTION(*NOSHOWSKP) が指定されると、この 2 つのステートメントは、 LINES EXCLUDED: 2 に置き換えられます。

追加の診断メッセージ

追加の診断メッセージ・セクションには、複数の行にわたるエラーを示すコンパイラー・メッセージがリストされます。可能であれば、メッセージは、エラーのあるソースの行番号と順序番号を示します。 516 ページの図 235 に例を示します。

```
         追 加 の 診 断 メ ッ セ ー ジ
 MSG ID     SV    番号   SEQ      メッセージ・テキスト
*RNF7066    00      8   000800    外部記述ファイルのレコード様式名は使用されない。
*RNF7066    00      8   000800    外部記述ファイルのレコード様式名は使用されない。
*RNF7086    00     60   000004    RPG は、ファイルのブロック化を処理する。
                                  INFDS は、データのブロックが転送された場合にだけ更新されます。
*RNF7086    00     60   000004    RPG は、ファイルのブロック化を処理する。
                                       INFDS は、データのブロックが転送された場合にだけ更新されます。
 * * * * *  追 加 の 診 断 メ ッ セ ー ジ の 終 わ り   * * * * *
```

*図 235. OPTION(*NOSRCSTMT) が指定された追加の診断メッセージ例*

OPTION(*SRCSTMT) が指定された場合、メッセージは、示されているステートメント番号だけをもちます。図 236 に例を示します。

```
         追 加 の 診 断 メ ッ セ ー ジ
 MSG ID     SV     STATEMENT      メッセージ・テキスト
*RNF7066    00        000800      外部ファイルのレコード様式名は使用されない。
*RNF7066    00        000800      外部ファイルのレコード様式名は使用されない。
*RNF7086    00       6000004      RPG は、ファイルのブロック化を処理する。
                                  INFDS は、データのブロック化が転送された場合にだけ更新されます。
*RNF7086    00       6000004      RPG は、ファイルのブロック化を処理する。
                                       INFDS は、データのブロックが転送された場合にだけ更新されます。
 * * * * *  追 加 の 診 断 メ ッ セ ー ジ の 終 わ り   * * * * *
```

*図 236. OPTION(*SRCSTMT) が指定された追加の診断メッセージ例*

出力バッファー位置

出力バッファー位置テーブルのフィールド位置は、ソースにプログラム記述出力仕様書が入っていればいつでもリストに含まれます。出力される各変数またはリテラルでは、テーブルには出力フィールド仕様書の行番号および出力バッファー中の開始と終了桁が含まれています。テーブルには長すぎるリテラルは切り捨てられ、'...' が後ろに付けられますが、終わりのアポストロフィは付きません (例えば、'Extremely long-litera...')。 図 237 は出力バッファー位置テーブルの例を示しています。

```
                出 力 バ ッ フ ァ ー の 位 置
行   START  END   フィールドまたは固定情報
番号 POS    POS
  58    1   100   FLDY
  59  101   132   FLDZ
  60  133   156   GRAPHFLD
 * * * * *  出 力 バ ッ フ ァ ー の 位 置 の 終 わ り   * * * * *
```

図 237. 出力バッファー位置テーブル

/COPY メンバー・テーブル

/COPY メンバー・テーブルは、ソースに指定された /COPY メンバーを指定し、これらの外部名をリストします。情報源識別コード番号を使用して、メンバーの名前および位置を検索することができます。このテーブルはまた、モジュール / プログラムによって使用されるメンバーのレコードとしても有用です。 517 ページの図 238 に例を示します。

```
                          /コピー・ メンバー
行     SRC     RPG 名  <--------- 外部名 ----------> CCSID  <- 最終変更 ->
番号   ID              ライブラリー  ファイル    メンバー      日付      時刻
  39        5  MYCPY   MYLIB         QRPGLESRC   MYCPY    37  98/07/24 16:20:04
         *****  /コピー・メンバーの終わり  *****
```

図 238. サンプル /コピー・メンバー・テーブル

コンパイル時データ

コンパイル時データのセクションには、ALTSEQ または NLSS テーブルについての情報およびテーブルと配列についての情報が含まれています。この例では、図 239 に示したように、代替照合順序および 2 つの配列が含まれています。

```
                       コンパイル時データ
    61 **                                                                                                    971104     003300
       *--------------------------------------------------------------------*
       * 代替照合順序テーブルのデータ:                                      *
       *--------------------------------------------------------------------*
    62 ALTSEQ    1122ACAB4B7C36F83A657D73                                                                    971104     003400
行   <---------------------- データ・レコード --------------------------------------------------------------->   変更  SRC  SEQ
番号 ....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+....9....+...10        日付   ID  番号
       *--------------------------------------------------------------------*
       * 代替照合順序テーブル:                                              *
       * 順序変更の文字数 . . . . . . . . . . . . . . . . . . . . . . . :  6 1 *
       * 2 0_ 1_ 2_ 3_ 4_ 5_ 6_ 7_ 8_ 9_ A_ B_ C_ D_ E_ F_                  *
       * _0  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .   _0            *
       * _1  .  22 3 .  .  .  .  .  .  .  .  .  .  .  .  .    _1            *
       * _2  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .   _2            *
       * _3  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .   _3            *
       * _4  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .   _4            *
       * _5  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .   _5            *
       * _6  .  .  .  F8 .  .  .  .  .  .  .  .  .  .  .  .   _6            *
       * _7  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .   _7            *
       * _8  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .   _8            *
       * _9  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .   _9            *
       * _A  .  .  .  65 .  .  .  .  .  .  .  .  .  .  .  .   _A            *
       * _B  .  .  .  .  7C .  .  .  .  .  .  .  .  .  .  .   _B            *
       * _C  .  .  .  .  .  .  .  .  .  .  AB .  .  .  .  .   _C            *
       * _D  .  .  .  .  .  .  .  73 .  .  .  .  .  .  .  .   _D            *
       * _E  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .   _E            *
       * _F  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .   _F            *
       *    0_ 1_ 2_ 3_ 4_ 5_ 6_ 7_ 8_ 9_ A_ B_ C_ D_ E_ F_                 *
       *--------------------------------------------------------------------*
    63 **                                                                                                    971104     003500
       *--------------------------------------------------------------------*
       * 配列 . . . : BLUE  4    代替配列  . . . . : GREEN      *
       *--------------------------------------------------------------------*
    64 1234ZZ                                                                                                971104     003600
    65 ABCDYY                                                                                                971104     003700
    66 5432XX                                                                                                971104     003800
    67 EDCBWW                                                                                                971104     003900
    68 ABCDEF                                                                                                0980728    004000
    69 **                                                                                                    971104     00410
       *--------------------------------------------------------------------*
       * 配列 . . . : RED                                                  *
       *--------------------------------------------------------------------*
    70 3861                                                                                                  971104     00420
    71 TJKL                                                                                                  971104     00430
 *****  コンパイル時データの終わり  *****
```

図 239. サンプル・コンパイル時データのセクション

1 変更された文字の合計数
　　分類順序が変更された文字の数を示します。

2 変更する文字
　　テーブルの行と桁の組み合わせで変更される文字を識別します。例えば、文字 3A の新しい値は 65 となり、列 3_ 行 _A にあります。

3 代替順序
選択された文字の新しい 16 進分類値。

4 配列/テーブル情報
コンパイラーがデータを必要とする配列またはテーブルの名前を指定します。代替配列が定義されていれば、その名前も示されます。

キー・フィールド情報

キー・フィールド情報セクションには、各キー付きファイルのキー・フィールドに関する情報が示されます。このセクションにはまた、複数のレコードに共通するキー (すなわち、共通キー) に関する情報も示されます。図 240 は、例を示しています。

```
                キ ー ・ フ ィ ー ル ド 情 報
      ファイル        内部          外部
        レコード      フィールド名  フィールド名  属性
    2  KEYL6
        共通キー:
                                              DATE *ISO- 10
                                              CHAR        3
        REC1
                      FLD12                   DATE *ISO- 10
                      FLD13                   CHAR        3
                      FLD15                   CHAR        5
                      FLDC                    UCS2        5 13488
                      FLDCV                   VUC2        5 13488
                      FLDG                    GRPH        5 835
        REC2
                      FLD22                   DATE *ISO- 10
                      FLD23                   CHAR        3
* * * * *  キ ー ・ フ ィ ー ル ド 情 報 の 終 わ り  * * * * *
```

図 240. キー・フィールド情報の例

相互参照表

相互参照表には、少なくとも次の 3 つのリストが入っています。

- ファイルおよびレコード
- 大域フィールド
- 標識

さらに、これには各サブプロシージャーによって使用されるローカル・フィールドが含まれます。ファイル、フィールドおよび標識がモジュール / プログラム内のどこで使用されるかをチェックするためにこのテーブルを使用してください。

識別コードが参照されない時に出される通知メッセージ RNF7031 は、リストの相互参照セクションおよびメッセージ要約にのみ表示されることに注意してください。これはリストのソース・セクションには表示されません。

122 桁を超える長さの名前は、複数行にわたるリストの相互参照セクションに表示されます。名前全体が、行の末尾に文字 '...' を付けて印刷されます。名前の最後の部分が 17 桁を超える場合は、次の行から開始して属性および行番号をリストします。 519 ページの図 241 は、2 つのサブプロシージャーをもつモジュール TRANSRPT の例を示しています。

この例では、相互参照表は、各参照の行番号を示します。OPTION(*NOSRCSTMT) ではなく OPTION(*SRCSTMT) を指定すると、各参照ごとにステートメント番号が表示され、相互参照リストは、リストの最初の 80 桁より長く拡張できます。

```
                        相  互  参  照  表
      ファイルおよびレコードの参照:
         ファイル          装置            参照 (D=定義済み)
           レコード
         CUSTFILE          DISK                 8D
           CUSTREC                            0      44
*RNF7031 CUSTRPT           DISK                 9D
           ARREARS                            0      60      79
      大域フィールド参照:
         フィールド        属性            参照 (D=定義 M=変更)
         *INZSR            BEGSR               63D
         AMOUNT            P(10,2)             56M     83      95
         CITY              A(20)               53D    132
         CURDATE           D(10*ISO-)          42D     64M     92
         CUSTNAME          A(20)               50D    122
         CUSTNUM           P(5,0)              49D    124
         DUEDATE           A(10)               57M     84      91
         EXTREMELY_LONG_PROCEDURE_NAME_THAT_REQUIRES_MORE_THAN_ONE_LINE_IN_THE_CROSS_REFERENCE_EVEN_THOUGH_THE_ENTIRE_LINE_UP_TO_...
            COLUMN_132_IS_USED_TO_PRINT_THE_NAME...
                           I(5,0)               9D
                           PROTOTYPE
         FMTCUST           PROTOTYPE           35D     59     113     114
                                               134
         INARREARS         A(1)                30D     58      85      86
                           PROTOTYPE           101
         LONG_FLOAT        F(8)                 7D     11M     12M
         NUMTOCHAR         A(31)               22D    124     130
                           PROTOTYPE
         RPTADDR           A(100)              59      82
         RPTNAME           C(100)              59      81
                           CCSID(13488)
         RPTNUM            P(5,0)              80
         SHORT_FLOAT       F(4)                8D     10M
*RNF7031 STATE             A(2)                54D
         STREETNAME        A(20)               52D    131
         STREETNUM         P(5,0)              51D    130
         THIS_NAME_IS_NOT_QUITE_SO_LONG...
                           A(5)                 7D
         UDATE             S(6,0)              64
*RNF7031 ZIP               P(5,0)              55D
      INARREARS フィールド参照:
         フィールド        属性            参照 (D=定義 M=変更)
         DAYSLATE          I(10,0)             88D     92M     94
         DATEDUE           D(10*ISO-)          89D     91M     92
      FMTCUST フィールド参照:
         フィールド        属性            参照 (D=定義 M=変更)
         NAME              A(100)             115D    122M
                           BASED(_QRNL_PST+)
         ADDRESS           A(100)             116D    130M
                           BASED(_QRNL_PST+)
      標識の参照:
         標識                              参照 (D=定義 M=変更)
*RNF7031 01                                 44D
       *****  相  互  参  照  表  の  終  わ  り  *****
```

*図 241. OPTION(*NOSRCSTMT) が指定された相互参照テーブル例*

外部参照リスト

外部参照セクションでは、バインド時に他のモジュールで必要かあるいは使用可能な外部プロシージャーおよびフィールドをリストします。このセクションは、ソースが静的にバインドされたプロシージャー、インポートされたフィールド、またはエクスポートされたフィールドを含んでいる時にはいつでも表示されます。

静的にバインドされたプロシージャー部にはプロシージャー名と、CALLB 命令または %PADDR 組み込み関数の名前参照、または CALLP に呼び出されるか式の内部のプロトタイプのバインドされたプロシージャーの名前参照が含まれます。

インポートされたフィールドとエクスポートされたフィールドの部分には、フィールド名、次元 (配列の場合)、フィールド属性、およびその定義参照を含みます。図 242 に例を示します。

```
                    外  部  参  照
     静的結合プロシージャー:
        プロシージャー                          参照
        PROTOTYPED                               2       2
        PADDR_PROC                               4
        CALLB_PROC                               6
     インポート・フィールド:
        フィールド        属性                 定義済み
        IMPORT_FLD        P(5,0)                  3
     エクスポート・フィールド:
        フィールド        属性                 定義済み
        EXPORT_ARR(2)     A(5)                    2
   *****  外  部  参  照  の  終  わ  り  *****
```

図 242. 外部参照の例

メッセージの要約

メッセージの要約には、起こったエラーの重大度による合計が入っています。OPTION(*SECLVL) が指定された番号に、これは第 2 レベル・メッセージ・テキストも提供します。図 243 は、例を示しています。

```
                         メ  ッ  セ  ー  ジ  の  要  約
 MSG ID  SV   番号 メッセージ・テキスト
*RNF7031 00     16 名前または標識が参照されていない
                   原因  . . . . . :   フィールド、サブフィールド、TAG、データ構造、
                   PLIST、KLIST、サブルーチン、標識、またはプロトタイプがプログラムの中で
                   定義されているが、参照されていません。
                   回復手順  . . . :   この項目を参照するか、あるいは項目をプログラムから
                   除去してください。コンパイルし直してください。
*RNF7066 00      2 外部記述ファイルのレコード様式名は使用されない。
                   原因  . . . . . :   正しい入力または出力操作には使用されない外部記述
                   ファイルのレコード様式名があります。
                   回復手順  . . . :   この外部記述ファイルのレコード様式名を入力または
                   出力に使用するか、あるいはその名前をキーワード IGNORE のパラメーター
                   として指定してください。コンパイルし直してください。
*RNF7086 00      2 RPG はファイルのブロック化を処理する。INFDS は、データのブロックが転送
                   された場合にだけ更新されます。
                   原因  . . . . . :   RPG では、UFCB (ユーザー・ファイル制御ブロック) に
                   MLTRCD(*YES) が指定されます。レコードは、RPG とデータ管理機能の間で
                   ブロック単位で渡されます。INFDS (ファイル情報データ構造) の 241 桁目
                   から終わりまでは、レコードのブロックが読み取りまたは書き出された場合にだけ
                   更新されます。
                   回復手順  . . . :   レコードの各読取りまたは書き出しの後にこの情報が必要
                   であった場合には、SEQONLY(*NO) をもつ OVRDBF コマンドをそのファイル
                   に対して指定してください。
      *****  メ  ッ  セ  ー  ジ  の  要  約  の  終  わ  り  *****
```

図 243. メッセージの要約の例

最終の要約

最終の要約セクションには、最終メッセージ統計とソース統計が提示されます。また、コンパイルの状況も指定されます。図 244 に例を示します。

```
                          最 終 の 要 約
  メッセージ合計:
    通知         (00) . . . . . . . :      20
    警告         (10) . . . . . . . :       0
    エラー       (20) . . . . . . . :       0
    重大エラー   (30+) . . . . . . . :       0
    -------------------------------  -------
    合計  . . . . . . . . . . . . . :      20
  ソース合計:
    レコード. . . . . . . . . . . . :      71
    仕様  . . . . . . . . . . . . . :      55
    データ・レコード  . . . . . . . :       8
    注記  . . . . . . . . . . . . . :       0
        * * * * *  最 終 の 要 約 の終 わ り * * * * *
プログラム MYSRC がライブラリー MYLIB に入れられました。最高の重大度は 00。98/07/28 の 14:21:03 に作成されました。
        * * * * *  E N D  O F  C O M P I L A T I O N * * * * *
```

図 244. 最終の要約の例

コード生成およびバインド・エラー

最終の要約セクションの後に、コード生成エラーまたはバインド・エラー (あるいはその両方) のセクションが示されることがあります。

コード生成エラーのセクションは、コンパイラーがモジュール・オブジェクトのコードを生成中にエラーが起こった時にだけ表示されます。一般には、このセクションは表示されません。バインド・エラーのセクションは、CRTBNDRPG コマンドのバインド中にメッセージが出るといつでも表示されます。一般的なエラーは、CRTBNDRPG コマンドが出された時にソースで参照されたすべての 外部プロシージャーおよびフィールドの場所を指定できないことです。

特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものであり、本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。

使用許諾については、下記の宛先に書面にてご照会ください。

〒106-0032
東京都港区六本木 3-2-31
IBM World Trade Asia Corporation
Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム（本プログラムを含む）との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Canada Ltd. Laboratory
Information Development
8200 Warden Avenue
Markham, Ontario L6G 1C7
Canada

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、 IBM より提供されます。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、さまざまなオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。お客様は、IBM のアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。

プログラミング・インターフェース情報

本書の目的は、お客様が RPG IV ソースを使用してプログラムの作成する手助けをすることです。本書には、ILE RPG コンパイラーが提供している汎用プログラミング・インターフェースとそれに関連する情報を記述しています。

汎用プログラミング・インターフェースにより、お客様が ILE RPG コンパイラー機能を使用するプログラムを書くことができます。

商標

以下は、IBM Corporation の商標です。

400	IBM
alphaWorks	IBMLink
AFP	ILE
AS/400	iSeries
C/400	MQSeries
DB2 Universal Database	Redbooks
e (ロゴ)	RPG/400
@server	VisualAge
GDDM	WebSphere

Domino は、IBM Corporation の商標です。

Java およびすべての Java 関連の商標およびロゴは、Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です。

Microsoft、Windows、Windows NT および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

UNIX は、The Open Group の米国およびその他の国における登録商標です。

他の会社名、製品名およびサービス名などはそれぞれ各社の商標または登録商標です。

登録商標および未登録の商標はそれぞれ、® および ™ で示しています。

参考文献

iSeries システムの ILE RPG プログラミングに関連したトピックについての詳細な説明は、次の IBM iSeries の資料を参照してください。

- 「*AS/400 プログラム開発管理機能 (PDM)*, SC88-5197-00」コピー、削除、名前変更などの操作を容易にするために、ライブラリーのリスト、オブジェクト、メンバー、およびユーザー定義のオプションを使って実行する、プログラム開発管理機能 (PDM) の使用法についての説明があります。PDM を学習するのに役立つ活動および参照情報が入っています。最も一般に使用される操作および機能キーについては、例を使用した詳細な説明があります。
- 「*AS/400 適用業務開発ツールセット AS/400 用原始ステートメント入力ユーティリティー 使用者の手引きと参照*, SD88-5047-00」ソース・メンバーを作成および編集するためのアプリケーション開発ツールセット原始ステートメント入力ユーティリティー (SEU) の使用法についての説明があります。この資料では、SEU セッションを開始および終了する方法と、このフルスクリーン・テキスト・エディターの多くの機能を使用する方法を説明しています。この解説書には、新規ユーザーと経験の深いユーザーの両方にとって簡単な行編集コマンドから高水準言語やデータ形式用の事前定義プロンプトの使用まで、各種の編集作業の実施に役立つ例が入っています。
- 「*AS/400 Advanced Series Application Display Programming*, SC41-5715-00」次のことについての情報を提供します。
  - アプリケーション用の画面を作成し保守するための DDS の使用法
  - システム上の表示装置ファイルの作成および処理
  - オンライン・ヘルプ情報の作成
  - アプリケーション用のパネルとダイアログを定義するための UIM の使用法
  - パネル・グループ、レコード、または文書の使用法
- 「*バックアップおよび回復の手引き*, SD88-5008-06」以下の設定およびその管理についての情報を提供します。
  - ジャーナル処理、アクセス・パス保護、およびコミットメント制御
  - ユーザー補助記憶域プール (ASP)
  - ディスク保護 (装置パリティー、ミラー保護、およびチェックサム)

  これは、バックアップ媒体および保管 / 復元操作に関するパフォーマンス情報を提供します。また、活動時保管サポートの使用、異なるリリースへの保管および復元、およびプログラミングのヒントおよび手法などの、拡張バックアップおよび回復のトピックについての説明があります。
- 「*CL プログラミング*, SD88-5038-05」オブジェクトとライブラリー、CL プログラミング、プログラム間の制御の流れと連絡、CL プログラムでのオブジェクトの処理、および CL プログラムの作成についての概要説明を含む iSeries プログラミングの広範な説明が入っています。その他のトピックとしては、事前定義メッセージと即時メッセージおよびメッセージの処理、ユーザー定義のコマンドとメニューの定義と作成、デバッグ・モード、ブレークポイント、追跡、および表示機能を含むアプリケーションのテストなどが入っています。
- 「*Communications Management*, SC41-5406-02」通信環境における実行管理機能、通信状況、通信障害の追跡および診断、エラーの処理と回復手順、パフォーマンス、特定の回線速度、およびサブシステム記憶域情報に関する説明が入っています。
- 「*GDDM Programming Guide*, SC41-0536-00」OS/400 図形データ表示管理プログラム (GDDM) を使って図形データ・アプリケーション・プログラムを作成する方法を説明します。多くのプログラム例およびこのプロダクトをデータ処理システムに適合させる方法を理解するのに役立つ情報が含まれています。

- 「*GDDM Reference*, SC41-3718-00」OS/400 図形データ表示管理プログラム (GDDM) を使って図形データ・アプリケーション・プログラムを作成する方法を説明します。この資料には、GDDM で使用可能なすべてのグラフィックス・ルーチンの詳細説明が入っています。また、GDDM への高水準言語インターフェースについての説明も入っています。
- 「*ICF Programming*, SC41-5442-00」iSeries 通信および OS/400 システム間通信機能 (OS/400-ICF) を使うアプリケーション・プログラムを作成するのに必要な情報を提供します。また、データ記述仕様書 (DDS) キーワード、システム提供の様式、戻りコード、ファイル転送サポートの説明、およびプログラム例も入っています。
- 「*IDDU Use*, SC41-5704-00」iSeries 対話式データ定義ユーティリティー (IDDU) を使ってデータ・ディクショナリー、ファイル、およびレコードをシステムに対して記述する方法を説明します。次の項目が含まれています。
  - コンピューター・ファイルおよびデータ定義の概念の紹介
  - 照会および文書で使用するデータを記述するための IDDU の使用法の紹介
  - データ・ディクショナリー、ファイル、レコード様式、およびフィールドの作成、保守、および使用に関する代表的な作業
  - 他のシステムで作成されたファイルを処理するための IDDU の使用法、エラー回復と問題の防止策に関する詳細説明
- 「*WebSphere Development Studio: ILE C/C++ Programmer's Guide*, SC09-2712-03」ILE C 言語を使ってアプリケーションを開発する方法を説明します。これには、プログラムの作成、実行、およびデバッグについての説明があります。また、言語をまたがるプログラムとプロシージャー呼び出し、ロケール、例外処理、データベース、外部記述ファイル、および装置ファイルのプログラミング上の考慮事項が解説されています。パフォーマンス上のヒントもいくつか説明されています。付録には、ソース・コードを EPM C/400® またはシステム C/400 から ILE C へマイグレーションする情報が含まれています。
- 「*WebSphere Development Studio: ILE COBOL プログラマーの手引き*, SD88-5045-03」iSeries システム上で ILE COBOL を作成、コンパイル、バインド、実行、デバッグ、および保守する方法を説明します。ここには、他の ILE COBOL および非 ILE COBOL プログラムの呼び出し方法、他のプログラムとのデータの共用方法、ポインターの使用法、および例外の処理方法についてのプログラミング情報が含まれています。 また、外部接続装置、データベース・ファイル、表示装置ファイル、および ICF ファイルに対する入出力操作の実行方法が説明されています。
- 「*ILE 概念*, SD88-5033-06」OS/400 ライセンス・プログラムの統合化言語環境® (ILE) 体系に関する概念および用語が説明されています。モジュールの作成、バインド、プログラムの実行、プログラムのデバッグ、および例外処理などをカバーするトピックが含まれています。
- 「*WebSphere Development Studio: ILE RPG 解説書*, SD88-5043-04」ILE RPG プログラミング言語についての説明があります。この解説書は位置ごとおよびキーワードごとに、すべての RPG IV 仕様書に有効な項目を説明し、またすべての演算コードおよび組み込み関数を詳細に説明しています。またこの解説書には、RPG の論理サイクル、配列とテーブル、編集機能、および標識の説明があります。
- 「*印刷装置プログラミング*, SD88-5073-02」印刷を理解し制御するために役立つ情報を提供します。iSeries システムの印刷要素およびそのシステムの概念の特定情報、印刷操作に対するプリンター・ファイルと印刷スプーリング・サポート、およびプリンターの接続性について説明しています。パーソナル・コンピューターの使用についての考慮事項、ビジネス・グラフィックス・ユーティリティー (BGU)、高機能印刷 (AFP™) などのその他の印刷機能、スプール出力をある出力待ち行列から別の出力待ち行列へ移動するといった iSeries システムの印刷要素の処理例が入っています。また、印刷負荷の管理に使用する制御言語 (CL) の付録も付いています。 iSeries システムとともに使用できるフォントも提供しています。フォント置換テーブルによって、接続されたプリンターがアプリケ

ーション指定のフォントをサポートしていない場合の、置き換えフォントの相互参照が提供されます。

- 「*iSeries 機密保護解説書*, SD88-5027-07」適切な権限をもっていないユーザーによってシステムおよびデータが使用されないように保護し、意識的または無意識によるデータの損傷や破損からデータを保護し、情報を最新に維持し、システム上での機密保護のセットアップのために、システム機密保護サポートをどのように使用できるかを説明しています。
- 「ソフトウェアの導入, SD88-5002-06」ライセンス・プログラム、プログラム一時修正 (PTF)、および IBM からの 2 次言語を導入する、初期導入のためのステップごとの手順を提供します。またこの解説書は、既にあるリリースが導入された iSeries システムに新しいリリースを導入するユーザーにも役立ちます。
- 「*Tape and Diskette Device Programming*, SC41-5716-02」入出力用にテープおよびディスケット装置を使うプログラムの開発やサポートをするユーザーを援助する情報を提供します。それには、装置ファイルの情報およびテープやディスケット装置の説明が含まれています。
- 「*Who Knew You Could Do That with RPG IV? A Sorcerer's Guide to System Access and More*」RPG IV および統合化言語環境 (ILE) の利点を最大に生かそうとしている iSeries システム・プログラマーに対するヒントを提供します。これは、次の IBM Redbooks Web サイトで利用できます。

  `http://www.redbooks.ibm.com/`

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

[サ行]

[タ行]

[ナ行]

[ハ行]

［マ行］

[ヤ行]

[ラ行]

[ワ行]

[数字]

A

B

C

D

E

F

G

I

J

L

M

N

O

P

Q

R

S

T

U

W

X

［特殊文字］

プログラム番号: 5722-WDS

Printed in Japan

SD88-5042-04

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12