

IBM i
バージョン 7.3

プログラミング
IBM Rational Development
Studio for i ILE RPG
プログラマーの手引き

IBM

IBM i
バージョン 7.3

プログラミング
IBM Rational Development
Studio for i ILE RPG
プログラマーの手引き

IBM

注

本書および本書で紹介する製品をご使用になる前に、A-1 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 Rational Development Studio for i (製品番号 5770-WDS)、および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。このバージョンは、すべての RISC モデルで稼働するとは限りません。また CISC モデルでは稼働しません。

本書にはライセンス内部コードについての参照が含まれている場合があります。ライセンス内部コードは機械コードであり、IBM 機械コードのご使用条件に基づいて使用権を許諾するものです。

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： IBM i
Version 7.3
Programming
IBM Rational Development Studio for i
ILE RPG Programmer's Guide

発行： 日本アイ・ビー・エム株式会社

担当： トランスレーション・サービス・センター

© Copyright IBM Corporation 1994, 2016.

目次

ILE RPG プログラマーの手引き

「ILE RPG プログラマーの手引き」につ

いて. 1-1

本書の対象読者. 1-1

前提条件および関連情報. 1-1

コメントの送付方法. 1-1

新着情報. 2-1

I このリリースにおける新しい機能. 2-1

7.2 での変更点. 2-4

7.1 での変更点. 2-12

6.1 での変更点. 2-16

V5R4 での変更点. 2-20

V5R3 での変更点. 2-24

V5R2 での変更点. 2-29

V5R1 での変更点. 2-32

V4R4 での変更点. 2-37

V4R2 での変更点. 2-41

V3R7 での変更点. 2-45

V3R6/V3R2 での変更点. 2-49

ILE RPG の概要. 3-1

RPG IV プログラミング言語の概要. 3-1

RPG IV 仕様書. 3-1

サイクル・プログラミング. 3-2

サブプロシージャーの論理. 3-4

標識. 3-4

命令コード. 3-4

ILE RPG プログラムの例. 3-5

IBM i の使用. 3-10

システムとの対話. 3-10

Rational Development Studio for i. 3-11

ILE における RPG プログラミング. 3-11

プログラムの作成. 3-12

プログラムの管理. 3-13

プログラムの呼び出し. 3-14

ソースのデバッグ. 3-15

バインド可能な API. 3-15

マルチスレッド・アプリケーション. 3-15

プログラムの作成方針. 3-17

方針 1: OPM 互換アプリケーション・プログラ

ム. 3-17

方法. 3-17

OPM 互換プログラムの例. 3-17

ILE の影響. 3-18

関連情報. 3-18

方針 2: CRTBNDRPG を使う ILE プログラム. 3-19

方法. 3-19

CRTBNDRPG を使った ILE プログラムの例. 3-19

ILE の影響. 3-20

関連情報. 3-21

方針 3: CRTRPGMOD を使う ILE アプリケー

ション. 3-21

方法. 3-21

単一言語の ILE アプリケーション・シナリ

オ. 3-22

混合言語の ILE アプリケーション・シナリ

オ. 3-23

拡張アプリケーション・プログラム・シナリ

オ. 3-24

関連情報. 3-25

避けるべき方針. 3-25

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. 3-26

複数プロシージャー・モジュール — 概要. 3-27

メイン・プロシージャーおよびサブプロシー

ジャー. 3-27

プロトタイプ呼び出し. 3-28

複数プロシージャーのあるモジュール例. 3-30

ARRSRPT プログラム全体. 3-34

コーディング上の考慮事項. 3-38

一般的な考慮事項. 3-38

プログラムの作成. 3-38

メイン・プロシージャーの考慮事項. 3-39

サブプロシージャーの考慮事項. 3-39

詳細情報について. 3-40

メイン・プロシージャー. 3-40

サブプロシージャー. 3-40

プロトタイプ呼び出し. 3-41

ILE RPG アプリケーション・プログラム

の作成と実行. 4-1

ソース・ファイルの使用. 4-1

ソース物理ファイルの使用. 4-1

ライブラリーおよびソース物理ファイルの作

成. 4-1

原始ステートメント入力キューティリティー

(SEU) の使用. 4-2

SQL ステートメントの使用. 4-5

IFS ソース・ファイルの使用. 4-7

組み込みファイル. 4-7

IFS 内での検索パス. 4-8

CRTBNDRPG コマンドによるプログラムの作成. 4-10

CRTBNDRPG コマンドの使用. 4-10

ソース・デバッグ用のプログラムの作成. 4-12

静的バインドによるプログラムの作成. 4-13

OPM 互換プログラム・オブジェクトの作成. 4-14

コンパイラー・リストの使用. 4-15

コンパイラー・リストの作成. 4-16

コンパイラー・リストのカスタマイズ. 4-17

ページ見出しのカスタマイズ	4-17	活動化グループの管理	4-57
スペーシングのカスタマイズ	4-17	活動化グループの指定	4-58
構造化命令の字下げ	4-18	OPM デフォルト活動化グループでの実行	4-59
コンパイル・エラーの訂正	4-19	OPM RPG/400 と ILE RPG プログラムの互換性の維持	4-59
インライン診断メッセージの使用	4-19	活動化グループの削除	4-59
追加の診断メッセージの使用	4-20	資源再利用コマンド	4-60
SEU を使用したコンパイラー・リストのブラウズ	4-20	動的に割り振られた記憶域の管理	4-60
実行時エラーの訂正	4-21	RPG 命令を使用したデフォルト・ヒープの管理	4-62
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	4-21	ヒープ記憶域の問題	4-67
保守のためのコンパイラー・リストの使用	4-21	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	4-68
RETURNCODE データ域のアクセス	4-22	プログラムおよびプロシーチャーの呼び出し	4-75
CRTRPGMOD および CRTPGM コマンドによるプログラムの作成	4-24	プログラム / プロシーチャー呼び出しの概要	4-76
モジュール・オブジェクトの作成	4-24	プログラムの呼び出し	4-77
CRTRPGMOD コマンドの使用	4-25	プロシーチャーの呼び出し	4-77
NOMAIN モジュールの作成	4-27	呼び出しスタック	4-78
ソース・デバッグのためのモジュールの作成	4-30	再帰呼び出し	4-78
追加の例	4-32	パラメーターの受け渡しについての考慮事項	4-80
バインドされた ILE RPG モジュールの動作	4-32	プロトタイプ呼び出しの使用	4-82
関連する CL コマンド	4-32	CALLP 命令の使用	4-83
プログラムへのモジュールのバインド	4-33	式の中での呼び出し	4-83
CRTPGM コマンドの使用	4-34	自由形式呼び出しの例	4-84
複数モジュールのバインド	4-36	プロトタイプ・パラメーターの受け渡し	4-84
追加の例	4-37	パラメーターの受け渡しスタイル	4-84
関連する CL コマンド	4-37	参照による受け渡し	4-85
バインダー・リストの使用	4-37	値による受け渡し	4-85
モジュールまたはプログラムの変更	4-38	読み取り専用参照による受け渡し	4-85
UPDPGM コマンドの使用	4-39	値または読み取り専用参照による受け渡しの利点	4-85
最適化レベルの変更	4-39	パラメーターの受け渡しスタイルの選択	4-86
プログラム識別情報の除去	4-40	操作記述子の使用	4-87
オブジェクト・サイズの縮小	4-40	パラメーターの省略	4-88
サービス・プログラムの作成	4-41	*OMIT の受け渡し	4-89
サービス・プログラムの概要	4-41	パラメーターの省略	4-89
サービス・プログラム作成の方針	4-42	渡されるパラメーターの数の検査	4-90
CRTSRVPGM を使用したサービス・プログラムの作成	4-42	%PARMS の使用	4-90
サービス・プログラムの変更	4-43	必要なデータより少ないデータの受け渡し	4-95
関連する CL コマンド	4-43	ファイル・パラメーターの受け渡し	4-96
サンプル・サービス・プログラム	4-44	評価の順序	4-96
サービス・プログラムの作成	4-47	言語間呼び出し	4-97
プログラムへのバインド	4-48	言語間呼び出しに関する考慮事項	4-98
サービス・プログラムの更新	4-49	固定形式の呼び出し命令の使用	4-98
サンプル・バインダー・リスト	4-50	CALL および CALLB の例	4-100
プログラムの実行	4-51	PARM および PLIST を使用したパラメーターの受け渡し	4-100
CL CALL コマンドを使用したプログラムの実行	4-52	PARM 命令の使用	4-100
CL CALL コマンドを使用したパラメーターの受け渡し	4-52	PLIST 命令の使用	4-101
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	4-54	呼び出されたプログラムまたはプロシーチャーからの戻り	4-101
ユーザー作成コマンドを使用したプログラムの実行	4-56	メイン・プロシーチャーからの戻り	4-102
実行時照会メッセージに対する応答	4-56	サイクル・メイン・プロシーチャーの正常終了	4-102
ILE プログラムの終了	4-57	サイクル・メイン・プロシーチャーの異常終了	4-103

サイクル・メイン・プロシージャーの未 終了の戻り	4-103
サブプロシージャーからの戻り	4-104
ILE バインド可能 API を使う戻り	4-104
バインド可能 API の使用	4-105
バインド可能 API の使用例	4-106
グラフィックス・ルーチンの呼び出し	4-106
特殊なルーチンの呼び出し	4-106
記憶域モデル	4-107
単一レベル記憶域モデルに関する考慮事項	4-107
テラスペース記憶域モデルに関する考慮事 項	4-107
継承記憶域モデルに関する考慮事項	4-107
プログラムとサービス・プログラムの記憶 域モデルに関する推奨事項	4-108
マルチスレッド化に関する考慮事項	4-108
複数のスレッドでの同時実行	4-108
複数のスレッドでの逐次化の実行	4-109
THREAD キーワードに関する活動化グルー プの考慮事項	4-109
複数のスレッド間で共有される記憶域	4-110
モジュール間のデッドロックの防止方法	4-110
All-Thread 静的変数	4-111
逐次化されたプロシージャーを使用する場 合	4-112
逐次化されたプロシージャーが十分な保護 を提供しない場合	4-113
共用リソースへのアクセスを手動で同期化 する場合の問題	4-114
スレッド関連の API の使用	4-114
例を作成する方法	4-114
例を実行する方法	4-114
RPG と e-business の世界	4-126
RPG と XML	4-126
XML 文書の処理	4-127
XML パーサー・エラー・コード	4-128
XML パーサーの制限	4-130
RPG と MQSeries	4-132
RPG と Java	4-132
Java と RPG についての概要	4-132
オブジェクト・データ・タイプおよび CLASS キーワード	4-133
Java メソッドのプロトタイピング	4-133
Java メソッドの ILE RPG からの呼び出し	4-136
オブジェクトの作成	4-141
独自クラスでのメソッドの呼び出し	4-142
Java 仮想マシンのセットアップ方法の制御	4-143
RPG ネイティブ・メソッド	4-144
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	4-146
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	4-146
RPG から Java を呼び出したときのコーデ ィング・エラー	4-147
RPG プロトタイプでのメソッド・パラメ ーターの不正確な指定	4-147

Java リソースの解放エラー	4-147
存在しないオブジェクトの使用	4-148
Java を使用するためのその他の RPG コー ディング	4-149
複数のオブジェクトの同時解放を Java に通知	4-149
一時オブジェクトの使用が終了したこと を Java に伝える	4-150
オブジェクトを永続にしたいことを Java に伝える	4-151
永続オブジェクトの使用が終了したこと を Java に伝える	4-152
Java 仮想マシン (JVM) の作成	4-152
JNI 環境ポインターの入手	4-152
JNI 例外の処理	4-156
追加の考慮事項	4-157
一般的な実行時エラー	4-157
デバッグのヒント	4-157
RPG での String オブジェクトの作成	4-157
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	4-158
拡張 JNI コーディング	4-158
jvalue 構造でのオブジェクト参照の設定	4-159
Java 文字データの変換	4-161
Java クラスの中のフィールドへのアクセ ス	4-161
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	4-163
PCML の使用による Java からの RPG プ ログラムの呼び出し	4-164
PCML の制約事項	4-165
デバッグおよび例外処理	5-1
プログラムのデバッグ	5-1
ILE ソース	5-2
デバッグ・コマンド	5-3
デバッグのためのプログラムの準備	5-4
ルート・ソース・ビューの作成	5-6
コピー・ソース・ビューの作成	5-6
リスト・ビューの作成	5-7
ステートメント・ビューの作成	5-8
ILE ソースの開始	5-8
STRDBG の例	5-9
デバッグ・オプションの設定	5-10
デバッグ・セッションでのプログラムの追加/除 去	5-11
デバッグ・セッションへのサービス・プログ ラムの追加例	5-11
ILE プログラムのデバッグ・セッションから の除去例	5-12
プログラム・ソースの表示	5-13
別のモジュールの表示	5-13
モジュールのビューの変更	5-15
ブレークポイントの設定と除去	5-16
無条件ジョブ・ブレークポイントの設定およ び除去	5-17

無条件ジョブ・ブレイクポイントの設定例	5-18	ILE RPG 例外処理	5-61
無条件スレッド・ブレイクポイントの設定および除去	5-19	サイクル・メイン・プロシーチャー内の例外処理プログラム	5-61
条件付きジョブ・ブレイクポイントの設定および除去	5-20	サブプロシーチャー内の例外処理プログラム	5-62
F13 を使った条件付きジョブ・ブレイクポイントの設定例	5-21	OPM と ILE RPG 例外処理との違い	5-63
BREAK コマンドを使用した条件付きジョブ・ブレイクポイントの設定例	5-21	例外処理プログラムの使用	5-63
国別言語分類順序 (NLSS)	5-22	例外処理プログラムの優先順位	5-64
ステートメント番号を使用したジョブ・ブレイクポイントの設定および除去	5-23	ネストされた例外	5-64
条件付きスレッド・ブレイクポイントの設定および除去	5-25	未処理例外	5-64
「モジュール・ブレイクポイントの処理」画面の使用	5-25	未処理エスケープ・メッセージの例	5-64
TBREAK または CLEAR デバッグ・コマンドの使用	5-26	未処理機能チェックの例	5-65
ジョブとスレッドすべてのブレイクポイントの除去	5-26	最適化に関する考慮事項	5-66
ウォッチ条件の設定および除去	5-26	RPG 特有の処理プログラムの使用	5-67
ウォッチの特性	5-26	エラー標識または 'E' 命令コード拡張の指定	5-67
ウォッチ条件の設定	5-27	MONITOR グループの使用	5-68
WATCH コマンドの使用	5-28	エラー処理サブルーチンの使用	5-70
活動状態のウォッチの表示	5-29	ファイル・エラー処理 (INFSR) サブルーチンの使用	5-71
ウォッチ条件の除去	5-29	プログラム・エラー処理サブルーチンの使用	5-73
ウォッチ条件の設定例	5-30	エラー処理サブルーチンでのループの防止	5-77
プログラム・オブジェクトのステップスルー	5-31	ENDSR 命令での戻り点の指定	5-79
呼び出しステートメントのステップオーバー	5-32	ILE 条件処理プログラム	5-80
呼び出しステートメントへのステップイン	5-33	条件処理プログラムの使用	5-80
F22 を使用した OPM プログラムのステップインの例	5-34	取り消し処理プログラムの使用	5-86
サブプロシーチャーへのステップインの例	5-35	ILE CL が通知および状況メッセージを監視する際の問題	5-89
データおよび式の表示	5-36	ダンプの入手	5-91
変数評価時の予期しない結果	5-38	ILE RPG 定様式ダンプの入手	5-91
配列の内容の表示	5-39	DUMP 命令コードの使用	5-92
テーブルの内容の表示	5-39	定様式ダンプの例	5-92
データ構造の表示	5-40	ファイルおよび装置の処理	6-1
標識の表示	5-42	ファイルの定義	6-1
16 進値としてのフィールドの表示	5-42	ファイル記述と入出力装置との関連付け	6-1
文字形式でのフィールドの表示	5-42	ファイルの名前指定	6-3
UCS-2 データの表示	5-43	ファイル記述のタイプ	6-3
可変長フィールドの表示	5-43	外部記述のファイルのプログラム記述としての使用	6-4
ポインターがアドレス指定するデータの表示	5-43	プログラムとファイルとの代表的な関係の例	6-5
基底付き変数の評価	5-44	外部記述ファイルの定義	6-5
ヌル可能フィールドの表示	5-45	レコード様式の名前変更	6-6
デバッグ組み込み関数の使用	5-46	フィールド名の変更	6-6
XML-SAX 処理手順のデバッグ	5-47	レコード様式の無視	6-7
フィールドの値の変更	5-48	入力仕様書を使用した外部記述の変更	6-8
フィールドの属性の表示	5-50	出力仕様書の使用	6-9
フィールド、式、またはコマンドと名前の等値化	5-51	レベル検査	6-10
ILE RPG のソース・デバッグの各国語サポート	5-52	プログラム記述ファイルの定義	6-11
デバッグ用サンプル・ソースの例	5-52	データ管理命令と ILE RPG 入出力命令	6-11
例外の処理	5-57	ファイルに関する一般的な考慮事項	6-13
例外処理の概要	5-58	ファイル入出力の一時変更および指定変更	6-13
		ファイル入出力の指定変更の例	6-14
		ファイルのロック	6-15
		レコードのロック	6-16
		オープン・データ・パスの共用	6-17

スプーリング	6-18	フェッチ・オーバーフローの指定	6-59
出力スプーリング	6-19	フェッチ・オーバーフローの指定例	6-59
RPG プログラム対 DDS ファイルの		プログラム記述ファイル中の用紙制御情報の	
SRTSEQ/ALTSEQ	6-19	変更	6-60
データベース・ファイルのアクセス	6-20	用紙制御情報の変更例	6-61
データベース・ファイル	6-20	テープ装置のアクセス	6-61
物理ファイルおよび論理ファイル	6-20	表示装置へのアクセス	6-62
データ・ファイルとソース・ファイル	6-20	順次ファイルの使用	6-62
外部記述ディスク・ファイルの使用	6-21	順次ファイルの指定	6-62
レコード様式仕様書	6-21	順次ファイルの指定例	6-62
アクセス・パス	6-21	SPECIAL ファイルの使用	6-63
レコードまたはファイルに有効なキー	6-24	SPECIAL ファイルの使用例	6-65
有効な検索指数	6-25	WORKSTN ファイルの使用	6-66
部分キーの参照	6-25	システム間通信機能	6-66
レコードのブロック化および非ブロック化	6-26	外部記述 WORKSTN ファイルの使用	6-66
プログラム記述ディスク・ファイルの使用	6-27	表示装置ファイルでの機能キー標識の指定	6-68
索引付きファイル	6-27	表示装置ファイルでのコマンド・キーの指定	6-69
有効な検索指数	6-28	外部記述 WORKSTN ファイルの処理	6-69
順次ファイル	6-29	サブファイルの使用	6-70
レコード・アドレス・ファイル	6-30	サブファイルの使用	6-72
限界値レコード	6-30	プログラム記述 WORKSTN ファイルの使用	6-73
相対レコード番号	6-30	様式名のあるプログラム記述 WORKSTN フ	
ディスク・ファイルの処理方式	6-31	ァイルの使用	6-74
連続処理	6-32	出力仕様書	6-74
キーによる順次処理	6-33	入力仕様書	6-74
キーによる順次処理の例	6-33	演算仕様書	6-75
キーによるランダム処理	6-38	追加の考慮事項	6-75
キーによるランダム処理の例	6-39	様式名のないプログラム記述 WORKSTN フ	
限界内順次処理	6-40	ァイルの使用	6-75
限界内順次処理の例	6-41	入力ファイル	6-75
相対レコード番号による処理	6-42	出力ファイル	6-75
有効なファイル命令	6-43	入出力共用ファイル	6-75
コミットメント制御の使用	6-45	有効な WORKSTN ファイル命令	6-76
コミットメント制御の開始および終了	6-46	EXFMT 命令	6-76
コミットメント制御のロック	6-46	READ 命令	6-76
コミットメント制御の有効範囲	6-47	WRITE 命令	6-76
コミットメント制御用のファイルの指定	6-47	複数装置ファイル	6-76
COMMIT 命令の使用	6-48	対話式アプリケーションの例	6-78
コミットメント制御の使用例	6-48	データベース物理ファイル	6-79
条件付きコミットメント制御の指定	6-49	メイン・メニュー照会	6-79
プログラム・サイクルでのコミットメント制		MAINMENU: 表示装置ファイルの DDS	6-79
御	6-50	CUSMAIN: RPG ソース	6-81
キー付きファイルを使用した予期しない結果	6-51	ファイルの維持	6-82
DDM ファイル	6-51	CUSMSTL1: 論理ファイルの DDS	6-83
V3R1 より前の DDM ファイルの使用	6-52	MNTMENU: 表示装置ファイルの DDS	6-84
外部接続装置へのアクセス	6-52	CUSMNT: RPG ソース	6-86
装置ファイルのタイプ	6-52	郵便番号による検索	6-93
プリンターのアクセス	6-53	CUSMSTL2: 論理ファイルの DDS	6-94
PRINTER ファイルの指定	6-53	SCHZIPD: 表示装置ファイルの DDS	6-95
ページ・オーバーフローの処理	6-54	SCHZIP: RPG ソース	6-97
プログラム記述ファイルでのオーバーフロ		名前による検索と照会	6-101
ー標識の使用	6-55	CUSMSTL3: 論理ファイルの DDS	6-102
すべてのページへの見出しの印刷例	6-56	SCHNAMD: 表示装置ファイルの DDS	6-103
すべてのページへのフィールドの印刷例	6-56	SCHNAM: RPG ソース	6-106
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	6-57		

付録. 7-1

付録 A. OPM RPG/400 と AS/400 用 ILE RPG との動作上の相違点	7-1
コンパイル	7-1
実行	7-1
デバッグおよび例外処理	7-3
入出力	7-3
文字フィールドの DBCS データ	7-6
付録 B. RPG III から RPG IV への変換援助プログラムの使用	7-6
変換の概要	7-7
ファイルに関する考慮事項	7-7
ソース・メンバー・タイプ	7-7
ファイルのレコード長	7-8
ファイル名およびメンバー名	7-8
ログ・ファイル	7-9
変換援助プログラム・ツールの要件	7-9
変換援助プログラムが行わないこと	7-9
ソースの変換	7-9
CVTRPGSRC コマンド	7-10
デフォルトの値を使用したメンバーの変換	7-15
1 ファイル中のすべてのメンバーの変換	7-16
1 ファイル中のいくつかのメンバーの変換	7-16
試行変換の実行	7-16
変換報告書の入手	7-17
報告書簡易作成機能ソース・メンバーの変換	7-17
組み込み SQL をもつソース・メンバーの変換	7-18
仕様書テンプレートの挿入	7-18
データ・ファイルからのソースの変換	7-18
ソース変換の例	7-18
変換の分析	7-21
変換報告書の使用	7-21
ログ・ファイルの使用	7-24
変換上の問題の分析解決	7-25
既存の RPG III コードのコンパイル・エラー	7-26
サポートされていない RPG III 機能	7-26
FREE 命令コードの変換	7-26
/COPY コンパイラー指示ステートメントの使用	7-27

組み合わせ問題	7-27
文脈に依存した問題	7-29
外部記述データ構造の使用	7-30
配列と外部記述 DS サブフィールドの組み合わせ	7-30
外部記述 DS サブフィールドの名前変更と初期化	7-31
実行時の相違点	7-32
付録 C. 作成コマンド	7-32
CL コマンドの使用	7-32
構文図の解釈法	7-33
CRTBNDRPG コマンド	7-34
CRTBNDRPG コマンドの説明	7-37
CRTRPGMOD コマンド	7-52
CRTRPGMOD コマンドの説明	7-55
付録 D. コンパイラー・リスト	7-56
コンパイラー・リストの読み方	7-57
プロローグ	7-57
ソース・セクション	7-59
追加の診断メッセージ	7-64
出力バッファー位置	7-65
/COPY メンバー・テーブル	7-65
コンパイル時データ	7-65
キー・フィールド情報	7-66
相互参照表	7-67
EVAL-CORR 要約	7-68
外部参照リスト	7-69
メッセージの要約	7-70
最終の要約	7-70
コード生成およびバインド・エラー	7-71
付録 E. プリプロセッサ・プロバイダーの情報	7-71

参考文献. 8-1

特記事項. A-1

プログラミング・インターフェース情報	A-3
商標	A-3
使用条件	A-3

索引 X-1

ILE RPG プログラマーの手引き

本書は、Integrated Language Environment[®] 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、 IBM[®] i (IBM i) オペレーティング・システムを使用する
IBM i 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。

注: このガイドには、いくつかの画面取りがあり、iSeries への古い参照、および以前のリリースからの用語が含まれている場合があります。

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 形式へのプログラムの変換
- コンパイラー・リストの解釈

「ILE RPG プログラマーの手引き」について

本書の概要については、このセクションをお読みください。

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、IBM i 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。言語の詳細な説明については、「*IBM Rational Development Studio for i: ILE RPG 解説書 (SD88-5043-08)*」を参照してください。

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 IBM i 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 (SD88-5033-09)*」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

IBM i 技術情報を検索する手始めとして、IBM i Information Center を使用します。Information Center には次の Web サイトからアクセスできます。

<http://www.ibm.com/systems/i/infocenter/>

Information Center には、ソフトウェアのインストール、Linux、WebSphere®、Java™、高可用性、データベース、論理区画、CL コマンド、およびシステム・アプリケーション・プログラミング・インターフェース (API) などの新規システム情報と更新済みのシステム情報が含まれています。さらに、システムのハードウェアとソフトウェアの計画、トラブルシューティング、および構成時に役立つ推奨事項と探索機能も提供します。

ILE RPG コンパイラー に最も関連の深い資料については、8-1 ページの『参考文献』にリストされています。

コメントの送付方法

IBM にお客様のご意見をお寄せください。本書や他の IBM i の資料に関するご意見をお持ちの際は、IBM 担当員にお知らせいただくか、次のいずれかの方法をお選びください。

- 郵便の場合は下記あてにご送付ください。
-
- 電子メールの場合は、以下の E メール・アドレスです。
 - 資料に関するご意見

RCHCLERK@us.ibm.com

– IBM i Information Center に関するご意見:

RCHINFOC@us.ibm.com

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

V3R1 以降の ILE RPG コンパイラーの各リリースでの新機能および変更された機能

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。

- | • 『このリリースにおける新しい機能』
- # • 2-4 ページの『7.2 での変更点』
 - 2-12 ページの『7.1 での変更点』
 - 2-16 ページの『6.1 での変更点』
 - 2-20 ページの『V5R4 での変更点』
 - 2-24 ページの『V5R3 での変更点』
 - 2-29 ページの『V5R2 での変更点』
 - 2-32 ページの『V5R1 での変更点』
 - 2-37 ページの『V4R4 での変更点』
 - 2-41 ページの『V4R2 での変更点』
 - 2-45 ページの『V3R7 での変更点』
 - 2-49 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: この製品の情報は、RPG IV の 7.3 リリース時点で最新のものです。コンパイラーの前のリリースを
- | 使用している場合は、ご使用のシステムでサポートされている機能を確認する必要があります。例えば、
- | 7.1 システムを使用している場合、7.3 リリースの新機能はサポートされません。

このリリースにおける新しい機能

- | このセクションでは、ILE RPG 7.3 で行われた機能強化について説明します。

| 完全自由形式ソースのサポート

- | 1 行目に特殊ディレクティブ ****FREE** が指定された RPG ソースには、自由形式コードのみが含まれます。コードは 1 桁目から開始し、行の終わりまで拡張できます。
- | 完全自由形式ソースの場合、ソース行の長さに実際的制限はありません。
- | 完全自由形式ソースでは固定形式コードは許可されませんが、**/COPY** ディレクティブまたは **/INCLUDE** ディレクティブを使用して、6 桁から 80 桁のみを使用する、桁制限ソースを組み込むことができます。

| 新しい組み込み関数 **%SCANR** (逆方向の走査)

- | **%SCANR** 組み込み関数は **%SCAN** 組み込み関数に似ていますが、この関数が検出するのは、検索引数の最初のオカレンスではなく、最後のオカレンスです。
- | 以下の例では、**%SCAN** を使用してストリング内の **"***"** の最初のオカレンスを検出し、**%SCANR** を使用して最後のオカレンスを検出しています。


```
string = 'The title is *** Chapter 1 ***.';
p1 = %SCAN ('***' : string);
p2 = %SCANR ('***' : string);
// p1 = 14
// p2 = 28
```

組み込み関数 %SCAN の長さパラメーター

長さパラメーターにより、検索するソース・ストリングの量を制限することができます。

以下の例では、最初の %SCAN 組み込み関数は 26 を返します。2 番目の %SCAN 組み込み関数は 0 を返します。この理由は、値 "abc" が、開始位置 1 と長さ 10 によって示されたサブストリング内で検出されないためです。

```
string = 'The alphabet begins with abc.';
p1 = %SCAN ('abc' : string);
p2 = %SCAN ('abc' : string : 1 : 10);
// p1 = 26
// p2 = 0
```

ファイルの拡張 ALIAS サポート

どの外部記述ファイルにも ALIAS キーワードを指定できるようになりました。

修飾されていないグローバル・ファイルに対して ALIAS キーワードが指定された場合、RPG プログラム内でフィールドの代替名が使用可能になります。

以下の例で、MYFILE ファイル内の REQALC フィールドには *REQUIRED_ALLOCATION* という代替名があります。ALIAS キーワードは、RPG プログラム内でのこのフィールドの名前が *REQUIRED_ALLOCATION* になることを示しています。

```
dcl-f myfile ALIAS;

read myfile;
if required_allocation <> 0
and size > 0;
...
```

入出力操作でのデータ構造の規則の緩和

- タイプ *ALL によって定義された外部記述データ構造または LIKEREK データ構造は、どの入出力操作の結果データ構造としても使用できます。

```
dcl-f myfile usage(*input : *output : *update);
dcl-ds ds extname('MYFILE' : *ALL);

read myfile ds;
update myfmt ds;
write myfmt ds;
```

- データ構造が、DISK ファイルのレコード様式に対して、2 番目のパラメーターなしの LIKEREK を使用して定義されており、出力バッファのレイアウトが入力バッファのレイアウトと全く同じ場合、そのデータ構造はどの入出力操作の結果データ構造としても使用できま

す。

```
dcl-f myfile usage(*input : *output : *update);
dcl-ds ds likerec(fmt);

read myfile ds;
update myfmt ds;
write myfmt ds;
```

ヌル可能フィールドに関連した機能拡張

- データ構造が EXTNAME キーワードまたは LIKERECE キーワードによって定義されている場合、サブフィールドがすべて標識であることを指定して、*NULL を追加の抜き出しタイプとしてコーディングすることができます。外部ファイルがデータベース・ファイルの場合、結果データ構造は、そのファイルのヌル・バイト・マップに一致します。
- NULLIND キーワードを使用して以下のことを実行します。
 - フィールドをヌル可能として定義する
 - 独自の標識フィールドを、フィールドのヌル標識になるように定義する
 - EXTNAME(*NULL) または LIKERECE(*NULL) によって定義された独自の標識データ構造を、別のデータ構造のヌル標識になるように定義する

PCML の機能拡張

- PGMINFO 制御仕様書キーワードの *DCLCASE パラメーターを指定して、プログラム・インターフェース情報内の名前が、RPG ソース・ファイルに定義されている名前と同じケース (大 / 小文字) で生成されるようにします。
- モジュールの作成時にプログラム・インターフェース情報に組み込むプロシージャに対してプロシージャ仕様書キーワードに PGMINFO(*YES) を指定するか、組み込まないプロシージャに対して PGMINFO(*NO) を指定します。

DCLOPT(*NOCHGDSLEN)

DCLOPT(*NOCHGDSLEN) を指定して、入力仕様書、出力仕様書、または演算仕様書を使用してデータ構造の長さが変更されないようにします。DCLOPT(*NOCHGDSLEN) を指定すると、より自由な形式の宣言で %SIZE(data-structure) を使用することができます。

表 2-1. 7.2 以降に変更された言語要素: 制御仕様書キーワード

要素	説明
PGMINFO キーワード	*DCLCASE プログラム・インターフェース情報内の名前を、RPG ソース・ファイルに定義されている名前と同じケース (大 / 小文字) で生成するための *DCLCASE パラメーター。

表 2-2. 7.2 以降に変更された言語要素: ファイル仕様書キーワード

要素	説明
ALIAS キーワード	すべての外部記述ファイルで許可されます。

表 2-3. 7.2 以降に変更された言語要素: 定義仕様書キーワード

要素	説明
EXTNAME キーワード	抜き出しタイプ *NULL。
LIKERECE キーワード	抜き出しタイプ *NULL。

新着情報

表 2-4. 7.2 以降で変更された言語要素: 組み込み関数

要素	説明
%SCAN 組み込み関数	%SCAN 組み込み関数は、検索対象の長さを示す、4 番目のパラメーターをサポートするようになりました。

表 2-5. 7.2 以降の新しい言語要素: ディレクティブ

要素	説明
特殊ディレクティブ **FREE	**FREE は、ソースが完全自由形式、すなわち 1 桁からソース行の最後までが RPG コードであることを示しています。

表 2-6. 7.2 以降の新しい言語要素: 制御仕様書キーワード

要素	説明
DCLOPT (*NOCHGDSLEN) キーワード	入力仕様書、出力仕様書、または演算仕様書を使用したデータ構造のサイズ変更を不許可にします。

表 2-7. 7.2 以降の新しい言語要素: 定義仕様書キーワード

要素	説明
NULLIND キーワード	ある項目を別の項目のヌル標識 (複数可) として関連付けます。

表 2-8. 7.2 以降の新しい言語要素: プロシージャ仕様書キーワード

要素	説明
PGMINFO キーワード	モジュールの作成中にどのプロシージャのインターフェースをプログラム・インターフェース情報内で記述するかを制御できます。

表 2-9. 7.2 : 移行の新しい言語要素: 組み込み関数

要素	説明
%SCANR (逆方向の走査)	別のストリング内で、あるストリングの最後のオカレンスを検出します。

7.2 での変更点

この節では、7.2 で ILE RPG に対して行われた機能強化について説明します。

自由形式の制御ステートメント、ファイル・ステートメント、定義ステートメント、およびプロシージャ・ステートメント

- # 自由形式の制御ステートメントは CTL-OPT で始まり、セミコロンで終わります。

```
# CTL-OPT OPTION(*SRCSTMT : *NODEBUGIO)  
# ALWNULL(*USRCTL);
```

- # 自由形式のファイル定義ステートメントは DCL-F で始まり、セミコロンで終わります。

以下のステートメントは 3 つのファイルを定義します。

- # 1. 入力および更新用にオープンされた外部記述 DISK ファイル。
- # 2. 入出力用にオープンされた外部記述 WORKSTN ファイル。

```

#      3. レコード長が 132 のプログラム記述 PRINTER ファイル。
#
#      DCL-F custFile usage(*update) extfile(custFilename);
#      DCL-F screen workstn;
#      DCL-F qprint printer(132) oflind(qprintOfFlow);
#
#      • 自由形式のデータ定義ステートメントは DCL-C、DCL-DS、DCL-PI、DCL-PR、または DCL-S
#      で始まり、セミコロンで終わります。
#
#      以下のステートメントはいくつかの項目を定義します。
#
#      1. 名前付き固定情報 MAX_ELEMS。
#
#      2. 独立の可変長文字フィールド fullName。
#
#      3. 整数サブフィールド num および UCS-2 サブフィールド address を持つ修飾されたデータ構
#      造。
#
#      4. プロシージャ「Qp0lRenameUnlink」のプロトタイプ。
#
#      DCL-C MAX_ELEMS 1000;
#      DCL-S fullName VARCHAR(50)
#              INZ('Unknown name');
#      DCL-DS ds1 QUALIFIED;
#              num INT(10);
#              address UCS2(100);
#      END-DS;
#      DCL-PR Qp0lRenameUnlink INT(10) EXTPROC(*DCLCASE);
#              oldName POINTER VALUE OPTIONS(*STRING);
#              newName POINTER VALUE OPTIONS(*STRING);
#      END-PR;
#
#      • 自由形式のプロシージャ定義ステートメントは DCL-PROC で始まり、セミコロンで終わります。
#      END-PROC ステートメントは、プロシージャを終了するために使用されます。
#
#      以下に、自由形式のサブプロシージャ定義の例を示します。
#
#      DCL-PROC getCurrentUserName EXPORT;
#      DCL-PI *n CHAR(10) END-PI;
#      DCL-S curUser CHAR(10) INZ(*USER);
#
#      RETURN curUser;
#      END-PROC;
#
#      • /FREE ディレクティブと /END-FREE ディレクティブは不要になりました。コンパイラーはこ
#      れらを無視します。
#
#      • 自由形式ステートメントと固定形式ステートメントを混在させることができます。
#
#      IF endDate < beginDate;
#      C      GOTO      internalError
#
#      ENDIF;
#      duration = %DIFF(endDate : beginDate : *days);
#      . . .
#      C      internalError TAG
#
# 英数字データの CCSID サポート
#
#      • モジュールのデフォルトの英数字 CCSID は、UTF-8 および 16 進数を含むさらに多くの
#      CCSID に設定することができます。
#
#      • CCSID を使用して英数字データを定義することができます。サポートされる CCSID には以下
#      のものが含まれます。
#
#      – 1 バイトおよび混合バイトの EBCDIC CCSID
#
#      – 1 バイトおよび混合バイトの ASCII CCSID
#
#      – UTF-8

```

新着情報

- 16 進数

外部英数字サブフィールドの CCSID

英数字サブフィールドにはファイル内のフィールドと同じ CCSID が必要であることを示すには、

外部記述データ構造に CCSID(*EXACT) を使用します。

16 進データでは CCSID 変換は実行されない

16 進データの暗黙的な変換または明示的な変換では、CCSID 変換は許可されません。

16 進データには、以下のものが含まれます。

• 16 進数リテラル

• CCSID(*HEX) を使用して定義された英数字データおよび図形データ

• ファイルで DATA キーワードが有効になっていて、ファイル内のフィールドの CCSID が

65535 である場合は、外部記述ファイルのバッファー内にある英数字データおよび図形データ

• ファイル内のフィールドの CCSID が 65535 である場合は、CCSID(*EXACT) を使用して定義

された外部記述データ構造内にある英数字データおよび図形データ

連結の暗黙的な変換

コンパイラーは、連結式で英数字データ、図形データ、および UCS-2 データの間の暗黙的な変換

を実行します。

ジョブ CCSID への変換を行わずにデータベース・ファイルをオープンする

入出力命令で英数字データと図形データがジョブ CCSID との間で変換されない状態でのデータベ

ース・ファイルのオープンを指定するには、制御キーワード OPENOPT(*NOCVTDATA) またはフ

ァイル・キーワード DATA(*NOCVT) を使用します。

デフォルトの CCSID、日付の形式、または時刻の形式を一時的に変更する

日付の形式、時刻の形式、および CCSID のデフォルト値を設定するには、/SET ディレクティブ

と /RESTORE ディレクティブを使用します。

%SUBDT によって返される長さを制御する

%SUBDT のオプションの 3 番目のパラメーターを使用すると、結果での桁数を指定することがで

きます。例えば、年の値を 4 桁の値で返すことができます (%SUBDT(MyDate:*YEARS:4))。

タイム・スタンプ・データの精度の向上

タイム・スタンプ・データは、0 から 12 までの小数点以下の秒とすることができます。

オープン・アクセス・ファイル

オープン・アクセス・ファイルは、そのすべての操作がオペレーティング・システムではなくユー

ザー作成プログラムまたはプロシージャによって処理されるファイルです。このプログラムまた

はプロシージャは、「オープン・アクセス・ハンドラー」または単に「ハンドラー」と呼ばれま

す。HANDLER キーワードはハンドラーを指定します。

新規の XML-INTO オプション

• XML 名前空間は、「ns」オプションと「nsprefix」オプションによってサポートされます。

• サブフィールド名で RPG によってサポートされない文字が含まれる XML 名は、

「case=convert」オプションによってサポートされます。

ソース文字がターゲット文字セットに存在しない場合にデータが失われる CCSID 変換のサポート

制御仕様書キーワード CCSIDCVT(*EXCP : *LIST)。

• ターゲット文字セット内に一致がないソース文字が原因で CCSID 変換によってデータが失われ

た場合に例外を出すには、CCSIDCVT(*EXCP) を使用します。

• 変換によってデータが失われる可能性があるかどうかを示す診断メッセージとともに、モジュール内のすべての CCSID 変換のリストを取得するには、CCSIDCVT(*LIST) を使用します。

日付データ、時刻データ、およびタイム・スタンプ・データでの作業時に RPG コンパイラーが妥当性検査ステップをスキップできるようにする VALIDATE(*NODATETIME)

RPG コンパイラーが妥当性検査を実行せずに日付データ、時刻データ、およびタイム・スタンプ・データを文字データとして処理できるようにするには、制御仕様書キーワード

VALIDATE(*NODATETIME) を使用します。

これによって、一部の日付操作、時刻操作、およびタイム・スタンプ操作のパフォーマンスが向上
することがあります。

注: 妥当性検査ステップをスキップすると、重大なデータ破損の問題が発生する可能性があります。
この機能は、日付データ、時刻データ、およびタイム・スタンプ・データが常に有効である
ことが確かな場合にのみ使用する必要があります。

表 2-10. 7.2 で変更された言語要素: 制御仕様書キーワード

# 要素	# 説明
# CCSID キーワード	<ul style="list-style-type: none"> • CCSID(*EXACT) は、モジュール内にあるすべての英数字データの CCSID に注意するようにコンパイラーに指示します。 <ul style="list-style-type: none"> – 英数字リテラルと図形リテラルには、ソース・ファイルの CCSID があります。 – 英数字データには常に CCSID があるとみなされます。 <p>CCSID(*EXACT) が指定されていない場合、RPG コンパイラーは、データベース・ファイルのリテラル、変数、または入力バッファーと出力バッファー内のデータの CCSID に関して誤った想定を立てることがあります。</p> <ul style="list-style-type: none"> • CCSID(*CHAR:ccsid) では、*HEX、*JOB RUN MIX、*UTF8、ASCII CCSID、および EBCDIC CCSID がサポートされます。 • CCSID(*GRAPH:ccsid) では、*HEX、*JOB RUN がサポートされます。 • CCSID(*UCS2:ccsid) では、*UTF16 がサポートされます。
# DFTACTGRP キーワード	自由形式の制御仕様書があり、ACTGRP キーワード、BNDDIR キーワード、または STGM DL キーワードの少なくとも 1 つが使用される場合、DFTACTGRP(*NO) が想定されます。
# OPENOPT キーワード	<p>OPENOPT(*{NO}CVT) は、データベース・ファイルの DATA キーワードのデフォルトを制御します。</p> <ul style="list-style-type: none"> • OPENOPT(*CVT DATA) は、ファイルの DATA キーワードが指定されていない場合に、DISK ファイルと SEQ ファイルに対して DATA(*CVT) を想定することを示します。 • OPENOPT(*NOCVT DATA) は、ファイルの DATA キーワードが指定されていない場合に、DISK ファイルと SEQ ファイルに対して DATA(*NOCVT) を想定することを示します。 <p>を参照してください。</p>

表 2-11. 7.2 で変更された言語要素: ディレクティブ

# 要素	# 説明
# /FREE ディレクティブおよび /END-FREE ディレクティブ	自由形式コードの始めと終わりを示すには、これらのディレクティブは不要になりました。これらはコンパイラーによって無視されます。

新着情報

表 2-12. 7.2 で変更された言語要素: 定義仕様書キーワード

#	要素	説明
# # # # #	CCSID キーワード	<ul style="list-style-type: none"> 英数字データでサポートされます。 英数字サブフィールドの CCSID を制御するために、外部記述データ構造でサポートされます。 パラメーターは、グラフィック・データの場合 *HEX および *JOB RUN になります。 パラメーターは、UCS-2 データの場合 *UTF16 になります。
# # # # #	DTAARA キーワード	自由形式の定義内 <ul style="list-style-type: none"> *VAR は使用されません。名前が引用符なしで指定されている場合、変数または名前付き固定情報の名前であると想定されます。 データ構造の場合、これがデータ域データ構造であることを指定するために *AUTO が使用されます。IN 命令、OUT 命令、および UNLOCK 命令を使用してデータ域を操作できることを指定するには、*USRCTL が使用されます。
# # # #	EXTFLD キーワード	自由形式のサブフィールド定義内 <ul style="list-style-type: none"> パラメーターはオプションです。 パラメーターが引用符なしで指定されている場合、以前に定義された名前付き固定情報の名前であると想定されます。
# # #	EXTNAME キーワード	自由形式のデータ構造定義内 <ul style="list-style-type: none"> file-name パラメーターまたは format-name パラメーターが引用符なしで指定されている場合、以前に定義された名前付き固定情報の名前であると想定されます。
# # #	EXPORT キーワード および IMPORT キーワード	自由形式の定義内 <ul style="list-style-type: none"> 外部名が、独立フィールドまたはデータ構造の指定方法と同じ (同じ大/小文字混合) であることを示す *DCLCASE を外部名に指定することができます。
# # # # #	EXTPROC キーワード	自由形式のプロトタイプ定義またはプロシージャ・インターフェース定義内 <ul style="list-style-type: none"> 外部名が、プロトタイプまたはプロシージャ・インターフェースの指定方法と同じ (同じ大/小文字混合) であることを示す *DCLCASE を外部プロシージャまたはメソッド名に指定することができます。 プロシージャ・インターフェース名が *N と指定されている場合、外部名は DCL-PROC ステートメントから取得されます。
# #	LIKE キーワード	自由形式の定義内では、LIKE キーワードには、長さ調整を指定するオプションの 2 番目のパラメーターがあります。
# #	LEN キーワード	自由形式の定義内では、LEN キーワードはデータ構造定義でのみ許可されます。その他の自由形式の定義では、長さはデータ・タイプ・キーワードの一部として指定されます。
# # # # # # #	CLASS キーワード、 DATFMT キーワード、 PROCPTR キーワード、 TIMFMT キーワード、 および VARYING キーワード	これらのキーワードは、自由形式の定義では使用されません。これらのキーワードによって指定される情報は、関連するデータ・タイプ・キーワードの一部として指定されます。
# # # #	FROMFILE キーワード、 PACKEVEN キーワード、 および TOFILE キーワード	これらのキーワードは、自由形式の定義では許可されません。
# #	OVERLAY キーワード	パラメーターは、自由形式のサブフィールド定義のデータ構造の名前にすることはできません。代わりに POS キーワードが使用されます。

#

表 2-13. 7.2 で変更された言語要素: リテラル

# 要素	# 説明
# タイム・スタンプ・リテラル	# タイム・スタンプ・リテラルは、0 から 12 までの小数点以下の秒とすることができます。

表 2-14. 7.2 で変更された言語要素: ステートメントの順序

# 要素	# 説明
# ファイル・ステートメントおよび定義ステートメント	# ファイル・ステートメントおよび定義ステートメントを混在させることができます。

表 2-15. 7.2 で変更された言語要素: 組み込み関数

# 要素	# 説明
# %CHAR	# オペランドがタイム・スタンプの場合、返される値の長さは、タイム・スタンプ内のバイト数によって異なります。 *ISO0 形式の場合、バイト数は 14 から 26 までにすることができます。 *ISO 形式の場合、バイト数は 19、または 21 から 32 までにすることができます。
# %DEC	# オペランドがタイム・スタンプの場合、桁数は、タイム・スタンプ内の小数点以下の秒数に応じて 14 から 26 までにすることができます。
# %DIFF	# オペランドがタイム・スタンプの場合、オプションの 4 番目のパラメーターは、返す小数点以下の秒数を指定します。
# %SECONDS	# タイム・スタンプに秒を追加するために %SECONDS が使用される場合、パラメーターには、小数点以下の秒数を指定する小数点以下の桁数を設定することができます。
# %SUBDT	# <ul style="list-style-type: none"> • オプションの 3 番目のパラメーターは、結果での桁数を指定します。 • 第 1 オペランドがタイム・スタンプで、第 2 オペランドが *SECONDS の場合、オプションの第 4 オペランドは、結果での小数点以下の秒数を示します。
# %TIMESTAMP	# <ul style="list-style-type: none"> • 最初のパラメーターはタイム・スタンプにできます。 • 最初のパラメーターは *SYS にできます。 • 最初のパラメーターが日付、タイム・スタンプ、または *SYS の場合、2 番目のオプション・パラメーターは、小数点以下の秒数を示す 0 から 12 までの値にすることができます。 • 最初のパラメーターが文字または数字の場合、3 番目のオプション・パラメーターは、小数点以下の秒数を示す 0 から 12 までの値にすることができます。

表 2-16. 7.2 で変更された言語要素: 固定形式の定義仕様書

# 要素	# 説明
# 長さ入力	# タイム・スタンプの長さ入力は、19、または 21 から 32 までの値にすることができます。
# 小数点以下の桁数の入力	# タイム・スタンプの小数点以下の桁数の入力は、0 から 12 までの値にすることができます。

新着情報

表 2-17. 7.2 の新しい言語要素: ディレクティブ

#	要素	説明
#	/SET ディレクティブ	以下の制御ステートメント・キーワードの新しい値を一時的に設定します。
#		• CCSID(*CHAR:ccsid)
#		• CCSID(*GRAPH:ccsid)
#		• CCSID(*UCS2:ccsid)
#		• DATFMT(format)
#		• TIMFMT(format)
#		これらの値は、定義で値が明示的に指定されていない場合に、定義ステートメントのデフォルト値を指定するために使用されます。
#	/RESTORE ディレクティブ	以前の設定を、値を設定した最後の /SET ディレクティブより前の値に復元します。
#		• CCSID(*CHAR)
#		• CCSID(*GRAPH)
#		• CCSID(*UCS2)
#		• DATFMT
#		• TIMFMT

表 2-18. 7.2 の新しい言語要素: 自由形式ステートメント

#	要素	説明
#	CTL-OPT	自由形式の制御ステートメントを開始します。
#	DCL-F	自由形式のファイル定義を開始します。
#	DCL-C	自由形式の名前付き固定情報定義を開始します。
#	DCL-DS	自由形式のデータ構造定義を開始します。
#	DCL-SUBF	自由形式のサブフィールド定義を開始します。サブフィールド名が自由形式の計算で許可される命令コードと同じである場合を除き、「DCL-SUBF」の指定はオプションです。
#	END-DS	自由形式のデータ構造定義を終了します。サブフィールドがない場合、これは、DCL-DS ステートメントの最後のキーワードの後で指定できます。
#	DCL-PI	自由形式のプロシージャ・インターフェース定義を開始します。
#	DCL-PR	自由形式のプロトタイプ定義を開始します。
#	DCL-PARM	自由形式のパラメーター定義を開始します。パラメーター名が自由形式の計算で許可される命令コードと同じである場合を除き、「DCL-PARM」の指定はオプションです。
#	END-PI	自由形式のプロシージャ・インターフェース定義を終了します。パラメーターがない場合、これは、DCL-PI ステートメントの最後のキーワードの後で指定できます。
#	END-PR	自由形式のプロトタイプ定義を終了します。パラメーターがない場合、これは、DCL-PR ステートメントの最後のキーワードの後で指定できます。
#	DCL-S	自由形式の独立フィールド定義を開始します。
#	DCL-PROC	自由形式のプロシージャ定義を開始します。
#	END-PROC	自由形式のプロシージャ定義を終了します。

表 2-19. 7.2 の新しい言語要素: 制御仕様書キーワード

#	要素	説明
#	CCSIDCVT(*EXCP *LIST)	異なる CCSID を持つデータ間の変換をコンパイラーがどのように処理するかを制御できます。

表 2-19. 7.2 の新しい言語要素: 制御仕様書キーワード (続き)

# 要素	説明
# VALIDATE # (*NODATETIME) #	日付データ、時刻データ、およびタイム・スタンプ・データを、使用前に検証する必要があるかどうかを指定します。

表 2-20. 7.2 の新しい言語要素: ファイル定義キーワード

# 要素	説明
# DATA(*{NO}CVT) #	データベースが英数字フィールドと図形フィールドについてジョブ CCSID との間で CCSID 変換を実行するようにファイルをオープンするかどうかを制御します。
# HANDLER(handler # {;communication- # area})	ファイルがオープン・アクセス・ファイルであることを指定します。
# DISK ((*EXT # record-length))	自由形式のファイル定義の装置タイプを指定する装置キーワード。 • デフォルトの装置タイプは DISK です。 • 各装置タイプ・キーワードのデフォルト・パラメーターは、これが外部記述ファイルであることを示す *EXT です。
# PRINTER ((*EXT # record-length))	
# SEQ ((*EXT # record-length))	
# SPECIAL ((*EXT # record-length))	
# WORKSTN ((*EXT # record-length))	
# USAGE(*INPUT # *OUTPUT *UPDATE # *DELETE)	自由形式のファイル定義におけるファイルの使用法を指定します。
# KEYED ((*CHAR : # key-length)) #	自由形式のファイル定義でファイルがキー付きであることを示します。

表 2-21. 7.2 の新しい言語要素: 自由形式のデータ・タイプ・キーワード

# 要素	説明
# CHAR(length)	固定長の英数字データ・タイプ
# VARCHAR(length # {;prefix-size})	可変長の英数字データ・タイプ
# GRAPH(length)	固定長の図形データ・タイプ
# VARGRAPH(length # {;prefix-size})	可変長の図形データ・タイプ
# UCS2(length)	固定長の UCS-2 データ・タイプ
# VARUCS2(length # {;prefix-size})	可変長の UCS-2 データ・タイプ
# IND	標識データ・タイプ
# INT(digits)	整数データ・タイプ
# UNS(digits)	符号なし整数データ・タイプ
# PACKED(digits # {;decimals})	パック 10 進数データ・タイプ

新着情報

表 2-21. 7.2 の新しい言語要素: 自由形式のデータ・タイプ・キーワード (続き)

# 要素	説明
# ZONED(<i>digits</i> # { <i>:decimals</i> })	ゾーン 10 進数データ・タイプ
# BINDEC(<i>digits</i> # { <i>:decimals</i> })	2 進 10 進数データ・タイプ
# FLOAT(<i>size</i>)	浮動データ・タイプ
# DATE {(<i>format</i>)}	日付データ・タイプ
# TIME {(<i>format</i>)}	時刻データ・タイプ
# TIMESTAMP # {(fractional seconds)}	タイム・スタンプ・データ・タイプ
# POINTER {(*PROC)}	ポインター・データ・タイプ。オプション・パラメーター *PROC は、これがプロシージャ # ー・ポインターであることを示します。
# OBJECT {(*JAVA : # <i>class-name</i>)}	オブジェクト・データ・タイプ。これが Java コンストラクターの戻りの型を定義している場 # 合、パラメーターはオプションです。

表 2-22. 7.2 の新しい言語要素: 自由形式のデータ定義キーワード

# 要素	説明
# EXT	データ構造が外部記述であることを示します。このキーワードは、EXTNAME キーワードが # データ構造定義の最初のキーワードとして指定されている場合はオプションです。
# POS(<i>subfield-start-</i> # <i>position</i>)	データ構造内でのサブフィールドの開始位置を指定します。
# PSDS	データ構造がプログラム状況データ構造であることを指定します。

#

7.1 での変更点

この節では、7.1 で ILE RPG に対して行われた機能強化について説明します。

データ構造配列のソートと検索

サブフィールドの 1 つをキーとして使用して、データ構造配列のソートと検索を行うことができます。

```
// Sort the custDs array by the amount_owing subfield
SORTA custDs(*).amount_owing;

// Search for an element in the custDs array where the
// account_status subfield is "K"
elem = %LOOKUP("K" : custDs(*).account_status);
```

配列のソート (昇順または降順)

配列を昇順にソートするには SORTA(A) を、降順にソートするには SORTA(D) を、それぞれ使用します。順次配列 (ASCEND キーワードまたは DESCEND キーワード) はソートできません。

```
// Sort the salary array in descending order
SORTA(D) salary;
```

新しい組み込み関数 %SCANRPL (スキャンして置き換え)

%SCANRPL 組み込み関数は、ストリング内に存在する特定の値をすべてスキャンし、その値を別の値で置き換えます。

```
// Replace NAME with 'Tom'
string1 = 'See NAME. See NAME run. Run NAME run.';
string2 = %ScanRpl('NAME' : 'Tom' : string1);
// string2 = 'See Tom. See Tom run. Run Tom run.'
```

%LEN(varying : *MAX)

%LEN 組み込み関数を使用して、可変長の文字フィールド、UCS-2 フィールド、または図形フィールドの最大文字数を確認することができます。

外部記述データ構造での ALIAS 名の使用

定義仕様書で ALIAS キーワードを使用して、外部記述データ構造のサブフィールドに代替名を使用する旨を示すことができます。ファイル仕様書で ALIAS キーワードを使用して、当該ファイルのレコードで定義された LIKERECD データ構造に代替名を使用する旨を示すことができます。

```
A          R CUSTREC
A          CUSTNM      25A      ALIAS(CUSTOMER_NAME)
A          CUSTAD      25A      ALIAS(CUSTOMER_ADDRESS)
A          ID          10P 0

D custDs          e ds          ALIAS
D                                     QUALIFIED EXTNAME(custFile)
/free
  custDs.customer_name = 'John Smith';
  custDs.customer_address = '123 Mockingbird Lane';
  custDs.id = 12345;
```

速くなった戻り値

RTNPARM キーワードで定義されたプロシージャは、戻り値を隠しパラメーターとして処理します。プロシージャが非常に大きな値、特に非常に大きな可変値を戻すようにプロトタイプ化されていると、そのプロシージャを RTNPARM キーワードで定義することにより、そのプロシージャの呼び出しパフォーマンスを大幅に向上させることができます。

```
D getFileData      pr          a  varying len(1000000)
D                                     rtnparm
D file            a  const varying len(500)
D data            S          a  varying len(1000)
/free
  data = getFileData ('/home/mydir/myfile.txt');
```

%PARMNUM 組み込み関数

%PARMNUM(parameter_name) 組み込み関数は、パラメーター・リスト内における当該パラメーターの順序数を返します。プロシージャが RTNPARM キーワードと一緒にコーディングされているときは、この組み込み関数を使用することが特に重要となります。

```
D          pi
D name          100a  const varying
D id            10i 0 value
D errorInfo          liked(errs_t)
D                                     options(*nopass)
/free
  // Check if the "errorInfo" parameter was passed
  if %parms >= %parmnum(errorInfo);
```

任意指定のプロトタイプ

プログラムまたはプロシージャが別の RPG モジュールによって呼び出されない場合は、プロトタイプの指定は任意です。以下のタイプのプログラムとプロシージャの場合は、プロトタイプを省略することができます。

- 出口プログラムまたはコマンドのコマンド処理プログラムとして使用されるようにのみ意図されているプログラム

新着情報

- 別のプログラミング言語から呼び出されるようにのみ意図されているプログラム
- モジュールからエクスポートされないプロシージャ
- モジュールからエクスポートされるが、別のプログラミング言語から呼び出されるようにのみ意図されているプロシージャ

任意のタイプのストリング・パラメーターの受け渡し

値または読み取り専用参照によって渡されたストリング・パラメーターに対しては、暗黙的な変換が実行されます。例えば、CONST UCS-2 パラメーターを持つようにプロシージャをプロトタイプ化することができ、そのプロシージャの呼び出し時に文字式をパラメーターとして渡すことができます。これにより、UCS-2 タイプでプロトタイプ化されたパラメーターと戻り値を持つ単一のプロシージャを作成できるようになります。このプロシージャを呼び出すために、任意のタイプのストリング・パラメーターを渡し、戻り値を任意のタイプのストリング変数に割り当てることができます。

```
// The makeTitle procedure upper-cases the value
// and centers it within the provided length
alphaTitle = makeTitle(alphaValue : 50);
ucs2Title = makeTitle(ucs2Value : 50);
dbcsTitle = makeTitle(dbcsValue : 50);
```

XML-INTO 用の 2 つの新しいオプション

- *datasubf* オプションを使用すると、属性も指定された XML 要素のテキスト・データを受け取るサブフィールドを指定することができます。
- *countprefix* オプションにより、*allowmissing=yes* オプションを指定する必要性が減少します。このオプションは、RPG 配列要素の数を受け取る追加サブフィールドの名前または XML-INTO 命令によって設定される非配列サブフィールドの名前の接頭部を指定します。

これらのオプションは 6.1 用の PTF によっても使用可能となります。

テラスペース記憶域モデル

テラスペース記憶域モデルを使用したり、呼び出し元の記憶域モデルを継承したりする目的で、
RPG モジュールや RPG プログラムを作成することができます。テラスペース記憶域モデルを使用
すると、自動記憶域に関するシステムしきい値が単一レベル記憶域モデルの場合と比べて大幅に高
くなります。単一プロシージャの自動記憶域の量に関する限度と、呼び出しスタック上の全プロ
シージャの自動記憶域の合計に関する限度があります。

CRTRPGMOD または CRTBNDRPG コマンドでは記憶域モデル (STGMDL) パラメーターを使用し、制御仕様書では STGMDL キーワードを使用します。

*TERASPACE

プログラムまたはモジュールはテラスペース記憶域モデルを使用します。

*SNGLVL

プログラムまたはモジュールは単一レベル記憶域モデルを使用します。

*INHERIT

プログラムまたはモジュールは呼び出し元の記憶域モデルを継承します。

CRTBNDRPG コマンドの ACTGRP パラメーターと制御仕様書の ACTGRP キーワードの変更

ACTGRP パラメーターと ACTGRP キーワードのデフォルト値が QILE から *STGMDL に変更されました。

ACTGRP(*STGMDL) は、活動化グループがプログラムの記憶域モデルによって決まることを指定します。記憶域モデルが *TERASPACE のときは、ACTGRP(*STGMDL) と ACTGRP(QILETS) が同じです。その他の場合は、ACTGRP(*STGMDL) と ACTGRP(QILE) が同じです。

注: ACTGRP パラメーターと ACTGRP キーワードに対する変更は、活動化グループをプログラムに割り当てるデフォルト方法には影響しません。 STGMDL パラメーターと STGMDL キーワードのデフォルト値は *SNGLVL であるため、ACTGRP パラメーターまたはキーワードが指定されていないと、プログラムの活動化グループは以前のリリースの場合と同じようにデフォルトの QILE となります。

テラスペース記憶域の割り振り

制御仕様書で ALLOC キーワードを使用して、モジュール内の RPG 記憶域管理命令がテラスペース記憶域を使用するか単一レベル記憶域を使用するかを指定することができます。テラスペース記憶域割り振りの最大サイズは、単一レベル記憶域割り振りの最大サイズより大幅に大きくなりました。

暗号化されたリスト・デバッグ・ビュー

モジュールのリスト・デバッグ・ビューが暗号化されていると、デバッグ担当者が暗号鍵を知っているときは、リスト・ビューを表示できるのはデバッグ・セッションの間だけとなります。このため、お客様がリスト・ビューによってソース・コードを確認できるようにしなくても、お客様にデバッグ可能プログラムを送信することができます。 CRTRPGMOD コマンド、CRTBNDRPG コマンド、または CRTSQLRPGI コマンドで DBGENCKEY パラメーターを使用します。

表 2-23. 6.1 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	ACTGRP(*STGMDL)	*STGMDL は ACTGRP キーワードおよびコマンド・パラメーターの新しいデフォルトです。プログラムがテラスペース記憶域モジュールを使用している場合は、活動化グループが QILETS となります。その他の場合は QILE となります。
組み込み関数	%LEN(varying-field : *MAX)	これを使用して、可変長フィールドの最大文字数を確認できるようになりました。
命令コード	SORTA(A D)	SORTA 命令コードで、配列を昇順 (A) にソートするか降順 (D) にソートするかを示す A 命令拡張および D 命令拡張を使用できるようになりました。

表 2-24. 6.1 以降の新しい言語要素

言語単位	要素	説明
制御仕様書キーワード	STGMDL(*INHERIT *TERASPACE *SNGLVL)	モジュールまたはプログラムの記憶域モデルを制御します。
	ALLOC(*STGMDL *TERASPACE *SNGLVL)	記憶域管理命令 %ALLOC、%REALLOC、DEALLOC、ALLOC、REALLOC の記憶域モデルを制御します。
ファイル仕様書のキーワード	ALIAS	LIKEREC キーワードで定義されたデータ構造のサブフィールドに代替フィールド名を使用します。

新着情報

表 2-24. 6.1 以降の新しい言語要素 (続き)

言語単位	要素	説明
定義仕様書キーワード	ALIAS	外部記述データ構造のサブフィールドに代替フィールド名を使用します。
	RTNPARM	プロシージャーの戻り値を隠しパラメーターとして処理するように指定します。
組み込み関数	%PARMNUM	パラメーター・リスト内におけるパラメーターの順序数を戻します。
	%SCANRPL	ストリング内に存在する特定の値をすべてスキャンし、その値を別の値で置き換えます。
XML-INTO のオプション	databuf	属性も持つ XML 要素のテキスト・データを受け取るサブフィールドに名前を付けます。
	countprefix	RPG 配列要素の数を受け取る追加サブフィールドの名前または XML-INTO 命令によって設定される非配列サブフィールドの名前の接頭部を指定します。

6.1 での変更点

この節では、6.1 で ILE RPG に対して行われた機能強化について説明します。

THREAD(*CONCURRENT)

モジュールの制御仕様書に THREAD(*CONCURRENT) が指定されていると、次のように複数のスレッドで並行して実行できる機能が提供されます。

- 複数のスレッドが同時にモジュールで実行できる。
- デフォルトでは、各スレッドが静的変数の独自のコピーを持つように静的変数が定義される。
- 個々の変数は、STATIC(*ALLTHREAD) を使用してすべてのスレッドで共用されるように定義できる。
- プロシージャー開始仕様書で SERIALIZE を指定することによって、一度に 1 つのスレッドのみがプロシージャーを実行できるように個々のプロシージャーを直列化することができる。

RPG サイクルを使用しないメイン・プロシージャーを定義する機能

サブプロシージャーは、制御仕様書で MAIN キーワードを使用して、プログラム入力プロシージャーとして識別することができます。これにより、RPG サイクルを使用するモジュールが存在しない場合でも RPG アプリケーションを開発することができます。

サブプロシージャーで定義されるファイル

ファイルは、サブプロシージャーでローカルに定義できます。ローカル・ファイルへの入出力は、データ構造を使用してのみ実行できます。I 仕様書および O 仕様書はサブプロシージャーで許可されず、コンパイラーは外部記述ファイルの I 仕様書および O 仕様書を生成しません。デフォルトでは、ローカル・ファイルは自動記憶域に関連付けられます。サブプロシージャーが戻るときに、ファイルがクローズされます。STATIC キーワードを使用して、ファイルに関連付けられる記憶域が静的であることを示すことができるため、サブプロシージャーのすべての呼び出しは同じ

ファイルを使用します。また、サブプロシージャーが戻ったときにファイルが開いている場合は、そのサブプロシージャーへの次の呼び出しのためにファイルは開いたままになります。

修飾レコード様式

ファイルが **QUALIFIED** キーワード付きで定義される場合、レコード様式は **MYFILE.MYFMT** のようにファイル名で修飾する必要があります。修飾ファイルにはコンパイラーによって生成される **I** 仕様書および **O** 仕様書はありません。入出力はデータ構造を介してのみ行われます。

他のファイルのように定義されるファイル

ファイルは、**LIKEFILE** キーワードを使用して、他のファイル仕様書と同じ設定を使用するように定義することができます。これは、ファイルをパラメーターとして渡すときに重要です。ファイルが外部記述される場合、**QUALIFIED** キーワードは暗黙となります。新規ファイルへの入出力は、データ構造を介してのみ実行できます。

パラメーターとして渡されるファイル

プロトタイプ・パラメーターは、**LIKEFILE** キーワードを使用して、ファイル・パラメーターとして定義できます。同じ **LIKEFILE** 定義を介して関連付けられたファイルは、パラメーターとしてプロシージャーに渡される場合があります。呼び出されたプロシージャーまたはプログラム内で、サポートされるすべての操作をそのファイルに実行することができます。入出力はデータ構造を介してのみ実行できます。

EXTDESC キーワードと EXTFILE(*EXTDESC)

EXTDESC キーワードは、ファイルの外部記述を取得するためにコンパイラーがコンパイル時に使用するファイルを識別します。ファイル名は、「**LIBNAME/FILENAME**」または「**FILENAME**」のいずれかの形式のリテラルとして指定されます。これにより、ファイルに対するコンパイル時指定変更が不要になります。

EXTFILE キーワードが特殊値 ***EXTDESC** を許可するように拡張されました。この特殊値は、**EXTDESC** によって指定されたファイルを実行時にも使用するように指示するものです。

外部記述データ構造のライブラリーを指定するための EXTNAME

EXTNAME キーワードが、リテラルを使用して外部ファイルのライブラリーを指定できるように拡張されました。 **EXTNAME('LIBNAME/FILENAME')** または **EXTNAME('FILENAME')** がサポートされています。これにより、ファイルに対するコンパイル時指定変更が不要になります。

EXFMT は結果データ構造を許可する

EXFMT 命令が、結果フィールドにデータ構造を指定できるように拡張されました。データ構造は、使用タイプ ***ALL** を使用して、レコード様式の外部記述データ構造 (**EXTNAME (file:fmt:*ALL)**) として、またはレコード様式の **LIKEREC (LIKEREC(fmt:*ALL)**) を使用して定義する必要があります。

データ構造、文字、UCS-2 およびグラフィック変数の制限の拡大

- データ構造のサイズは最大 16,773,104 とすることができる。
- 文字定義の長さは最大 16,773,104 とすることができる。(この制限は可変長文字定義では 4 少ない。)
- UCS-2 定義の長さは最大 8,386,552 UCS-2 文字とすることができる。(この制限は可変長 UCS-2 定義では 2 少ない。)
- グラフィック定義の長さは最大 8,386,552 DBCS 文字とすることができる。(この制限は可変長グラフィック定義では 2 文字少ない。)

新着情報

- VARYING キーワードは、長さの接頭部を保持するために使用されるバイト数を示す 2 または 4 のいずれかのパラメーターを許可します。

%ADDR(varying : *DATA)

%ADDR 組み込み関数が、*DATA を 2 番目のパラメーターとして使用して可変長フィールドのデータ部分のアドレスを取得できるように拡張されました。

DIM および OCCURS の制限の拡大

配列または複数回繰り返しデータ構造は、合計サイズが 16,773,104 より大きくなければ、最大 16,773,104 要素を持つことができます。

文字、UCS-2 および DBCS リテラルの制限の拡大

- 文字リテラルの最大長が 16380 文字になりました。
- UCS-2 リテラルの最大長が 8190 文字 (UCS-2) になりました。
- 図形リテラルの最大長が 16379 文字 (DBCS) になりました。

ファイルおよび定義のための TEMPLATE キーワード

ファイルおよび変数の定義が、他のファイルまたは変数を定義するのに名前が LIKEFILE、LIKE、または LIKEDS キーワードと一緒にのみ使用されることを指示するように、TEMPLATE キーワードをコーディングすることができます。コンパイラーは、コンパイル時にテンプレート定義のみを使用して他のファイルおよび変数を定義し、それらのファイルや変数に関連するコードを生成しないので、プロトタイプ呼び出しのタイプを定義する際にはテンプレート定義が便利です。

テンプレート・データ構造では、データ構造およびそのサブフィールドに対して INZ キーワードをコーディングでき、それにより INZ(*LIKEDS) の使用が容易になります。

一部の UCS-2 規則の緩和

コンパイラーが文字値、UCS-2 値、およびグラフィック値の間の暗黙的な変換を行うため、多くの場合、%CHAR、%UCS2、または %GRAPH のコーディングは不要となります。この機能拡張は、V5R3 および V5R4 の PTF でも利用することができます。暗黙的な変換の現在のサポート対象は以下のとおりです。

- EVAL および EVALR による代入
- 式の中の比較演算。
- 固定形式演算 IFxx、DOUxx、DOWxx、WHxx、CASxx、CABxx、COMP を使用した比較。
- 変換命令 MOVE および MOVEL に関しては、暗黙的な変換がすでにサポートされています。

UCS-2 変数の初期化が、%UCS2 組み込み関数を使用せずに文字リテラルまたは図形リテラルを使用して行えるようになりました。

コンパイル済みオブジェクトからの未使用変数の除去

新しい値 *UNREF と *NOUNREF が、CRTBNDRPG コマンドおよび CRTRPGMOD コマンドの OPTION キーワードと、制御仕様書の OPTION キーワードに追加されました。デフォルトは *UNREF です。*NOUNREF は、参照されない変数を RPG モジュールに生成しないよう指示します。これにより、プログラム・サイズは削減でき、インポートされた変数が参照されない場合、モジュールをプログラムまたはサービス・プログラムにバインドするためにかかる時間を削減できます。

PCML をモジュールに保管可能

Program Call Markup Language (PCML) を、ストリーム・ファイルだけでなく、モジュールにも保管できるようになりました。RPG プログラマーは、PGMINFO コマンド・パラメーターまたは制

御仕様書の新規 PGMINFO キーワード、あるいはその両方の組み合わせを使用することにより、PCML 情報の保管先を選択することができます。PCML 情報がモジュール内に置かれた場合は、後で QBNRPII API を使用して取り出すことができます。この拡張は V5R4 の PTF によっても使用できますが、制御仕様書キーワードを介してのみ可能です。

表 2-25. V5R4 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	OPTION(*UNREF *NOUNREF)	未使用の変数はモジュールに生成してはならないことを指定します。
	THREAD(*CONCURRENT)	新規パラメーター *CONCURRENT では、複数のスレッドで同時に実行可能です。
ファイル仕様書のキーワード	EXTFILE(*EXTDESC)	EXTDESC キーワードの値を EXTFILE キーワードでも使用するよう指定します。
組み込み関数	%ADDR(varying-field : *DATA)	現在は、可変長変数のデータ部分のアドレスを取得するために使用できます。
定義仕様書キーワード	DIM(16773104)	配列は最大 16773104 要素を持つことができます。
	EXTNAME('LIB/FILE')	ファイル名のリテラルを許可します。リテラルにはファイルのライブラリーを含むことができます。
	OCCURS(16773104)	複数回繰り返しデータ構造は最大 16773104 要素を持つことができます。
	VARYING {(214)}	長さ接頭部のバイト数を示すパラメーターを取ることが可能になりました。
定義仕様書	長さ入力	データ構造、およびタイプ A、C、または G の定義に対して最大 9999999 とすることができます。(より長い項目を定義するには、LEN キーワードを使用する必要があります。)
入力仕様書	長さ入力	英数字フィールドの場合は最大 99999 で、UCS-2 および図形フィールドの場合は 99998 とすることができます。
演算仕様書	長さ入力	英数字フィールドの場合は最大 99999 とすることができます。
命令コード	EXFMT フォーマット { result-ds }	結果入力にデータ構造を持つことができます。

新着情報

表 2-26. V5R4 以降の新しい言語要素

言語単位	要素	説明
制御仕様書キーワード	MAIN(subprocedure-name)	プログラムのプログラム入力プロシージャを指定します。
	PGMINFO(*NO *PCML { : *MODULE })	プログラム情報をモジュールに直接置くかどうかを示します。
ファイル仕様書のキーワード	STATIC	サブプロシージャへの呼び出し間で、ローカル・ファイルがプログラム状態を保存することを示します。
	QUALIFIED	ファイルのレコード様式名は FILE.FMT のようにファイル名で修飾されていることを示します。
	LIKEFILE(ファイル名)	ファイルは他のファイルと同じように定義されることを示します。
	TEMPLATE	ファイルは後の LIKEFILE 定義でのみ使用されることを示します。
	EXTDESC(constant-filename)	外部定義用にコンパイル時に使用される外部ファイルを指定します。
定義仕様書キーワード	STATIC(*ALLTHREAD)	静的変数の同じインスタンスが、モジュールで実行中のすべてのスレッドによって使用されることを示します。
	LIKEFILE(ファイル名)	パラメーターがファイルであることを示します。
	TEMPLATE	定義は LIKE 定義または LIKEDS 定義に対してのみ使用されることを示します。
	LEN(長さ)	データ構造、または定義 A、C、または G の長さを指定します。
プロシージャ仕様書キーワード	SERIALIZE	プロシージャが一度に 1 つのスレッドのみを実行できることを示します。

V5R4 での変更点

V5R4 で ILE RPG に対して加えられた機能拡張について、以下にリストして記載します。

新規の命令コード EVAL-CORR

```
EVAL-CORR{(EH)} ds1 = ds2
```

新規命令コード EVAL-CORR は、ソース・データ構造のサブフィールドからのデータおよびヌル標識をターゲット・データ構造のサブフィールドに割り当てます。割り当てられるサブフィールドは、両方のデータ構造で同じ名前と互換データ・タイプを持つサブフィールドです。

例えば、データ構造 DS1 が文字サブフィールド A、B、および C を持ち、データ構造 DS2 が文字サブフィールド B、C、および D を持つ場合、ステートメント EVAL-CORR DS1 = DS2; はサブフィールド DS2.B および DS2.C からのデータを DS1.B および DS1.C に割り当てます。また、

EVAL-CORR 命令の影響を受ける、ターゲット・データ構造内のヌル可能サブフィールドのヌル標識は、ソース・データ構造のサブフィールドのヌル標識から割り当てられ、ソース・サブフィールドがヌル可能でない場合は *OFF に設定されます。

```
// DS1 subfields          DS2 subfields
//   s1 character          s1 packed
//   s2 character          s2 character
//   s3 numeric
//   s4 date                s4 date
//                           s5 character
//
EVAL-CORR ds1 = ds2;
// This EVAL-CORR operation is equivalent to the following EVAL operations
//   EVAL ds1.s2 = ds2.s2
//   EVAL ds1.s4 = ds2.s4
// Other subfields either appear in only one data structure (S3 and S5)
// or have incompatible types (S1).
```

EVAL-CORR は、外部記述ファイルおよびレコード様式への入出力操作の結果のデータ構造を簡単に使用できるようにし、レコード様式のレイアウトが違うか、サブフィールドのタイプが少し違う場合に、異なるレコード様式のデータ構造間でのデータの自動転送を可能にします。

新規プロトタイプ・パラメーター・オプション **OPTIONS(*NULLIND)**

OPTIONS(*NULLIND) がパラメーターに指定された場合、そのパラメーターと共にヌル・バイト・マップが渡され、着呼側プロシーチャーが発呼者のパラメーターのヌル・バイト・マップに直接アクセスできるようになります。

新規組み込み関数 **%XML**

```
%XML (xmldocument { : options } )
```

%XML 組み込み関数は、XML 文書を記述し、その文書の構文解析方法を制御するオプションを指定します。 *xmldocument* パラメーターには文字式または UCS-2 式が可能であり、値は XML 文書または XML 文書を含む IFS ファイルの名前とすることができます。 *xmldocument* パラメーターの値がファイルの名前である場合は、"doc=file" オプションを指定する必要があります。

新規組み込み関数 **%HANDLER**

```
%HANDLER (handlingProcedure : communicationArea )
```

%HANDLER は、イベントまたは一連のイベントを処理するプロシーチャーを識別するために使用されます。 %HANDLER は値を戻しません。また、XML-SAX および XML-INTO の第 1 オペランドとしてのみ指定できます。

第 1 オペランド *handlingProcedure* は、処理プロシーチャーのプロトタイプを指定します。プロトタイプによって指定された戻り値およびパラメーターは、処理プロシーチャーが必要とするパラメーターと一致している必要があります。要件は、%HANDLER が指定されている命令によって決まります。

第 2 オペランド *communicationArea* は、処理プロシーチャーのすべての呼び出しでパラメーターとして渡される変数を指定します。このオペランドは、参照によって渡されるプロトタイプ・パラメーターの検査に使用される規則と同じ規則に従って、処理プロシーチャーの最初のプロトタイプ・パラメーターと完全に一致していなければなりません。通信域パラメーターは、配列およびデータ構造を含む、任意のタイプであってかまいません。

新規命令コード **XML-SAX**

```
XML-SAX{ (e) } %HANDLER(eventHandler : commArea ) %XML(xmldoc { : options } );
```

XML-SAX は、%XML 組み込み関数により指定された XML 文書に対して SAX 構文解析を開始します。 XML-SAX 命令は、文書の構文解析を開始する XML パーサーを呼び出すことにより開始しま

新着情報

す。要素の開始の検出、属性名の検出、および要素の終了の検出などのイベントがパーサーにより発見された場合、パーサーはそのイベントを記述したパラメーターで **eventHandler** を呼び出します。

commArea オペランドは、パラメーターとして **eventHandler** に渡される変数であり、XML-SAX 命令コードが処理プロシージャーと通信する方法を提供します。**eventHandler** が戻ると、次のイベントが検出されて再度 **eventHandler** が呼び出されるまでパーサーは構文解析を続行します。

新規命令コード XML-INTO

```
XML-INTO{ (EH) } variable %XML(xmlDoc { : options });  
XML-INTO{ (EH) } %HANDLER(handler : commArea ) %XML(xmlDoc { : options });
```

XML-INTO は、以下の 2 つの方法のいずれかで XML 文書からデータを読み込みます。

- 変数に直接。
- %HANDLER で指定されるプロシージャーに渡す配列パラメーターに徐々に。

命令を制御するためにさまざまなオプションを指定することができます。

第 1 オペランドには、構文解析されるデータのターゲットを指定します。変数名または % HANDLER 組み込み関数を含むことができます。

第 2 オペランドには、XML 文書のソースを指定する %XML 組み込み関数、およびその文書の構文解析方法を制御する任意のオプションを指定します。XML データを含むか、XML データのロケーションを含むことができます。このオペランドが何を指定するかを示すために、doc オプションを使用します。

```
// Data structure "copyInfo" has two subfields, "from"  
// and "to". Each of these subfields has two subfields  
// "name" and "lib".  
// File cpyA.xml contains the following XML document  
// <copyinfo>  
//   <from><name>MASTFILE</name><lib>CUSTLIB</lib></from>  
//   <to><name>MYFILE</name><lib>*LIBL</lib>  
// </copyinfo>  
xml-into copyInfo %XML('cpyA.xml' : 'doc=file');  
// After the XML-INTO operation, the following  
// copyInfo.from .name = 'MASTFILE' .lib = 'CUSTLIB'  
// copyInfo.to .name = 'MYFILE' .lib = '*LIBL'
```

フィールド名の先頭から文字を除去するための PREFIX キーワードの使用

```
PREFIX(' : number_of_characters)
```

ファイル仕様書および定義仕様書で PREFIX キーワードの第 1 パラメーターとして空の文字リテラル (間に文字を入れずに指定された 2 つの単一引用符) が指定されていると、指定された数の文字がフィールド名から除去されます。例えば、XRNAME、XRIDNUM、および XRAMOUNT というフィールドを含むファイルの場合は、ファイル仕様書で PREFIX('':2) と指定すると、内部フィールド名が NAME、IDNUM、および AMOUNT になります。

ファイル固有の接頭部以外は名前が同じであるサブフィールドを含むファイルが 2 つ存在する場合は、この機能を使用して、それらのファイルに基づいて定義された外部記述データ構造のサブフィールドの名前から接頭部を除去することができます。これにより、EVAL-CORR を使用して一方のデータ構造から他方のデータ構造に同じ名前のサブフィールドを割り当てることが可能となります。例えば、ファイル FILE1 にフィールド F1NAME があり、ファイル FILE2 にフィールド F2NAME があり、さらに FILE1 の外部記述データ構造 DS1 および FILE2 の外部記述データ構造 DS2 に対して PREFIX('':2) が指定された場合、サブフィールド F1NAME および F2NAME は両方とも NAME になります。データ構造 DS1 および DS2 間の EVAL-CORR 命令は NAME サブフィールドを割り当てます。

DEBUG キーワードの新規値

```
DEBUG { ( *INPUT *DUMP *XMLSAX *NO *YES ) }
```

DEBUG キーワードは、モジュールに生成されるデバッグ援助機能を決定します。*NO および *YES が既存の値です。*INPUT、*DUMP および *XMLSAX によって *YES よりも細分性が提供されます。

*INPUT

入力仕様書にあって、モジュール内の他のどの場所でも使用されないフィールドは、入力操作中
にプログラム・フィールドに読み込まれます。

*DUMP

(A) 拡張がない DUMP 操作が実行されます。

*XMLSAX

SAX イベント名の配列がモジュールに生成され、SAX イベント・ハンドラーのデバッグ時に使用されます。

*NO

デバッグ援助機能をモジュールに生成しないように指示します。DEBUG(*NO) を指定しても、DEBUG キーワードを省略した場合と同じになります。

*YES

この値は、互換性目的のために保持されています。DEBUG(*YES) を指定することは、パラメータなしで DEBUG を指定するか、または DEBUG(*INPUT : *DUMP) を指定することと同じです。

フリー・フォーム計算用の構文検査

SEU では、フリー・フォーム・ステートメントが正しい構文であるか検査されるようになりました。

修飾されたデータ構造のヌル可能サブフィールド用の改良されたデバッグ・サポート

ヌル可能サブフィールドを持つ修飾されたデータ構造をデバッグする際に、すべてのヌル可能サブフィールド用に標識サブフィールドを持つ同じデータ構造として、NULL 標識が編成されるようになりました。データ構造の名前は、_QRNU_NULL_ データ構造名 (例えば _QRNU_NULL_MYDS) です。データ構造のサブフィールド自身がヌル可能サブフィールドを持つデータ構造である場合は、NULL 標識データ構造は、標識サブフィールドを持つデータ構造サブフィールドを同様に持ちます。例えば、データ構造 DS1 がヌル可能サブフィールド DS1.FLD1、DS1.FLD2、および DS1.SUB.FLD3 を持つ場合、デバッグ命令を使用してデータ構造全体にすべての NULL 標識を表示することができます。

```
====> EVAL _QRNU_NULL_DS
> EVAL _QRNU_NULL_DS1
   _QRNU_NULL_DS1.FLD1 = '1'
   _QRNU_NULL_DS1.FLD2 = '0'
   _QRNU_NULL_DS1.SUB.FLD3 = '1'
====> EVAL _QRNU_NULL_DS.FLD2
   _QRNU_NULL_DS1.FLD2 = '0'
====> EVAL _QRNU_NULL_DS.FLD2 = '1'
====> EVAL DSARR(1).FLD2
   DSARR(1).FLD2 = 'abcde'

====> EVAL _QRNU_NULL_DSARR(1).FLD2
   _QRNU_NULL_DSARR(1).FLD2 = '0'
```

共用ファイルを持つファイルの終わりの振る舞いの変更

モジュールが共用ファイルに対してキー順次入力操作を実行した結果が EOF 状態になり、別のモジュールが SETLL などの位置決め操作を使用してファイル・カーソルを設定した場合、最初のモジュール

新着情報

による後続の順次入力操作は正常に実行されます。この変更の前では、最初の RPG モジュールは、他のモジュールが共用ファイルを位置変更したという事実を無視していました。

この振る舞いの変更は、リソース V5R2M0 および V5R3M0 の PTF (それぞれ、SI13932 および SI14185) で使用可能です。

表 2-27. V5R3 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	DEBUG(*INPUT*DUMP *XMLSAX *NO*YES)	新規パラメーター *INPUT、*DUMP および *XMLSAX は、デバッグ補助機能のためのさらなるオプションを提供します。
ファイル仕様書のキーワード	PREFIX('':2)	PREFIX キーワードの第 1 パラメーターとして空のリテラルを指定し、名前の先頭から文字を除去することができます。
定義仕様書キーワード	OPTIONS(*NULLIND)	NULL 標識がパラメーターと共に渡されることを示します。
	PREFIX('':2)	PREFIX キーワードの第 1 パラメーターとして空のリテラルを指定し、名前の先頭から文字を除去することができます。

表 2-28. V5R3 以降の新しい言語要素

言語単位	要素	説明
組み込み関数	%HANDLER(プロトタイプ: パラメーター)	イベント用の処理手順を指定します。
	%XML(文書 {:オプション})	XML 文書およびそれを構文解析する方法を制御するオプションを指定します。
命令コード	EVAL-CORR	ソース・データ構造のサブフィールドからのデータおよび NULL 標識をターゲット・データ構造のサブフィールドに割り当てます。
	XML-INTO	XML 文書からのデータをプログラム変数に直接読み込みます。
	XML-SAX	XML 文書の SAX 構文解析を開始します。

V5R3 での変更点

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の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** による、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書 **CCSID(*CHAR : *JOB RUN)**:

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 *JOB RUN である必要があります。CCSID(*CHAR : *JOB RUN) 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:*JOB RUN) が指定されていると、文字データはジョブ CCSID 内にあると見なされ、CCSID(*CHAR : *JOB RUN) が指定されていないと、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。

- トリミング対象の文字を示す **%TRIM**、**%TRIMR**、および **%TRIML** の 2 番目のパラメーター:

%TRIM が拡張され、トリミング対象の文字のリストを指定するために、オプションとして 2 番目のパラメーターを使用できるようになりました。

例:

```
trimchars = '*-.';
data = '***a-b-c-.'
result = %trim(data : trimchars);
// now result = 'a-b-c'. All * - and . were trimmed from the ends of the data
```

- 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)**:

新着情報

プロトタイプ・パラメーターに `OPTIONS(*TRIM)` が指定されていると、渡されるデータの先行および末尾のブランクがトリムされます。 `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 = 'rst.' を指すポインター (この . は x'00' です)
```

• 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。以前の上限は 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。
- 命令コード `CHAIN`、`READ`、`READE`、`READP`、および `READPE` の外部記述ファイル名に対する I/O の結果フィールドにデータ構造を指定することができます。

例:

- 次のプログラムは、外部記述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(0_)
/free
  O_FLD1 = 'ABCDE';
  O_FLD2 = 7;
  write outrec outrecDs;
  *inlr = *on;
/end-free
```

- 次のプログラムは、複数形式の論理ファイルを読み取り、それぞれのレコード様式のフィールドを保持する 2 つの重複するサブフィールドを含むデータ構造 `INPUT` に、そのファイルを書き込みます。

```
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';
      // handle rec1
    elseif recname = 'REC2';
      // ハンドル rec2
    endif;
    read log input;
  enddo;
  *inlr = *on;
/end-free

```

- プログラム/モジュールが共用ファイルに対してキー順次入力操作を実行した結果が EOF 状態になった場合は、同じプログラム/モジュールによる後続の順次入力操作が試みられることがあります。入力要求がデータベースに送信され、入力に使用可能なレコードがある場合は、データがプログラム/モジュールに移動され、EOF 状態はオフに設定されます。
- **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** により、RPG ユーザーは JVM を開始するために使用される Java プロパティを明示的に設定することができます。

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、どのオプション・ストリングにも存在しない文字によって区切られ、かつ終了する 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する場合:** **os400.stdout** オプションもデフォルト値以外の値に設定したい場合には、環境変数を次の値に設定することができます。

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

この例では、区切り文字および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** により、RPG ユーザーは、Java メソッドに対する RPG 呼び出しが例外で終了したとき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' が含まれている場合は、RPG から Java メソッドが呼び出されたときに Java 例外が発生したり、呼び出された Java メソッドが呼び出し元に例外をスローしたりすると、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ます、**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;'
```

新着情報

- **RPG プリプロセッサにより、SQL プリプロセッサ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 などの条件付きコンパイル・ディレクティブが受け入れるオリジナルのソース行が含まれます。さらに、/COPY ステートメントによって組み込まれたファイルのソース行も含まれ、オプションとして、/INCLUDE ステートメントによって組み込まれたソース行が含まれます。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、SQL プリコンパイラは /COPY、条件付きコンパイル・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 プリプロセッサを使用します。これにより、SQLRPGLE ソースは、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持つことができます。

表 2-29. 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:パラメーター *UCS2:数値 *CHAR:*JOB RUN)	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 *JOB RUN を指定できるようになりました。
組み込み関数	%DEC(式 {形式})	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(式:式)	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	長さおよび小数点以下の桁数の入力	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。

表 2-29. V5R2 以降に変更された言語要素 (続き)

言語単位	要素	説明
演算仕様書	長さおよび小数点以下の桁数の入力	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 命令	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 命令	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 命令	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 2-30. 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(配列:開始要素 { :要素の数 })	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

V5R2 で ILE RPG に対して加えられた機能拡張について、以下にリストして記載します。

- 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使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を使用できるようになりました。これにより、配列の配列、つまり構造の副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。

新着情報

- キーによる入出力の拡張

プログラマーは、次の 2 つの新しい方法で、/FREE 演算におけるキー付き入出力命令の検索指数を指定できます。

1. リストで検索指数 (式も可能) を指定する
2. 検索指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行えませんでした。

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 命令が拡張され、拡張 N をサポートするようになりました。これは、この命令で、負荷の高いディスク書き出しを強制せず、ブロック化したバッファをローカルに単に書き出すように指示します。

- データ域アクセスの拡張

DTAARA キーワードが拡張され、データ域の名前とライブラリーが実行時に判別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 を含む IFS ファイルを生成し、パラメーターをプログラム (CRTBNDRPG) またはエクスポートされたプロシージャ (CRTRPGMOD) に表します。

表 2-31. 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(式)	型を表す文字のパラメーターを取れるようになった。
	%DECH(式)	
	%FLOAT(式)	
	%INT(式)	
	%INTH(式)	
	%UNS(式)	
	%UNSH(式)	
定義仕様書キーワード	DTAARA({*VAR:}データ域名)	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(ファイル名{:外部レコード名} {:*ALLI*INPUTI*OUTPUTI*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	長さおよび小数点以下の桁数の入力	バック・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 31 まで指定できる。
命令コード	CHAIN、DELETE、READE、READPE、SETGT、SETLL	自由形式演算では、演算項目 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	長さおよび小数点以下の桁数の入力	バック・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 31 まで指定できる。

表 2-32. 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(内部レコード名{:*ALLI*INPUTI*OUTPUTI*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。

新着情報

表 2-32. 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(式 : 式)	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返す。
	%BITNOT(式)	引数のビットとは逆のビットの結果を返す。
	%BITOR(式 : 式)	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返す。
	%BITXOR(式 : 式)	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返す。
	%FIELDS(名前{:名前...})	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(データ構造)	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 Rational® Development Studio for i 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、コンパイラ指示 /FREE および /END-FREE で区切ります。

注: これらのディレクティブは不要になりました。

- ファイル仕様に 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できるようになりました。

このリリースでは、これ以外の拡張も行われています。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はなりますが、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。
- %PADDD 組み込み関数は、その引数に応じて、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。

新着情報

- 新しい指示 `/INCLUDE` は、`/INCLUDE` が SQL プリプロセッサ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 2-33. 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。
定義仕様書キーワード	EXTPROC(*JAVA:クラス名:プロシージャ名)	これを指定すると、Java メソッドが呼び出されます。
	EXTPROC(*CL:プロシージャ名)	プロシージャが戻り値に ILE CL 規則を使用することを指定します。
	EXTPROC(*CWIDEN:プロシージャ名)	プロシージャが、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:プロシージャ名)	プロシージャが、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化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。

表 2-33. V4R4 以降に変更された言語要素 (続き)

言語単位	要素	説明
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 2-34. 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。
コンパイラ指示	/FREE ... /END-FREE	/FREE... /END-FREE コンパイラ指示は、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は、組み込み SQL またはホスト変数を持つことはできません。
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。
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。

新着情報

表 2-34. 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(引数:配列 {:開始索引 {: 要素の数}))	指定された引数に近い引数、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(引数:検索テーブル {:代 替テーブル}))	指定された引数に近い引数、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 2-34. 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 以降の RPG IV の主な拡張機能は、スレッド化された環境内で ILE RPG モジュールを安全に実行するためのサポート、新しい 3 桁および 20 桁の符号付きおよび符号なし数字データ・タイプ、さらに新しい汎用文字セット バージョン 2 (UCS-2) データ・タイプのサポートと、UCS-2 フィールドと図形フィールドまたは 1 バイト文字フィールドとの間の変換のサポートです。

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。3 および 20U 符号なし整数。
 - これらの新しい整数データ・タイプによって、整数値の範囲が広がり、それとともに整数計算のパフォーマンスが向上し、64 ビット AS/400 RISC 処理装置を最大限に活用できます。
 - 新しい 3U タイプを使用すると、値によって渡された 1 バイト文字 (CHAR) の戻りの型とパラメータを持つ、ILE C プロシージャとの通信がより簡単にできるようになります。
 - 新しい INTPREC 制御仕様書のキーワードによって、式の中の整数および符号なし 2 進算術演算の中間値に、20 桁の精度を指定できます。
 - 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、多くの文字言語の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 のサポートを追加することによって、多国籍企業用に 1 つのアプリケーションを開発できるようになり、コード・ページ変換の必要性が最小限になりました。Unicode を使用することにより、保全性を失わずに、複数のスクリプトでの文字の処理が可能になります。
 - MOVE 命令と MOVEL 命令の使用による、UCS-2 フィールドと図形文字フィールドまたは 1 バイト文字フィールドとの間の変換と、新しい %UCS2 組み込み関数および %GRAPH 組み込み関数のサポート。

新着情報

- コード化文字セット ID (CCSID) の異なる UCS-2 フィールド間または図形フィールド間での、EVAL、MOVE、および MOVEL 命令を使用する変換と、新しい %UCS2 組み込み関数のサポート。

このリリースでは、これ以外の拡張も行われています。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 を指定すると、コンパイラー・リストでソース ID および SEU 順序番号からデバッグ用のステートメント番号を割り当てられます。(ステートメント番号は、コンパイラー・リスト内のエラーを識別するためにデバッガーが使用し、また、実行時エラーが発生したステートメントを識別するためにも使用されます。) *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 を使用して、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すると、デバッガーにおける READ ステートメントの STEP によって、入力仕様のステップを進むのではなく、次の計算に進みます。
- 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) によって、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 内のデフォルト値を使用できます。
 - 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化されます。
- 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 命令コードは、式を評価し、その結果を、固定長文字または図形結果に割り当てます。この割り当てにより、データは、結果内で右揃えされます。
- 新しい FOR 命令コード。反復ループを実行して、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 命令コードを使用すると、サブルーチン内の任意の地点から終了できます。
- 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 2-35. 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る文字、図形、または UCS-2 値を、プロシージャー呼び出しで渡す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。
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、多数の書き込み言語の文字をコーディングできます。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。
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。

新着情報

表 2-36. 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、コンパイル時データ、およびプログラム記述入出力フィールドと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、コンパイル時データ、およびプログラム記述入出力フィールドと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)	定義に図形と UCS-2 の CCSID を設定します。
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、右揃えされます。
	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 命令によって開始された命令の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、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 0 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 の追加
- 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。

新着情報

表 2-37. 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(*JOB RUN '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。
	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。 <div><div>'c'</div><div>Years</div><div>-----</div><div>01900-1999</div><div>12000-2099</div><div>⋮⋮</div><div>⋮⋮</div><div>92800-2899</div></div>
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 2-38. 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。

表 2-38. V3R7 以降の新しい言語要素 (続き)

言語単位	新規	説明
	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。
	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性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(*JOB RUN *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 *JOB RUN *LANGIDUNQ *LANGIDSHR 'sort-table-name')	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。
	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。

新着情報

表 2-38. V3R7 以降の新しい言語要素 (続き)

言語単位	新規	説明
	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' が指定された命令コード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。
	%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。

表 2-38. V3R7 以降の新しい言語要素 (続き)

言語単位	新規	説明
	%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、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

新着情報

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、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

自由形式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、自由形式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)
- ナル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。UPDATE 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) の日付に初期化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 2-39. 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 2-40. 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)

新着情報

表 2-40. V3R6 以降の新しい言語要素 (続き)

言語単位	新規	説明
	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返します。
	%DEC および %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返します。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。
	%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返します。
	%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返します。
	%FLOAT	数値式の値を、浮動形式に変換します。
	%INT および %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返します。
	%NULLIND	ヌル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索引数の 1 桁目、またはそれが見付からない場合には 0 を返します。
	%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 および %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。

表 2-40. V3R6 以降の新しい言語要素 (続き)

言語単位	新規	説明
	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？簡単に言えば、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

新着情報

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著作権を示す機能

このキーワードを使用して指定した著作権情報は、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 2-41. 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 2-42. 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('著作権ストリング')	著作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)

表 2-42. V3R1 以降の新しい言語要素 (続き)

言語単位	新規	説明
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。

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 Integrated Language Environment® 構成要素の役割
- Integrated Language Environment® プログラムの作成方針
- 複数のプロシージャ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

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、「*IBM Rational Development Studio for i: 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、プログラムで使用するデータを記述します。
4. **入力仕様書**には、プログラムで使用す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、プログラムで使用す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・セクション

1. **プロシージャ仕様書**は、サブプロシージャの始めと終わりにマークを付け、サブプロシージャ名およびエクスポートするかどうかを指示します。

RPG IV の概要

2. **定義仕様書**には、サブプロシージャーで使用す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する方法があります。

注: 制御仕様書に MAIN または 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。

3-3 ページの図 3-1 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。

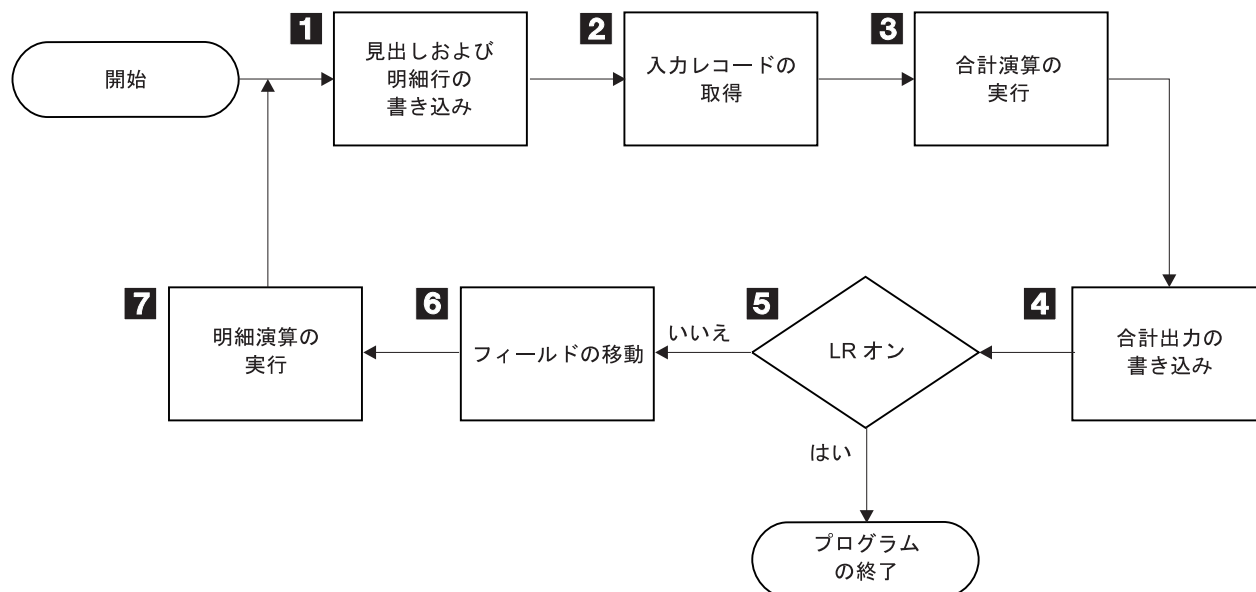


図 3-1. RPG プログラムの論理サイクル

- 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。

RPG IV の概要

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、もっとも単純です。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、あるいは特殊名 *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を使用すると、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tring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。

EMP_REC

EMP_NUMBER	EMP_NAME	EMP_RATE	
1	6	22	27

```
*.. 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('EMPLOYEE NAME')
A          EMP_RATE         5  2       TEXT('EMPLOYEE RATE')
A          K EMP_NUMBER
```

図 3-2. 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。

TRN_REC

TRN_NUMBER	TRN_HOURS	TRN_BONUS	
1	6	10	16

ILE RPG プログラムの例

```
*.. 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 3-3. 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 0 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; 1
#   *IN99 = NOT %found(employee); 2
#   if %found(employee); 3
#     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。

1. 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。
2. *IN99 には %FOUND の反対が割り当てられます。後の出力仕様書は、標識 99 によって条件設定されます。CHAIN 命令によってレコードが検出されなかった場合、標識 99 には値 *ON を指定する必要があります。
3. CHAIN 命令が正常に行われる (つまり、%FOUND が *ON を返す) 場合、その従業員の給料が評価されます。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
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'
```

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。

ILE RPG プログラムの例

- 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つかったことを示す) 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

サブプロシージャは、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。

プロシージャ仕様書はプロシージャの始めと終わりを指示します。定義仕様書は、プロシージャの戻りタイプ、プロシージャへのパラメーター、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                                           *
*-----*
FTRANSACTION IP E K DISK
FEMPLOYEE IF E K DISK
FQSYSVRT 0 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 ' ' ' '
*-----*
* 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 (TRANSACTION) 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
*-----*
OQSYSVRT H 1P 2 3
0 35 '給与計算表'
0 *DATE Y 60
0 H 1P 2
0 60 Heading1
0 H 1P 2
0 60 Heading2
0 D N1PN99 2
0 TRN_NUMBER 5
0 EMP_NAME 24
0 EMP_RATE L 33
0 TRN_HOURS L 40
0 TRN_BONUS L 49
0 Pay 60 '$ 0. '
0 D N1P 99 2
0 TRN_NUMBER 5
0 35 '** 社員ファイル上にない **'
0 T LR
0 33 'END OF LISTING'
*-----*
* 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

```

IBM i の使用

IBM i Information Center 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 IBM i と呼びます。ワークステーションから、IBM i 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

IBM i Information Center で IBM i システムを説明する資料の全リストの入手が可能です。

システムとの対話

コマンド言語 (CL) を使って IBM i 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 3-1. 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

表 3-1. よく使用される CL コマンド (続き)

処置	CL コマンド 結果
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 CRTL 論理ファイルを作成

Rational Development Studio for i

IBM Rational Development Studio for i は、IBM i 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。

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

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

ILE における RPG プログラミング

ILE RPG により Integrated Language Environment® 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、IBM i 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、IBM i でプログラミングを行うためのアプローチです。IBM i マシン・アーキテクチャーおよび
 # IBM i 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。3-12 ページの表 3-2 は、IBM i 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。

ILE における RPG プログラミング

表 3-2. IBM i でサポートされているプログラミング言語

Integrated Language Environment® (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チャーを、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チ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・ディレクトリによりプログラム・サイズを減少させることができます。

図 3-5 はプログラム作成の 2 つの方法を示したものです。

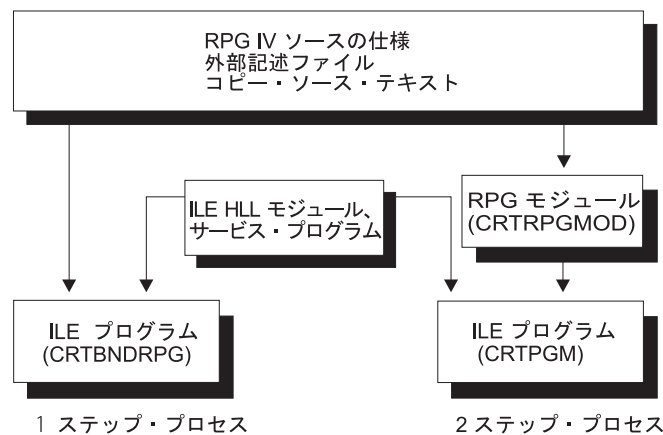


図 3-5. ILE におけるプログラム作成

一度プログラムを作成しておく、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、4-10 ページの『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、4-24 ページの『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、4-41 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。

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

ILE における RPG プログラミング

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、4-57 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、4-60 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間の呼び出しが含まれます。プロシージャは、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が後に続く任意指定のメイン・プロシージャから構成されます。プロシージャ名はバインド時 (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、4-51 ページの『プログラムの実行』を参照してください。プログラム/プロシージャ呼び出しの詳細については、4-75 ページの『プログラムおよびプロシージャの呼び出し』を参照してください。

ソースのデバッグ

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ステップスルーし、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、5-1 ページの『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、4-51 ページの『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

ILE RPG には、マルチスレッド環境で操作するための、同時と逐次化という 2 つのモードがあります。それぞれのモードには利点と欠点があります。ご使用のアプリケーションの各モジュールごとに、適合する操作モードを選択することができます。

スレッドの RPG サポートにより、静的記憶域が確実にスレッド・セーフな方法で処理されます。ただし、アプリケーションのスレッド・セーフティーのその他の側面については、ユーザーの責任で行ってください。アプリケーションが使用するすべてのプログラムがスレッド・セーフであること、共用記憶域はすべてスレッド・セーフな方法で管理すること、およびスレッド・セーフなシステムの側面のみを使用することを確認する必要があります。

ILE における RPG プログラミング

表 3-3. RPG におけるスレッド・セーフティー・モードの比較

問題	THREAD(*CONCURRENT)	THREAD(*SERIALIZE)
スレッド・セーフティーを達成するためにソース変更は必要ですか (THREAD キーワードのコーディング以外)。	いいえ。ただし、並行スレッドの数が非常に大きい場合、モジュールが使用する静的記憶域の量を減らすために、ソース・コード変更が必要となる場合があります。	いいえ
モジュール内での静的記憶域の処理によるデッドロックのリスクはありますか。	SERIALIZE がプロシージャ仕様書でコーディングされている場合は、あります。	はい。リスクは高いです。デッドロックはモジュール・レベルで発生する可能性があります。THREAD_A がモジュール MOD_1 にあり、THREAD_B がモジュール MOD_2 にある場合、各スレッドは他のモジュール内のプロシージャを呼び出そうとします。
モジュールはマルチスレッドの実行によって利益を得ますか。	はい	いいえ
ボトルネックのリスクがありますか。	SERIALIZE キーワードがプロシージャ仕様書でコーディングされている場合は、あります。	はい。リスクは高いです。モジュールへのアクセスの逐次化により、モジュールはアプリケーション内のボトルネックになる可能性があります。あるスレッドがモジュールでアクティブになっている場合、他のスレッドは、すべてのプロシージャにおいて最初のスレッドがモジュールで実行されなくなるまで待つ必要があります。
スレッド・ローカル記憶域はサポートされていますか。	はい。これは静的記憶域のデフォルト・タイプです。	いいえ
all-thread 静的記憶域はサポートされていますか。	はい	はい。これは唯一サポートされている静的記憶域のタイプです。
RPG プログラマーは、静的変数がスレッド・ローカルであるか、またはすべてのスレッドによって共用されるかを選択することができますか。	はい	いいえ。all-thread 静的記憶域のみがサポートされています。
実行時に必要なメモリー量に関する問題はありますか。	可能性があります。モジュールに必要な静的記憶域の量は、モジュールを使用しているスレッドの数によって乗算されます。	いいえ。すべてのスレッドが同じ静的記憶域を使用します。
対象ユーザーは誰ですか。	複数のスレッドでの実行によるパフォーマンスの利点を求めており、各スレッドが使用するスレッド・ローカル静的記憶域の量を受け入れる用意があるか、RPG モジュールを再作成して、できるだけ少量の静的記憶域を使用したい (あるいはその両方) RPG プログラマー。	モジュールを再作成して静的記憶域の量を減らすことを望まない RPG プログラマー、または THREAD(*CONCURRENT) で必要となるスレッドごとの追加記憶域について懸念する RPG プログラマー。複数のスレッドが同時にモジュール内でプロシージャを実行する場合、RPG プログラマーは、モジュールがボトルネックになる可能性があるという事実を受け入れます。

詳細については、4-108 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。

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して行うことができます。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 の拡張機能を利用できますが、すべての ILE の機能拡張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

3-18 ページの図 3-6 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。

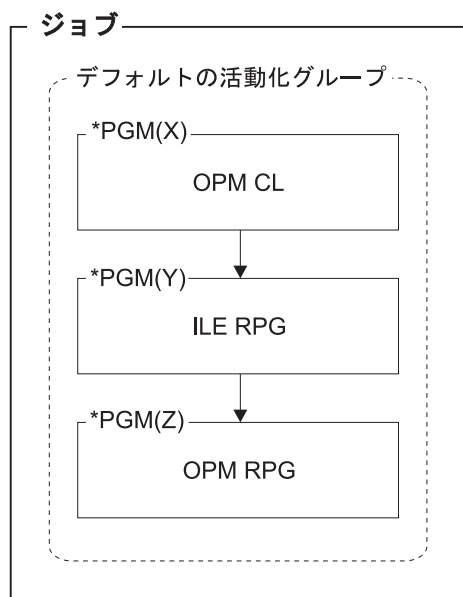


図 3-6. OPM 互換アプリケーション・プログラム

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。

プログラムの呼び出し

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クリーンアップするためには、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

7-9 ページの『ソースの変換』

ワン・ステップ作成処理

4-10 ページの『CRTBNDRPG コマンドによるプログラムの作成』

ILE 静的バインド

4-75 ページの『プログラムおよびプロシーチャーの呼び出し』。また「*ILE* 概念」

例外処理の相違

5-63 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、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

3-20 ページの図 3-7 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。

CRTBNDRPG を使う ILE プログラム

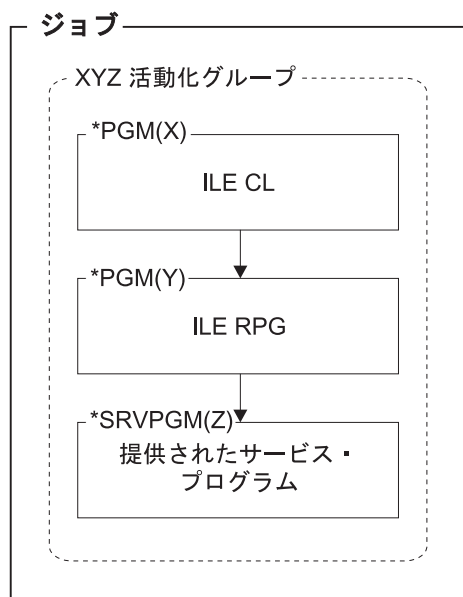


図 3-7. CRTBNDRPG を使う ILE プログラム

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。

プログラムの呼び出し

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

7-9 ページの『ソースの変換』

ワン・ステップ作成処理

4-10 ページの『CRTBNDRPG コマンドによるプログラムの作成』

活動化グループ

4-57 ページの『活動化グループの管理』

RCLRSC

4-60 ページの『資源再利用コマンド』

ILE 静的バインド

4-75 ページの『プログラムおよびプロシージャの呼び出し』。また「*ILE* 概念」

例外処理の相違

5-63 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・デフォルトの範囲設定

6-13 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 6-17 ページの『オープン・デフォルト・パスの共用』。また「*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せんが、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、3-20 ページの『ILE の影響』で説明したのと同じです。

この方法を使用する前に「*ILE* 概念」で基本的な ILE 概念について理解して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。

CRTRPGMOD を使う ILE アプリケーション

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と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 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・プログラムと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。3-23 ページの図 3-8 は、このアプリケーションの実行時のビューです。

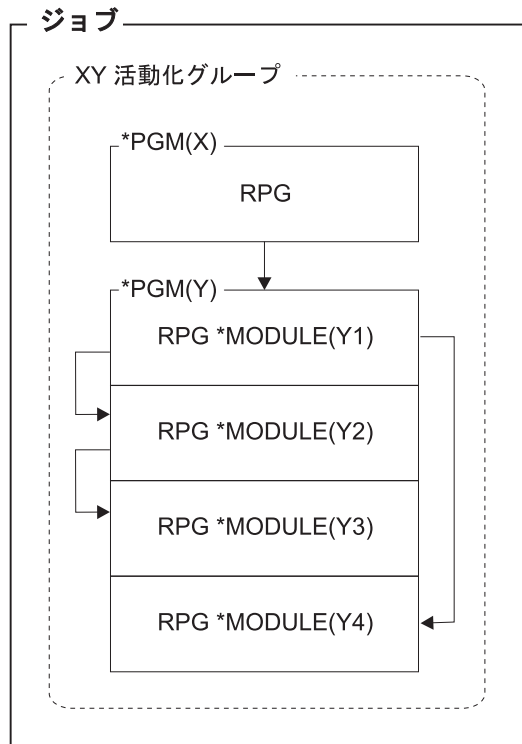


図 3-8. CRTRPGMOD および CRTPGM を使用する単一言語のアプリケーション・プログラム

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、3-20 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チ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。

3-24 ページの図 3-9 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。

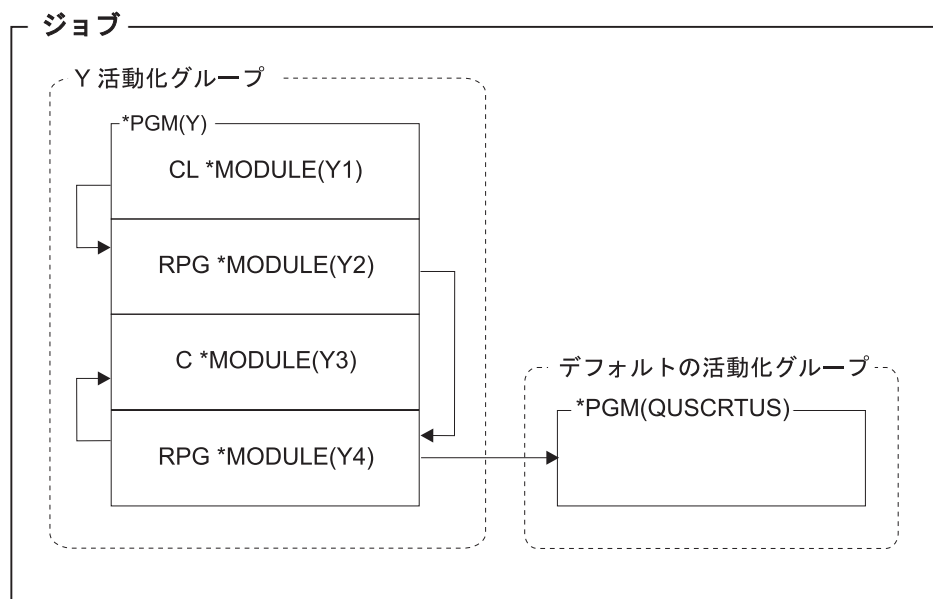


図 3-9. 混合言語のアプリケーション・プログラム

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、3-20 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。

3-25 ページの図 3-10 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。

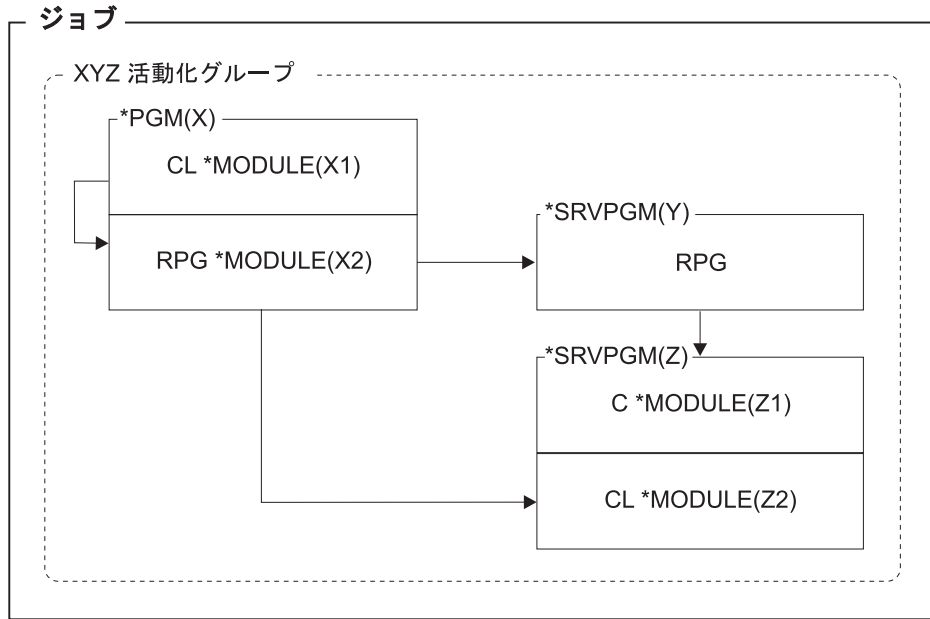


図 3-10. 拡張アプリケーション・プログラム

プログラム X からサービス・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、3-20 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

4-24 ページの『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

4-57 ページの『活動化グループの管理』

ILE 静的バインド

4-75 ページの『プログラムおよびプロシーチャーの呼び出し』。また「ILE 概念」

例外処理

5-57 ページの『例外の処理』。また「ILE 概念」

サービス・プログラム

4-41 ページの『サービス・プログラムの作成』。また「ILE 概念」

プログラムの更新

4-39 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、3-26 ページの図 3-11 に示すようなシナリオは避けてください。

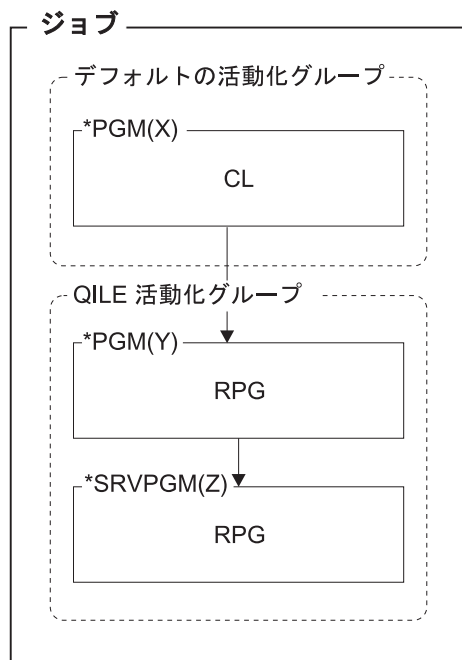


図 3-11. 避けるべきシナリオ： 1 つのアプリケーション・プログラムに、 OPM のデフォルトの活動化グループ内の CL プログラムと、指定の活動化グループ内の ILE プログラムが入っています。

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。

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・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から構成されています。

1. **プロシージャ**とは、バインドされた呼び出しを使用して呼び出される内蔵タイプの計算ユニットのことです。
2. **RPG コンパイラー**は、**RPG プログラマー**が、バインドされた呼び出しでリニア・メイン・プロシージャを呼び出さないように制限します。代わりに、リニア・メイン・プロシージャに対するバインドされた呼び出しは、コンパイラーが提供するプログラムの**プログラム入力プロシージャ (PEP)**によって行われます。リニア・メイン・プロシージャのプロトタイプは常に **EXTPGM** キーワードを使用するため、プロトタイプを使用する呼び出しがプログラム呼び出しを実行します。

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

メイン・プロシージャおよびサブプロシージャ

ILE RPG モジュールはゼロ以上のサブプロシージャおよびメイン・プロシージャ (オプション) で構成されています。メイン・プロシージャとは、プログラム入力プロシージャとして指定できる (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) プロシージャです。サイクル・メイン・プロシージャは、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・セクションで定義できます。リニア・メイン・プロシージャはサブプロシージャ・セクションで指定でき、また制御仕様書の **MAIN** キーワードで特別に指定することができます。プロシージャおよびプログラム・ロジック・サイクルについての追加情報は、「WebSphere Development Studio ILE RPG 解説書」を参照してください。

サブプロシージャは、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です。サブプロシージャは次の点で基本的にメイン・プロシージャと異なります。

- サブプロシージャ内で定義された名前は、サブプロシージャ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。
- サブプロシージャへの呼び出しはバインドされたプロシージャ呼び出しでなければならない。
- P、F、D、および C 仕様書しか使用することができない。
- バインドされた呼び出しではなく、プログラム呼び出しを介して呼び出される以外は、リニア・メイン・プロシージャは他のサブプロシージャと同じです。

サブプロシージャは、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。

サブプロシージャは別の機能を提供します。値によってサブプロシージャ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を呼び出すことができます。3-28 ページの図 3-12 は、複数プロシージャ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。

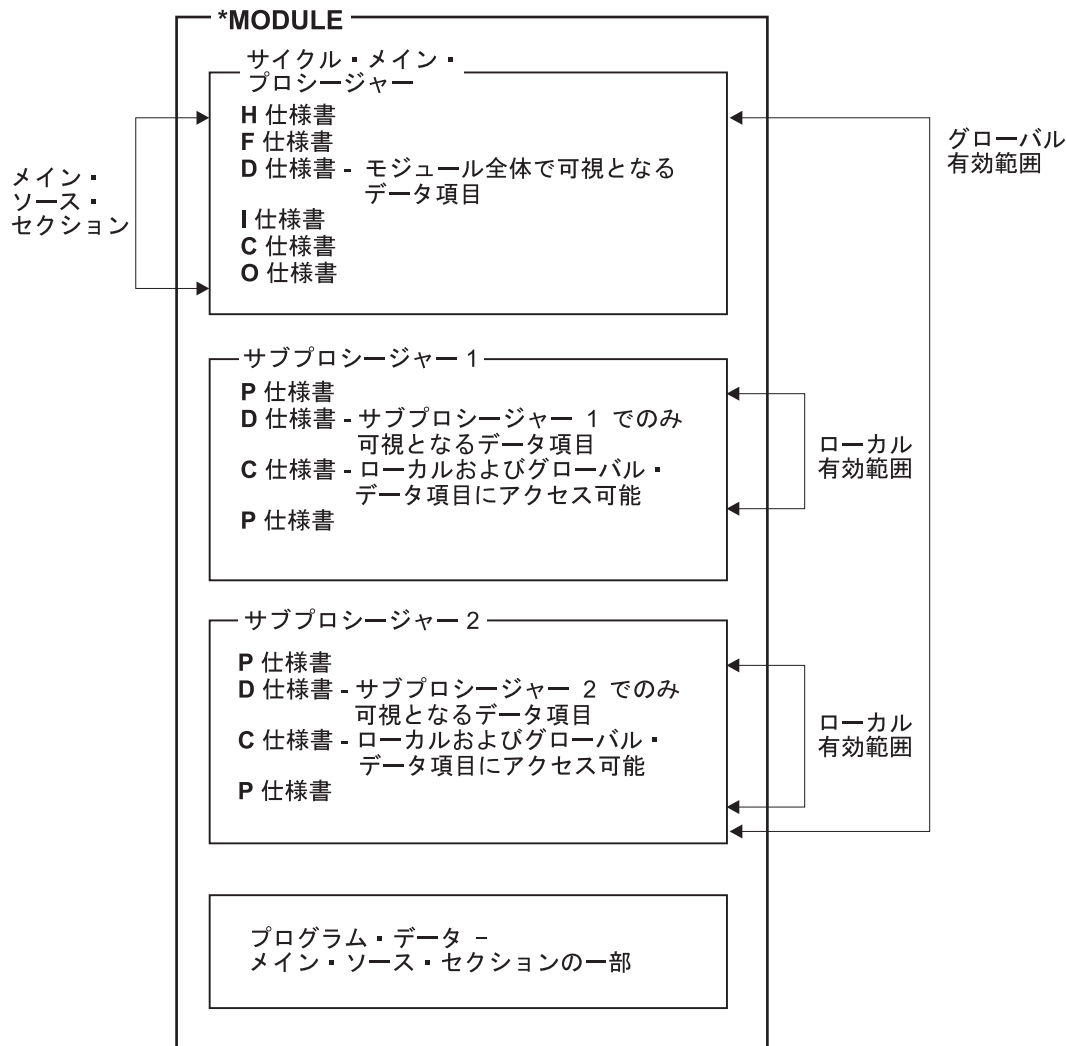


図 3-12. 複数プロシージャーのある ILE RPG サイクル・モジュール

図から分かりますとおり、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 3-13 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
// 注意してください)。これには 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 3-13. `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 3-14 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 3-14. `FmtAddr` プロシージャーのプロトタイプ

プロシージャーが呼び出しと同じモジュールにコーディングされた場合は、プロトタイプの指定は任意です。プロトタイプが指定されていないと、コンパイラーはプロシージャー・インターフェースからプロトタイプを生成します。ただし、プロシージャーがエクスポートされ、かつ別の `RPG` モジュールまたはプログラムからも呼び出される場合は、コピー・ファイルでプロトタイプを指定し、そのコピー・ファイルを読み出しモジュールとプロシージャーをエクスポートするモジュールとの両方にコピーするようにしてください。

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いたい場合は式で、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 3-15 は `FmtCust` への呼び出しを示します。図 3-13 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 3-15. `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。

複数プロシージャ・モジュール

- 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
- 同じモジュール内のサブプロシージャ

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 3-16 はこの呼び出しを示しています。

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

図 3-16. FmtAddr プロシージャの呼び出し

上図のように、値を戻すためのプロシージャの使用によって、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）

複数プロシージャのあるモジュール例

ここで、複数プロシージャ・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 3-31 ページの図 3-17 に示されたものと類似したものになります。

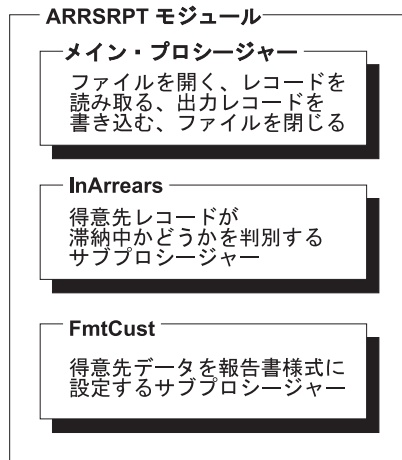


図 3-17. ARRSRPT モジュールの構成要素

ここで、3-32 ページの図 3-18 に示されている最初の子プロシージャ 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、サブプロシージャはメイン・プロシージャに 'I' を戻します。

複数プロシージャのあるモジュール例

```
//-----  
// 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 3-18. サブプロシージャ *InArrears* のソース

図 3-18 はすべてのサブプロシージャ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はプロシージャ仕様書で始まり、終わります。
- 2 開始プロシージャ仕様書 (プロシージャ仕様書の 24 桁目に B) の後に、プロシージャ・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 サブプロシージャで使用される変数またはプロトタイプは、プロシージャ・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、サブプロシージャは '0' をメイン・プロシージャに戻します。

すべてのサブプロシージャの場合、およびプロトタイプ入りロパラメーターのあるサイクル・メイン・プロシージャの場合も、プロシージャ・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・インターフェース定義**は、プロトタイプが指定されていた場合は、プロシージャの定義内でのプロトタイプ情報の繰り返しとなります。これは、プロシージャの入ロ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・インターフェース定義は、プロシージャ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。プロトタイプが指定されていないと、コンパイラーがプロシージャ・インターフェースからプロトタイプを生成するため、プロシージャ・インターフェース定義は内部定義と外部定義の両方を提供します。InArrears の場合には、入ロパラメーターはありません。

3-33 ページの図 3-19 に示す次のサブプロシージャ 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 3-19. サブプロシージャ 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 FmtAddr を考えてみます。3-31 ページの図 3-17 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を入れるユーティリティー・モジュールです。

3-34 ページの図 3-20 に、モジュール FMTPROCS のソースを示します。プロシージャ FmtAddr は別のモジュールから呼び出されるので、プロトタイプが必要となります。プロトタイプを共用できるように、プロトタイプを /COPY ファイルに入れてあります。このファイルは、プロシージャの呼び出し方法に関する情報を提供する目的で呼び出しモジュールと、プロトタイプがプロシージャ・インターフェースと一致するようにする目的でプロシージャを定義するモジュールの両方にコピーされます。

複数プロシージャのあるモジュール例

```
//=====
// 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・プロシージャを持ちません。
//=====
H NOMAIN
//-----
// プロトタイプ・プロシージャ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 QRPGLSRC,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 3-20. サブプロシージャ *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、サブプロシージャだけから構成されていて、メイン・プロシージャはありません。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、3-38 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。3-35 ページの図 3-21 はミニ・アプリケーションの異なる部分を示します。

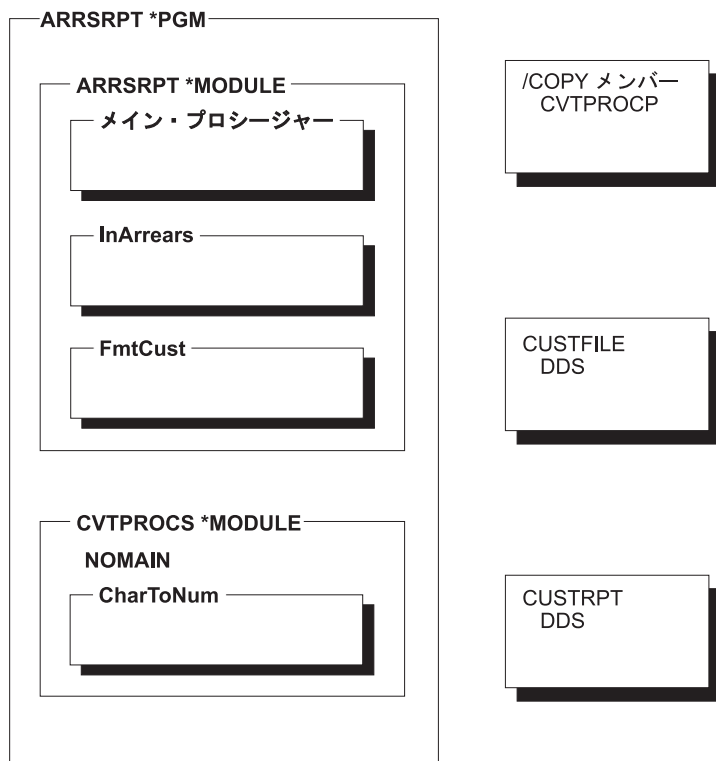


図 3-21. ARRSRPT アプリケーション・プログラム

3-36 ページの図 3-22 は ARRSRPT モジュールのソース全体を示したものです。

複数プロシージャーのあるモジュール例

```
//=====
// モジュール ARRSRPT のソース。サイクル・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
// CUSRPT - プリンター・ファイル (様式 ARREARS を使用)
//-----
FCUSTFILE  IP      E          DISK
FCUSRPT    0      E          PRINTER
*-----*
* プロトタイプ
*-----*
/COPY QRPGLF,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
*-----*
* 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              MOVE    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);
// 入力ファイルの中で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。
```

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 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・プロシージャに続く各サブプロシージャには、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、図 3-23 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 3-23. ARRSRPT の出力

図 3-24 および 3-38 ページの図 3-25 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 3-24. CUSTFILE の DDS

コーディング上の考慮事項

```
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
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 3-25. 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、Application Development Manager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を含めることを要求し、メイン・プロシージャしかプログラム入力プロシージャ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・プロシージャの考慮事項

メイン・プロシージャ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。

次の考慮事項は、サイクル・メイン・プロシージャにのみ適用されます。

- サイクル・メイン・プロシージャ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であるため、モジュール内のすべてのプロシージャ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。
- サイクル・メイン・プロシージャは常にエクスポートされます。このことは、プログラム内の他のプロシージャがバインド呼び出しの使用によってメイン・プロシージャを呼び出すことができることを意味します。
- サイクル・メイン・プロシージャの呼び出しインターフェースは次の 2 つの方法のいずれかで定義することができます。
 1. プロシージャ・インターフェースと任意指定のプロトタイプの使用
 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、プロシージャ呼び出しを使用して呼び出されます。

サブプロシージャの考慮事項

これらの考慮事項は、特に指定のない限り、通常のサブプロシージャおよびリニア・メイン・プロシージャに適用されます。

- 演算命令のどれもサブプロシージャでコーディングすることができます。ただし、入力仕様書および出力仕様書はサブプロシージャでサポートされていないため、サブプロシージャでローカルに定義されたファイルに対するファイル入出力操作には、データ構造結果フィールドを使用する必要があります。すべてのデータ域は、サブプロシージャで使用することができますが、メイン・ソース・セクション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

コーディング上の考慮事項

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せん。

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 (リニア・メイン・プロシージャを含む) の実行時の動作は、サブプロシージャのためのサイクル・コードがないために、サイクル・メイン・プロシージャの動作とは一部異なります。
 - サブプロシージャが終了すると、自動記憶域内で開かれているローカル・ファイルがクローズされます。ただし、グローバル・ファイルのクローズなどの終了活動はいずれも、サブプロシージャ自身と関連したサイクル・メイン・プロシージャ (存在する場合) が終了するまでは発生しません。活性化グループの終了前に確実にグローバル・ファイルがクローズするようにするには、アプリケーション終了時のプログラム入力プロシージャ、およびプログラム入力プロシージャ用に使用可能な取り消し処理プログラムの両方によって呼び出される『終結処置』サブプロシージャをコーディングします。

終結処理プロシージャの使用に代わる方法は、暗黙ファイルのオープンまたはデータ域ロックが発生しないように、また任意のサブプロシージャ内でオープンがクローズと対応し、`IN` が `OUT` と、`CRT` が `DLT` と対応するというように、モジュールをコーディングすることです。サイクル・メイン・プロシージャがアクティブでないときに、サブプロシージャがアクティブになっている可能性がある場合に、サイクル・モジュールに対してこの代替処理を考慮する必要があります。

- サブプロシージャ内の例外処理は、サブプロシージャ用のデフォルトの例外処理プログラムがないために、基本的にサイクル・メイン・プロシージャと異なっています。結果として、サイクル・メイン・プロシージャ用にデフォルトの処理プログラムが呼び出された場合には、サブプロシージャの異常終了が発生します。

詳細情報について

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

トピック

参照先

例外処理

5-61 ページの『サイクル・メイン・プロシージャ内の例外処理プログラム』

メイン・プロシージャの終了

4-102 ページの『メイン・プロシージャからの戻り』

サブプロシージャ

トピック

参照先

定義 「*IBM Rational Development Studio for i: ILE RPG 解説書*」のサブプロシージャに関する章。

NOMAIN モジュール

4-27 ページの『NOMAIN モジュールの作成』

例外処理

5-62 ページの『サブプロシージャ内の例外処理プログラム』

プロシージャ仕様書

「*IBM Rational Development Studio for i: ILE RPG 解説書*」のプロシージャ仕様書に関する章。

プロシージャ・インターフェース

「*IBM Rational Development Studio for i: ILE RPG 解説書*」のデータおよびプロトタイプの変数に関する章。

サブプロシージャの終了

4-104 ページの『サブプロシージャからの戻り』

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

4-82 ページの『プロトタイプ呼び出しの使用』

一般情報

「*IBM Rational Development Studio for i: ILE RPG 解説書*」の第 24 章

パラメーターの受け渡し

4-84 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

「*IBM Rational Development Studio for i: ILE RPG 解説書*」のデータおよびプロトタイプの変数に関する章。

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の呼び出し

以降に示すページで、多くの Integrated Language Environment® 用語および概念が簡単に説明されています。これらの用語と概念の詳細な説明は、「*ILE 概念*」にあります。

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

最初に、ソース・ステートメントを QRPGLSRC と呼ばれるファイルに入れる必要があります。ファイル QRPGLSRC 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 QRPGLSRC です。IBM は、ライブラリー QGPL のソース・ファイル QRPGLSRC 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 4-1 に示すように新しい ILE RPG 構造を考慮に入れています。

12	80	20
順序番号/日付	コード	コメント

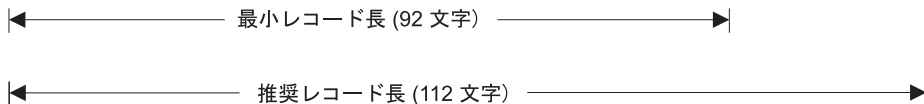


図 4-1. ILE RPG レコード長の説明

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の詳細については、資料「ADTS for AS/400: Source Entry Utility」および資料「ADTS/400: Programming Development Manager」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、資料「ADTS for AS/400: Source Entry Utility」を参照してください。

ソース・ファイルを QRPGLSRC 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するには、次の入力を行います。

```
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. QRPGLSRC というソース物理ファイルを作成するには、次の入力を行います。

```
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。

- 編集セッションを開始し、ソース・メンバー EMRPRT 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(EMRPRT)
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 EMRPRT 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 4-2 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+++++
***** データの初め *****
:
:
:
:
:
:
:
:
:
:
:
:
:
:
:
:
:
***** 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  F11=切り替え
F16=検索の反復  F17=変更の反復  F24=キーの続き
メンバー EMRPRT がファイル MYLIB/QRPGLESRC に追加された。  +
```

図 4-2. 新しいメンバーの編集画面

- 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 — プロシージャール仕様書の場合

SEU の使用

```

=====
* 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 4-3. 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。

4-5 ページの図 4-4 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 4-4. 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自由形式または固定形式で入れます。
- 自由形式の場合
 - 区切り文字「EXEC SQL」を使用して SQL ステートメントを開始します。
 - SQL ステートメントは複数の行に入れることができます。継続文字は不要です。
 - SQL ステートメントの終了を示すにはセミコロンを使用します。
- 固定形式の場合
 - 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。
 - 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。
- 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。

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。

4-6 ページの図 4-5 と 4-6 ページの図 4-6 は組み込み SQL ステートメントの例です。

SQL ステートメントの使用

```
...+....1....+....2....+....3....+....4....+....5....+....6....+....7..
/FREE
  X = Y + Z; // ILE RPG calculation operations

  // The entire SQL statement is on one line
  EXEC SQL INSERT INTO MYLIB/MYFILE (FLD1) VALUES(12);

  // The SQL statement begins on the same line as
  // EXEC SQL and then it is split across several lines
  EXEC SQL   INSERT
              INTO MYLIB/MYFILE
              (FLD1) VALUE(12);

  // The SQL statement begins on the line after
  // EXEC SQL and then it is split across several lines
  EXEC SQL
      INSERT INTO MYLIB/MYFILE
              (FLD1) VALUE(12);

  X = Y + Z; // ILE RPG calculation operations
/END-FREE
```

図 4-5. ILE RPG プログラムの中の自由形式 SQL ステートメント

```
...+....1....+....2....+....3....+....4....+....5....+....6....+....7..
* ILE RPG calculation operations
C              EVAL      X = Y + Z

* The entire SQL statement is on one line
C/EXEC SQL INSERT INTO MYLIB/MYFILE (FLD1) VALUES(12)
C/END-EXEC

* The SQL statement begins on the same line as
* EXEC SQL and then it is split across several lines
C/EXEC SQL   INSERT
C+           INTO MYLIB/MYFILE
C+           (FLD1) VALUE(12)
C/END-EXEC

* The SQL statement begins on the line after
* EXEC SQL and then it is split across several lines
C/EXEC SQL
C+   INSERT INTO MYLIB/MYFILE
C+   (FLD1) VALUE(12)
C/END-EXEC

* ILE RPG calculation operations
C              EVAL      X = Y + Z
C
```

図 4-6. ILE RPG プログラムの中の固定形式 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/systems/i/infocenter/> で **IBM i Information Center** の中の『データベースおよびフ
ァイル・システム』カテゴリの『*DB2 for i*』の節を参照してください。

SEU による SQL ステートメントの構文検査の処理方法については、資料「*ADTS for AS/400: Source Entry Utility*」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・ファイルを QSYS ファイル・システム内または統合ファイル・システム内のいずれかに指定できるパラメーターが含まれています。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 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"/QRPGLSRC,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 4-1. 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/QRPGLSRC) 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/QRPGLSRC) 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 は統合ファイル・システムの有効な名前であることを注意)
/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)

IFS ソース・ファイルの使用

表 4-1. QSYS および IFS の /Copy ファイル名解釈 (続き)

/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 "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ん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/r1.2/inc:/driver/r1.1/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 4-2. /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/r1.2/inc/file.rpg /driver/r1.1/inc/file.rpg /home/auser/temp/file.rpg /home/auser/src/file.rpg QSYS の場合: FILE(*LIBL/QRPGLESRC) MBR(FILE.RPG)
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/r1.2/inc/file /driver/r1.2/inc/file.rpgleinc /driver/r1.2/inc/file.rpgle /driver/r1.1/inc/file /driver/r1.1/inc/file.rpgleinc /driver/r1.1/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

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、プロシーチャーを呼び出し (他のモジュールまたはサービス・プログラム)、プロシーチャー・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・プロシーチャーを呼び出す
- 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チャーを呼び出すことができます。

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、4-24 ページの『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。

『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』 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。

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

表 4-3. プログラム識別

パラメーター	説明
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。

表 4-4. プログラムの作成

パラメーター	説明
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。
OPTION(*UNREF)	*UNREF/*NOUNREF は、参照されないフィールドをプログラム・オブジェクトに配置するかどうかを決める。
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。
DBGENCKEY(*NONE)	プログラムのリスト・デバッグ・ビューに対して暗号化を指定する。
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。
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。
STGMDDL(*SINGLVL)	プログラムの記憶域モデルを指定する。

CRTBNDRPG コマンドの使用

表 4-5. コンパイラー・リスト

パラメーター	説明
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。

表 4-6. データ変換オプション

パラメーター	説明
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)	ヌル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。

表 4-7. 実行時の考慮事項

パラメーター	説明
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(*STGMDL)	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)	使用する分類順序テーブルを指定する。
LANGID(*JOB RUN)	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)	固定長形式操作でバック 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。2 番目のパラメーターは、ストリーム・ファイルまたはモジュールのどちらに生成されるかを示す。
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、7-32 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 4-4 ページの図 4-3 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事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 4-7 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			
プログラム :	EMPRPT	ライブラリー:	MYLIB
		モジュール:	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 EMPTYTYPE	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=変数監視 F18=監視の処理 F24=キーの続き			

図 4-7. 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、5-1 ページの『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。

CRTBNDRPG コマンドの使用

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、4-41 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。

OPM 互換プログラム・オブジェクトの作成

この例では、4-15 ページの図 4-8 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                                             *
*-----*
FTRANSACTION IP   E           K DISK
FEMPLOYEE    IF   E           K DISK
FQSPRT       0    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 (TRANSACTION) 内の各レコードごとに、
* 社員が見つけ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
*-----*
OQSPRT      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 4.8. 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

コンパイラー・リストの使用

- 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 7-56 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・リスト中に含まれます。

*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わるかの詳細については、『*NOSRCSTMT ソース見出し』および『*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

コンパイラー・リストの使用

ト出力内の行スペーシングを制御します。これらのディレクティブの詳細については、「*IBM Rational Development Studio for i: 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 4-9 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 番号	ソースの仕様	注記	Src Seq ID	番号
33 C	*****		26 - 35	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 4.9. 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場合があります。 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つけた場合には、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 4-10 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	番号	ソースの仕様	注記	Do	Page	変更	Src	Seq	日付	ID	番号
63	C	SETOFF	12								003100
=====			aabb								
=====			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 4-10. サンプル・フィンガー・インライン診断メッセージ

コンパイラー・リストの使用

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 'ccccc' 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、'ccccc' 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 4-11 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	日付	ID	番号
番号	1...2...3...4...5...6...7...8...9...10										
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 4-11. 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。

可能な時は、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・リストのブラウズの詳細については、「ADTS for AS/400: Source Entry Utility」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る場合は、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、7-65 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。

コンパイラー・リストの使用

- 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 (4-25 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 データ域にアクセスするには、*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 コマンドまたは 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

RETURNCODE データ域のアクセス

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

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

Rational Developer for i を使用します。これは推奨される方法であり、ILE RPG プログラムの作成に関する資料が製品のオンライン・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、およびモジュールのすべてのプロシージャ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 **サイクル・メイン・プロシージャ**。サイクル・メイン・プロシージャには、独自の LR 意味構造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**。サブプロシージャは RPG サイクルを使用しません。サブプロシージャそれ自体には使用可能なローカル記憶域があります。サイクル・メイン・プロシージャがコーディングされていない場合、サブプロシージャの 1 つは、リニア・メイン・プロシージャとして指定される場合があります。

メイン・プロシージャ (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だけがこれを読み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がこれを読み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。

モジュールの概念の詳細については、「*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。

『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』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 7-32 ページの『付録 C. 作成コマンド』に入っています。

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

表 4-8. モジュールの指定

パラメーター	説明
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(*MODULE)	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モジュールのリストを示す。

モジュール・オブジェクトの作成

表 4-8. モジュールの指定 (続き)

パラメーター	説明
TEXT(*SRCMBRTXT)	モジュールの簡単な説明を用意する。

表 4-9. モジュールの作成

パラメーター	説明
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。
OPTION(*UNREF)	*UNREF/*NOUNREF は、参照されないフィールドをモジュールに配置するかどうかを決める。
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。
DBGENCKEY(*NONE)	モジュールのリスト・デバッグ・ビューに対して暗号化を指定する。
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。
STGMDL(*INHERIT)	モジュールの記憶域モデルを指定する。

表 4-10. コンパイラー・リスト

パラメーター	説明
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。

表 4-11. データ変換オプション

パラメーター	説明
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)	ヌル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。

表 4-12. 実行時の考慮事項

パラメーター	説明
SRTSEQ(*HEX)	使用する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(*JOB RUN)	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。
INFOTMF(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。2 番目のパラメーターは、ストリーム・ファイルまたはモジュールのどちらに生成されるかを示す。
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、4-15 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 7-56 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 4-29 ページの図 4-12 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、4-30 ページの図 4-13 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。

モジュール・オブジェクトの作成

- 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:  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 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
*-----
* サブプロシージャ 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 4-12. TRANSSVC メンバーのソース・ステートメント

モジュール・オブジェクトの作成

```
* 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 4-13. 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 のプロシージャを呼び出します。このモジュールのソースが 4-31 ページの図 4-14 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、5-4 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:  TRANSRPT
*   プロシージャ 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
*   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  0  F  80      PRINTER      OFLIND(*INOF)
/COPY QRPGL,TRANSP
* プロシージャ '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 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
O          12 'Product name'
O          40 'Quantity'
O          54 'Income'
OQSYSPRT  H  1P          1
O          OR  OF
O
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 4-14. 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 4-32 ページの図 4-15 に示します。/COPY メンバーは 4-30 ページの図 4-13 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 4-15. 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、4-44 ページの『サンプル・サービス・プログラム』。
- サービス・プログラムに使用するモジュールの作成例については、4-48 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、4-68 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、5-52 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、サイクル・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、サイクル・メイン・プロシージャーを持つ各 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 つのサイクル・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。リニア・メイン・プロシージャーを持つ、またはメイン・プロシージャーを持たない RPG モジュールは、RPG サイクルを使用しません。また、他のモジュールのサイクルにも影響しません。

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/systems/i/infocenter/> で **IBM i 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、IBM i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、プロシージャは始動プロシージャ、すなわち**プログラム入力プロシージャ**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に関連するユーザーのコードが、ユーザー入りロプロシージャです。

ILE RPG モジュールにメイン・プロシージャがあると、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、入りロモジュールとして指定することができます。

図 4-16 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 TRANSVC 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 は入りロモジュールです。

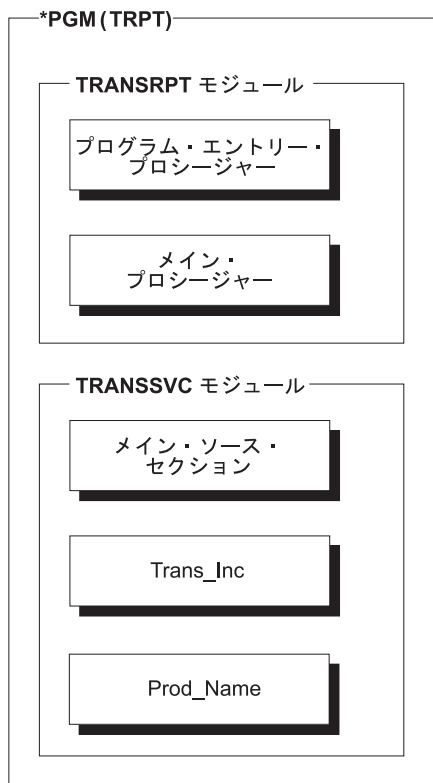


図 4-16. プログラム TRPT の構造

バインドされたオブジェクト内では、プロシージャは静的プロシージャ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

プログラムへのモジュールのバインド

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す。サービス・プログラムの作成についての情報は、4-41 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。

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、CRTCLMOD、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、4-58 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 4-13 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詳細な説明は、Web サイト <http://www.ibm.com/systems/i/infocenter/> で **IBM i Information Center** の中の『プログラミング』カテゴリの『CL および API』の節を参照してください。

表 4-13. 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。

プログラムへのモジュールのバインド

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の変数またはプロシージャ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 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、4-31 ページの図 4-14、4-29 ページの図 4-12、および 4-32 ページの図 4-15 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。

4-37 ページの図 4-17 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 4-17. 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、4-48 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、5-52 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 (UPDPM)
- 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/systems/i/infocenter/> で **IBM i**
Information Center 中の『プログラミング』カテゴリの『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 4-14. 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

バインダー・リストの使用

表 4-14. 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 (続き)

セクション名	*BASIC	*EXTENDED	*FULL
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
相互参照リスト			X
バインディング統計			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、4-50 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・リストの詳細については、「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。変更必要箇所は、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チ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た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場合があります。

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、5-66 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。

モジュールまたはプログラムの変更

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チャー・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/systems/i/infocenter/> で IBM i
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。

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

Rational Developer for i 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によって呼び出すことができる、一組のプロシージャ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。その代わりに、サービス・プログラムの関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・

サービス・プログラムの概要

リストを使って、プロシーチャーの呼び出しに使用可能な変数およびプロシーチ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

サブプロシーチ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・プログラムを著作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・プロシーチ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、4-44 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、マスク・エクスポート、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 4-15 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詳細な説明は、Web サイト
<http://www.ibm.com/systems/i/infocenter/> で **IBM i 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 4-15. 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(library name/service program name) 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、4-47 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、4-38 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。

4-45 ページの図 4-18 はサービス・プログラムのソースを示したものです。4-47 ページの図 4-19 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チ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、サブプロシーチ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. プロシーチ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チ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、4-87 ページの『操作記述子の使用』を参照してください。

```

=====
* CvtToHex - 入カストリングを 16 進出カストリングに変換
=====
H COPYRIGHT(' (C) Copyright MyCompany 1995')
D/COPY RPGGUIDE/QRPGLE,CVTHEXPR
-----
* メイン入カパラメーター
* 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                  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 4-18. サービス・プログラム CvtToHex のソース


```

*-----*
* 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

*-----*
* 入カ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

```

=====			
* 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 4-19. 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 4-20 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 4-20. 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に 1 つが必要です。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の詳細については、「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、4-45 ページの図 4-18 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 4-20 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/QRVSRSC)
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 QSRVSRSC 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・ディレクトリは不要であることに注意してください。

サンプル・サービス・プログラム

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、4-49 ページの図 4-21 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まれます。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。(入カストリングは '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 文字の入カストリングを使用 *
* 2. 10 文字の 16 進ストリングに変換 (結果が入カストリング全体に *
*   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 0 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
0 'Result14+++++'
OQSYSPRT H 1P
0 'Result10++'
OQSYSPRT E
0 ResultDS
0 Comment +5

```

図 4-21. 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。

サンプル・サービス・プログラム

1. 新しいプロシージャ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を加えるには、次のようにします。

1. 新しいプロシージャのモジュール・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・プログラムの更新の詳細については、「*ILE* 概念」を参照してください。

サンプル・バインダー・リスト

4-51 ページの図 4-22 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・リストの詳細については、4-37 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				MYLIB/CVTHEXPGM	ISERIES1	08/15/02	ページ 1
5769WDS V5R2M0 020719 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・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・プログラム	ライブラリー	サービス・プログラム	ライブラリー	サービス・プログラム	ライブラリー	サービス・プログラム	ライブラリー
CVTTOHEX	MYLIB						
バインド・ディレクトリー	ライブラリー	バインド・ディレクトリー	ライブラリー	バインド・ディレクトリー	ライブラリー	バインド・ディレクトリー	ライブラリー
*NONE							
プログラムの作成				MYLIB/CVTHEXPGM	ISERIES1	08/15/02	ページ 2
5769WDS V5R2M0 020719 23:24:00							
簡易要約表							
プログラム入力プロシージャ	:	1					
記号	タイプ	ライブラリー	オブジェクト ID				
	*MODULE	MYLIB	CVTHEXPGM _QRNP_PEP_CVTHEXPGM				
複数指定	:	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				MYLIB/CVTHEXPGM	ISERIES1	08/15/02	ページ 3
5769WDS V5R2M0 020719 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 4.22. CVTHEXPGM の基本バインダー・リスト

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。資料「CL プログラミング (SD88-5038-06)」には、このメニューに関する情報が含まれています。
- PDM 開始 (STRPDM) コマンド。資料「ADTS/400: Programming Development Manager」には、このコマンドに関する情報が含まれています。
- QCMDExc プログラム。資料「CL プログラミング」には、このプログラムに関する情報が含まれています。

サンプル・サービス・プログラム

- 高水準言語。4-75 ページの『プログラムおよびプロシーチャーの呼び出し』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チャーの呼び出しについての情報があります。

注: Rational Developer for i 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/systems/i/infocenter/> で **IBM i 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、IBM i 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 プロシーチャーがプログラム入力プロシーチ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、唯一 *ENTRY PLIST が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。

パラメーターの処理方法の詳細な説明については、資料「CL プログラミング」で『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。4-53 ページの図 4-23 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      0      F      80      PRINTER
FEMPMST     IP      E              K DISK
IEMPREC          01

*-----*
* 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
C      SETON
C      ENDIF
C      LR99
OPRINT      H      1P              2  6
0
0      H      1P
0
0              12 'NAME'
0              34 'SERIAL #'
0              45 'DEPT'
0              56 'TYPE'
0      D      01N99
0              ENAME      20
0              ENUM      32
0              EDEPT      45
0              ETYPE      55
0      D      99
0              16 '***'
0              40 'Invalid Password Entered'
0              43 '***'
```

図 4-23. 実行時にパラメーターを要求する ILE RPG プログラム

CL CALL コマンドを使用したプログラムの実行

図 4-24 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 4-24. 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 4-25 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=取り消し
F13=情報援助 F16=システム・メイン・メニュー

図 4-25. アプリケーション・メニューの例

図 4-25 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」）。

4-55 ページの図 4-26 は、上図の 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 4-26. アプリケーション・メニューのデータ記述仕様書

図 4-27 は、4-54 ページの図 4-25 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 4-27. 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 4-27 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。コマンド定義の使用の詳細については、資料「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。これはサイクル・メイン・プロシージャ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。5-61 ページの『サイクル・メイン・プロシージャ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が取り消され、サブプロシージャ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 (リニア・メイン・プロシージャとして指定されるものも含む) 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。

サブプロシージャの呼び出し元が RPG プロシージャの場合には、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 サイクル・メイン・プロシージャがデフォルト・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追加できるすべての ILE RPG 照会メッセージを見つけるには、以下のコマンドを使用します。

```
# DSPMSGD RANGE(RNQ0100 RNQ9999)
# MSGF(QRNXMSG) DETAIL(*BASIC) OUTPUT(*PRINT)
```

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 *SYSRPLYL 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 (CHGJOBDD)
- ジョブ記述作成 (CRTJOBDD)
- 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/systems/i/infocenter/> で **IBM i 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、5-58 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。活動化グループの詳細については、「ILE 概念」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。新しいコピーはそれぞれ独自のデータを持ったりファイルをオープンしたりします。ただし、「それ自体」への呼び出しを終了するための何らかの方法を確保する必要があります。さもないと、新しい活動化グループが作成され続けるだけで、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せん。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

IBM i 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。

詳しくは、7-9 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、3-17 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。

活動化グループの管理

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/systems/i/infocenter/> で **IBM i 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の詳細については、「*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、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ている状態となり、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 コマンドの詳細については、Web サイト <http://www.ibm.com/systems/i/infocenter/> にある **IBM i Information Center** の『プログラミング』カテゴリーの中の『CL および API』セクションを参照してください。RCLRSC および活動化グループの詳細については、「*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。

```

* Two counters are kept:
* 1. The current number of array elements
* 2. The number of array elements that are allocated for the array
D arrInfo          DS          QUALIFIED
D   pArr           *          INZ(*NULL)
D   numElems       10I 0 INZ(0)
D   numAlloc       10I 0 INZ(0)
D arr              S          20A  VARYING DIM(32767)
D                               BASED(arrInfo.pArr)
D i                S          10I 0
/free
// Allocate storage for a few array elements
// (The number of elements that the array is considered to
// actually have remains zero.)
arrInfo.numAlloc = 2;
arrInfo.pArr = %alloc(arrInfo.numAlloc * %size(arr));

// Add two elements to the array
if arrInfo.numAlloc < arrInfo.numElems + 2;
// There is no room for the new elements.
// Allocate a few more elements.
arrInfo.numAlloc += 10;
arrInfo.pArr = %realloc (arrInfo.pArr
                        : arrInfo.numAlloc * %size(arr));
endif;
arrInfo.numElems += 1;
arr(arrInfo.numElems) = 'XYZ Electronics';
arrInfo.numElems += 1;
arr(arrInfo.numElems) = 'ABC Tools';

// Search the array
i = %lookup ('XYZ Electronics' : arr : 1 : arrInfo.numElems);
// i = 1

// Sort the array
sorta %subarr(arr : 1 : arrInfo.numElems);

```

図 4-28. 動的に割り振られた配列の割り振り、ソートおよび検索

```

// Search the array again
i = %lookup ('XYZ Electronics' : arr : 1 : arrInfo.numElems);
// Now, i = 2, since the array is now sorted

// Remove the last element from the array
arrInfo.numElems -= 1;

// Clear the array
// This can be done simply by setting the current number of
// elements to zero. It is not necessary to actually clear
// the data in the previously used elements.
arrInfo.numElems = 0;

// Free the storage for the array
dealloc arrInfo.pArr;
reset arrInfo;

return;

```

動的に割り振られた記憶域の管理

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。ユーザー作成ヒープおよびその他の ILE 記憶域管理の概念の詳細については、「*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。

4-63 ページの図 4-29 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?'        DPLY      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 4-29. メモリー管理 - 名前のリンク・リストの作成

```

*-----*
* 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@

```



```

*-----*
* 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

```

* 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		

ヒープ記憶域の問題

4-68 ページの図 4-30 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 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ができません。

  Fld1 = *ALL'a';
/END-FREE

```

図 4.30. 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、元のポインタに新しい値を設定するまでは、このポインタを使用できません。
- ヒープ記憶域へのポインタ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が *INZSR に設定されていると、その後でポインタの RESET を行うと、ポインタが、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が (消去されたり、例えば ALLOC 命令によって新しいポインタ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。ポインタが失われた後では、このポインタ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成については、「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)、CEEGETST (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/systems/i/infocenter/> で **IBM i 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 4-31 では、DYNARRAY 内のプロシージャ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 4-31. DYNARRAY モジュールのプロトタイプを含む /COPY ファイル DYNARRI

4-70 ページの図 4-32 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バック (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 4-32. DYNARRAY のグローバル変数とローカル・プロトタイプ

*-----			
* 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			
* プロシーチ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	

4-72 ページの図 4-33 は DYNARRAY にあるサブプロシーチャーを示したものです。

動的に割り振られた記憶域の管理

```
=====
* DYNALNIT: 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 DYNALNIT      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    INITIALLOC    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、CEEGETST 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      CEEGETST(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 DYNALNIT      E
```

図 4-33. DYNARRAY のサブプロシージャー

```
=====
* DYNALNIT: 配列処理を終了させます。
*
* 機能:      ヒープを削除します。
*=====
P DYNALNIT      B      EXPORT
C              CALLP      CEEDSHP(HeapId : *OMIT)
C              RESET      HeapVars
P DYNALNIT      E
```



```

=====
* 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
* 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      01dElems      5 0

```

動的に割り振られた記憶域の管理

```

*
* 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

```

```

=====
* 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
*-----
* このプロシーチ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

```

このサブプロシーチ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、プロシーチ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。プロシーチャーは、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。

```
CRTPRGMOD MODULE(MYLIB/DYNARRAY) SRCFILE(MYLIB/QRPGLESRC)
```

このプロシーチャーは、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 4-34 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 4-34. DYNARRAY のプロシーチャーを使用するサンプル・モジュール

プログラムおよびプロシーチャーの呼び出し

ILE では、プログラムまたはプロシーチ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チ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)。

動的に割り振られた記憶域の管理

プログラムまたはプロシーチ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チ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。

プログラムまたはプロシーチャーの RPG 呼び出し元が存在しない場合や、プロシーチャーがモジュールからエクスポートされない場合は、プロトタイプの指定は任意です。プロトタイプが指定されていないと、RPG コンパイラーはプロシーチャー・インターフェースからプロトタイプを生成します。プロシーチャーにプロシーチャー・インターフェースが含まれていない場合、RPG コンパイラーは、戻り値とパラメーターのないプロトタイプを生成します。明示プロトタイプを持たないようなプロシーチャーを呼び出すこともプロトタイプ呼び出しと見なされます。

次の場合には、プログラムまたはプロシーチ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チャーを呼び出す
- 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チャーから戻る
- 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 / プロシーチ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チャー・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 つ以上のプロシーチャーから成ります。ILE RPG モジュールには任意指定のメイン・プロシーチャーおよびゼロかそれ以上のサブプロシーチャーが含まれます。この章では、「プロシーチャー」という語はメイン・プロシーチャーおよびサブプロシーチャーの両方に適用されます。

ILE「プログラム呼び出し」はプロシーチ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チャーに対する呼び出しです。プログラム入力プロシーチャーは、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チャーです。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・プロシーチャーが、プログラムの呼び出し直後にプログラム入力プロシーチ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。

- プロシーチ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がプロシージャ呼び出し用に使用可能となります。

プロシージャの呼び出し

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、他のプロシージャを呼び出すために、静的プロシージャ呼び出しまたはプロシージャ・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。

静的プロシージャ呼び出しとは、バインド中にプロシージャ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・ポインター呼び出しは、プロシージャ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・ポインターをパラメーターとして別のプロシージャに渡すことができ、このプロシージャ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を実行することになります。また、プロシージャ名の配列または異なるプロシージャへプロシージャ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が同一の活動化グループにある場合には、プロシージャ・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呼び出しのコストと同じです。

プロシージャ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
- 別々の ILE サービス・プログラムのプロシージャ

静的プロシージャ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は、プロシージャ・ポインターによっても呼び出すことができます。

静的プロシージャ呼び出しの使用例のリストについては、4-84 ページの『自由形式呼び出しの例』および 4-100 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・ポインターの使用例については、「*IBM Rational Development Studio for i: ILE RPG 解説書*」でプロシージャ・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。プロシージャが値を戻す場合には、式でプロトタイプ・プロシージャを呼び出すこともできま

プログラム / プロシージャ呼び出しの概要

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呼び出しを実行し、関連したユーザーのプロシージャ (メイン・プロシージャ) が追加されます。プロシージャが呼び出される時に、ユーザーのプロシージャ (メイン・プロシージャまたはサブプロシージャ) 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のオーバーヘッドはありません。

図 4-35 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・プロシージャは RPG サブプロシージャを呼び出し、次に C プロシージャを呼び出します。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。

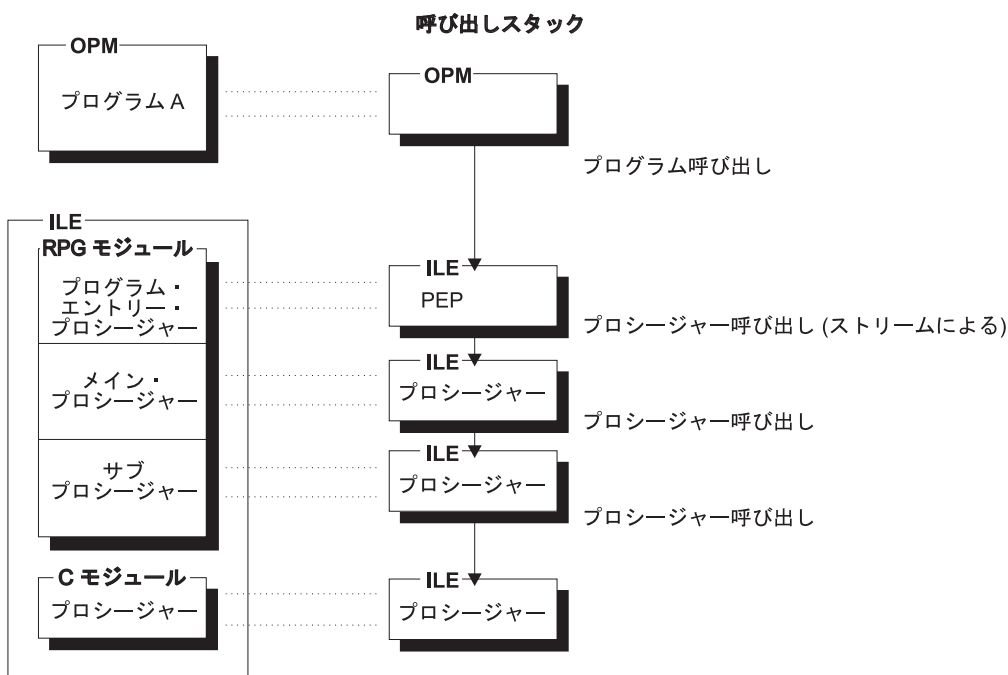


図 4-35. 呼び出しスタック上のプログラム呼び出しとプロシージャ呼び出し

注: ユーザー入りロプロシージャ (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とユーザー入りロプロシージャ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に対して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 A がそれ自身を呼び出すか、あるいはプロシージャ B を呼び出し、次にプロシージャ B が再びプロシージャ A を呼び出す呼び出しです。各再帰呼び出しによって、プロシージャの新しい呼び出しが呼

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、サブプロシーチ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。新規の呼び出しは、以前の呼び出しと同じ静的記憶域 (モジュールのグローバル静的記憶域とプロシーチャー内のローカル静的記憶域の両方) を使用します。

また、再帰呼び出しは、メイン・プロシーチャーがリニア・メイン・プロシーチャーであるプログラムにも使用することができます。リニア・メイン・プロシーチャーはプログラム呼び出しを介してのみ呼び出すことができるため、リニア・メイン・プロシーチャーが再帰的にそれ自体を呼び出す場合、リニア・メイン・プロシーチャーが含まれるプログラムが再び呼び出されます。それ以外の場合、自身を再帰的に呼び出すリニア・メイン・プロシーチャーの動作は、それ自体を再帰的に呼び出す通常の子プロシーチャーの場合と同じです。

呼び出しスタックにあるサイクル・メイン・プロシーチャーは、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、既に活動中のサイクル・メイン・プロシーチャーを呼び出す可能性のあるプロシーチ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、図 4-36 に示すように、3 つのモジュールがあるとして。

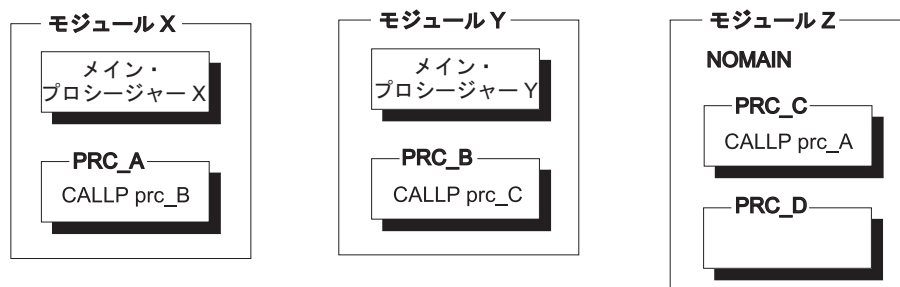
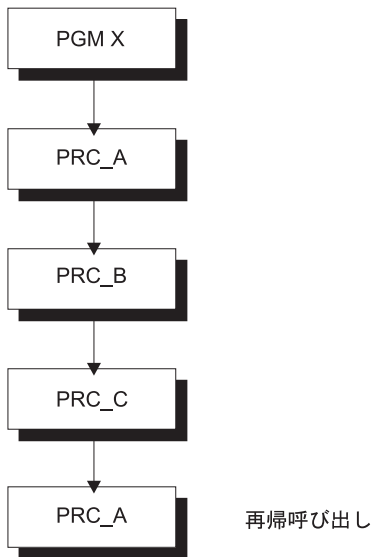


図 4-36. それぞれがサブプロシーチャーをもつ 3 つのモジュール

モジュール X のプロシーチャー A はモジュール Y のプロシーチャー B を呼び出すプログラムを実行しているとします。プロシーチャー B があるフィールドを処理すること以外、B が何をするか分かりません。一方、プロシーチャー B はプロシーチャー C を呼び出し、C はプロシーチャー A を呼び出します。プロシーチャー C がプロシーチャー 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 4-80 ページの図 4-37 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。

プログラム / プロシージャ呼び出しの概要



呼び出しスタック (一番下の項目が一番新しい)

図 4-37. 避けるべき再帰呼び出しスタック

サブプロシージャ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
- 正しいファイルがファイル・パラメーターに渡されていること
- 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)、安全に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。

RPG ファイル・パラメーターは、その他の HLL のファイル・パラメーターに関連していません。RPG ファイル・パラメーターは他の RPG プログラムまたはプロシージャにのみ渡すことができます。

表 4-16 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 4-16. 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		4-84 ページの『プロトタイプ・パラメーターの受け渡し』
参照による受け渡し	はい	はい	4-85 ページの『参照による受け渡し』
値による受け渡し	はい (b)		4-85 ページの『値による受け渡し』
読み取り専用参照による受け渡し	はい		4-85 ページの『読み取り専用参照による受け渡し』

プログラム / プロシージャ呼び出しの概要

表 4-16. パラメーター受け渡しオプション (続き)

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
操作記述子の受け渡し	はい (b)	はい (b)	4-87 ページの『操作記述子の使用』
*OMIT の受け渡し	はい	はい (b)	4-88 ページの『パラメーターの省略』
パラメーター省略の制御	はい	はい	4-89 ページの『パラメーターの省略』
渡されたパラメーターの数の取得	はい	はい	4-90 ページの『渡されるパラメーターの数の検査』
正しくないパラメーター長の使用不能	はい		4-95 ページの『必要なデータより少ないデータの受け渡し』
ファイル・パラメーターを渡す	はい		ファイル・パラメーターの受け渡し
注: (b) – バインドされたプロシージャ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。このプロトタイプは、明示的に指定されるか、あるいは、プロシージャが呼び出しと同じモジュールに指定されている場合にコンパイラーによってプロシージャ・インターフェースから暗黙的に生成されます。プロトタイプ呼び出し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
- 同じモジュール内のサブプロシージャ

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、組み込み関数の引数に非常によく似た、呼び出しの引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・プロシージャを入れてください。値を戻すプロシージャに対し 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、4-84 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・プロシーチャーの呼び出し) 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チ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チ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. 呼び出されるプログラムまたはプロシーチ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。このステップは、プロシーチャーが呼び出しと同じモジュールに存在し、かつプロシーチャーを呼び出すモジュールがほかにはない場合はオプションです。
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チ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、プロシーチャー Switch の呼び出しを示します。このプロシーチャーは、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、プロシーチャー呼び出しでは、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・プロシーチャー呼び出しです。

CALLP を使用して値を戻すプロシーチ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・プロシーチ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・プロシーチャーを定義し、この戻り値を使用したい場合には式の中でプロシーチ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、プロシーチャー名を使用してください。例えば、プロシーチ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チャーの呼び出しは、数値の必要な個所でなければなりません。

4-84 ページの図 4-38 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チャー CVTCHR のプロトタイプです。4-84 ページの図 4-39 は、式の中でプロシーチャーがどのように使用されるかを示しています。

プロトタイプ呼び出しの使用

```
* 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 4-38. 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 4-39. 式の中のプロトタイプ・プロシーチャーの呼び出し

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。

- 3-36 ページの図 3-22
- 4-47 ページの図 4-19
- 5-54 ページの図 5-31
- 4-94 ページの図 4-47
- 5-84 ページの図 5-44

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。

- 3-9 ページの図 3-4
- 3-33 ページの図 3-19
- 4-31 ページの図 4-14
- 5-54 ページの図 5-31

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同じプロトタイプを使用してコンパイルされる場合、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、プロシーチ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: IBM i 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。4-87 ページの図 4-42 は、ILE CEE API CEETSTA プロトタイプ定義 (省略引数のテスト) 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。

プロトタイプ・パラメーターの受け渡し

- 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、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
- 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・プログラムまたはプロシージャ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。


```

*-----
* プロシーチャーは 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 4-40. VALUE パラメーターのプロシーチャー DO_CALC のプロトタイプ

```

P DO_CALC          B          EXPORT
*-----
* このプロシーチャーは、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 4-41. DO_CALC プロシーチャーのプロシーチャー・インターフェース定義

```

*-----
* CEETSTA (省略引数のテスト) -- ILE CEE API
*   1. 存在フラグ          出力 2 進 (4)
*   2. 引数番号            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 4-42. 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チ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チ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チャーに記述情報を提供するために、**操作記述子**を

プロトタイプ・パラメーターの受け渡し

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タイプの操作記述子しかサポートしません。操作記述子は配列または表、数字タイプのデータ、日付、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。さらに、操作記述子は、CALLB を使用して作成された非プロトタイプの呼び出し用のデータ構造では使用できません。ただし、プロトタイプの呼び出しに対しては、データ構造は文字データと見なされ、操作記述子は使用可能です。

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 4-43 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 4-43. 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、4-44 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

プログラムまたは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、4-87 ページの図 4-42 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、4-44 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はこれを予定している必要があります、このことは、そのプロシージャ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、4-90 ページの図 4-44 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 4-44. 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。 プロシーチ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チ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チャーを書いて、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 が戻す値には *OMIT が渡されたパラメーターも含まれ、さらに、RTNPARM キーワードが指定されているプロシーチャーの戻り値を最初に処理する追加のパラメーターも含まれます。サイクル・メイン・プロシーチャーの場合、%PARMS は PSDS の *PARMS フィールドに含まれている値と同じものを戻します。ただし、*PARMS フィールドを使用するためには、PSDS のコーディングも行う必要があります。

特定のパラメーターがプロシーチャーに渡されたかどうかを調べたい場合は、%PARMNUM 組み込み関数を使用して、そのパラメーターの番号を確認することができます。 %PARMNUM が戻す値は、そのプロシーチャーに RTNPARM キーワードが指定されていれば、正確なパラメーター番号を反映します。

*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・プロシーチ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。 サブプロシーチャーから *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を使用するため、プロシーチャー FMTADDR が数回変更されています。FMTADDR は 3 つの異なるプロシーチャーによって呼び出されます。 プロシーチャーは、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チ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もので、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。4-92 ページの図 4-45 は、このプロシージャーのソースを示したものです。4-93 ページの図 4-46 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。

プロトタイプ・パラメーターの受け渡し

```

*=====
* 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          バック (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 4-45. プロシージャ FMTADDR のソース


```

=====
* サブルーチン: GetStreet#
* 番地の文字形式を入手し、左寄せして、右に空白を
* 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

```

```

=====
* 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 4-46. プロシージャ FMTADDR のプロトタイプでの /COPY メンバーのソース

4-94 ページの図 4-47 にはプロシージャ PRTADDR のソースを示してあります。このプロシージャは、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 つのプロシージャがこの 1 つのプロシージャ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 PRTADDR は終了します。

プロトタイプ・パラメーターの受け渡し

```

=====
* 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  0    F    80          PRINTER
-----
* FmtAddr のプロトタイプ
-----
D FmtAddr          PR
D  addr            70
D  strno           5  0
D  st              20
D  cty             15  OPTIONS(*NOPASS)
D  prov            15  OPTIONS(*NOPASS)
D Address          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 4-47. プロシージャ PRTADDR のソース

```

*-----*
* '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。 *
*-----*
OQSYSVRT   E
0          Address

**
00123Bumble Bee Drive
01243Hummingbird Lane
**
00003Cowslip Street      Toronto
01150Eglinton Avenue    North York
**
00012Jasper Avenue      Edmonton      Alberta
00027Avenue Road        Sudbury        Ontario

```

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. 4-92 ページの図 4-45 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、4-94 ページの図 4-47 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。呼び出された側が、最大長よりも短いパラメーターを指定できることを (文書またはそのプロトタイプによって) 指示した場合、短いパラメーターを安全に渡すことができます。

4-96 ページの図 4-48 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

プロトタイプ・パラメーターの受け渡し

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 4-48. *VARSIZE パラメーターのシステム API QCMDEXC のプロトタイプ

ファイル・パラメーターの受け渡し

LIKEFILE キーワードを使用して、プロトタイプ化したパラメーターがファイルであることを示すことができます。プロトタイプを使用して呼び出しを行う場合、渡すファイルは、プロトタイプの LIKEFILE パラメーターで指定されたファイルであるか、または LIKEFILE ファイル仕様書キーワードを介してそのファイルに関係しているファイルでなければなりません。例えば、プロトタイプで LIKEFILE(MYFILE) を指定し、LIKEFILE(MYFILE) を使用してファイル OTHERFILE を定義する他のファイル仕様書を持っている場合、その呼び出しで MYFILE または OTHERFILE を受け渡すことができます。

ファイル・パラメーター およびファイルに関連する変数 の詳細については、「WebSphere Development Studio ILE RPG 解説書」の一般的なファイルの考慮事項の章を参照してください。

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。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。『RPG パラメーター受け渡し方式』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。

RPG パラメーター受け渡し方式

表 4-17. 参照による受け渡し

呼び出しメカニズム	例
ILE RPG – プロトタイプ	<pre> D proc PR D parm 1A C CALLP proc(fld) </pre>
ILE C	<pre> void proc(char *parm); proc(&fld); </pre>
ILE COBOL	CALL PROCEDURE "PROC" USING BY REFERENCE PARM
RPG – 非プロトタイプ	<pre> C CALL 'PROC' C PARM FLD </pre>
ILE CL	CALL PROC (&FLD)

表 4-18. 値による受け渡し

呼び出しメカニズム	例
ILE RPG – プロトタイプ	<pre> D proc PR D parm 1A VALUE C CALLP proc('a') </pre>
ILE C	<pre> void proc(char parm); proc('a'); </pre>
ILE COBOL	CALL PROCEDURE "PROC" USING BY VALUE PARM
RPG – 非プロトタイプ	N/A
ILE CL	N/A

表 4-19. 読み取り専用参照による受け渡し

呼び出しメカニズム	例
ILE RPG – プロトタイプ	<pre> D proc PR D parm 1A CONST C CALLP proc(fld) </pre>
ILE C	<pre> void proc(const char *parm); proc(&fld); </pre>
ILE COBOL	N/A ¹
RPG – 非プロトタイプ	N/A
ILE CL	N/A

注:

プロトタイプ・パラメーターの受け渡し

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 プロシージャが ILE CL プロシージャと正しくやりとりを行なうことを確認するには、ILE CL プロシージャのプロトタイプにおいて、または ILE CL プロシージャによって呼び出される RPG プロシージャにおいて、EXTPROC(*CL:'procedurename') とコーディングします。
2. RPG プロシージャが ILE C プロシージャと正しくやりとりを行なうことを確認するには、ILE C プロシージャのプロトタイプにおいて、または ILE C プロシージャによって呼び出される RPG プロシージャにおいて、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. RPG プロシージャは、8 バイト・ポインターを使用している ILE C/C++ プロシージャと対話することができます。ただし、ILE C/C++ プロシージャは 16 バイト・ポインターをパラメーターとして使用する必要があります。「IBM Rational Development Studio for i: ILE C/C++ コンパイラー参照」を参照してください。
5. LIKEFILE キーワードでプロトタイプ化された RPG ファイル・パラメーターは、他の言語からのファイル・パラメーターと交換可能ではありません。例えば、RPG ファイルを、FILE または RFILE パラメーターを予測している C 関数に渡すことはできません。同様に、RPG パラメーターが LIKEFILE キーワードでプロトタイプ化されている場合、C 関数は FILE または RFILE パラメーターを RPG プロシージャに渡すことはできません。

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に対するプロシージャ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チャー内の、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: プロシーチャー・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チャーのアドレスを含むプロシーチャー・ポインターの名前を指定します。

- 1 つのプロシーチャーに、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 プロシーチャー (プログラム入力プロシーチャーを含む) が最初に呼び出されると、そのフィールドが初期化され、そのプロシーチ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、同じプロシーチャーの後続の呼び出しでは、呼び出されるプロシーチ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 プロシーチ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 プロシーチャーがプログラム (*PGM) の中にバインドされる場合には、どのプロシーチ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チ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チャーはシステム・プロシーチャーです。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チ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。

固定形式の呼び出し命令の使用

操作記述子の詳細については、4-87 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細な説明については、「*IBM Rational Development Studio for i: 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、5-52 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チャーの呼び出し例については、4-49 ページの図 4-21
- バインド可能 API の呼び出し例については、4-72 ページの図 4-33
- 各種の RPG プロシーチャーを呼び出すメイン照会プログラムの例は、6-81 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 命令はプロシーチ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チ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/プロシーチャーか、あるいは呼び出されるプログラム/プロシーチ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 4-20 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 4-20. 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 プロシーチャー	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 プロシーチャー	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チ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。

プロシーチ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チ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、4-90 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、4-88 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細な説明については、「*IBM Rational Development Studio for i: 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。

- 4-53 ページの図 4-23
- 4-45 ページの図 4-18
- 5-81 ページの図 5-43

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の入力パラメーター・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・プロシージャにだけ指定することができます。

PLIST 命令の例については、4-53 ページの図 4-23 と 5-81 ページの図 5-43 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からの戻り

プログラムまたはプロシージャが戻る場合には、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も除去されます)。プロシージャ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、プロシージャが異常終了します。例えば、これは ILE RPG プロシージャ X が、X を呼び出すプロシージャに対して直接、エスケープ・メッセージを出す (CL プロシージャのような) 別のプロシージャを呼び出す場合に起こります。これはま

呼び出されたプログラムまたはプロシージャからの戻り

た、プロシージャ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・プロシージャとサブプロシージャ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・プロシージャからの戻り

メイン・プロシージャから戻ると、次のことが起こります。

- これがサイクル・メイン・プロシージャで、LR がオンの場合には、グローバル・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。
- プロシージャがプログラム入力プロシージャ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も呼び出しスタックから除去される。

サイクル・メイン・プロシージャは、次のいずれかの方法で呼び出しプロシージャに制御権を戻します。

- 正常終了
- 異常終了
- 未終了

呼び出されたサイクル・メイン・プロシージャから戻する方法について次に説明します。

LR、H1 から 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の詳細については、「*IBM Rational Development Studio for i: 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。

サイクル・メイン・プロシージャの正常終了: LR 標識がオンで、H1 から H9 標識がオンでない時には、サイクル・メイン・プロシージャが正常に終了し、制御権が呼び出し元プロシージャに戻されます。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、サイクル・メイン・プロシージャ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。

サイクル・メイン・プロシージャ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グローバル・ファイルがクローズされる。
- このプロシージャ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。

サイクル・メイン・プロシーチャーへの次の呼び出しでは、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 EXPORT キーワードを使用して定義さ
れます。これらの変数は、プログラムが活動化グループで最初に活動化されるときに 1 回だけ初期設定さ
れます。たとえば 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。

サイクル・メイン・プロシーチャーの異常終了: 次のいずれかが起こると、サイクル・メイン・プロシーチャーが異常終了し、制御権が呼び出し元プロシーチャーに戻されます。

- 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、5-79 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。

サイクル・メイン・プロシーチ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グローバル・ファイルがクローズされる。
- このプロシーチ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サイクル・メイン・プロシーチ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。

このプロシーチ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、5-67 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)。

サイクル・メイン・プロシーチャーの未終了の戻り: LR 標識または H1 から 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、サイクル・メイン・プロシーチャーが未終了で、制御権を呼び出し元プロシーチャーに戻することができます。

- 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チャーに戻されます (RT 標識の詳細については、「IBM Rational Development Studio for i: ILE RPG 解説書」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からの戻り

サイクル・メイン・プロシージャを呼び出し、それが未終了で戻った場合には、プロシージャを再び呼び出した時に、このプロシージャのすべてのフィールド、標識、およびファイルはプロシージャが途中終了した時と同じ値を持っています。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、プロシージャが途中終了し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、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ちらかがオンで、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

サブプロシージャからの戻り

このセクションは通常のサブプロシージャおよびリニア・メイン・プロシージャに適用されます。

RETURN 命令が正常に実行された時、あるいはプロシージャ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、**サブプロシージャは正常に戻ります**。サブプロシージャが自動記憶域にローカル・ファイルを持っている場合、それらはサブプロシージャの終了時にクローズされます。それ以外の場合、呼び出しスタックからのサブプロシージャの除去以外には、プログラムのサイクル・メイン・プロシージャ (存在する場合) 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、サイクル・メイン・プロシージャ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、**サブプロシージャが異常終了**し、制御権は呼び出し元プロシージャに戻されます。自動記憶域内のローカル・ファイルがすべてクローズされます。それ以外は、サイクル・メイン・プロシージャが終了するまでこれ以上の処理は起こりません。

モジュールがサイクル・モジュールで、メイン・プロシージャ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、サブプロシージャの終了時に呼び出す終了プロシージャ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。プロシージャが CEETREC を使用して終了した場合には、メイン・プロシージャおよびサブプロシージャ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、プロシージャ内でプロシージャ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。

サイクル・メイン・プロシージャが活動状態でない場合、あるいはサイクル・メイン・プロシージャがない場合には、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/systems/i/infocenter/> で **IBM i Information Center** 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
- ストレージ管理
- 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、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/systems/i/infocenter/> で **IBM i Information Center** 中の『プログラミング』カテゴリーの『CL および API』節の API の説明を参照してください。図 4-49 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 4-49. ILE バインド可能 API のサンプル呼び出し構文

バインド可能 API の使用

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/systems/i/infocenter/> で **IBM i 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。

- 4-44 ページの『サンプル・サービス・プログラム』(CEEDOD の使用例)
- 4-68 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)
- 5-80 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)
- 5-86 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 IBM i 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は、IBM i 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。

IBM i 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の詳細については、資料「*GDDM Programming Guide*」および資料「*GDDM Reference*」を参照してください。

注: CALL 命令を使用して、IBM i 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。

記憶域モデル

統合化言語環境 (ILE) は 2 つの記憶域モデル (単一レベル記憶域モデルとテラスペース記憶域モデル) を提供します。これらの記憶域モデルの 1 つを使用するためにモジュール、プログラム、およびサービス・プログラムを作成することもできれば、呼び出し元の記憶域モデルを継承するためにモジュール、プログラム、およびサービス・プログラムを作成することもできます。

活動化グループに呼び込まれたプログラムおよびサービス・プログラムは、すべて同じ記憶域モデルを使用する必要があります。活動化グループに最初に呼び込まれたプログラムまたはサービス・プログラムが単一レベル記憶域モデルを使用している場合は、同じ活動化グループに属する他のすべてのプログラムおよびサービス・プログラムは単一レベル記憶域モデルまたは継承記憶域モデルのいずれかを使用する必要があります。同様に、活動化グループに最初に呼び込まれたプログラムまたはサービス・プログラムがテラスペース記憶域モデルを使用している場合は、同じ活動化グループに属する他のすべてのプログラムおよびサービス・プログラムはテラスペース記憶域モデルまたは継承記憶域モデルのいずれかを使用する必要があります。

プログラムやサービス・プログラムは、単一レベル記憶域モデルと継承記憶域モデルを使用しているモジュールから作成することもできれば、テラスペース記憶域モデルと継承記憶域モデルを使用しているモジュールから作成することもできます。単一レベル記憶域モデルとテラスペース記憶域モデルの両方を使用しているモジュールからプログラムやサービス・プログラムを作成することはできません。

単一レベル記憶域モデルに関する考慮事項

- 1 つのプロシージャについては 16MB の自動記憶域という制限があります。
- 呼び出しスタック上の全プロシージャについては合計 16MB の自動記憶域という制限があります。

テラスペース記憶域モデルに関する考慮事項

- 自動記憶域に対する実質的な制限はありません。
- テラスペース記憶域モデルを使用すると、高速の 8 バイト・ポインターを使用している C および C++ で書かれたサービス・プログラムにアクセスすることができます。ただし、サービス・プログラム内の C または C++ 関数は 16 バイト・ポインターをパラメーターとして使用する必要があります。「IBM Rational Development Studio for i: ILE C/C++ コンパイラ参照」を参照してください。

継承記憶域モデルに関する考慮事項

- 活動化グループは *CALLER でなければなりません。
- 継承記憶域モデルを持つプログラムやサービス・プログラムは、任意の記憶域モデルを使用しているプログラムやサービス・プログラムから呼び出すことができます。

継承記憶域モデルに関する考慮事項

- 実際の記憶域モデルは呼び出し元の記憶域モデルによって実行時に決定されます。
- コンパイル時には 1 つのプロシージャーについて 16MB の自動記憶域という制限があります。
- 自動記憶域の実行時制限は、実行時に実際の記憶域モデルによって決まります。

プログラムとサービス・プログラムの記憶域モデルに関する推奨事項

- プログラムやサービス・プログラムがテラスペース記憶域モデルの利点を必ずしも必要としない場合は、ACTGRP(*CALLER) プログラムおよびサービス・プログラムに対して STGMDL(*INHERIT) を使用することを検討してください。
- プログラムとサービス・プログラムが名前付き活動化グループとともに作成される場合は、命名規則を使用してテラスペース活動化グループを識別することを検討してください。例えば、テラスペース活動化グループ名を TS で終わるようにすることができます。これは、CRTBNDRPG または CRTPGM に ACTGRP(*STGMDL) を指定したときに活動化グループ名の選択方法に従っています。その場合、テラスペース記憶域モデルには QILETS が使用され、単一レベル記憶域モデルには QILE が選択されます。
- 同じ活動化グループ属性を持つテラスペース記憶域モデル・プログラムと単一レベル記憶域モデル・プログラムを作成しないようにしてください。例えば、TERAPGM がテラスペース記憶域モデル・プログラムであり、SLSPGM が単一レベル記憶域モデル・プログラムであり、TERAPGM と SLSPGM が両方とも ACTGRP(MYACTGRP) を使用するようにコンパイルされるとします。TERAPGM が先に呼び出された場合は、活動化グループ MYACTGRP がテラスペース記憶域モデルの活動化グループとして作成され、記憶域モデルの不一致のために SLSPGM の呼び出しは失敗します。同様に、SLSPGM が先に呼び出された場合は、活動化グループ MYACTGRP が単一レベル記憶域モデルの活動化グループとして作成され、記憶域モデルの不一致のために TERAPGM を呼び出す試みはすべて失敗します。

詳細については、「*ILE* 概念」を参照してください。

マルチスレッド化に関する考慮事項

マルチスレッド環境で実行する場合、考慮すべきスレッド・セーフティー問題が多数あります。マルチスレッド化されたアプリケーションの作成を開始する前に、マルチスレッド化の概念を十分に理解する必要があります。 <http://www.ibm.com/systems/infocenter/> でマルチスレッド・アプリケーションに関する情報を参照してください。また、マルチスレッド・アプリケーションのコーディング時に発生する一般的なプログラミング・エラーを理解しておく必要があります。 <http://www.ibm.com/systems/infocenter/> で『一般的なマルチスレッド・プログラミング・エラー』を参照してください。データベース・ファイルを使用する際、およびマルチスレッド環境でコミットメント制御を使用する際に発生する可能性がある問題に特に注意してください。

スレッド・セーフティーの 1 つの重要な問題は、静的記憶域の処理です。ILE RPG モジュール内で静的記憶域のスレッド・セーフ処理を確実に実行するには、2 つの方法があります。THREAD(*CONCURRENT) を指定することによって各スレッドに対して別々の静的記憶域を設定することができます。あるいは、THREAD(*SERIALIZE) を指定することによってモジュールへのアクセスを一度に 1 つのスレッドに制限することができます。スレッド・セーフティーのこれらの 2 つの方式の比較については、マルチスレッド・アプリケーションを参照してください。

以降の説明は、これらの概念を理解していることが前提です。

複数のスレッドでの同時実行

THREAD(*CONCURRENT) を指定する場合、2 つ以上のスレッドが同じモジュール、同じプロシージャー、または同じステートメントで異なるプロシージャーを実行できます。2 つ以上のスレッドが同じモジュールまたは同じプロシージャーで実行されている場合、それぞれデフォルトで、モジュールとそのプロシージャーに静的記憶域の独自のインスタンスを持ちます。この記憶域は、スレッド・ローカル記憶域と呼ば

れます。例えば、2 つのスレッドが以下のループで実行されている場合、1 つのスレッドは、値 7 を持つ変数 "count" を使用して、ファイルの 20 番目のレコードを読み取った後、"IF" ステートメントを処理することができます。もう 1 つのスレッドは、値 1 を持つ変数 "count" を使用して、ファイルの 4 番目のレコードを読み取った後、"READ" ステートメントを処理することができます。

```
read rec ds;
count = 0;
dow not %eof(file);
  if (ds.amtOwing > ds.max.Owing);
    handleAccount (ds);
    count += 1;
  endif;
  read rec ds;
enddo;
```

STATIC(*ALLTHREAD) を使用して変数を定義することを選択した場合、すべてのスレッドがその変数の同じインスタンスを使用します。

注意:

RPG は、2 つのスレッドが同時に **all-thread** 静的変数を変更しようとする動作、または別のスレッドがその値を検査しているときにあるスレッドが変数を変更しようとする動作に対して保護を提供しません。詳しくは、**All-Thread** 静的変数を参照してください。

あるコードが一度に 1 つのスレッドによってのみ使用されるようにするには、そのコードを逐次化プロシージャに置くことができます (プロシージャ開始仕様書の **SERIALIZE** キーワード)。逐次化された各プロシージャには独自の逐次化メカニズムがあることに注意してください。あるスレッドを逐次化された 1 つのプロシージャで実行し、別のスレッドを、同じモジュール内の逐次化された別のプロシージャで実行することができます。

コードが一度に 1 つのスレッドによってのみ使用されるようにする他の方法は、プロシージャ内のそのコードをスレッド逐次化モジュールに入れることです。

複数のスレッドでの逐次化の実行

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。スレッド逐次化モジュールは、スレッドがモジュールでプロシージャの実行を開始するとロックされ、モジュール内でスレッドが実行されなくなったときにはじめてアンロックされます。モジュールがロックされると、他のスレッドはモジュールでプロシージャを実行できません。他のスレッドがモジュールでプロシージャを呼び出そうとする場合、プロシージャを実行するには、モジュールがアンロックされるのを待たなければなりません。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スレッド逐次化モジュール内で、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。

THREAD キーワードに関する活動化グループの考慮事項

- THREAD キーワードが指定されているときは、デフォルトの活動化グループを実行しないようにしてください。
 - THREAD(*CONCURRENT) の場合、DFTACTGRP(*YES) でコンパイルされたプログラムは、別のスレッドで現在実行中であると、LR オンで戻ることも異常で戻ることもできません。プログラムの非活動化をこのプログラム自体が試みた場合、非活動化は MCH4405 で失敗します。
 - THREAD(*SERIALIZE) の場合、またはプロシージャ仕様書に SERIALIZE キーワードが指定されている THREAD(*CONCURRENT) の場合は、プログラムがデフォルトの活動化グループで実行されると、RPG ランタイムは、モジュールまたはプロシージャへのアクセスを直列化するために使用された mutex を破棄できません。場合によっては、RPG プログラムの終了後に、モジュールの静的記

マルチスレッド化に関する考慮事項

憶域に入っている、mutex を指すポインターが割り振り解除または上書きされることがあります。その場合は、mutex を指すポインターが失われたとき、mutex に関連付けられているシステムの記憶域が解放されないため、記憶域不足になる可能性があります。RPG プログラムが再び呼び出されると、RPG プログラムは新しい mutex を作成します。

このような状況は以下の場合に起こることがあります。

- プログラムが THREAD(*SERIALIZE) および DFTACTGRP(*YES) でコンパイルされ、プログラムが LR オンで終了するか、または異常終了して、プログラムが非活動化される。プログラムが非活動化されると、プログラムの静的記憶域が割り振り解除されます。
- RCLRSC コマンドが使用され、mutex を使用してモジュールまたはプロシージャーを直列化するプログラムがデフォルトの活動化グループで実行された。プログラムが DFTACTGRP(*YES) でコンパイルされるか、または呼び出し元がデフォルトの活動化グループに存在する ACTGRP(*CALLER) でコンパイルされると、プログラムはデフォルトの活動化グループで実行されます。
- セカンダリー・スレッドで実行されるプログラムの場合は、ACTGRP(*NEW) でコンパイルしないようにしてください。ACTGRP(*NEW) でコンパイルされたプログラムがセカンダリー・スレッド内で終了すると、ジョブはメッセージ CEE0200 が終了します。

複数のスレッド間で共有される記憶域

以下のいずれかが真の場合、2 つ以上のスレッドが同じ記憶域にアクセスできます。

- 変数が STATIC(*ALLTHREAD) キーワードで定義されている場合
- EXPORT/IMPORT キーワードがスレッド逐次化モジュールの定義仕様書で使用されている場合
- 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
- ファイルが SHARE(*YES) で作成またはオーバーライドされている場合。この場合、共用記憶域を表すフィードバック領域です。RPG はファイル操作中に常にフィードバック領域を参照するため、そのファイル自身へのアクセスを同期化する必要があります。

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、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。この共用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、スレッド関連の API の使用を参照してください。

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、スレッド逐次化モジュールまたは逐次化されたプロシージャー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、4-111 ページの図 4-50 で示される状況を考慮します。この状況では、2 つのスレッドがそれぞれ異なるスレッド逐次化モジュールでプロシージャーを実行しています (プロシージャー PROC1 はモジュール MOD1、プロシージャー PROC3 はモジュール MOD2)。また、MOD1 はプロシージャー PROC2 を持ち、MOD2 はプロシージャー PROC4 も持っています。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。

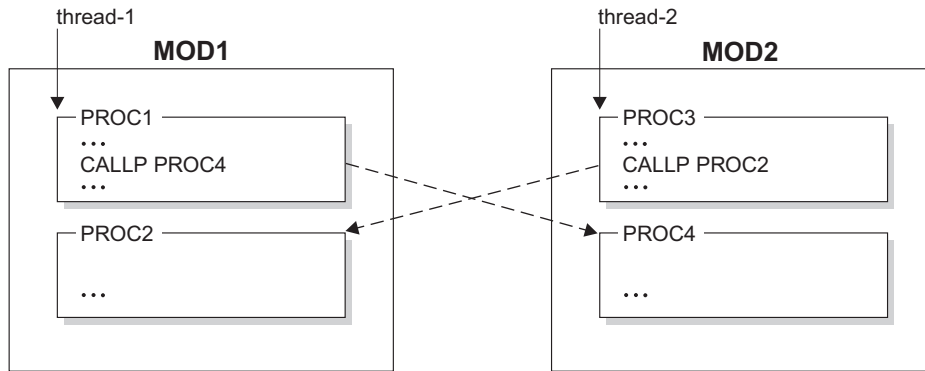


図 4-50. THREAD(*SERIALIZE) モジュールのデッドロック例

この例は、同じスレッド逐次化モジュール内の複数のプロシージャに同時にアクセスしようとする場合に、どのようにデッドロックが発生するかを示します。

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作成するためには、『スレッド関連の API の使用』で説明する技法を使用して、モジュールの同期化を制御できます。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、必要なプロシージャを呼び出す (PROC1 または 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に同時にアクセスすることはできませんが、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。

All-Thread 静的変数

STATIC(*ALLTHREAD) キーワードを使用して変数を定義する場合、変数がスレッド・セーフな方法で使用されるようにする責任があります。変数の有効範囲および変数の使用法によって、変数へのアクセスを同期化するのに役立つ追加変数の設定が必要になる場合があります。

- 変数が逐次化されたプロシージャに対してローカルである場合、逐次化のために一度に変数にアクセスできるスレッドは 1 つだけです。そのため、余分な同期を追加する必要はありません。
- 変数がモジュールに対してグローバルであり、コードの 1 か所でのみ変更されることを保証でき、さらに他のスレッドが変数を使用するには、その前にその変数を変更するコードが実行されることを保証できる場合、その変数に対する同期を追加する必要はありません。
- それ以外の場合、mutex またはセマフォなどの同期技法で使用するよう追加の変数を追加する必要があります。<http://www.ibm.com/systems/infocenter/> および『スレッド関連の API の使用』で、スレッドに関する情報を参照してください。

マルチスレッド化に関する考慮事項

他の変数へのアクセスを同期化するために同期変数を追加する必要がある場合、以下のことを確認する必要があります。

- 同期変数は、変数へのアクセスが行われる前に初期化する必要があります。
- 変数を処理するときは、セマフォまたは `mutex` をロックすることによって、まず変数へのアクセスを取得する必要があります。変数の処理が終了したら、セマフォまたは `mutex` をアンロックする必要があります。
- 変数がモジュールからエクスポートされる場合、変数をインポートするすべてのモジュールも同期変数を使用できることを確認する必要があります。これを行うには、同期変数をエクスポートするか、変数を使用する必要があるモジュールによって呼び出すことが可能なエクスポート・モジュールにエクスポート済みのロックおよびアンロック・プロシーチャーを追加します。

ヒント: 同期が必要な変数、およびその同期変数またはロックおよびアンロック・プロシーチャーに対して命名規則を設定します。例えば、同期が必要な変数の先頭に `SN_` を付け、その同期変数またはプロシーチャーに対して同じ名前を使用するが、異なる接頭部を使用するという規則を使用できます。例えば、変数 `SN_nextIndex` は、ロックおよびアンロック・プロシーチャー `LOCK_nextIndex` および `UNLOCK_nextIndex` を持つことができます。そのような規則を使用し、さらにその使用を厳しく強制することによって、プログラマーが正しい同期プロトコルに従わずに、同期が必要な変数を使用する可能性を減らすことができます。

- デッドロック状態は回避する必要があります。例えば、1 つのスレッドが `FLD1` に対してロックを持ち、`FLD2` のロックを取得しようとし、別のスレッドが `FLD2` のロックを持ち、`FLD1` 上でロックを取得しようとする場合、どちらのスレッドも永遠に待機することになります。

逐次化されたプロシーチャーを使用する場合

逐次化されたプロシーチャーを使用して、共用リソースへのアクセスを同期化することができます。共用リソースがジョブ内の他の場所で使用される場合、追加の手動同期の追加が必要になる場合があります。

以下の例では、グローバルな `all-thread` 静的変数がファイルからロードされ、アプリケーションでのその他の使用はすべて変数の値を参照するだけです。同時に複数のスレッドによって変更される可能性がある場合、または 1 つのスレッドが値を使用している間に他のスレッドがその値を変更する可能性がある場合、`all-thread` 静的変数へのアクセスを制御する必要があることを覚えておいてください。ただし、

「`first-time-only setup`」コードで一度だけ変更される変数の特殊なケースでは、逐次化されたプロシーチャーは、複数のスレッドによるアクセスを制御するために十分です。すべてのスレッドが `first-time-only setup` プロシーチャーを呼び出し、プロシーチャー自身がローカルの `all-thread` 静的変数を使用してセットアップが行われたかどうかを追跡します。一度に 1 つのスレッドのみがプロシーチャーを実行するようにプロシーチャーが逐次化されるため、逐次化されたプロシーチャーでローカルの `all-thread` 静的変数へのアクセスを制御するための手動同期は必要ありません。

`getCustList` プロシーチャーは `first-time-only setup` プロシーチャーの例です。これが制御する共用リソースは、2 つのグローバルな `all-thread` 静的変数の `ATS_custList` および `ATS_numCusts` です。このプロシーチャーは `SERIALIZE` キーワードで定義されます。これは、カスタマーのリストが含まれるファイルを読み取り、そのリストを配列に保存します。これはローカルの `all-thread` 静的変数 `ATS_done` を使用して、このリストが既に取得されたかどうかを追跡し、既に取得されている場合には、即時に戻ります。リストが取得される前に、複数のスレッドが同時にプロシーチャーを呼び出そうとすると、1 つのスレッドが制御を取得し、他のスレッドは最初のスレッドがプロシーチャーを完了するまで待機します。他のスレッドは、最終的に制御を取得すると、`ATS_done` に値 `*ON` があることを検知するため、1 つずつ即時に戻ります。

```
* !!! Warning !!!
* These global ATS_xxx variables are in all-thread static storage.
* They are setup in getCustList().
* They should not be used before that procedure is called,
```

```

* and they should not be changed after that procedure is called.
D ATS_custList    S          100A  VARYING DIM(500)
D                  STATIC(*ALLTHREAD)
D ATS_numCusts    S          10I 0  INZ(0)
D                  STATIC(*ALLTHREAD)
/free
// Ensure that the all-thread static variables ATS_custList
// and ATS_numCusts have been set up
getCustList();

// Search for the customer name in the customer list
if %lookup(custname : ATS_custList : 1 : ATS_numCusts);
...
/end-free

P getCustList      B          SERIALIZE
FcustList IF      E          DISK
D custInfo         DS          LIKERECD(custRec)
* ATS_done is shared by all threads running this procedure.
* It doesn't need special thread-control because the procedure
* is serialized.
D ATS_done         S          N      INZ(*OFF)
D                  STATIC(*ALLTHREAD)
/free
// Only load the customer array once
if ATS_done;
return;
endif;
// Fetch the list of customers into the ATS_custList array
read custList custInfo;
dow not %eof(custList);
ATS_numCusts += 1;
ATS_custList(ATS_numCusts) = %trim(custInfo.custName);

read custList custInfo;
enddo;

// Set on the "first-time-only" indicator
ATS_done = *ON;
/end-free
P getCustList      E

```

逐次化されたプロシージャが十分な保護を提供しない場合

グローバルな all-thread 静的変数を持っている場合、その変数に対して逐次化された "get" および "set" プロシージャを持つことによって、その変数へのアクセスを適切に制御できように見える場合があります。しかし、プロシージャは個々に逐次化され、それぞれが独自の個別の制御メカニズムを持つため、残念ながら、この方法では適切な保護は提供されません。1 つのスレッドが "get" プロシージャを実行している場合、別のスレッドは同時に "set" プロシージャを実行することができます。

"get" および "set" プロシージャを使用する場合、両方のプロシージャにコードを追加して、変数へのアクセスを手動で同期化する必要があります。

別の方法では、1 つの get-set プロシージャで "get" と "set" の両方を結合します。これは、必要な関数を示すために別々のパラメーターを持つことができます。または、オプションのパラメーターを持つことができ、これが渡されると、"set" 関数を提供します。プロシージャは常に値を戻すため、"get" 関数は常に提供されます。

ただし、単一の "get-set" プロシージャを使用する場合でも、その変数に対して適切なスレッド・セーフティが提供されない場合があります。以前の値を使用して変数を変更する場合（値を変数に追加するなど）、変数の値を取得してから、同じステートメントで新規の値にそれを設定することで適切に対処できる

マルチスレッド化に関する考慮事項

と考える場合があります。ただし、他のスレッドが、プロシーチャーに対する 2 つの呼び出しの間にそのプロシーチャーを呼び出す場合があります。この場合、プロシーチャーに対する 2 番目の "set" 呼び出しは、他のスレッドによって設定された値を誤って上書きします。

```
// If myFld has the value 2 before this statement is run, the first call
// would return 2. The second call would set the value to 3. If another
// thread had set the value to 15 in between the calls, the second call
// should logically set it to 16, not to 3.
getSetMyFld          // second call to getSetMyFld, to set the value
(getSetMyFld() + 1); // first call to getSetMyFld, to get the value
```

変数に対して複数のアクセスを実行し、かつ操作の実行中に、他のスレッドがその変数を取得または設定できないようにする必要がある場合、何らかの手動同期を使用して、変数に対するすべてのアクセスを制御する必要があります。その変数のユーザーはすべて、同じ同期メカニズムを使用する必要があります。

共用リソースへのアクセスを手動で同期化する場合の問題

共用リソースへのアクセスを正常に制御することは非常に難しい作業です。共用リソースのユーザーはすべて、同じ同期メカニズムを使用することに同意しなければなりません。共用リソースの可視性の有効範囲が広いほど、アクセスの制御は難しくなります。例えば、ローカルの all-thread 静的変数へのスレッド・セーフ・アクセスを持つことはきわめて容易です。これは、当該プロシーチャーだけがアクセス可能なためです。エクスポート済みの all-thread 静的変数へのスレッド・セーフ・アクセスを持つことは非常に困難です。これは、あらゆるプロシーチャーがアクセス可能なためです。

スレッド関連の API の使用

システム API を呼び出して、スレッドを開始し、スレッドの完了を待ち、さらに、共用リソースへのアクセスを同期化することができます。

以下の例では、いくつかのスレッドを作成し、2 つの異なる同期化技法を使用していくつかの共用変数へのアクセスを制御します。この例を十分に理解するには、<http://www.ibm.com/systems/infocenter/> で『マルチスレッド・アプリケーション』を参照してください。これらの例は、セマフォおよび mutex の使用方法を示す C の例に似ています。

セマフォの例は、パラメーターを thread-start プロシーチャー threadSem に渡す方法を示します。通常、thread-start プロシーチャー・パラメーターは、サブフィールドがパラメーターの代わりをして、通常のプロシーチャーが使用する値を戻すデータ構造体です。この例には、2 つのサブフィールドがあります。"val" サブフィールドは、thread-start プロシーチャーへの入力で、"result" サブフィールドはプロシーチャーによる出力です。例では、thread-start プロシーチャーは 2 で乗算した入力値に結果を設定します。

例を作成する方法:

1. 例のソースをソース・メンバーにコピーする。以降の説明では、これらがファイル MYLIB/MYSRCFILE、メンバー THREADMTX および THREADSEM RPGLE に入っていることが前提です。
2. プログラムをコンパイルする。
 - CRTBNDRPG MYLIB/THREADMTX SRCFILE(MYLIB/MYSRCFILE)
 - CRTBNDRPG MYLIB/THREADSEM SRCFILE(MYLIB/MYSRCFILE)

例を実行する方法: サンプル・プログラムは、マルチスレッド対応のジョブで実行する必要があります。ジョブで複数のスレッドを許可するように ALWMLTTHD(*YES) を指定して、SBMJOB を使用してプログラムを呼び出します。

- SBJJOB CMD(CALL MYLIB/THREADMTX) ALWMLTTHD(*YES)
- SBJJOB CMD(CALL MYLIB/THREADSEM) ALWMLTTHD(*YES)

```

/UNDEFINE LOG_ALL_RESULTS

H THREAD(*CONCURRENT) MAIN(threadMtx)
H BNDDIR('QC2LE')
/IF DEFINED(*CRTBNDRPG)
H DFTACTGRP(*NO)
/ENDIF
H OPTION(*SRCSTMT : *NOUNREF)

/COPY QSYSINC/QRPGLESRC,PTHREAD
D NUMTHREADS      C              3

D threadMtx      PR              EXTPGM('THREADMTX')

D mtxThread      PR              *  EXTPROC('mtxThread')
D   parm         *  VALUE

D handleThreads  PR              EXTPROC('handleThreads')

D checkResults   PR              EXTPROC('checkResults')
D   string       1000A  VARYING CONST
D   val          10I 0  VALUE

D threadMsg      PR              EXTPROC('threadMsg')
D   string       1000A  VARYING CONST

D print          PR              EXTPROC('print')
D   msg          1000A  VARYING CONST

D CEETREC        PR
D   cel_rc_mod   10I 0  OPTIONS(*OMIT)
D   user_rc      10I 0  OPTIONS(*OMIT)

D sleep          PR              EXTPROC(*CWIDEN:'sleep')
D   secs         10I 0  VALUE

D fmtThreadId    PR              17A  VARYING

```

図 4-51. RPG ソース・ファイル *THREADMTX*

マルチスレッド化に関する考慮事項

```
*-----
* Thread-scoped static variables (the STATIC keyword
* is implied because the definition is global)
*-----
D psds          SDS
D   pgmName          10A   OVERLAY(psds : 334)

*-----
* Job-scoped static variables
*-----

* Shared data that will be protected by the mutex
D sharedData     S          10I 0 INZ(0)
D                                     STATIC(*ALLTHREAD)
D sharedData2     S          10I 0 INZ(0)
D                                     STATIC(*ALLTHREAD)

* A mutex to control the shared data
D mutex          DS          LIKEDS(pthread_mutex_t)
D                                     STATIC(*ALLTHREAD)

// Program entry procedure
P threadMtx      B
/free
    print ('Enter ' + pgmName);
    handleThreads ();
    print ('Exit  ' + pgmName);
/end-free
P threadMtx      E

P handleThreads  B
D handleThreads  PI

D thread         DS          LIKEDS(pthread_t)
D                                     DIM(NUMTHREADS)
D rc             S          10I 0 INZ(0)
D i             S          10I 0 INZ(0)
/free

    print ('"handleThreads" starting');
```

```

print ('Test using a mutex');

// Initialize the mutex
mutex = PTHREAD_MUTEX_INITIALIZER;

print ('Hold Mutex to prevent access to shared data');
rc = pthread_mutex_lock (mutex);

checkResults('pthread_mutex_lock()' : rc);

print ('Create/start threads');
for i = 1 to NUMTHREADS;
    rc = pthread_create(thread(i) : *OMIT
                        : %paddr(mtxThread) : *NULL);
    checkResults ('pthread_create()' : rc);
endfor;

print ('Wait a bit until we are "done" with the shared data');
sleep(3);
print ('Unlock shared data');
rc = pthread_mutex_unlock (mutex);
checkResults('pthread_mutex_unlock()' : rc);

print ('Wait for the threads to complete, '
      + 'and release their resources');
for i = 1 to NUMTHREADS;
    rc = pthread_join (thread(i) : *OMIT);
    checkResults('pthread_join( ' + %char(i) + ' )' : rc);
endfor;
print ('Clean up the mutex');
rc = pthread_mutex_destroy (mutex);

print ('"handleThreads" completed');
return;

/end-free
P handleThreads    E

P mtxThread        B
D mtxThread        PI          *
D    parm          *    VALUE

```

マルチスレッド化に関する考慮事項

```
D rc          S          10I 0
D
/free

    threadMsg ('Entered');

    rc = pthread_mutex_lock (mutex);
    checkResults ('pthread_mutex_lock()' : rc);
    //***** Critical Section Begin *****
    threadMsg ('Start critical section, holding lock');

    // Access to shared data goes here
    sharedData += 1;
    sharedData2 -= 1;

    threadMsg ('End critical section, release lock');
    //***** Critical Section End *****

    rc = pthread_mutex_unlock (mutex);
    checkResults ('pthread_mutex_unlock()' : rc);

    return *NULL;
/end-free
P mtxThread      E

P checkResults    B          EXPORT
D checkResults    PI
D  string          1000A  VARYING CONST
D  val             10I 0  VALUE
D msg             S          1000A  VARYING
/free
    if val <> 0;
        print (string + ' failed with ' + %char(val));
        CEETREC (*OMIT : *OMIT);
    else;
        /if defined(LOG_ALL_RESULTS)
            print (string + ' completed normally with ' + %char(val));
        /endif
    endif;
/END-FREE
P checkResults    E
```

```

P threadMsg      B              EXPORT
D threadMsg      PI
D   string      1000A  VARYING CONST
/Free
    print ('Thread(' + fmtThreadId() + ') ' + string);
/END-FREE
P threadMsg      E

P print          B              EXPORT
D print          PI
D   msg          1000A  VARYING CONST
D printf         PR          *  EXTPROC('printf')
D   template     *          *  VALUE OPTIONS(*STRING)
D   string       *          *  VALUE OPTIONS(*STRING)
D   dummy        *          *  VALUE OPTIONS(*NOPASS)
D NEWLINE        C          x'15'
/free
    printf ('%s' + NEWLINE : msg);
/end-free
P print          E

P fmtThreadId    B              EXPORT
D fmtThreadId    PI          17A  VARYING
D pthreadId      DS          LIKEDS(pthread_id_np_t)
D buf            S          1000A
D sprintf        PR          *  EXTPROC('sprintf')
D buf            *          *  VALUE
D template      *          *  VALUE OPTIONS(*STRING)
D num1           10U 0  VALUE
D num2           10U 0  VALUE
D dummy         *          *  OPTIONS(*NOPASS)
/Free
    pthreadId = pthread_getthreadid_np();
    // get the hex form of the 2 parts of the thread-id
    // in "buf", null-terminated
    sprintf (%addr(buf)
        : '%.8x %.8x'
        : pthreadId.intId.hi
        : pthreadId.intId.lo);
    return %str(%addr(buf));
/END-FREE
P fmtThreadId    E

```


マルチスレッド化に関する考慮事項

```
/UNDEFINE LOG_ALL_RESULTS

H THREAD(*CONCURRENT) MAIN(threadSem)
H BNDDIR('QC2LE')
/IF DEFINED(*CRTBNDRPG)
H DFTACTGRP(*NO)
/ENDIF
H OPTION(*SRCSTMT : *NOUNREF)

/COPY QSYSINC/QRPGLESRC,PTHREAD
/COPY QSYSINC/QRPGLESRC,SYSSEM
/COPY QSYSINC/QRPGLESRC,SYSSTAT
D NUMTHREADS      C      3

D threadSem      PR      EXTPGM('THREADSEM')

D semThreadParm_t...
D      DS      QUALIFIED TEMPLATE
D  val      10I 0
D  result      10I 0
D semThread      PR      *      EXTPROC('semThread')
D  parm      LIKEDS(semThreadParm_t)

D handleThreads  PR      EXTPROC('handleThreads')

D checkResults  PR      EXTPROC('checkResults')
D  string      1000A  VARYING CONST
D  val      10I 0  VALUE

D checkResultsErrno...
D      PR      EXTPROC('checkResultsErrno')
D  string      1000A  VARYING CONST
D  cond      N      VALUE

D threadMsg      PR      EXTPROC('threadMsg')
D  string      1000A  VARYING CONST

D print      PR      EXTPROC('print')
D  msg      1000A  VARYING CONST
```

図 4-52. セマフォの使用を示す RPG プログラム *THREADSEM*

```

D CEETREC          PR
D   cel_rc_mod      10I 0 OPTIONS(*OMIT)
D   user_rc          10I 0 OPTIONS(*OMIT)

D sleep            PR          EXTPROC(*CWIDEN:'sleep')
D   secs            10I 0 VALUE

D fmtThreadId       PR          17A  VARYING

*-----
* Thread-scoped static variables (the STATIC keyword
* is implied because the definition is global)
*-----
D psds              SDS
D   pgmName          10A  OVERLAY(psds : 334)

*-----
* Job-scoped static variables
*-----

* Shared data that will be protected by the mutex
D sharedData        S          10I 0 INZ(0)
D                               STATIC(*ALLTHREAD)
D sharedData2        S          10I 0 INZ(0)
D                               STATIC(*ALLTHREAD)

* A semaphore to control the shared data
D semaphoreId        S          10I 0 STATIC(*ALLTHREAD)

* Simple lock operation. 0=which-semaphore, -1=decrement, 0=noflags
* Will be set to { 0, -1, 0} in main procedure before threads are created
D lockOperation      DS          LIKEDS(struct_sembuf)
D                               DIM(1)
D                               STATIC(*ALLTHREAD)

```

マルチスレッド化に関する考慮事項

```
* Simple unlock operation. 0=which-semaphore, 1=increment, 0=noflags
* Will be set to { 0, 1, 0} in main procedure before threads are created
D unlockOperation...
D                DS                LIKEDS(struct_sembuf)
D                DIM(1)
D                STATIC(*ALLTHREAD)

// Program entry procedure
P threadSem      B
/free
  print ('Enter ' + pgmName);
  handleThreads ();
  print ('Exit ' + pgmName);
/end-free
P threadSem      E

P handleThreads  B
D handleThreads  PI

D thread         DS                LIKEDS(pthread_t)
D                DIM(NUMTHREADS)
D rc             S                10I 0 INZ(0)
D i              S                10I 0 INZ(0)
D parms          DS                LIKEDS(semThreadParm_t)
D                DIM(NUMTHREADS)
/free

  print ('"handleThreads" starting');

  print ('Test using a semaphore');

  lockOperation(1).sem_num = 0;
  lockOperation(1).sem_op = -1;
  lockOperation(1).sem_flg = 0;

  unlockOperation(1).sem_num = 0;
  unlockOperation(1).sem_op = 1;
  unlockOperation(1).sem_flg = 0;
```

```

// Create a private semaphore set with 1
// semaphore that only I can use
semaphoreId = semget(IPC_PRIVATE : 1 : 0 + S_IRUSR + S_IWUSR);
checkResultsErrno ('semget' : semaphoreId >= 0);

// Set the semaphore count to 1.
// Simulate a mutex
rc = semctl(semaphoreId : 0 : CMD_SETVAL : 1);
checkResults('semctl(SETVAL)' : rc);

print ('Wait on semaphore to prevent access to shared data');
rc = semop(semaphoreId : lockOperation(1) : 1);
checkResultsErrno('main semop(lock)': rc = 0);

parms(1).val = 5;
parms(2).val = -10;
parms(3).val = 421;

print ('Create/start threads');
for i = 1 to NUMTHREADS;
    rc = pthread_create(thread(i) : *OMIT
                        : %paddr(semThread) : %addr(parms(i)));
    checkResults ('pthread_create()' : rc);
endfor;

print ('Wait a bit until we are "done" with the shared data');
sleep (3);
print ('Unlock shared data');
rc = semop (semaphoreId : unlockOperation(1) : 1);
checkResultsErrno ('main semop(unlock)' : rc = 0);

print ('Wait for the threads to complete, '
      + 'and release their resources');
for i = 1 to NUMTHREADS;
    rc = pthread_join (thread(i) : *OMIT);
    checkResults('pthread_join( ' + %char(i) + ')' : rc);
endfor;
print ('Clean up the semaphore');
rc = semctl(semaphoreId : 0 : IPC_RMID);
checkResults ('semctl(removeID)' : rc);

```

マルチスレッド化に関する考慮事項

```
    print ('Result(1) = ' + %char(parms(1).result));
    print ('Result(2) = ' + %char(parms(2).result));
    print ('Result(3) = ' + %char(parms(3).result));

    print ("handleThreads" completed');
    return;

/end-free
P handleThreads      E

P semThread          B
D semThread          PI          *
D   parm              LIKEDS(semThreadParm_t)

D rc                  S          10I 0
D
/free

    threadMsg ('Entered + parm.val = ' + %char(parm.val));
    // Set the output subfields of the parameter
    parm.result = parm.val * 2;

    rc = semop (semaphoreId : lockOperation(1) : 1);
    checkResultsErrno ('thread semop(lock)' : rc = 0);

    //***** Critical Section Begin *****
    threadMsg ('Start critical section, holding semaphore');

    // Access to shared data goes here
    sharedData += 1;
    sharedData2 -= 1;

    threadMsg ('End critical section, release semaphore');
    //***** Critical Section End *****

    rc = semop (semaphoreId : unlockOperation(1) : 1);
    checkResultsErrno ('thread semop(unlock)' : rc = 0);

    threadMsg ('Exiting');

    return *NULL;
/end-free
```

```

P semThread      E

P checkResults   B      EXPORT
D checkResults   PI
D  string        1000A    VARYING CONST
D  val           10I 0    VALUE
D msg            S      1000A    VARYING
/FREE
  if val <> 0;
    print (string + ' failed with ' + %char(val));
    CEETREC (*OMIT : *OMIT);
  else;
    /if defined(LOG_ALL_RESULTS)
      print (string + ' completed normally with ' + %char(val));
    /endif
  endif;
/END-FREE
P checkResults   E

P checkResultsErrno...
P               B
D checkResultsErrno...
D               PI
D  string        1000A    VARYING CONST
D  cond          N      VALUE
D getErrnoPtr    PR      *    EXTPROC('__errno')
D errnoVal       S      10I 0    based(threadErrnoPtr)
/FREE
  if not cond;
    threadErrnoPtr = getErrnoPtr();
    print (string + ' Errno(' + %char(errnoVal) + ')');
    CEETREC (*OMIT : *OMIT);
  else;
    /if defined(LOG_ALL_RESULTS)
      print (string + ' completed normally');
    /endif
  endif;
/END-FREE
P checkResultsErrno...
P               E

```


RPG と XML

```
P threadMsg      B                      EXPORT
D threadMsg      PI
D   string              1000A  VARYING CONST
/Free
    print ('Thread(' + fmtThreadId() + ') ' + string);
/End-Free
P threadMsg      E
P print          B                      EXPORT
D print          PI
D   msg              1000A  VARYING CONST
D printf         PR      *  EXTPROC('printf')
D   template              *  VALUE  OPTIONS(*STRING)
D   string              *  VALUE  OPTIONS(*STRING)
D   dummy              *  VALUE  OPTIONS(*NOPASS)
D NEWLINE         C      x'15'
/free
    printf ('%s' + NEWLINE : msg);
/end-free
P print          E

P fmtThreadId    B                      EXPORT
D fmtThreadId    PI      17A  VARYING
D pthreadId      DS      LIKEDS(pthread_id_np_t)
D buf            S      1000A
D sprintf        PR      *  EXTPROC('sprintf')
D buf            *  VALUE
D template              *  VALUE  OPTIONS(*STRING)
D num1           10U 0  VALUE
D num2           10U 0  VALUE
D dummy          *  OPTIONS(*NOPASS)
/Free
    pthreadId = pthread_getthreadid_np();
    // get the hex form of the 2 parts of the thread-id
    // in "buf", null-terminated
    sprintf (%addr(buf)
        : '%.8x %.8x'
        : pthreadId.intId.hi
        : pthreadId.intId.lo);
    return %str(%addr(buf));
/End-Free
P fmtThreadId    E
```

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』
- 4-132 ページの『RPG と MQSeries』
- 4-132 ページの『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 文書进行处理するために、XML-INTO および XML-SAX 命令コードを使用することができます。詳細については、4-127 ページの『XML 文書処理』を参照してください。

XML Toolkit (5733-XT1) 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。

XML 文書の処理

XML-INTO または XML-SAX ステートメントを使用して RPG プログラムから XML 文書を処理することができます。このステートメントは、高速 XML パーサーへの RPG 言語インターフェースです。RPG で現在使用されているパーサーは、多数の適格性エラーに対して XML 文書を検査しますが、非有効化パーサーです。XML パーサーについて詳しくは、「*ILE COBOL プログラマーの手引き*」の付録『XML 参照資料』にある『XML 適合性』節を参照してください。

XML 文書は、文字でも UCS-2 RPG 変数でもよく、または統合ファイル・システムのファイル内にあっておかまいません。

パーサーは、SAX パーサーです。SAX パーサーは、XML 文書を文字単位で読み込んで動作します。要素名、または属性値などの XML 文書の断片を見つけたときはいつでも、パーサーの発呼者が提供する処理手順に答えて、見つけた XML の断片に関する情報を渡します。例えば、パーサーが XML 要素名を見つけた場合、パーサーは、“イベント” が “start element” イベントであることを示す処理手順を呼び出し、エレメントの名前を渡します。

処理手順は、情報を処理し、別のイベントで処理手順を呼び出すのに十分な情報を持つまで XML 文書を読み込み続けるパーサーに戻ります。このプロセスは、XML 文書全体が構文解析されるか、処理手順が構文解析を終了する必要があることを示すまで、繰り返されます。

例えば、以下の XML 文書について考えてみてください。

```
<email type="text">
  <sendto>JohnDoe@there</sendto>
</email>
```

以下は、パーサーが読み込むテキストの断片、パーサーが生成するイベント、および各イベントに関連するデータです。注: 用語 “whitespace” は、行の終わりの文字、タブ文字およびブランクを指します。

構文解析されたテキスト	イベント	イベント・データ
	start document	
<email	start element	"email"
type=	attribute name	"type"
"text"	attribute value	"text"
>whitespace	element content	the whitespace
<sendto>	start element	"sendto"
JohnDoe@there	element content	"JohnDoe@there"
</sendto>	end element	"sendto"
whitespace	element content	the whitespace
</email>	end element	"email"
	end document	

XML-SAX および XML-INTO 命令コードでは、XML パーサーを使用することができます。

1. XML-SAX 命令では、イベント処理手順を指定し、パーサーが生成するすべてのイベントを処理できます。XML 文書に何が含まれているかを事前に知らない場合は、これが便利です。

RPG と XML

例えば、XML 文書が *type* という名前の XML 属性を含むことが分かっていて、この属性の値を知りたい場合は、処理手順は "attribute name" イベントが "type" の値を持つまで待つことができます。次回呼び出されるハンドラーは、必要なデータ (上記の例では "text") を持つ "attribute value" イベントである必要があります。

2. XML-INTO 命令では、XML 文書の内容を RPG 変数に直接読み込むことができます。XML 文書の形式が分かっており、文書の XML 要素の名前が RPG 変数に付けた名前と同じになることが分かっている場合は、これが便利です。

例えば、XML 文書が常に上記の文書の形式を持っていることが分かっている場合は、名前 "email"、およびサブフィールド "type" および "sendto" を持つ RPG データ構造を定義することができます。次に、XML-INTO 命令を使用して、XML 文書をデータ構造に直接読み込むことができます。命令が完了したとき、"type" サブフィールドは値 "text" を持ち、"sendto" サブフィールドは値 "JohnDoe@there" を持ちます。

3. XML-INTO 命令では、繰り返される XML エLEMENT の不明な数の値を取得することもできます。ユーザーは、処理手順が呼び出されるたびに要素の固定数の値を受け取る処理手順を提供します。XML 文書が一連の同一の XML 要素を含むことが分かっており、それがいくつになるか事前に分からない場合は、これが便利です。

XML データは、常にテキスト形式でパーサーによって戻されます。データが数値データ、または日付データなどの他のデータ・タイプを表すことが分かっている場合、XML-SAX 処理手順は、%INT または %DATE などの変換関数を使用してデータを変換する必要があります。

XML-INTO 命令は、文字データをレシーバーとして指定してフィールドまたはサブフィールドのタイプに自動的に変換します。

XML-SAX および XML-INTO 命令の両方で、命令を制御する一連のオプションを指定できます。このオプションは、以下の形式の 1 つの文字式で指定されます。

```
'opt1=val1 opt2=val2'
```

各命令は、独自の有効なオプションのセットを持っています。両方の命令コードに共通なオプションは以下のとおりです。

doc

"doc" オプションは、ユーザーが命令に提供した XML 文書が、文書を含む統合ファイル・システムのファイルの名前なのか、文書自身であるのかを指定します。デフォルトは、ユーザーが実際の XML 文書を提供したことを示す "doc=string" です。オプション "doc=file" は、ユーザーが実際の XML 文書を含むファイルの名前を提供したことを示すために使用します。

ccsid

"ccsid" オプションは、XML パーサーがデータを戻す CCSID を指定します。XML-SAX 命令の場合、パーサーがサポートする任意の CCSID を指定できます。XML-INTO 命令の場合、構文解析が 1 バイト文字または UCS-2 のいずれかで実行されるのかのみを制御できます。このそれぞれの命令の "ccsid" オプションについて詳しくは、「*ILE RPG 解説書*」の情報を参照してください。

XML パーサー・エラー・コード: XML パーサーが構文解析時に XML 文書でエラーを検出した場合、メッセージ RNX0351 が出力されます。メッセージから、エラーが発見された文書内のオフセットと同様に、エラーに関連する特定のエラー・コードを得ることができます。

以下の表は、各パーサー・エラー・コードの意味を示します。

XML パーサー・エラー・コード	説明
1	パーサーは、要素コンテンツの外側の空白文字をスキャン中に無効な文字を見つけました。
2	パーサーは、要素コンテンツの外側に、処理命令、要素、コメント、または文書タイプ宣言の無効な開始を見つけました。
3	パーサーは、重複属性名を見つけました。
4	パーサーは、属性値にマークアップ文字 '<' を見つけました。
5	要素の開始および終了タグ名が一致しませんでした。
6	パーサーは、要素コンテンツに無効文字を見つけました。
7	パーサーは、要素コンテンツに、要素、コメント、処理命令、または CDATA セクションの無効な開始を見つけました。
8	パーサーは、要素コンテンツに、マッチする開始文字シーケンス '<![CDATA[' のない CDATA 終了文字シーケンス ']]>' を見つけました。
9	パーサーは、コメントに無効文字を見つけました。
10	パーサーは、コメントに、'>' が続かない文字シーケンス '--' (2 つのハイフン) を見つけました。
11	パーサーは、処理命令データ・セグメントに無効文字を見つけました。
12	処理命令ターゲット名は、小文字、大文字、または大/小文字混合の 'xml' でした。
13	パーサーは、16 進文字参照に無効桁を見つけました (形式 ෝ、例えば ັ)。
14	パーサーは、小数点文字参照に無効桁を見つけました (形式 &#ddd;.)。
15	文字参照は、合法的な XML 文字を参照しませんでした。
16	パーサーは、エンティティー参照名に無効文字を見つけました。
17	パーサーは、属性値に無効文字を見つけました。
18	パーサーは、起こりうる文書タイプ宣言の無効開始を見つけました。
19	パーサーは、2 番目の文書タイプ宣言を見つけました。
20	要素名は正しく指定されませんでした。最初の文字が文字 '_' または ':' でなかったか、または、パーサーが要素名内または要素名に続いて無効文字を見つけました。
21	属性は正しく指定されませんでした。属性名の最初の文字が文字 '_' または ':' でなかったか、または '=' 以外の文字が属性名に続いて見つかったか、値の区切り文字のいずれかが不正確だったか、無効文字がその名前内またはその名前に続いて見つかりました。
22	空の要素タグが '/' が続く '>' で終了しませんでした。
23	要素終了タグが正しく指定されませんでした。最初の文字が文字 '_' または ':' でなかったか、またはタグが '>' で終了しませんでした。
24	パーサーは、要素コンテンツにコメントまたは CDATA セクションの無効な開始を見つけました。
25	処理命令ターゲット名は、正しく指定されませんでした。処理命令ターゲット名の最初の文字が文字 '_' または ':' でなかったか、または、パーサーが処理命令ターゲット名内またはそれに続いて無効文字を見つけました。
26	処理命令が終了文字シーケンス '?>' で終了しませんでした。
27	パーサーは、文字参照またはエンティティー参照で、'&' に続いて無効文字を見つけました。
28	XML 宣言にバージョン情報が存在しませんでした。
29	XML 宣言で 'version' が正しく指定されませんでした。'version' の後に '=' が続かなかったか、値が欠落していたか、不適切に区切り文字で区切られていたか、値が不正な文字を指定したか、開始および終了区切り文字が一致しなかったか、パーサーが XML 宣言でバージョン情報値終了区切り文字に続いて無効文字を見つけました。

RPG と XML

XML パーサー・エラー・コード	説明
30	パーサーは、XML 宣言にオプションのエンコード宣言の代わりに無効属性を見つけました。
31	XML 宣言のエンコード宣言値が欠落しているか、不正でした。値が小文字または大文字の A から Z で始まらなかったか、'encoding' の後に '=' が続かなかったか、値が欠落していたか、不適切に区切り文字で区切られていたか、不正な文字を指定したか、開始および終了区切り文字が一致しなかったか、パーサーが終了区切り文字に続いて無効文字を見つけました。
32	パーサーは、XML 宣言にオプションのスタンドアロン宣言の代わりに無効属性を見つけました。
33	XML 宣言で 'standalone' 属性が正しく指定されませんでした。'standalone' の後に '=' が続かなかったか、値が欠落していたか、または不適切に区切り文字で区切られていたか、値が 'yes' でも 'no' でもなかったか、値が不正な文字を指定したか、開始および終了区切り文字が一致しなかったか、パーサーが終了区切り文字に続いて無効文字を見つけました。
34	XML 宣言は、適切な文字シーケンス '>' で終了しなかったか、無効属性を含んでいました。
35	パーサーは、ルート要素の終了の後に文書タイプ宣言の開始を見つけました。
36	パーサーは、ルート要素の終了の後に要素の開始を見つけました。
300	パーサーは、文書が完了する前に文書の終わりに到達しました。
301	XML-INTO または XML-SAX の %HANDLER プロシージャラーが非ゼロ値を戻し、XML 構文解析が終了しました。
302	パーサーが要求された CCSID 値をサポートしないか、XML 文書の最初の文字が '<' ではありませんでした。
303	文書は、パーサーが処理するには大き過ぎました。パーサーは不完全な文書を構文解析しようとしたましたが、構文解析の完了には文書の終わりのデータが必要でした。
500-999	外部パーサーでの内部エラーです。サービス技術員にエラーを報告してください。
10001-19999	パーサーでの内部エラーです。サービス技術員にエラーを報告してください。

XML パーサーの制限:

- RPG 文字変数の長さは 16773104 バイトのみです。プログラムがそれよりも長い XML データへのポインターを持つ場合 (例えば MQSeries[®] 呼び出しから)、統合ファイル・システムの一時ファイルに XML データを書き込んで、一時ファイルから XML データを構文解析する必要があります。これを行うサンプル・プロシージャラーについては、4-131 ページの図 4-53 を参照してください。
- 構文解析が 1 バイト文字 CCSID で実行される場合、パーサーが処理できる最大文字数は、2147483408 です。
- 構文解析が UCS-2 で実行される場合、パーサーが処理できる最大 UCS-2 文字数は、1073741704 です。
- パーサーは、すべての CCSID をサポートするわけではありません。ジョブの CCSID が、パーサーが処理しない CCSID のいずれかである場合、文書を UCS-2 で構文解析する必要があります。
 - 1047、37、1140、273、1141、277、1142、278、1143、280、1144、284、1145、285、1146、297、1147、500、1148、871、および 1149 の EBCDIC CCSID はサポートされます。
 - 819、813、920 の ASCII CCSID はサポートされます。
 - 1200、13488、17584 の Unicode CCSID はサポートされます。

パーサーでは、5 つの事前定義の参照 (&、'、>、<、および ") 以外のエンティティ
参照はサポートされません。パーサーが不明なエンティティ参照を検出した場合、「不明参照」イ
ベントまたは「不明属性参照」イベントのいずれかを生成します。イベントの値は、形式 "&name;" の
参照です。

- パーサーは、DOCTYPE 宣言を構文解析しません。 DOCTYPE 宣言のテキストは、"DOCTYPE 宣言" イベント用のデータ値として渡されます。
- パーサーは、名前空間をサポートしません。 XML 要素および属性名のコロンを無視します。
- パーサーは、"接頭部マッピングの開始" および "接頭部マッピングの終了" イベントを生成しません。 XML 要素および属性名のコロンを無視します。

```
* Parameters:
* 1. path      : a pointer to a null-terminated string containing
*               the path to the file to be written
* 2. dataPtr   : a pointer to the data to be written
* 3. dataLen   : the length of the data in bytes
* 4. dataCcsid : the CCSID of the data
* 5. fileCcsid : the desired CCSID of the file
* Sample RPG coding:
*   ifsWrite ('/home/mydir/temp.xml' : xmlPtr : xmlLen : 37 : 37);
*   xml-into ds %xml('/home/mydir/temp.xml' : 'doc=file');
* To delete the file, use the system command
*   rmvlnk '/home/mydir/temp.xml'
```

図 4-53. 統合ファイル・システムのファイルへのデータの書き込み

```
* Note: This module requires BNDDIR(QC2LE)
P ifsWrite      B          EXPORT
D ifsWrite      PI
D path          * VALUE OPTIONS(*STRING)
D dataPtr       * VALUE
D dataLen       10I 0 VALUE
D dataCcsid     10I 0 VALUE
D fileCcsid     10I 0 VALUE

D O_CREAT       C          x'00000008'
D O_TRUNC       C          x'00000040'
D O_WRONLY      C          x'00000002'
D O_RDWR        C          x'00000004'
D O_CCSID        C          x'00000020'
D O_TEXT_CREAT  C          x'02000000'
D O_TEXTDATA    C          x'01000000'
D O_SHARE_NONE  C          x'00080000'

D S_IRUSR        C          x'0100'
D S_IROTH        C          x'0004'
D S_IRGRP        C          x'0020'
D S_IWUSR        C          x'0080'
D S_IWOTH        C          x'0002'
```


RPG と XML

```
D ssize_t      S          10I 0
D size_t       S          10U 0

D open         PR          10I 0 EXTPROC('open')
D path         *          VALUE OPTIONS(*STRING)
D flag         10I 0 VALUE
D mode         10I 0 VALUE
D fileCcsid    10I 0 VALUE options(*nopass)
D dataCcsid    10I 0 VALUE options(*nopass)
D writeFile    PR          LIKE(ssize_t)
D              EXTPROC('write')
D handle       10I 0 VALUE
D data         *          VALUE
D len          VALUE LIKE(size_t)
D closeFile    PR          10I 0 EXTPROC('close')
D handle       10I 0 VALUE

D oflag        S          10I 0
D omode        S          10I 0
D handle       S          10I 0
D rc           S          10I 0

D sysErrno     PR          *      EXTPROC('__errno')
D errno        S          10I 0 BASED(pErrno)
/FREE
  pErrno = sysErrno();
  oflag = 0 + 0_WRONLY + 0_CREAT + 0_TEXT_CREAT + 0_TRUNC
    + 0_CCSDID + 0_TEXTDATA + 0_SHARE_NONE;
  omode = 0 + S_IRUSR + S_IWUSR + S_IRGRP + S_IROTH;

  handle = open(path : oflag : omode : fileCcsid : dataCcsid);
```

```
// insert error handling if handle is less than zero
  rc = writeFile (handle : dataPtr : dataLen);
// insert error handling if rc is not zero
  rc = closeFile (handle);
// insert error handling if rc is not zero
/END-FREE
P ifswrite      E
```

RPG と MQSeries

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。

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。

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。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ない、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          0  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          0  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らなければなりません。

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ない、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ない、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。

QSYSINC/QRPGLESRC で JNI メンバーをコピーする場合、そのファイル内の定義のように、LIKE を使用して RPG 変数およびパラメーターを定義することができます。例えば、Java "int" タイプなどの変数を

RPG と Java

定義するには、それを LIKE(jint) と定義します。RPG および Java に関する以降の説明では、LIKE(jxxxx) で定義したすべての定義はモジュール内に QSYSINC/QRPGLESRC,JNI の /COPY を持つことが前提です。この /COPY ファイルの使用について詳しくは、4-149 ページの『Java を使用するためのその他の RPG コーディング』の節を参照してください。

表 4-21. Java データ・タイプの RPG 定義

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、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いません。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` を使用して指定できません。Java は配列をオブジェクトと見なすため、配列へのパラメーター・マッピングも参照によって渡す必要があります。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ます。

```
D tostring      PR          0  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ない、`VALUE` キーワードが必要です。

RPG と Java

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ます。

```
D getint          PR          0   EXTPROC(*JAVA:
D                                     'java.lang.Integer':
D                                     'getInteger')
D                                     CLASS(*JAVA:'java.lang.Integer')
D                                     STATIC
D   string         0   CLASS(*JAVA:'java.lang.String') CONST
D   num            0   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ます。

```
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ます。

```
D equals          PR          N   EXTPROC(*JAVA:
D                                     'java.lang.Integer':
D                                     'equals')
D   obj           0   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 プロシーチャーが 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。4-152 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 プロシーチャーから 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。4-149 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、4-150 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** 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チャーであっても、静的記憶域に強く依存します。この静的記憶域を正しく取り扱うには **THREAD** キーワードが必要です。このことは、アプリケーションの Java 部分が複数のスレッドで実行されている可能性がある場合、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、4-149 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ます。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ます、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 bdccreate1      PR              0   EXTPROC(*JAVA:
D                                     'java.math.BigDecimal':
D                                     *CONSTRUCTOR)
D   str              0   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 bdccreate2      PR              0   EXTPROC(*JAVA:
D                                     'java.math.BigDecimal':
D                                     *CONSTRUCTOR)
D   double          8F   VALUE

```

図 4-54. *BigDecimal* Java クラスを呼び出す RPG コーディング例


```

*   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              0   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              0   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              0   EXTPROC(*JAVA:
D                                     'java.math.BigDecimal':
D                                     'add')
D                                     CLASS(*JAVA:'java.math.BigDecimal')
D   bd1            0   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              0   CLASS(*JAVA:'java.math.BigDecimal')
D
D double          S              8F   INZ(1.1)
D fld1            S              10A
* Define a prototype to retrieve the String version of the BigDecimal
D getBdString     PR              0   CLASS(*JAVA:'java.lang.String')
D                                     EXTPROC(*JAVA:
D                                     'java.lang.BigDecimal':
D                                     'toString')
* Define a prototype to retrieve the value of a String
D getBytes        PR              65535A VARYING
D                                     EXTPROC(*JAVA:
D                                     'java.lang.String':
D                                     'getBytes')
* Define a variable to hold the value of a BigDecimal object
D bdVal           S              63P 5

```

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('123456789012345678901234567890')
*
* 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      bdn1 = 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      bdn2 = 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。
* bdn1 は呼び出しを行なうために使用するオブジェクトで、
* bdn2 はパラメーターです。
*
C          EVAL      sum = add(bdn1:bdn2)
* ここで sum には bdn1 + bdn2 の値を持つ BigDecimal
* オブジェクトが入ります。
C          EVAL      bdVal = %DECH(getBdString(sum) : 63 : 5)
* val には sum の値が入ります。
* sum の値が、変数 "val" が持つことができる数より大きい場合、
* オーバーフロー例外が発生します。

```

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')
D makestring   PR          0    EXTPROC(*JAVA:
D                                     'java.lang.String':
D                                     *CONSTRUCTOR)
D                                     CLASS(*JAVA:'java.lang.String')
D parm          65535A    CONST VARYING *
* 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          65535A 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 4-55. 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 4-56. 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() メソッドが静的メソッドである場合は、呼び出しは次の例のようになります。

```

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。

RPG と Java

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          0  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            S          0  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");  
    }  
}
```

`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 4-57. 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。IBM i Information Center の『Java システム・プロパティ』の節を参照してください。

- これらのオプションは SystemDefault.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 SystemDefault.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せんが、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。

一部のパラメーターと戻り値には、ジョブの CCSID と Java がバイト配列の場合に使用する CCSID との間の変換が必要です。file.encoding Java プロパティは、Java が使用する CCSID を取得するために RPG が使用するものです。file.encoding プロパティがジョブの CCSID に対して正しく設定されるようにしてください。このプロパティは、ジョブの属性を使用して Java が暗黙的に設定するようにすることもできます。例えば、'-Dfile.encoding=ISO8859_1' または '-Dfile.encoding=Cp948' を QIBM_RPG_JAVA_PROPERTIES 環境変数に追加することができます。file.encoding プロパティについて詳しくは、Information Center の『IBM Developer Kit for Java』トピックを参照してください。

```

# Java 仮想マシンを制御するために指定できるプロパティ、および Java が正しく機能するために必要に
# なる可能性があるその他の環境変数 (QIBM_USE_DESCRIPTOR_STDIO 環境変数など) の詳細について
# は、Information Center で IBM Developer Kit for Java のトピックを参照してください。使用する特定の
# Java クラスの資料も参照しなければならない場合があります。

```

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 サブプロシージャを書きます。ネイティブ・メソッドの場合、プロシージャ開始仕様書で 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と同じようにネイティブ・メソッドを書きます。図 4-58 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 4-58. ネイティブ・メソッドを呼び出す Java クラス

4-145 ページの図 4-59 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 4-59. 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と同様にコーディングされます。図 4-60 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 4-60. 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りません。
- 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を呼び出している場合は、「通常のプロシージャ」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 `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 `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・プロシージャについての詳細は、4-151 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 4-152 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 (『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 4-61 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
D               EXTPROC(*JAVA
D               : 'java.lang.String'
D               : 'getBytes')
D stringLength   PR
D               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 4-61. 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

RPG プロトタイプでのメソッド・パラメーターの不正確な指定: Java メソッドのプロトタイプをコード化する際、戻り値およびパラメーターのタイプを正確に指定しないと、RPG コンパイラーは、メソッド・シグニチャーを不正確にビルドします。プログラムが実行されると、間違ったメソッドが呼び出されるか、呼び出しが `NoSuchMethodError` Java 例外で失敗します。

呼び出しが `NoSuchMethodError` Java 例外で失敗する場合、RPG エラー・メッセージは、メソッドの呼び出しで使用された署名を示します。以下の表は、Java タイプとメソッド・シグニチャー値の間のマッピングを示します。Java タイプと RPG タイプ間のマッピングについては、4-134 ページの表 4-21を参照してください。

Java タイプ	署名
<code>boolean</code>	<code>Z</code>
<code>byte</code>	<code>B</code>
<code>char</code>	<code>C</code>
<code>short</code>	<code>S</code>
<code>int</code>	<code>I</code>
<code>long</code>	<code>J</code>
<code>float</code>	<code>F</code>
<code>double</code>	<code>D</code>
<code>any object</code>	<code>Lclass;</code>
<code>any array</code>	<code>[type</code>

Java クラスのメソッド用の有効な署名のリストを表示するには、`QSH` コマンドを使用します。

```
javap -s classname
```

ここで、`classname` はパッケージと共に指定されます。例: `java.lang.String` クラスが標準の `classpath` にない場合は、`javap` に `classpath` オプションを指定できます。

```
javap -s classname -classpath classlocation
```

メソッド用の有効な署名をメソッド呼び出し用の RPG で使用される署名を比べ、マッピング・テーブルから作業することによって、プロトタイプのエラーを判別できます。

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 (4-149 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。

RPG と Java

```
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、4-149 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、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、サブプロシーチ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、この処理を行なういくつかのサンプル RPG ラッパーを示し、これら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
H bnmdir('QC2LE')
/define OS400_JVM_12
/copy qsysinc/qrpglesrc,jni
/copy JAVAUTIL
```

JNI 関数用の以下の RPG ラッパーを説明します。詳しい作業例については、以下の 4-155 ページの図 4-67 を参照してください。ラッパー機能のプロトタイプおよび定数が含まれるコピー・ファイルについては、Copy-file JAVAUTIL を参照してください。

- 『複数のオブジェクトの同時解放を Java に通知』
- 4-150 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』
- 4-151 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』
- 4-152 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』
- 4-152 ページの『Java 仮想マシン (JVM) の作成』
- 4-152 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。

```
*-----
* 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;
```

RPG と Java

```
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 refObjectP       o          class(*java:'java.lang.Object')
D                  const
D                  options(*nopass)
D newObjectP       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 4-152 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 (java によって呼ばれる) が戻るときにネイティブ・メソッドのために発生しますが、そうでない場合は、オブジェクトを必要としなくなった Java に明示的に通知しない限り、決して発生しません。これは、RPG ラッパー・プロシージャ `freeLocalRef` を呼び出すことによって行ないます。

```
CALLP freeLocalRef (JNIEnv_P : string);
```

4-151 ページの図 4-62 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      0      CLASS(*JAVA
D                  : 'java.lang.Object')
D                  VALUE

/free
  jniEnv_P = env;
  DeleteLocalRef (env : localRef);
/end-free

P freeLocalRef...
P          E

```

図 4-62. *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 4-152 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・プロシーチャー *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);

```

図 4-63 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          0      CLASS(*JAVA
D                  : 'java.lang.Object')
D env          *      VALUE
D localRef      0      CLASS(*JAVA
D                  : 'java.lang.Object')
D                  VALUE

/free
  jniEnv_P = env;
  return NewGlobalRef (env : localRef);
/end-free
P getNewGlobalRef...
P          E

```

図 4-63. *getNewGlobalRef* のソース・コード

RPG と Java

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。

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 `freeGlobalRef` を呼び出します。

```
CALLP freeGlobalRef (JNIEnv_P : globalString);
```

図 4-64 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    0  CLASS(*JAVA
D                  : 'java.lang.Object')
D                  VALUE

/free
  jniEnv_P = env;
  DeleteGlobalRef (env : globalRef);
/end-free
P freeGlobalRef...
P          E
```

図 4-64. `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 がユーザーの代わりに JVM を作成します。JVM の開始時に設定される Java クラスパスおよびその他の Java プロパティを制御する方法については、『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。

JNI 環境ポインターの入手: 何らかの JNI 機能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・ポインターを介して呼び出されます。プロシージャ・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 `getJNIEnv` を呼び出します。

```
EVAL  JNIEnv_P = getJNIEnv();
```

4-153 ページの図 4-65 に、`getJNIEnv` のサンプル・ソース・コードを示します。


```

*-----
* getJniEnv - get the JNI environment pointer
* Note: This procedure will cause the JVM to be created if
*       it was not already created.
*-----
P getJniEnv      b          export
D getJniEnv      pi          *

D attachArgs     ds          likeds(JavaVMAttachArgs)
D env            s          * inz(*null)
D jvm            s          like(JavaVM_p) dim(1)
D nVms           s          like(jsize)
D rc             s          10i 0
D obj            s          o class(*java
D                :          : 'java.lang.Integer')
D newInteger     pr          o extproc(*java
D                :          : 'java.lang.Integer'
D                :          : *constructor)
D value          10i 0 value
/free
  monitor;
    // Get the current JVM
    rc = JNI_GetCreatedJavaVMs(jvm : 1 : nVms);
    if (rc <> 0);
      // Some error occurred
      return *null;
    endif; if (nVms = 0);
      // The JVM is not created yet. Call a Java
      // method to get the RPG runtime to start the JVM
      obj = newInteger(5);

      // Try again to get the current JVM
      rc = JNI_GetCreatedJavaVMs(jvm : 1 : nVms);
      if (rc <> 0)
        or nVms = 0;
        // Some error occurred
        return *null;
      endif;
    endif;
    // Attach to the JVM
    JavaVM_P = jvm(1);
    attachArgs = *allx'00';
    attachArgs.version = JNI_VERSION_1_2;
    rc = AttachCurrentThread (jvm(1) : env
                             : %addr(attachArgs));

    if (rc <> 0);
      return *null;
    endif;

    // Free the object if we created it above while
    // getting the RPG runtime to start the JVM
    if obj <> *null;
      freeLocalRef (env : obj);
    endif;
  on-error;
    return *null;
  endmon;
  return env;
/end-free
P getJniEnv      e

```

図 4-65. getJniEnv のソース・コード

```

*-----
* Copy file JAVAUTIL
*-----
/ if defined(JAVAUTIL_COPIED)
/ eof
/ endif
/ define JAVAUTIL_COPIED
D JNI_GROUP_ADDED...
D          c              0
D JNI_GROUP_NOT_ADDED...
D          c              -1
D JNI_GROUP_ENDED...
D          c              0
D beginObjGroup   pr      10i 0 extproc('beginObjGroup')
D   env          *      const
D   capacityParm  10i 0 value options(*nopass)
D endObjGroup     pr      10i 0 extproc('endObjGroup')
D   env          *      const
D   refObjectP    o      class(*java:'java.lang.Object')
D                  const
D                  options(*nopass)
D freeLocalRef...
D          pr          extproc('freeLocalRef')
D   env          *      value
D   localRef     o      CLASS(*JAVA
D                  : 'java.lang.Object')
D                  value
D getNewGlobalRef...
D          pr          o      class(*JAVA
D                  : 'java.lang.Object')
D                  extproc('getnewGlobalRef')
D   env          *      value
D   localRef     o      class(*JAVA
D                  : 'java.lang.Object')
D                  value
D freeGlobalRef...
D          pr          extproc('freeGlobalRef')
D   env          *      value
D   globalRef    0      class(*JAVA
D                  : 'java.lang.Object')
D                  value
D getJniEnv       pr      *      extproc('getJniEnv')

```

図 4-66. Copy-file JAVAUTIL

```

Java class

class TestClass{
    String name = "name not set";

    TestClass (byte name[]) {
        this.name = new String(name);
    }

    void setName (byte name[]) {
        this.name = new String(name);
    }

    String getName () {
        return this.name;
    }
}

```

図 4-67. JNI 関数用のラッパーの使用

```

RPG program

H THREAD(*SERIALIZE)
H BNDDIR('JAVAUTIL')
// (JAVAUTIL is assumed to the binding directory that lists
// the service program containing the procedures described
// below)
/copy JAVAUTIL
// (JAVAUTIL is assumed to be the source member containing the
// prototypes for the procedures described below)
D TestClass          C          'TestClass'
D StringClass        C          'java.lang.String'
D newTest            PR          0      EXTPROC(*JAVA : TestClass
D                               : *CONSTRUCTOR)
D   name              25A      VARYING CONST
D
D getName            PR          0      CLASS(*JAVA : StringClass)
D
D                               extproc(*JAVA : TestClass
D                               : 'getName')
D setName            PR          0      extproc(*JAVA : TestClass
D                               : 'setName')
D   newName           25A      VARYING CONST
D
D newString           PR          0      EXTPROC(*JAVA : StringClass
D                               : *CONSTRUCTOR)
D   value             65535A     VARYING CONST
D
D nameValue           PR          25A     VARYING
D                               extproc(*JAVA : StringClass
D                               : 'getBytes')
D
D myTestObj           S          LIKE(newTest)
D myString            S          LIKE(newString)
D env                 S          LIKE(getJniEnv)
/free

// Get the JNI environment pointer so that JNI functions
// can be called.

```

```

env = getJniEnv();

// Set the beginning marker for an "object group"
// so that any objects created between now and the
// "end object group" can be freed all at once.
beginObjGroup (env);

// Create a Test object to work with
// We do not want this object to be freed with the
// other objects in the object group, so we make it
// a permanent object
myTestObj = newTest ('RPG Dept');
myTestObj = getNewGlobalRef (env : myTestObj);

// Get the current "name" from the Test object
// This creates a local reference to the Name object
myString = getName (myTestObj);
dsply (nameValue(myString));

// Change the name
setName (myTestObj : 'RPG Department');

// Get the current "name" again. This will cause
// access to the previous local reference to the old name
// to be lost, making it impossible for this RPG
// program to explicitly free the object. If the object
// is never freed by this RPG program, Java could never
// do garbage-collection on it, even though the old String
// object is not needed any more. However, endObjGroup
// will free the old reference, allowing garbage collection
myString = getName (myTestObj);
dsply (nameValue(myString));

// End the object group. This will free all local
// references created since the previous beginObjGroup call.
// This includes the two references created by the calls
// to getName.
endObjGroup (env);

// Since the original Test object was made global, it can
// still be used.
setName (myTestObj : 'RPG Compiler Dept');

// The original Test object must be freed explicitly
// Note: An alternative way to handle this situation
//       would be to use nested object groups, removing
//       the need to create a global reference
//       beginObjGroup -----
//       create myTestObj
//       beginObjGroup -----
//       ...
//       endObjGroup -----
//       use myTestObj again
//       endObjGroup -----
freeGlobalRef (env : myTestObj);

return;
/end-free

```

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・プロシーチ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          0    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          0    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
```

RPG と Java

```
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。『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 (4-161 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)。

4-159 ページの図 4-68 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         ds              likes(jvalue) dim(3)

/free
parms(1).i = 10;                // parameter 1 is an int
parms(2).l = refToInt(string);  // parameter 2 is an object
parms(3).d = 2.5e3;             // parameter 3 is a double
CallVoidMethodA (JNIEnv_P : objectId : methodId : parms);
/end-free

```

図 4-68. RPG における JNI 呼び出し

ポインター JNIEnv_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。

jvalue 構造でのオブジェクト参照の設定: jvalue 構造は次のようになります。

```

D jvalue          DS              QUALIFIED
D                  BASED(jvalue_P)
... more subfields ...
D 1                LIKE(jint)
D                  OVERLAY(jvalue:1)

```

jvalue 構造の「1」フィールドはオブジェクト参照を表しますが、RPG はオブジェクト・タイプのサブフィールドをサポートしません。オブジェクト参照は実際には 4 バイトの整数で、「jvalue」データ構造の「1」サブフィールドはオブジェクトとしてではなく、4 バイトの整数として定義されます。RPG オブジェクト・タイプを jvalue.1 サブフィールドに割り当てるには、RPG コンパイラーがオブジェクト参照を整数として処理するようにするプロシージャおよび「spoofing」プロトタイプを書き込む必要があります。単純に整数パラメーターを取り、整数を戻す 1 つのプロシージャを作成します (以下の例ではプロシージャ「refIntConv」)。次に、プロシージャ・ポインターを使用してこのプロシージャを呼び出す 2 つのプロトタイプを作成します。1 つ目のプロシージャは戻りタイプをオブジェクト・タイプとして定義し (以下の例ではプロシージャ「intToRef」)、もう 1 つのプロシージャはパラメーターをオブジェクト・タイプとして定義します (以下の例ではプロシージャ「refToInt」)。オブジェクト・タイプと整数タイプを変換するには、プロトタイプ refToInt または IntToRef のいずれかを呼び出します。

RPG と Java

```
*-----
* refIntConv_procptr:
*   This procedure pointer is set to the address of a
*   procedure that takes an integer parameter and returns
*   an integer.
*   Since an object reference is actually an integer, we
*   can define prototypes that define either the return value
*   or the parameter as an object reference, to trick the RPG
*   compiler into allowing the Object reference to be passed
*   to or returned from the procedure.
*   Note: This type of trickery is not normally recommended,
*   but it is necessary in this case to circumvent the RPG
*   restriction against Object subfields.
*-----
D refIntConv_name...
D           c              'refIntConv'
D refIntConv_procptr...
D           s              *   procptr
D           inz(%paddr(refIntConv_name))
*-----
* refToInt - convert an object reference to an integer
*-----
D refToInt      pr          10i 0 extproc(refIntConv_procptr)
D   ref          o          class(*java:'java.lang.Object')
D               value
*-----
* intToRef - convert an integer to an object reference
*-----
D intToRef      pr          o   class(*java:'java.lang.Object')
D               extproc(refIntConv_procptr)
D   int          10i 0 value
```

図 4-69. *spoofing* プロシージャークラスがオブジェクト・タイプと整数タイプを変換するためのプロトタイプを持つ /COPY JNICONV_PR

H NOMAIN

/COPY JNICONV_PR

```
*-----
* refIntConv is used with prototypes refToInt and intToRef
* to convert between Object and integer types in RPG.
* See JNICONV_PR for more details.
*-----
D refIntConv      pr          10i 0 extproc(refIntConv_name)
D   parm          10i 0 value
*-----
* The procedure simply returns its parameter.
*-----
P refIntConv      B              export
D refIntConv      pi          10i 0
D   parm          10i 0 value
/free
  return parm;
/end-free
P refIntConv      E
```

図 4-70. オブジェクト・タイプと整数タイプを変換するためのプロシージャークラス

```

/copy QSYSINC/QRPGLESRC,JNI
/copy JNICONV_PR
D jvals          ds          likeds(jvalue) dim(5)
D myString        s          o    class(*java:'java.lang.String')
D newString       pr         o    extproc(*java:'java.lang.String'
D                  : *constructor)
D   val           100a      const varying
/free
  myString = newString('Hello');
  // Set the myString reference in the first jvalue element
  jvals(1).1 = refToInt (myString);
  . . .
  // Set the myString reference from the second jvalue element
  myString = intToRef(jvals(2).1);
  . . .
  return;

```

図 4-71. 変換プロトタイプの使用

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 EBCDIC 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 JAVAUTIL
*-----
* JAVA クラスおよびメソッド
*-----

```

図 4-72. 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

```

D Rectangle      C          'java.awt.Rectangle'
D NewRectangle   PR          0  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 PROCPT
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          0  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 機能呼び出す準備
*-----

```

```

C          EVAL      JNIEnv_P = getJniEnv ()
C          EVAL      rectClass = FindClass (JNIEnv_P
C                                     : ASCII_Rectangle)
C
C -----
C * 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 * -----
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
C
C -----
C * Cleanup プロシーチャーを呼び出します。
C * -----
C          callp      Cleanup()
C          eval      *INLR = '1'
C
C -----
C * Cleanup。 * - 必要であればオブジェクトを解放します。
C * -----
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
C
C -----
C * 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。
C * -----
P CanHdlr      B
D CanHdlr      PI
D   CommArea          *   CONST
C          callp      Cleanup()
P CanHdlr      E

```

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。この関数は、少なくとも 1 つのパラメーターを OPTIONS(*NOPASS) を指定してコード化する必要があります。メソッド・パラメーターをオプションにしない場合は、余分の “dummy” パラメーターを OPTIONS(*NOPASS) を指定して追加します。例えば、次のメソッドの場合、

```
void mymethod (int len, String str);
```

CallVoidMethod に対して以下のプロトタイプをコーディングすることができます。

```
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
D dummy           1a  OPTIONS (*NOPASS)
```

...

```
CallMyMethod (JNIEnv_P : objectId : methodId : 10 : string);
```

図 4-73. 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 4-159 ページの図 4-68 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を呼び出す場合には、RPG プログラムまたはプロシージャ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。

ILE RPG コンパイラーは、コマンドまたは制御仕様書で PGMINFO(*PCML) コンパイラー・パラメーターを指定する場合、ILE RPG プログラムまたはモジュール用に PCML 情報を生成します。コマンドで PGMINFO パラメーターの Location 部分に *STMF または *ALL を指定する場合、PCML 情報をストリーム・ファイルに生成することができます。INFOSTMF コマンド・パラメーターでストリーム・ファイル

の名前を指定します。コマンドで PGMINFO パラメーターの Location 部分に *MODULE または *ALL を指定する場合、または制御仕様書で PGMINFO キーワードを指定する場合、PCML 情報をモジュールに直接生成できます。この情報は、後で QBNRPDI API を使用して取り出すことができます。

CRTBNDRPG の場合、PCML は *ENTRY PLIST つまりメイン・プロシージャの「プロシージャ・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 (Java 固有のメソッドは除く) の「プロシージャ・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 での大/小文字混合名の獲得

制御仕様書の PGMINFO キーワードのパラメーターの 1 つとして *DCLCASE を指定した場合、PCML は、プログラム内のコーディングで使用されたケース (大/小文字) と同じケースの名前で生成されます。例えば、以下のソースの場合、PGMINFO(*DCLCASE) が指定されているため、PCML 内の名前は "myFunction" および "parm1" になります。

```
CTL-OPT PGMINFO(*DCLCASE) NOMAIN;
DCL-PROC myFunction export;
DCL-PI *N;
parm1 CHAR(10);
END-PI;
END-PROC;
```

Java コードは、以下のように、それらの名前に同じケース (大/小文字) を使用します。

```
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。

- ポインター

- プロシージャ・ポインター

- 1 バイト整数

- 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。

RPG と Java

- *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 -->
```

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。

- Integrated Language Environment® ソース・デバッガーの使用による、Integrated Language Environment® 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

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。

Rational Developer for i でデバッガーを使用します。これは推奨される方法であり、プログラムのデバッグ
に関する資料が製品のオンライン・ヘルプに記載されています。IBM i 上で実行されているプログラムを
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
ガーの実行前に、ソース中に直接停止点を設定することができます。統合 Rational Developer for i・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・ブレイクポイントの設定、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、異なる言語で作成された
複数のアプリケーションであっても、単一の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
- ブレイ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、UPDPDOD 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、UPDPDOD パラメーターを *NO に指定します。

デバッグおよび例外処理

- 「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、および最適レベルの効果を含め) ILE ソースの詳細については、「*ILE* 概念」のデバッグに関する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

ILE ソースはプログラム・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用されます。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。

ソースを使用するためには、その前に **CRTRPGMOD** または **CRTBNDRPG** を使用してモジュール・オブジェクトやプログラム・オブジェクトを作成する時にデバッグ・ビューを選択しなければなりません。デバッグ開始後に、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、4-39 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 ソースには使用可能な多数のデバッグ・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チャーの 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。

ILE ソース

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・コマンドのオンライン・ヘルプ情報を表示します。

ILE ソースのオンライン・ヘルプには、デバッグ・コマンドの記述、それらの可能な省略形の説明、各コマンドの構文図が示されます。また、ソースを使用して変数の表示や変更を行う、各 ILE 言語で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** ソースを使用し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。詳細については、5-8 ページの『**ILE** ソースの開始』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 5-1. 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

デバッグのためのプログラムの準備

表 5-1. デバッグ・ビュー (続き)

デバッグ・ビュー	デバッグ・データ	DBGVIEW パラメーター値
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。

```
CRTBNDPRG 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と同じです。 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はなく、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。

リスト・ビューを暗号化することによって、担当者が暗号鍵を知らない限りデバッグ・セッション中にリスト情報を表示できないようにすることができます。CRTBNDRPG、CRTRPGMOD、または CRTSQLRPGI コマンドの DBGENCKEY パラメーターを使用して暗号鍵を指定します。プログラムをデバッグするときに同じ鍵を指定してリスト・ビューのテキストを表示します。

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)
```

デバッグのためのプログラムの準備

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/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 つの条件を満たしていないときは、OPM システムを使用して 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 5-10 ページの図 5-1 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、cpoc、および RPGPGM のソースは、5-52 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。

ILE ソースの開始

```

                                モジュール・ソースの表示
プログラム :  DEBUGEX          ライブラリ:  MYLIB          モジュール:  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=変数監視   F18=監視の処理  F24=キーの続き
```

図 5-1. 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表示 (**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、次の入力を行います。

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、4-44 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 5-12 ページの図 5-2 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。

デバッグ・セッションでのプログラムの追加/除去

- 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。
- 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	プログラム/モジュール	ライブラリ	タイプ		
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=検索 F12=取消し					

図 5-2. 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。

- 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。

- F14 (モジュール・リストの処理) キーを押して 5-13 ページの図 5-3 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。
- 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。
- 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	プログラム/モジュール	ライブラリ	タイプ	
	*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	

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

終わり

図 5-3. 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 ソースが変更先のビューの位置と等価の位置にマップされます。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。

プログラム・ソースの表示

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 5-10 ページの図 5-1 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。図 5-4 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、5-15 ページの図 5-5 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	プログラム/モジュール	ライブラリ	タイプ	
	*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=検索 F12=取消し

図 5-4. 別のモジュールへの変更

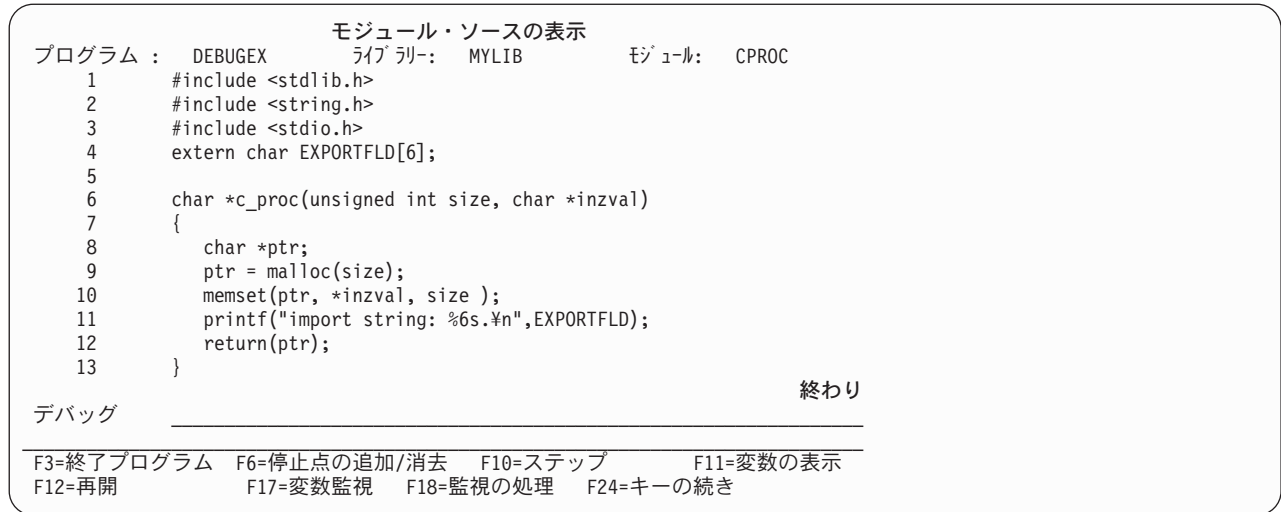


図 5-5. ILE C プロシージャ cproc のソース・ビュー

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 5-16 ページの図 5-6 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 5-16 ページの図 5-6 に示されます。

図 5-6. モジュールのビューの変更

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。

- ## ブレイクポイントの設定と除去

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。

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、5-23 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。

ブレイクポイントの設定と除去

無条件ジョブ・ブレイクポイントを設定・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・ブレイ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、ブレイ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・コマンドを使用して無条件ジョブ・ブレイクポイントを除去するには、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、ブレイ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。ジョブ・ブレイ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・ブレイクポイントの設定例:

この例では F6 (ブレイクポイントの追加/消去) を使って、無条件ジョブ・ブレイクポイントを設定します。ブレイ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・ブレイクポイントを設定したいときは、ステップ 3 を続けます。別のモジュールでジョブ・ブレイ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イ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 (ブレイクポイントの追加/消去) キーを押してください。行 88 にブレイクポイントがなければ、5-19 ページの図 5-7 に示すように、無条件ブレイクポイントがその行に設定されます。その行にブレイクポイントがある場合には、除去されます。

注: 最初の 演算仕様書でブレイ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イクポイントはその行に設定されることになります。

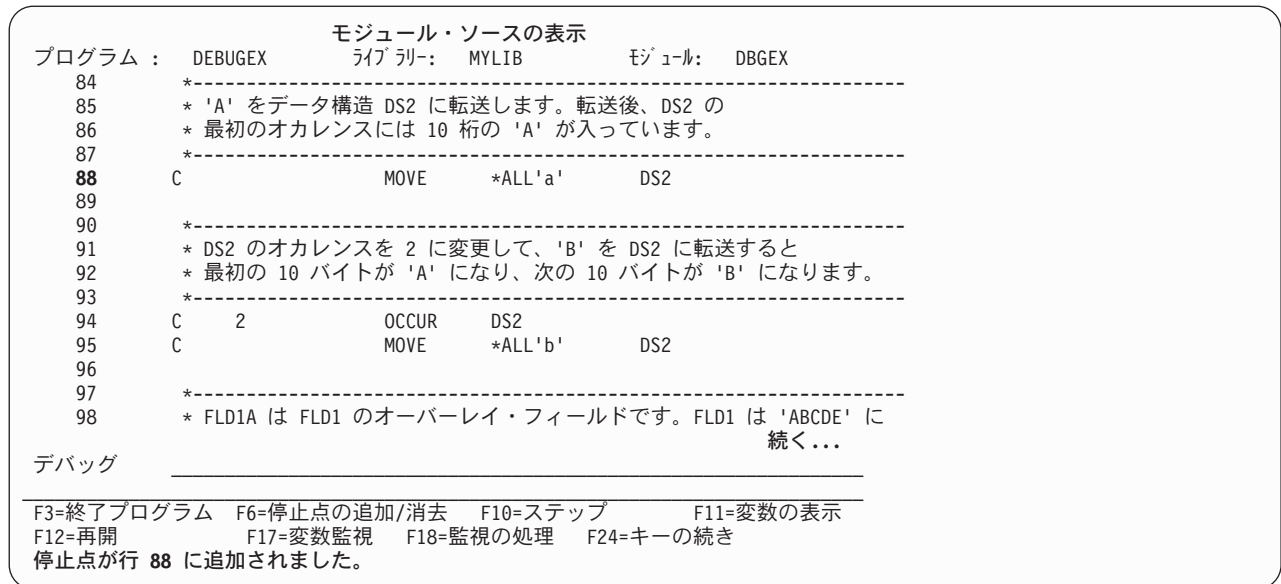


図 5-7. 無条件ジョブ・ブレイクポイントの設定

- ブレイ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。ブレイクポイントは除去されません。
- プログラムを呼び出してください。ブレイクポイントに達するとプログラムは停止し、ブレイ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、プログラムをステップスルーするか、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・ブレイ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・ブレイ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・ブレイクポイントの処理」画面
- 現行スレッドのスレッド・ブレイクポイントを設定する **TBREAK** デバッグ・コマンド
- スレッド・ブレイクポイントを除去する **CLEAR** デバッグ・コマンド

「モジュール・ブレイクポイントの処理」画面を使って、無条件スレッド・ブレイ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・ブレイ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・ブレイ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・ブレイ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。ブレイ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。

ブレイクポイントの設定と除去

無条件スレッド・ブレイ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・ブレイ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・ブレイ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・ブレイ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・ブレイクポイントの処理」画面
- ジョブ・ブレイクポイントを設定する BREAK デバッグ・コマンド
- ブレイクポイントを除去するための CLEAR デバッグ・コマンド

注: 条件付きブレイ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・ブレイ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・ブレイクポイントの処理」画面によるものです。「モジュール・ブレイクポイントの処理」画面には、「モジュール・ソースの表示」画面で F13 (モジュール・ブレイクポイントの処理) 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・ブレイクポイントの追加、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 5-21 ページの図 5-8 に示します。

ジョブ・ブレイ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・ブレイ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、ブレイ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・ブレイ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・ブレイ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・ブレイ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、より多くのブレイ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。

条件付きブレイ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イ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、ブレイ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* はブレイクポイントが出てくる時に評価される条件付き式です。条件式ブレイ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イ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、5-22 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イ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、ブレイ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・ブレイ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・ブレイクポイントを設定します。

1. 条件付きジョブ・ブレイ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イクポイントを追加します。
3. *IN02='1' の時に条件付きブレイクポイントを行 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 5-8 は、条件付きブレイ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			
プログラム . . . : DEBUGEX		ライブラリー . . : MYLIB	システム: XXXXXXXX
モジュール . . . : DBGEX		タイプ : *PGM	
オプションを入力して、Enter 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 5-8. 条件付きジョブ・ブレイクポイントの設定

条件付きジョブ・ブレイクポイントが行 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。

既存のブレイクポイントは常に、同じ位置に入力された新しいブレイクポイントによって置き換えられます。

5. ブレイ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・ブレイクポイントの処理」画面を出てください。F3 (プログラム終了) キーを押して、ILE ソースを終了してください。ブレイ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。ブレイ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、プログラムをステップスルーするか、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・ブレイ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・ブレイクポイントを指定するには、次のようにします。

ブレークポイントの設定と除去

1. 「モジュール・ソースの表示」画面から次の入力を行います。

```
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

プログラム : DEBUGEX ライブラリ: MYLIB モジュール: DBGEX

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

*-----

* 次の SETON 命令後に、*IN02 = '1'.

*-----

C SETON

C IF *IN02

C MOVE '1994-09-30' BigDate

C ENDIF

*-----

* ARRY の 2 番目のセルに新しい値を入れる。

*-----

C MOVE 4 Arry

*-----

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=変数監視 F18=監視の処理 F24=キーの続き

図 5-9. 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、5-23 ページの表 5-2 を参照してください。

ソース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、ソースは分類順序を決定するために、その文字の 16 進数値を使用します。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代替順序は、ILE ソースには使えません。その代わりに、ソースは *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、分類順序テーブルにアクセスするために、ソース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 *JOB RUN 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にアクセスできない場合には、ソースは *HEX 分類順序テーブルを使用します。

表 5-2. 非数値条件付きブレイクポイントの式

タイプ	可能な式
Char-8	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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
<p>注:</p> <ol style="list-style-type: none"> 1. 文字リテラルは 'abc' の形式になっています。 2. X'16 進数字' 形式の 16 進リテラル。 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・ブレイ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・ブレイクポイントを設定、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、これが重要です。

BREAK デバッグ・コマンドを使って無条件ジョブ・ブレイ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、ブレイクポイントを設定するプロシーチャーの名前です。ILE RPG では、1 つのモジュールにつき複数のプロシーチャーが許されるため、*procedure-name* は、メイン・

ブレイクポイントの設定と除去

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* はブレイ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 5-10 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 5-11 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 番号	<----- ソースの仕様 ----->	26 - 35	>----->	注記 ----->	Src Seq	Id	番号
1 C	ソ ー ス ・ リ ス ト						
2 C	MOVE	'123'	BI_FLD1				000100
	SETON			LR----			000200
	***** ソ ー ス の 終 わ り *****						

図 5-10. OPTION(*NOSRCSTMT) が指定された場合のリストのセクション例

SEQ 番号	<----- ソースの仕様 ----->	26 - 35	>----->	注記 ----->	Statement	番号
000100 C	ソ ー ス ・ リ ス ト					
000200 C	MOVE	'123'	BI_FLD1			000100
	SETON			LR----		000200
	***** ソ ー ス の 終 わ り *****					

図 5-11. 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 についてブレイ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イ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イ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、ブレイクポイントは 'SETON LR----' 行に設定されます。

モジュール・ソースの表示			
プログラム : TEST	ライブラリ: MYLIB	モジュール: 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=変数監視	F18=監視の処理	F24=キーの続き
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 5-12. ステートメント・ビューを使用したブレイクポイントの設定

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イ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、ブレイクポイントは 'P FmtCust B' 行に設定されます。

モジュール・ソースの表示

プログラム :	MYPGM	ライブラリ:	MYLIB	モジュール:	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

```

More...

デバッグ **BREAK fmtcust/2600**

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=変数監視 F18=監視の処理 F24=キーの続き
停止点が行 34 に追加されました。

図 5-13. ステートメント番号を使用したブレイクポイントと、*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・ブレイ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 WHEN expression

と入力します。変数の *procedure-name* および *statement-number* は無条件ブレイクポイントのものと同じです。変数の *expression* はブレイ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イクポイントまたは条件付きブレイ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 procedure-name/statement-number

と入力します。

条件付きスレッド・ブレイ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・ブレイ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・ブレイクポイントの処理」画面
- 現行スレッドの条件付きスレッド・ブレイ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・ブレイクポイントを除去する **CLEAR** デバッグ・コマンド

「モジュール・ブレイクポイントの処理」画面の使用: 「モジュール・ブレイクポイントの処理」画面を使って条件付きスレッド・ブレイ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・ブレイクポイントと同様に埋めます。
4. 実行 (キー) を押します。

ブレイクポイントの設定と除去

「モジュール・ブレイクポイントの処理」画面を使って条件付きスレッド・ブレイクポイントを除去するには、次のようにします。

1. 除去したいブレイクポイントの次の *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・ブレイ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・ブレイ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・ブレイ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・ブレイ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イクポイントの除去

「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・ブレイ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イ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、条件付きブレイ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・ブレイ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過しているときに、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るし、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ステップイントゥする必要があります。

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 (変数監視)。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。

ウォッチ条件の設定および除去

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 :

オプションを入力して、Enter 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

長さ
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 5-14. 「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

ウォッチの処理

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

プロシーチャー: 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 5-15. 「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

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 :	
-----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 5-16. 「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
```

ウォッチ条件の設定および除去

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なります。

- IBM i の初期プログラム・ロード (IPL) では、システム全体にわたるすべてのウォッチ条件が除去されます。

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 5-17 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。

プログラム : PAYROL

52 C
53 C
54 C
55 C
56 C
57 C
58 C
59 C
60 C
61 C
62 C
63 C
64 C
65 C
66 C
67 C

モジュール・ソースの表示
ライブラリ: MYLIB モジュール: PAYROLL
eval cnt = 1
dow (cnt < EMPMAX)
eval Pay_exmpt(cnt) = eflag(cnt)
eval cnt = cnt + 1
enddo

eval index = 1
dow index <= cnt
if Pay_exmpt(index) = 1
eval SALARY = 40 * Pay_wage(index)
eval numexmpt = numexmpt + 1
else
eval SALARY = Pay_hours(index)*Pay_wage(index)
endif
eval index = index + 1
enddo

デバッグ

続く ...

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=変数監視 F18=監視の処理 F24=キーの続き

行 65 の監視番号 1 , 変数 : SALARY

図 5-17. 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= プログラム終了 F12= 再開 F14= モジュール・リストの処理 F18= 監視の処理
F21= コマンド入力 F22= ステップイン F24= 出力の表示
命令 18 の監視番号 1 , 変数: SALARY

図 5-18. 「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

プログラム・オブジェクトのステップスルー

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

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が再度停止する前に、プロシージャ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

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いければなりません。

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

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。ソースを開始した後、「モジュール・ソースの表示」画面から、次を入力します。

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。

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。

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 5-19 に示すようにブレークポイント 102 で停止します。

図 5-19. *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

- 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			
プログラム :	DEBUGEX	ライブラリ:	MYLIB
		モジュール:	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=変数監視 F18=監視の処理 F24=キーの続き			
行 145 でステップが完了した。			

図 5-21. 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

スカラー・フィールド	RPG 定義
> EVAL String STRING = 'ABCDEF'	6A INZ('ABCDEF')
> EVAL Packed1D0 PACKED1D0 = -093.40	5P 2 INZ(-93.4)
> EVAL ZonedD3D2 ZONEDD3D2 = -3.21	3S 2 INZ(-3.21)
> EVAL Bin4D3 BIN4D3 = -4.321	4B 3 INZ(-4.321)
> EVAL Int3 INT3 = -128	3I 0 INZ(-128)
> EVAL Int5 INT5 = -2046	5I 0 INZ(-2046)
> EVAL Int10 INT10 = -31904	10I 0 INZ(-31904)
> EVAL Int20 INT20 = -463972	20I 0 INZ(-463972)
> EVAL Unsigned3 UNSIGNED3 = 128	3U 0 INZ(128)
> EVAL Unsigned5 UNSIGNED5 = 2046	5U 0 INZ(2046)
> EVAL Unsigned10 UNSIGNED10 = 31904	10U 0 INZ(31904)
> EVAL Unsigned20 UNSIGNED20 = 463972	20U 0 INZ(463972)
> EVAL DBCSString DBCSSTRING = 'BBCCDD'	3G INZ(G'~BBCCDD~')
> EVAL NullPtr NULLPTR = SYP:*NULL	* INZ(*NULL)
基底付きフィールド	
> EVAL String STRING = 'ABCDEF'	6A INZ('ABCDEF')
> EVAL BasePtr BASEPTR = SPP:C01947001218	* INZ(%ADDR(String))
> EVAL BaseString BASESTRING = 'ABCDEF'	6A BASED(BasePtr)
日付、時刻、タイム・スタンプ・フィールド	
> EVAL BigDate BIGDATE = '9999-12-31'	D INZ(D'9999-12-31')
> EVAL BigTime BIGTIME = '12.00.00'	T INZ(T'12.00.00')
> EVAL BigTstamp BIGTSTAMP = '9999-12-31-12.00.00.000000'	Z INZ(Z'9999-12-31-12.00.00.000000')

図 5-23. 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

コマンドで `OPTION(*NOUNREF)` を指定するか、または制御仕様書キーワードを指定する場合、プログラム・ソース内の参照されない変数は *RPG* モジュールに生成されません。デバッガーで参照されない変数を評価しようとする場合、デバッガーは、変数が存在しないことを示します。

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、パラメーターなし、または *INPUT か *YES のいずれかのパラメーターで 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 5-24 は、DBGEX 中で配列に EVAL を使用した例を示したものです。

```
> EVAL Array                                3S 2 DIM(2) INZ(1.23)
  ARRAY(1) = 1.23    ** 配列全体の表示 **
  ARRAY(2) = 1.23
> EVAL Array(2)      ** 2 番目の要素の表示 **
  ARRAY(2) = 1.23
> EVAL Array(1..2)   ** 要素の範囲の表示 **
  ARRAY(1) = 1.23
  ARRAY(2) = 1.23
```

図 5-24. 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。

5-40 ページの図 5-25 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 5-25. 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-index* 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 5-26 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_DS1_DS1 ** 現行指標値の判別 **
  _QRNU_DS1_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 5-26. 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 5-27 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 5-27. 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

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

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 = 'aaaaaaaaabbbbbbbbb'
```

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在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。

自身がポインター `PTR2` に基づいている基底付きポインター `PTR1` に基づく変数 `FLD1` を持っている場合、デバッガーで単純な `EVAL` コマンドを使用して `FLD1` を評価することはできません。

その代わりに、デバッガーに基底付きポインターのチェーンを明示的に提供する必要があります。

```
====> EVAL PTR2->PTR1->FLD1
```

例えば、次のような定義がある場合、

```
D pPointers      S          *
D pointers       DS          based(pPointers)
D p1             S          *
D p2             S          *
D data1          S          10A  based(p1)
D data2          S          10A  based(p2)
```


プログラム・オブジェクトのステップスルー

これらのコマンドをデバッガーで使用して、DATA1 および DATA2 の値を次のように表示または変更することができます。

```
====> eval pPointers->p1->data1
====> eval pPointers->p2->data2 = 'new value'
```

デバッガーで指定する式を決定するには、評価したい値を持つ式の終わりから開始します。

```
data1
```

次に左に移動して、data1 を定義するために、BASED キーワードに表示される名前 (p1) を追加します。

```
p1->data1
```

次に再び左に移動して、p1 を定義するために、BASED キーワードに表示される名前 (pPointers) を追加します。

```
pPointers->p1->data1
```

指定したポインターが BASED キーワードで定義されていない場合、式は完了します。この場合、pPointers は基底として定義されないため、デバッグ式が完了します。

```
====> eval pPointers->p1->data1
```

基底付き変数の評価

変数がポインターに基づく場合、その変数は通常の EVAL コマンドを使用したデバッガーによる評価に使用不可である場合があります。基底付きポインター自身が基底付きである場合、または基底付きポインターが参照で渡される入力パラメーターである場合 (CONST キーワードを使用した読み取り専用参照を含む)、これが発生することがあります。

例えば、以下のプログラムで、"basedFld" は、入力パラメーターであるポインター "parmPtr" に基づいています。

```
/copy myPgmProto
D myPgm          pi
D parmPtr        *
D basedFld       s      5a  based(parmPtr)
```

デバッガーで "basedFld" を評価するには、以下のメソッドのいずれかを使用します。

1. 5-43 ページの『ポインターがアドレス指定するデータの表示』に説明されている :c または :x 表記を使用して基底付きポインターを評価します。例えば、次のとおりです。

```
====> eval parmPtr:c
====> eval parmPtr:x
```

注: このメソッドは、パック、整数、または UCS-2 データなどの自然な表示と似ていない 16 進数表示を持つデータで正しく機能しません。

2. 基底付きポインターを明示的に指定するには、デバッガーの "矢印" 表記を使用します。このメソッドは、変数を変更するためにも使用できます。

```
====> eval parmPtr->basedFld
====> eval parmPtr->basedFld = 'abcde'
```

変数が 3 レベル以上の基底付き変数を持っている場合、2 番目のメソッドを使用する必要があります。例えば、以下のプログラムで、変数 "basedVal" は、3 レベルの基底付きポインターを持っています。これは、ポインター "p1" に基づいており、"p1" はポインター "p2" に基づいており、"p2" はさらにポインタ

ー "p3" に基づいています。変数 "basedVal" は、単純に "EVAL basedVal" を使用してデバッガーで評価することはできません。

D	storage	s	5a	inz('abcde')
D	val	s	5a	
D	basedVal	s	5a	based(p1)
D	p1	s	*	based(p2)
D	p2	s	*	based(p3)
D	p3	s	*	
D	ptr1	s	*	inz(%addr(storage))
D	ptr2	s	*	inz(%addr(ptr1))
D	ptr3	s	*	inz(%addr(ptr2))
C		eval	p3 = ptr3	
C		eval	p2 = ptr2	
C		eval	p1 = ptr1	
C		eval	val = basedVal	

"basedVal" などの変数を表示するには、デバッガーの *p1->p2->name* 表記を使用して、基底付きポインターを明示的に指定します。この表記を使用するには、表示する変数を指定してから、左に向かって作業し、基底付きポインター名と矢印 (->) を指定します。基底付きポインター自身が基底付きである場合は、2 番目の基底付きポインターと矢印を、以前の基底付きポインターの左に指定します。

例えば、basedVal を評価するには、以下のようにします。

```

===> EVAL p3->p2->p1->basedVal
          aaaaaaaa
          bbbb
          cccc
          dddd
a. variable name
b. basing pointer of variable ->
c. basing pointer of basing pointer ->
d. and so on

```

ヌル可能フィールドの表示

EVAL デバッグ・コマンドを使用してヌル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。

デバッガーがヌル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'

```

データ構造が null が可能なサブフィールドを持っている場合、データ構造のすべての null 可能なサブフィールド用の null 標識は、データ構造 _QRNU_NULL_dsname のサブフィールドとしてそれ自身保管されます。

データ構造が修飾されていない場合は、null 標識データ構造は修飾されていません。null が可能なサブフィールドの名前は、形式 _QRNU_NULL_subfieldname の中にあります。

例えば、修飾されたデータ構造 DS1 が null が可能なサブフィールド FLD1 および FLD2 および null が可能でないサブフィールド FLD3 を持っている場合、データ構造 _QRNU_NULL_DS1 は、標識サブフィ

プログラム・オブジェクトのステップスルー

ールド `_QRNU_NULL_NULLFLD1` および `NULL2` を持ちます。データ構造のすべての `null` 可能なサブフィールドを表示するには、以下のデバッグ・コマンドを使用します。

```
EVAL _QRNU_NULL_DS1      Result:  _QRNU_NULL_FLD1 OF _QRNU_NULL_DS1 = '1'
                             _QRNU_NULL_FLD1 OF _QRNU_NULL_DS1 = '0'
```

データ構造が修飾されている場合は、`null` 標識データ構造は修飾されています。`null` が可能なサブフィールドの名前は、データ構造サブフィールドの名前と同じです。

例えば、修飾されたデータ構造 `DS2` が `null` が可能なサブフィールド `F1` および `F2` および `null` が可能でないサブフィールド `F3` を持っている場合、データ構造 `_QRNU_NULL_DS2` は、標識サブフィールド `F1` および `F2` を持ちます。データ構造のすべての `null` 可能なサブフィールドを表示するには、以下のデバッグ・コマンドを使用します。

```
EVAL _QRNU_NULL_DS2      Result:  _QRNU_NULL_DS2.F1 = '0'
                             _QRNU_NULL_DS2.F2 = '1'
```

変数の `null` 標識を表示するには、最外部の名前を `_QRNU_NULL_name` で置換して、変数自身にアクセスするときと同じ `EVAL` 式をデバッガーで使用します。

```
EVAL FLD1                  Result: 'abc'
EVAL _QRNU_NULL_FLD1      Result: '0'
```

```
EVAL SUBF2                 Result: 0
EVAL _QRNU_NULL_SUBF2     Result: '1'
```

```
EVAL ARR(3)                Result: 13
EVAL _QRNU_NULL_ARR(3)    Result: '1'
```

```
EVAL DS3.INFO(2).SUB4      Result: 'xyz'
EVAL _QRNU_NULL_DS3.INFO(2).SUB4 Result: '0'
```

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

文字列・フィールドのサブ文字列

%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、文字列変数のサブ文字列を作成することができます。最初のパラメーターは文字列識別コード、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

図 5-28 は、5-54 ページの図 5-31 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tring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 characters)
  FLD1 = 'ABCDE'
> EVAL String        (6 characters)
  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 5-28. 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、5-40 ページの図 5-25 および 5-41 ページの図 5-26 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。

XML-SAX 処理手順のデバッグ

XML-SAX イベント処理手順に渡される 2 番目のパラメーターは、どの SAX イベントがパーサーで発見されたかを示す数値です。

RPG コードでは、*XML_START_ELEMENT および *XML_DOCTYPE_DECL のような特殊値を使用して、イベント値をテストすることができます。

ただし、この特殊値はデバッガーでは使用不可です。代わりに、制御仕様に DEBUG(*XMLSAX) キーワードをコード化する場合に使用可能にされる特殊な配列を使用することができます。配列の名前は _QRNU_XMLSAX です。配列エレメントの値は、先頭の "**XML_" を引いた、特殊語の名前と同じです。

プログラム・オブジェクトのステップスルー

例えば、XML-SAX イベント処理手順の 2 番目のパラメーターの名前が "xmlEvent" の場合、以下のデバッガー式を使用して、イベントの名前を決定します。

```
EVAL _QRNU_XMLSAX(xmlEvent)
```

```
Result: _QRNU_XMLSAX(XMLEVENT) = 'START_DOCUMENT'
```

イベント・ハンドラーに渡される 3 番目のパラメーターは、データへのポインターです。5-43 ページの『ポインターがアドレス指定するデータの表示』を参照し、4 番目のパラメーターの値を使用して、バイト単位でのデータの長さを決定します。

「例外」 イベントの場合、5 番目のパラメーターは、構文解析例外に関連するエラー・コードを保持します。エラー・コードの意味については、4-127 ページの『XML 文書の処理』を参照してください。

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* は変更したい変数の名前であり、*value* は変数 *field-name* に割り当てたい ID、リテラル、または定数値です。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 進数字' 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。これによって、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。

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。

変数の `null` 標識を変更するには、最外部の名前を `_QRNU_NULL_name` で置換して、変数自身にアクセスするときと同じ EVAL 式をデバッガーで使します。

```
EVAL FLD1 = 3
EVAL _QRNU_NULL_FLD1 = '0'
```

```
EVAL SUBF2 = 5
EVAL _QRNU_NULL_SUBF2 = '0'
```

```
EVAL ARR(3) = 0
EVAL _QRNU_NULL_ARR(3) = '1'
```

```
EVAL DS3.INFO(2).SUB4 = 'some value'
EVAL _QRNU_NULL_DS3.INFO(2).SUB4 = '0'
```

ヌル可能フィールドのデバッグの詳細については、5-45 ページの『ヌル可能フィールドの表示』を参照してください。

5-50 ページの図 5-29 は、5-54 ページの図 5-31 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。

フィールドの属性の表示

```
** ターゲットの長さ = ソースの長さ **
> EVAL String='123456'      (6 characters)
  STRING='123456' = '123456'
> EVAL ExportFld           (6 characters)
  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 characters)
  STRING = 'ABCDEF'
> EVAL LastName            (10 characters)
  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 characters)
  STRING = '123456'
> EVAL TableA              (3 characters)
  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 5-29. 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。

5-51 ページの図 5-30 は、5-54 ページの図 5-31 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 Array
  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 5-30. 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。別の式で使った場合には、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・コマンドに対応させるには、デバッグ・コマンド行に、

```
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

```
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。

フィールド、式、またはコマンドと名前の等値化

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

```
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 の制約事項の詳細については、「*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

5-54 ページの図 5-31 は、プログラム DEBUGEX のメイン・プロシーチ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。5-57 ページの図 5-32 および 5-57 ページの図 5-33 は、呼び出されるプログラム RPGPGM とプロシーチャー 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. 5-54 ページの図 5-31 のソースを使ってモジュール DBGEX を作るには、次の入力を行います。

```
CRTTRPGMOD 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. 5-57 ページの図 5-33 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. 5-57 ページの図 5-32 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。

デバッグ用サンプル・ソースの例

```

=====
*  DEBUGEX - ILE RPG ソースでの ILE ソース・デバッグ・
*            プログラムの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 Packed100       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"')
* 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 Array           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 F1d1            S          5A   INZ('ABCDE')
D F1d1a           S          1A   DIM(5) OVERLAY(F1d1)
D F1d2            S          5B 2  INZ(123.45)
*
D DS2             DS          10   OCCURS(2)

```

図 5-31. モジュール *DBGEX* のソース： *DBGEX* はプログラム *DEBUGEX* のメイン・モジュールです。

```

*
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 c_proc
*  および switch のプロトタイプ定義
*-----*
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
*-----*
* 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 の最初のカレンスには 10 桁の 'a' が入っています。
*-----*
C              MOVE      *ALL 'a'      DS2
*-----*
* DS2 のカレンスを 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')

```


デバッグ用サンプル・ソースの例

```

*-----*
* 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)

```

```

*-----*
* 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
*-----*
* Array の 2 番目のセルに新しい値を入れます。
*-----*
C          MOVE       4          Array
*-----*
* 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

```

```

=====
*  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 5-32. 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 5-33. C プロシージャ cproc のソース： cproc は DBGEX によって呼び出されます。

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。

デバッグ用サンプル・ソースの例

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。

- 5-61 ページの『ILE RPG 例外処理』
- 5-67 ページの『エラー標識または 'E' 命令コード拡張の指定』
- 5-71 ページの『ファイル・エラー処理 (INFSR) サブルーチンの使用』
- 5-68 ページの『MONITOR グループの使用』
- 5-73 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。

- 『例外処理の概要』(一般概念について)
- 5-67 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 サイクルについては、「*IBM Rational Development Studio for i: 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、4-78 ページの『呼び出しスタック』を参照してください)。

5-60 ページの図 5-34 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ルから構成される 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。

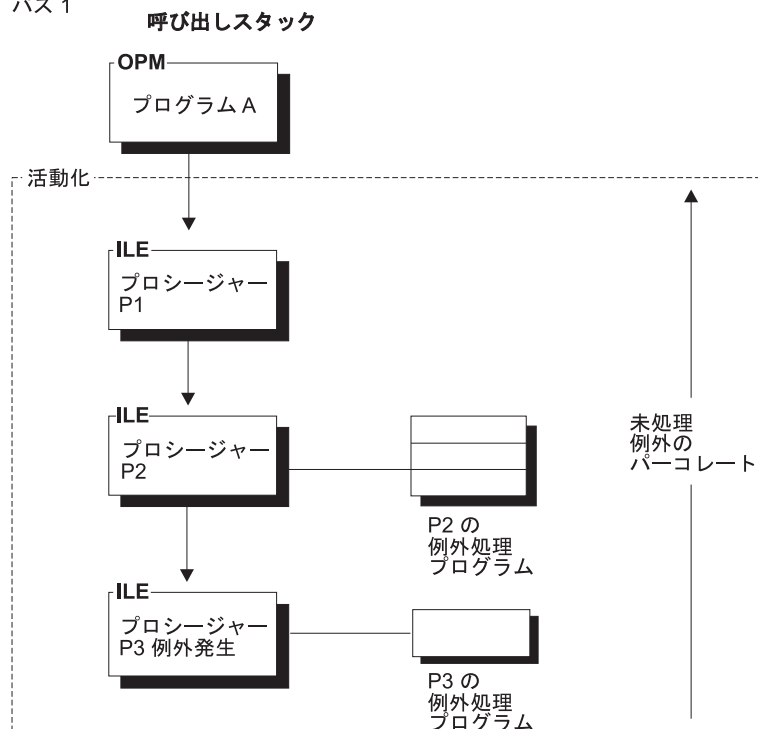
一般に、例外が起こった時には、呼び出しスタック項目に関連した処理プログラムが例外を処理できるようになります。例外がリスト上のどの処理プログラムによっても処理されなかった場合には、例外は未処理と見なされ、その点で、未処理例外について次のようなデフォルトの処置が取られます。

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。 5-60 ページの図 5-34 では、プロシージャ P1 が制御境界です。

例外処理の概要

パス 1



パス 2

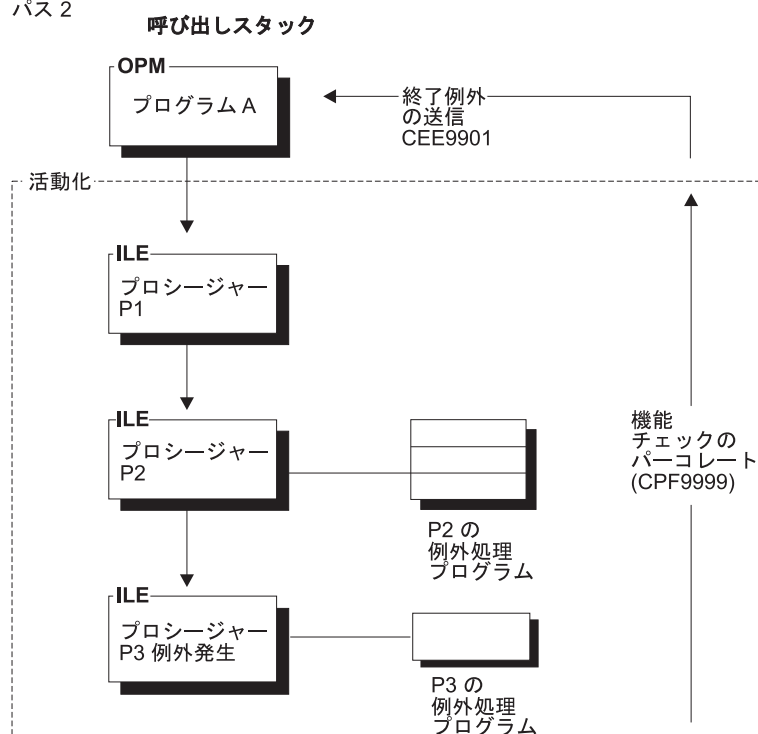


図 5-34. 呼び出しスタックおよび例外メッセージのパーコレーション

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・サブルーチンは、ファイルと同じ有効範囲でコーディングする必要があります。サイクル・モジュール内のグローバル・ファイルは、サイクル・メイン・プロシージャー内にサブルーチンを持つ必要があります。ローカル・ファイルはファイルと同じサブプロシージャー内にサブルーチンを持つ必要があります。サブプロシージャーで使用するグローバル・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。

サイクル・メイン・プロシージャー内の例外処理プログラム: 例外がサイクル・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。

例外処理の概要

- 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」拡張も存在せず、例外を生成するコードが MONITOR グループの MONITOR プロック内にある場合、制御は MONITOR グループの 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、

or

- 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、プロシーチャーの適切な時点 ('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、プロシーチ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、5-64 ページの『未処理例外』を参照してください。

サブプロシーチャー内の例外処理プログラム: サブプロシーチャー内の例外処理プログラム (リニア・メイン・プロシーチャーとして指定されたものを含む) は、サイクル・メイン・プロシーチャー内の例外処理プログラムとは次の点で異なります。

- グローバル・ファイルを使用している場合、ユーザーはそのファイルの INFSR サブルーチンをコーディングできないため、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 グループを使用し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。

サブプロシーチャー内の例外処理は、サブプロシーチャー用に生成された RPG サイクル・コードがないために、基本的にサイクル・メイン・プロシーチャーと異なります。結果として、サブプロシーチャー用のデフォルトの例外処理プログラムがないために、サイクル・メイン・プロシーチ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、サブプロシーチャーの異常終了が発生します。これは次のことを意味します。

- サブプロシーチャー内の *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。 サイクル・メイン・プロシーチ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。 サブプロシーチャーでは、ENDSR に達した場合には、サブプロシーチャーが異常終了し、サブプロシーチ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。
- サブプロシージャ内のエラーについての照会メッセージは出されないで、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。

グローバル・ファイルの PSDS および INFDS がモジュールの有効範囲を持っていることに注意してください。メイン・プロシージャおよびサブプロシージャともにこれらに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をコーディングし、各ローカル *PSSR からそのプロシージャ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、7-1 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。

例外処理プログラムの使用

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 (サイクル・メイン・プロシージャのみ)

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・タイプがサイクル・メイン・プロシージャに関連した **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。図 5-35 は、このシナリオと発生するイベントを説明しています。

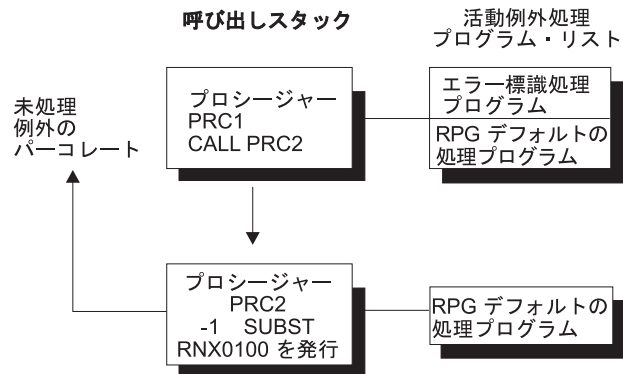


図 5-35. 処理不能エスケープ・メッセージのシナリオ

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。

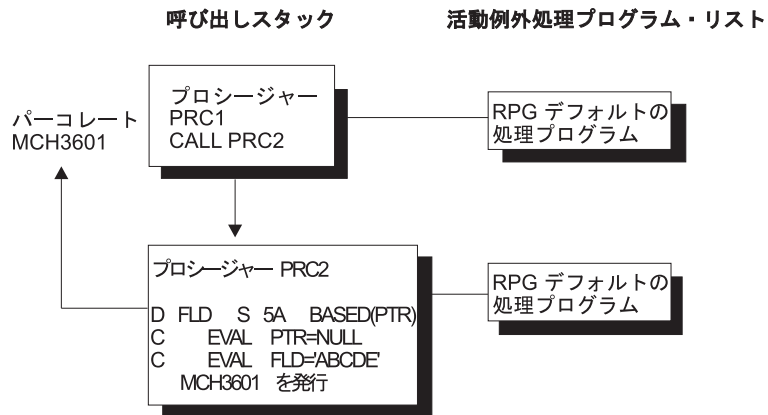
未処理機能チェックの例: 次のシナリオでは、サイクル・メイン・プロシージャーで機能チェックが起こり、処理されなかった時に発生するイベントについて説明します。このシナリオでは次のことが想定されています。

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。

例外処理プログラムの使用

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・エラーが起きます。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すると、機能チェックが起きます。図 5-36 は、このシナリオと発生するイベントを説明しています。

パス 1



パス 2

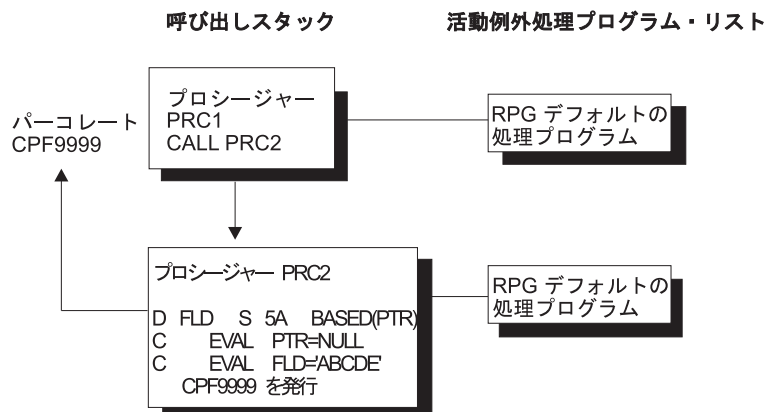


図 5-36. 処理不能の機能チェックのシナリオ

ここで次のことが起きます。

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と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 キーワードの詳細については、「*IBM Rational Development Studio for i: 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、4-39 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。「*IBM Rational Development Studio for i: ILE RPG 解説書*」には、*PSSR エラー処理サブルーチンと INFSR エラー処理サブルーチン、EXSR 命令コード、および INFDS データ構造と PSDS データ構造に関する詳細が記載されています。

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: エラー標識または「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)、どのエラー標識または「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、このプログラムをサポートする命令の任意の位置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、このプログラムをサポートする命令コードの
いずれかで「E」(または「e」)を指定します(例えば、CHAIN(E))。'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。

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チャーの呼び出しを含んでおり、かつそのサブプロシーチャーにエラーがある場合は、そのサブプロシーチャーのエラー処理が優先順位を有します。例えば、サブプロシーチャーに *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、サブプロシーチ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、またはプロシージャ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の特定期間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
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 5-37. 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、サイクル・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ブルーチンのコーディング方法については、 5-77 ページの『エラー処理サブ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: 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グローバル・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、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一サブ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、5-79 ページの『ENDSR 命令での戻り点の指定』を参照してください。演算項目 2 はサブプロシージャー内のファイル・エラー・サブブルーチンでは使用できません。
5. ファイル・エラー処理サブ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・エラー処理サブブルーチンで ILE RPG コンパイラ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・エラー処理サブブルーチンの最後の仕様書でなければなりません。

5-72 ページの図 5-38 には、INFSR エラー処理サブ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	0	E			DISK				
FPRINT	0	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 5-38. ファイル例外処理の例

* 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			
0		TRNPRDNO			
0				29	'NOT IN PRDMAS FILE'

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チャーのどれか (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チャーに固有です。

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。サブプロシーチャーでは、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、5-79 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。

5-75 ページの図 5-39 には、サイクル・メイン・プロシーチャーのプログラム・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・プロシーチ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チャーを終了させます。
*-----*
C      ELSE
C      MOVE            '*CANCL'   ReturnPt
C      ENDIF
C      ENDSR          ReturnPt

```

図 5-39. サイクル・メイン・プロシーチャーの *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、プロシーチャーが終了します。

5-76 ページの図 5-40 および 5-77 ページの図 5-41 では、サブプロシーチ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、プロシージャは正常に実行されません。
*-----*
C      ENDSR
P      E

```

図 5-40. GOTO のある *PSSR サブルーチンのサブプロシージャの例

```

*-----*
* サブプロシージャ定義の開始
*-----*
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、プロシージャは正常に実行されません。
*-----*
C      ENDSR
P      E

```

図 5-41. RETURN のある *PSSR サブルーチンのサブプロシージャの例

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。

5-78 ページの図 5-42 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

```

=====
* NOLLOOP: *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: プロシーチ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、プロシーチャーを *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チ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 5-42. エラー処理サブルーチンでのループの防止

図 5-42 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、次の入力を行います。

```

CRIBNDRPG PGM(MYLIB/NOLLOOP) DBGVIEW(*SOURCE)
STRDBG PGM(MYLIB/NOLLOOP)

```

*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、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チャーが取り消されます。
5. ENDSR 命令が制御を受け取り、プロシーチ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

サイクル・メイン・プロシージャで 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 として以下のいずれか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。 サブプロシージャで処理を再開したい場合には、サブプロシージャの本体の 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サイクル・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用方法については、「*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。

図 5-43 にはプロシージャ RPGHDLR のソースを示してあります。このプロシージャは 3 つのプロシージャ・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 QSYSVRT 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 QSYSVRT 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。                                *
*   このプロシージャ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 プロシージャの例外処理プロシージャ        *
*       です。                                                       *
=====
FQSYSVRT  0    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・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の PSDS へのポインター            *
*   b. スtring・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 5-43. 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

ILE 条件処理プログラム

```

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 パーコレート进行处理します。 *
* メッセージ番号 (MsgNo) の値は 16 進であることを注意。          *
*-----*
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

*=====*
* プロシーチャー出力                                *
*=====*
OQSYSVRT  E
0
0          ProcName          'HDLR: In Handler for '
OQSYSVRT  E          Handling
0          'HDLR: Handling...'
OQSYSVRT  E          Perclating
0          'HDLR: Percolating...'

```

5-84 ページの図 5-44 は、条件処理プログラム RPGHDLR が登録されているプロシーチャー SHOWERR のソースを示します。

プロシーチャー・パラメーターには、RPGHDLR へのプロシーチャー・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・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。

ILE 条件処理プログラム

```

*=====
* 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。                             *
*=====
FQSYSPT  0    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 5-44. 条件処理プログラム登録のソース

* 処理プログラムを登録し、エラーを生成します。 *					

C	EXSR	RegHndl r			

* サブstring・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、 *					
* 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、プロシーチ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	DeRegHndl r			
C	SETON				LR

* RegHdlr - 処理プログラムを登録する API の呼び出し *					

C	RegHdlr	BEGSR			
C	CALLP	CEEHDLR(pConHdlr : %ADDR(CommArea) : *OMIT)			
C	ENDSR				

* DeRegHdlr - 処理プログラムを登録解除する API の呼び出し *					

C	DeRegHdlr	BEGSR			
C	CALLP	CEEHDLU(pConHdlr : *OMIT)			
C	ENDSR				

ILE 条件処理プログラム

```
*****
* *PSSR: プロシーチ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'

*****
* プロシーチャー出力*
*****
OQSYSPT E      ImBack
0
OQSYSPT E      InPssr
0      'I'm Back'
OQSYSPT E      Cancelling
0      'In PSSR'
OQSYSPT E      Cancelling
0      'Cancelling...'
```

これらのプロシーチ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. 5-81 ページの図 5-43 に示されたソースを使用してプロシーチャー **RPGHDLR** を作成するために、次を入力してください。

```
CRTRPGMOD MODULE(MYLIB/RPGHDLR)
```

2. 5-84 ページの図 5-44 に示されたソースを使用してプロシーチャー **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 **ERRORTTEST** を作成するために、次を入力してください。

```
CRTPGM PGM(MYLIB/ERRORTTEST) MODULE(SHOWERR RPGHDLR)
```

4. プログラム **ERRORTTEST** を実行するために、次を入力してください。

```
OVRPRTF FILE(QSYSPT) SHARE(*YES)
CALL PGM(MYLIB/ERRORTTEST)
```

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、プロシーチ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チャー登録 (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チャー (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。これらの ILE バ
```

インド可能 API についての詳細は、Web サイト <http://www.ibm.com/systems/i/infocenter/> で **IBM i Information Center** の中の『プログラミング』カテゴリの『CL および API』の節を参照してください。

図 5-45 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、同じ方法でサイクル・メイン・プロシージャーに対して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			
* handler.			
*-----			
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 5-45. 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

*-----
 * 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	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	
C	PARM	
C		Handler
C		*OMIT
C	EVAL	Msg = 'Array index error'
C	EVAL	Array(Zero) = 'x'
C	ENDSL	
P Enabler	E	

*-----			
* 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が設定した			
* handler.			
*-----			
C	'Cancel Hdlr'	DSPLY	Msg
P	CanHdlr	E	

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 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 5-46. 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 プロシージャが同一活動化グループ内の ILE CL プロシージャ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、デフォルト応答の T 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が、RPG プロシージャ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 プロシージャとは異なる活動化グループにあるようにする。
2. RPG プロシージャで 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 5-47 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 5-47. CPF4906 を無視する ILE 条件処理プログラム

5-91 ページの図 5-48 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 5-48. 状況および通知メッセージを無視する方法

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」で『System Reply List』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。サイクル・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。

- 定様式ダンプを実行するのではなく、プログラムを実行する十分な権限をユーザーに与えます。
*OBJOPR および *EXECUTE 権限を与えることによって、これを実行します。

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 *DUMP または *YES で指定された場合に、生成されます。そうでない場合には、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 (5-52 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

```

プログラム状況域:
プロシージャ名 .....:  DBGEX2
プログラム名 .....:  TEST
ライブラリー .....:  MYLIB
モジュール名 .....:  DBGEX2
プログラム状況 .....:  00202
直前の状況 .....:  00000
エラーのステートメント .....:  00000088
RPG ルーチン .....:  RPGPGM
パラメーターの数 .....:
メッセージ・タイプ .....:  MCH
追加のメッセージ情報 .....:  4431
メッセージ・データ .....:
  プログラム記号の違反
RNK9001 の原因となった状況 .....:
最終使用ファイル .....:
最終ファイル状況 .....:
最終ファイル操作 .....:
最終ファイル・ルーチン .....:
最終ファイル・ステートメント .....:
最終ファイル・レコード名 .....:
ジョブ名 .....:  MYUSERID
ユーザー名 .....:  MYUSERID
ジョブ番号 .....:  002273
システム開始日付 .....:  09/30/1995
開始した日付 .....:  *N/A*
開始した時刻 .....:  *N/A*
コンパイル日付 .....:  123095
コンパイル時刻 .....:  153438
コンパイラ・レベル .....:  0001
ソース・ファイル .....:  QRPGLSRC
ライブラリー .....:  MYLIB
メンバー .....:  DBGEX2
  
```

図 5-49. 定様式ダンプのプログラム状況情報セクション

- A** プロシージャ識別: プロシージャ名、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、プログラム内に情報がないフィールドを示しています。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

定様式ダンプの例

```

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
ファイル操作 . . . . . : 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 . . . . . : 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

```

図 5-50. 定様式ダンプのフィードバック域セクション

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:

```

0000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *

```

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/systems/i/infocenter/> で **IBM i Information Center** の中の『データベースおよびファイル・システム』カテゴリの『DB2 for i』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・バス:									
0000	64800000	00001AF0	00001800	00000000	00000140	000001C6	00000280	000002C0	* 0 F *
0020	00000530	00000000	00000000	00000380	00000000	06000000	00000000	00000000	* *
0040	00000800	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 5-51. 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・バ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

定様式ダンプの例

ILE RPG/400 定様式ダンプ

Module Name. : DBGEX2

Optimization Level : *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
ARRAY	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
BIGSTAMP	TIMESTAMP(26)	'9999-12-31-12.00.00.000000'	
	VALUE IN HEX	'F9F9F9F960F1F260F3F160F1F24BF0F04BF0F04BF0F0F0F0F0'X	
BIN4D3	BIN(4,3)	-4.321	'EF1F'X
BIN9D7	BIN(9,7)	98.7654321	'3ADE68B1'X
DBCSTRING	GRAPHIC(3)	' BBCCDD '	'C2C2C3C3C4C4'X

図 5-52. 定様式ダンプのデータ・セクション

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'	'8181818181818181'X
	(2)	'bbbbbbbbbb'	'8282828282828282'X
DS3	DS	T	
FIRSTNAME	CHAR(10)	'Fred'	'C69985844040404040'X
LASTNAME	CHAR(10)	'Jones'	'D1969585A240404040'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

定様式ダンプの例

```
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 * * * * *
```

- 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** ヌル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・ポインターを区別しません。
- Z** サブプロシージャ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。

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 つ以上のプロシーチャーを含む Integrated Language Environment® プログラムのことです。

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

IBM i でのすべての入出力操作にとって重要な要素は、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、ディスクット・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。

ファイル記述と入出力装置との関連付け

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 사용할 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 6-1 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 6-1. 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 IBM i 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 6-1. RPG 装置タイプと IBM i のファイル・タイプとの相互関連

RPG 装置タイプ	IBM i 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、ディスクット、保管、プリンター、データベース
SPECIAL	N/A

図 6-2 は、図 6-1 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。

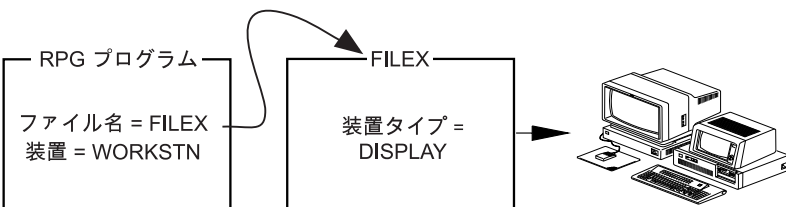


図 6-2. ファイル名と表示装置ファイル記述との関連付け

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 6-3 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。

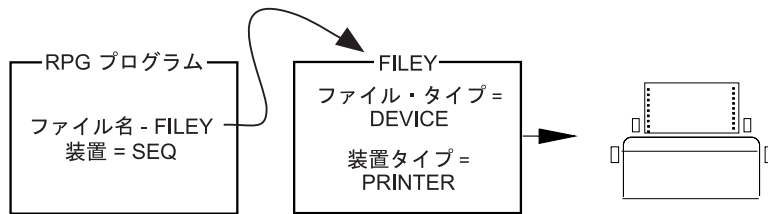


図 6-3. ファイル名と表示装置ファイル記述との関連付け

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、6-13 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

IBM i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、ソース内のファイルに使用される名前はファイル記述仕様書の 7 から 16 桁目に指定されます。ファイル名は 1 から 10 文字の長さとすることができ、その有効範囲（グローバルまたはローカル）内で固有な名前にしなければなりません。EXTFILE キーワードを使用して、実行時にファイルを位置指定することができます。EXTFILE キーワードが指定されない場合、ライブラリー・リスト内のファイルを位置指定するために、実行時に同じ名前が使用されます。外部記述ファイルの場合、コンパイル時にファイルを位置指定するために、EXTDESC キーワードが使用されます。実行時に同じファイルを使用したい場合は、EXTFILE(*EXTDESC) を指定することができます。

実行時のファイルの位置指定に EXTFILE キーワードを指定しない場合、またはコンパイル時のファイルの位置指定に EXTDESC キーワードを指定しない場合は、ファイル指定変更コマンドを使用して、特定の名前、ライブラリー、またはメンバーを指定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、6-13 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、データ構造体を使用してファイル操作のデータを保持することができ、グローバル・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できます。

ファイル記述のタイプ

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うことができます。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。 詳しくは、6-10 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)、またはデータ構造体のサブフィールドとして、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、コンパイラーはユーザーが定義したデータ構造体、またはファイル用にコーディングした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

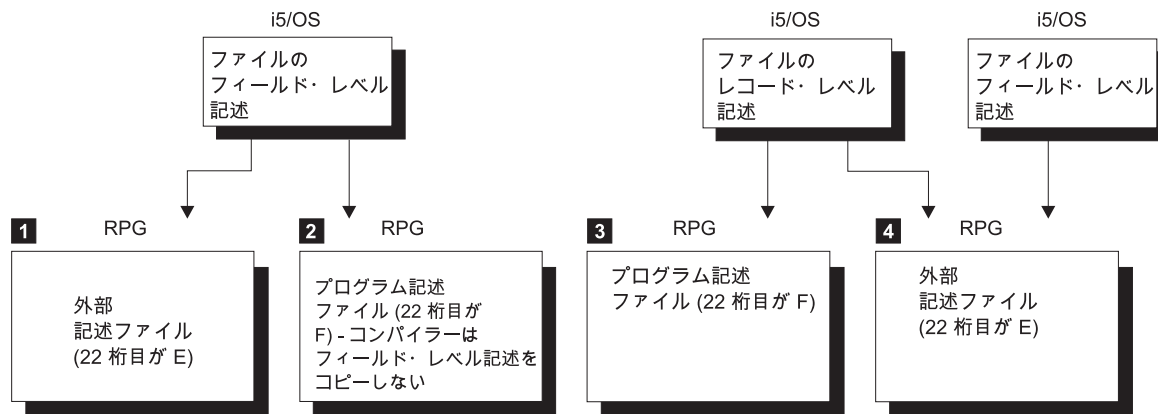


図 6-4. IBM iでの RPG プログラムとファイル間の代表的な関係

- 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に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。詳しくは、6-13 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 IBM i 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

外部記述ファイルの定義

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)。

ファイルが QUALIFIED キーワードで定義されている場合、ファイルの定義に使用されるキーワード内の様式名を指定する場合を除き、様式名はプログラム内で filename.formatname の様式で指定されます。例えば、プログラム内でファイルに MYFILE という名前が付けられ、FMT1、FMT2、および FMT3 の様式を持つと想定します。FMT3 を NEWFMT3 に名前変更する場合、RENAME(FMT3:NEWFMT3) を指定し、FMT2 を無視する場合、IGNORE(FMT2) を指定します。計算中または LIKEREK キーワードの指定時に、名前 MYFILE.FMT1 および MYFILE.NEWFMT3 の修飾様式を使用します。

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。入力および出力仕様書は、グローバルな外部記述ファイルの場合には任意指定であり、サブプロシーチャー内の外部記述ファイルまたは修飾ファイルでは使用できないことに注意してください。

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。プログラム中のファイル名および非修飾レコード様式名は、それぞれ固有の記号名でなければなりません。ファイルが修飾されており、レコード様式が filename.fmtname という様式で指定される場合、様式の名前はプログラム内で固有である必要はありません。

レコード様式名を変更するためには、図 6-5 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。ファイルが修飾されている場合でも、RENAME キーワードが付いた名前の修飾様式は使用しないということを覚えておいてください。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++
FITMMSTL IP E K DISK RENAME(ITEMFORMAT:MSTITM)
*
```

図 6-5. 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 6-5 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。同じ名前を持つ、異なるファイルからのレコード様式が存在するという問題の他の解決策としては、QUALIFIED キーワードを使用して、修飾されているようにファイルを定義します。修飾ファイルのレコード様式は、filename.formatname という様式で指定され、様式名がプログラム内の他の名前と同じであっても問題はありません。

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。6-7 ページの図 6-6 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 6-6. 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。図 6-7 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。ファイルが修飾されている場合でも、**IGNORE** または **INCLUDE** キーワードが付いた名前の修飾様式は使用しないということを覚えておいてください。

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 6-7. 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

- 図 6-8 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。
- 図 6-8 に示されているようにフィールド名を変更する。
- 図 6-8 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。
- 6-9 ページの図 6-9 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。
- 6-9 ページの図 6-9 に示されているようにフィールド標識を割り当てる。

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。

図 6-8. 外部記述への RPG 機能の一時変更および追加

- ## 6-8 IBM i: ILE RPG プログラマーの手引き

```

*.. 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6-9. 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、出力仕様書は任意指定です。また、サブプロシージャー内のローカル・ファイルまたは修飾ファイルでは使用できません。RPG は、外部記述ファイルの出力仕様書を必要とせず、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詳細については、Web サイト <http://www.ibm.com/systems/i/infocenter/> で **IBM i 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 for i』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。 6-10 ページの図 6-10を参照してください。

外部記述ファイルの定義

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書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 命令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 ...+... *
OFilenam++DF..N01N02N03Excnam++++B++A++Sb+Sa+.....*
OITMREC      D      20
0.....N01N02N03Field++++++YB.End++PConstant/editword/DTformat++
0
*ALL 1
*
OSLSREC      D      30
0
0
0
0
15
*
SLSNAM 2
COMRAT
BONUS
```

図 6-10. 外部記述ファイルの出力仕様書

- 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。
- 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レコード様式は、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、または指定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、5-57 ページの『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/systems/i/infocenter/> で **IBM i Information Center** の『データベースおよびファイル・システム』カテゴリの『DB2 for i』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. グローバル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. グローバル出力ファイルの場合には、出力仕様書で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、いずれかのファイル作成コマンドを使用します。これらのコマンドは、Web サイト <http://www.ibm.com/systems/i/infocenter/> で **IBM i Information Center** の中の『プログラミング』カテゴリの『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。表 6-2 は、IBM i 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 6-2. 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

データ管理命令と ILE RPG 入出力命令

表 6-2. データ管理命令および対応する RPG 入出力命令 (続き)

データ管理命令	ILE RPG 入出力命令
READ	
相対	READ、CHAIN
レコード番号による	
キーによる	READ、READE、CHAIN、プライマリーおよびセカンダリー・ファイル
順次	READ
前	READP、READPE
NEXT	READ、READE
送信勧誘装置	READ
WRITE-READ	EXFMT
WRITE	
相対	WRITE
レコード番号による	WRITE、EXCEPT、プライマリーおよびセカンダリー・ファイル
キーによる	
順次	WRITE、EXCEPT
FEOD	FEOD
UPDATE	
相対	UPDATE、プライマリーおよびセカンダリー・ファイル
レコード番号による	
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	DELETE、プライマリーおよびセカンダリー・ファイル
レコード番号による	
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

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 IBM i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
- IBM i 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、IBM i 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。

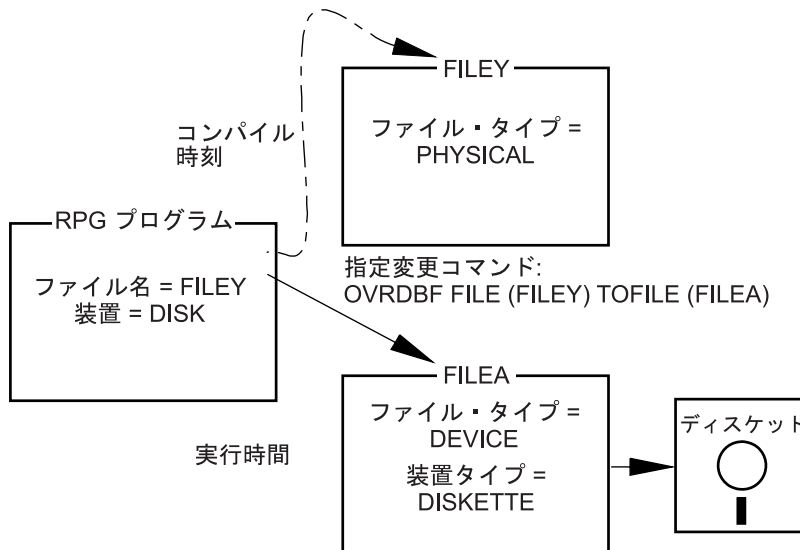


図 6-11. ファイル入出力の一時変更例

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。

```
# OVRDBF コマンドを SHARE(*YES) とともに使用し、プログラムの 1 つが DATA(*NOCVT) を有効にし
# てファイルを定義する場合、そのファイルを開くすべてのプログラムが DATA(*NOCVT) を使用してファ
# イルを定義する必要があります。別のデータ変換オプションを使用してファイルを開くと、実行時に
# CPF417C が表示されることがあります。
```

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。IBM i システムで

ファイル入出力の一時変更および指定変更

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、IBM i 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。

IBM i 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/systems/i/infocenter/> で IBM i 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 for i』の節を参照してください。

「ILE 概念」にも、一時変更に関する情報、および活動化グループとジョブ・レベルの有効範囲に関する情報があります。

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が、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。

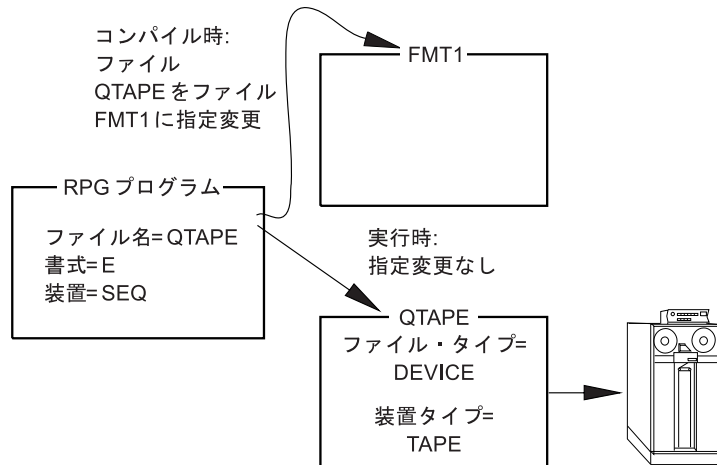


図 6-12. ファイル入出力の指定変更例

ファイルのロック

IBM i 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、Web サイト <http://www.ibm.com/systems/i/infocenter/> で **IBM i 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 for i*』の節を参照してください。

IBM i 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。6-17 ページの『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: サブプロシーチャーは、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、6-45 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。

他のプログラムまたはモジュールで使えるように、ファイルを共用している場合には、プログラムおよびモジュール間でファイルをパラメーターとして渡すことを考慮してください。 ファイル・パラメーターの引き渡しを参照してください。

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。

プログラムが順次入力操作を実行し、ファイルの終わり状態になる場合、通常の操作とは、同じモジュール内の任意の後続の順次入力操作で、データベースへの物理的入力要求なしに即時にファイルの終わり状態になることです。 ただし、ファイルが共有される場合は、RPG ランタイムは常時物理的入力要求をデータベースに送信し、共用ファイルを使用してファイルが別のプログラムまたはモジュールへの呼び出しで位置変更されている場合に、入力操作は成功します。

オープン・データ・パスの共用により、IBM i 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・パスの共用

- 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 6-3 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 6-3. 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・パスの共用、および活動化グループとジョブ・レベルの有効範囲の詳細については、資料「*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。IBM iは、入
出力スプーリング機能を提供します。各ファイル記述には、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/systems/i/infocenter/> で **IBM i 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 for i』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。

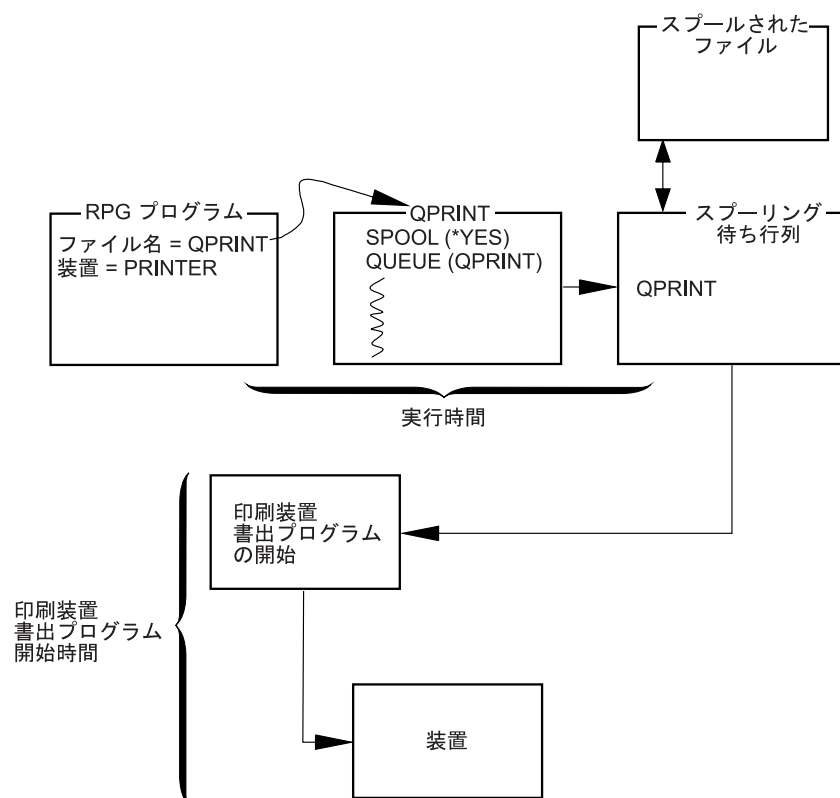


図 6-13. 出力スプーリング例

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、IBM i 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。ディスク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。

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、IBM i 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 IBM i 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詳細については、Web サイト <http://www.ibm.com/systems/i/infocenter/> で **IBM i Information Center** の中の『データベースおよびファイル・システム』カテゴリの『DB2 for i』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进行处理する場合には、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 IBM i 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/systems/i/infocenter/> で **IBM i Information Center** の中の『データベースおよびファイル・システム』カテゴリの『DB2 for i』の節を参照してください。

6-22 ページの図 6-14 は、データベース・ファイルの DDS の例を示し、6-23 ページの図 6-15 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、6-22 ページの図 6-14 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://www.ibm.com/systems/i/infocenter/> で **IBM i Information Center** の中の『データベースおよびファイル・システム』カテゴリの『DB2 for i』の節を参照してください。

外部記述ディスク・ファイルの使用

```
*.. 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                                  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 6-14. 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 6-23 ページの図 6-15 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          CITY            R          COLHDG('Customer Address')
A          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 6-15. 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、6-22 ページの図 6-14 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。

外部記述ディスク・ファイルの使用

- 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数とファイルまたはレコードのキーを比較し、検索引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。

有効な検索指数: 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、検索指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、検索指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指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、検索指数はフィールド A と同じ属性だけをもつ単一フィールドとしかありません。

注: ヌル可能キー・フィールドは、ALWNULL(*YES) または ALWNULL(*INPUTONLY) と共に使用できません。

レコード名に対する命令の場合には、検索指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。

検索指数が 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指数の場合には、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 を使用して検索指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指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、検索指数に対する %NULLIND() の現在の値が検索に使用されます。

検索指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。7-66 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 の検索指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、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指数を指定することもできます。このような検索指数は部分キーと呼ばれます。

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。

外部記述ディスク・ファイルの使用

```
DName+++++ETDsFrom+++To/L+++IDc.Keywords+++++
D keys          DS          LIKERE(rec : *KEY)
CL0N01Factor1+++++Opcod&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指数の指定規則は次のとおりです。

- 検索指数は、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指数のキー・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指数はフィールド A、およびフィールド A と B になります。
- 検索指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するリストまたは %KDS の検索指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、バックまたはゾーン) があります。
- 検索指数でキー・フィールドの一部分を参照することはできません。

検索指数で部分キーを参照した場合には、ファイルは検索指数を満たす最初のレコードに位置付けられるか、あるいは検索指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指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、検索指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、検索指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指数を満たす場合に、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指数を満たす場合に、その前のレコードが検索されます。上の命令コードの詳細については、「*IBM Rational Development Studio for i: 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、上記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 IBM i 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 (上記の 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の詳細については、「*IBM Rational Development Studio for i: 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。

プログラム記述ディスク・ファイルの使用

有効な検索指数: プログラム記述ファイルの場合には、検索指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、検索指数は、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、検索指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 6-16 および 6-29 ページの図 6-17 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 6-16. 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 6-17. (第 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 6-18. (第 2/2)。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

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索引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 命令の検索索引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 命令の検索索引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。

プログラム記述ディスク・ファイルの使用

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进行处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同じで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。IBM i 環境からのレコード・アドレス・ファイルを使用する場合は、各フィールドの長さは 4 バイトであるため、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 6-4 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 6-4. 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

ディスク・ファイルの処理方式

表 6-4. 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			
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	ブランク
レコード・アドレス・ ファイルとして (相対レコード番号)	ブランク	ブランク	T
レコード・アドレス限界値 ファイルとして		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、Web サイト <http://www.ibm.com/systems/i/infocenter/> で **IBM i**
- # **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 for i』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が、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、Web サイト <http://www.ibm.com/systems/i/infocenter/> で **IBM i Information Center** の中の『データベースおよびファイル・システム』カテゴリの『DB2 for i』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): 図 6-19 および 6-34 ページの図 6-20 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。6-34 ページの図 6-21 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 6-19. データベース・ファイル EMPMST (物理ファイル) の DDS

ディスク・ファイルの処理方式

```
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 6-20. 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 6-21. 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: YTD RPT1 *
* 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 0 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 6-22. 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。

ディスク・ファイルの処理方式

```

*****
* プログラム名:  YTD RPT2                                *
* 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      0   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、ファイル終わリルーチン进行处理
* 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
C              ENDIF                                12

* 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

OPRINT      H      1P                                2 6
O              40 'EMPLOYEE WEEKLY WORKING '
O              52 'HOURS REPORT'
O              H      01                                1
O              12 'EMPLOYEE: '
O              32
O              H      01                                1
O              12 'SERIAL #: '
O              17
O              ENUM                                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 6-23. キーによる順次処理、例 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: YTD RPT5 *
* 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 0 F 80 PRINTER
FEMPMST IP E K DISK
FTRWEEK IS E K DISK
IEMPREC 01
I ENUM M1
IRCWEEK 02
I ENUM M1
```

図 6-24. キーによる順次処理、例 3

ディスク・ファイルの処理方式

```

C 01          Z-ADD 0          TOTHR5 1
C 01          Z-ADD 0          TOTOV5 1
C 01          SETOFF          12
C*
C MR          IF      (*IN02='1')
C          ADD      EHWRK      TOTHR5
C EHWRK      SUB      ENHRS      OVTHR5 4 111
C 11          ADD      OVTHR5      TOTOV5
C          SETON          12
C          ENDIF
C OPRINT      H      1P          2 6
0          50 'YTD PAYROLL SUMMARY'
0          D      01          1
0          12 'EMPLOYEE: '
0          ENAME          32
0          D      01          1
0          12 'SERIAL #: '
0          ENUM          17
0          27 'DEPT: '
0          EDEPT          30
0          40 'TYPE: '
0          ETYPE          41
0          D      02 MR          1
0          8 'WEEK #'
0          WEEKNO          10
0          32 'HOURS WORKED = '
0          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 (TOTHR5 と TOTOV5) はゼロと等しくなります。
0          D      01NMR          1
0          70 'YTD HOURS WORKED = '
0          TOTHR5          3 78
0          D      01NMR          1
0          70 'YTD OVERTIME HOURS = '
0          TOTHR5          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 (TOTHR5 と TOTOV5)
* が印刷されます。

0          T      01 12          1
0          OR      LR 12
0          70 'YTD HOURS WORKED = '
0          TOTHR5          3 78
0          T      01 12          1
0          OR      LR 12
0          70 'YTD OVERTIME HOURS = '
0          TOTOV5          3 78

```

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指数を、CHAIN 命令の演算仕様書の演算項目 1 に指定します。6-40 ページの図 6-26 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ない、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。(6-27 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。6-33 ページの図 6-19 および図 6-25 は、EMSTUPD (6-40 ページの図 6-26) 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 6-25. 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。

ディスク・ファイルの処理方式

```
*****
* 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) が検索索引数として使用され、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つ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 6-26. 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: 図 6-27 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。

6-42 ページの図 6-29 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。

6-33 ページの図 6-19 には、プログラム ESWLIM1 (図 6-27) および ESWLIM2 (6-42 ページの図 6-29) 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 0 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
0 12 'SERIAL #'
0 22 'NAME'
0 45 'DEPT'
0 56 'TYPE'
0 D 01 1
0 ENUM 10
0 ENAME 35
0 EDEPT 45
0 ETYPE 55
```

図 6-27. 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): 6-42 ページの図 6-28 には、プログラム ESWLIM2 (6-42 ページの図 6-29) 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。

ディスク・ファイルの処理方式

```
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 6-28. 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   0   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
0           12 'SERIAL #'
0           22 'NAME'
0           45 'DEPT'
0           56 'TYPE'
0           D    01                1
0           ENUM                  10
0           ENAME                  35
0           EDEPT                  45
0           ETYPE                  55
0*
```

図 6-29. 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 6-5 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 6-44 ページの表 6-6 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一時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 6-5. 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
U	P/S			K/A/P/G/ D/T/Z/F	UPDATE、DELETE、CLOSE、FEOD、FORCE

有効なファイル命令

表 6-5. 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	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 IBM i CREATE FILE コマンドで 既に作られている必要があります。					

表 6-6. キーによらない処理方式で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
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

表 6-6. キーによらない処理方式で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	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 IBM i 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。

- IBM i 上では、

コミットメント制御の使用

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 IBM i 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) を使用します。これらのコマンドについての詳細は、Web サイト <http://www.ibm.com/systems/i/infocenter/> で **IBM i 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については、資料「バックアップおよび回復」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・レベルの詳細については、『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、6-49 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://www.ibm.com/systems/i/infocenter/> で **IBM i 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ができますが、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://www.ibm.com/systems/i/infocenter/> で **IBM i Information Center** 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。詳細については、資料「バックアップおよび回復」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の有効範囲の詳細については、「**ILE 概念**」で『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。

コミットメント制御の使用

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の機能の詳細な説明については、「*IBM Rational Development Studio for i: 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、IBM i 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で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 ...+... *
CLON01Factor1+++++Opcode(E)+Factor2+++++Result+++++Len++D+HiLoEq....
C
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 6-30. 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。

コミットメント制御の使用

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 6-31 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 ...+... *
CLON01Factor1+++++0opcode(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 6-31. 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。

キー付きファイルを使用した予期しない結果

同等のために READE、READPE、SETLL を使用するか、レコード・アドレス・ファイルによる制限内順次処理の場合、通常、キー比較はデータ管理機能レベルで実行されます。ただし、キー比較をデータ管理機能レベルで実行できない状態があります。データ管理機能がキー比較を実行できない場合、16 進数照合順序を使用して比較が実行されます。これは、予期しない結果を引き起こすことがあります。例えば、ABSVAL を数字キーに使うと、ファイルの中で 1 の値を持つキーに対して、-1 と 1 の両方の検索指数が一致してしまいます。16 進の照合順序を使うと、-1 の検索指数は 1 の値を持つ実際のキーには一致しません。

異なるキー比較を引き起こす機能の一部を以下に示します。

- 「複数の読み取り (Read Multiple)」に続く「次の等しいキーの取得 (Get Next Key Equal)」で、検索キーが提供される必要がありません。この状態を回避するには、「複数の読み取り」が 1 つのレコードのみを読み取るように、SEQONLY(*NO) または SEQONLY(*YES 1) のいずれかを指定して、OVRDBF コマンドを実行します。
- キー付きフィードバックは、ファイルに対して開いたときに要求されませんでした。
- 「読み取り」要求は、データの「グループ別」ビューを介して実行されました。この状態を回避するために、「グループ別」データの物理的コピーを使用します。
- ファイルは分散データ管理 (DDM) ファイルであり、リモート・ファイルは、バージョン 3 リリース 1 モディフィケーション 0 の前に作成されました。

データ管理機能で実行されるキー比較と異なる 16 進数キー比較を引き起こす機能の一部を以下に示します。

- ファイルに ALTSEQ が指定された
- 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
- ALWNULL(*USRCTL) が制御仕様上のキーワードまたはコマンド・パラメーターとして指定され、レコードまたは検索指数のキーが null 値を持っている。ファイルまたは検索指数のキーが null 値を持っている。これは、外部記述ファイルにのみ適用されます。
- ファイル用の SRTSEQ が 16 進数でない
- 番号記号がシステムの優先記号と異なる
- ファイル内のキーの CCSID がジョブの CCSID と異なる

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に関する詳細に必要な情報が提供されます。DDM

DDM ファイル

の使用および DDM ファイルの作成についての詳細は、Web サイト <http://www.ibm.com/systems/i/infocenter/> で **IBM i Information Center** の中の『データベースおよびファイル・システム』カテゴリの『DB2 for i』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 つ以上の検索指数が一致してしまう **DDS** 機能を使うと、予想と異なる結果になることがあります。例えば、**ABSVAL** を数字キーに使うと、ファイルの中で 1 の値を持つキーに対して、-1 と 1 の両方の検索指数が一致してしまいます。16 進数の照合順序を使うと、-1 の検索指数は 1 の値を持つ実際のキーには一致しません。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指数のキーに **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。

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、ディスク装置、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、6-66 ページの『**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 6-7 にリストされた装置ファイルを使用します。

表 6-7. IBM i 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
ディスク・ファイル	ディスク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、IBM i 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 ファイルに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。外部記述ファイルで有効な出力仕様書項目については、「*IBM Rational Development Studio for i: 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。

プリンターのアクセス

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/systems/i/infocenter/> で **IBM i Information Center** の中の『プログラミング』カテゴリの『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 **Information Center** の中の『データベースおよびファイル・システム』カテゴリの『DB2 for i』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詳細な説明については、「*IBM Rational Development Studio for i: ILE RPG 解説書*」を参照してください。

ページ・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、ページ・オーバーフローです。外部記述 PRINTER ファイルの場合には、ページ・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。
- 行番号およびページ・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。詳細については、「*IBM Rational Development Studio for i: 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 INFSD 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、5-57 ページの『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

6-56 ページの表 6-8 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ものでなければ、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。6-57 ページの図 6-33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ため、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。

プリンターのアクセス

- 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 6-8. 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 6-32 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
0.....N01N02N03Field++++++YB.End++PConstant/editword/DTformat++
0 OR L2
0 8 'DATE'
0 18 'ACCOUNT'
0 28 'N A M E'
0 46 'BALANCE'
0*

図 6-32. 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: 6-57 ページの図 6-33 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
0           OR      L2
0.....N01N02N03Field+++++++YB.End++PConstant/editword/DTformat++
0                  ACCT                  8
0*
```

図 6-33. 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。

6-58 ページの図 6-34 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。

プリンターのアクセス

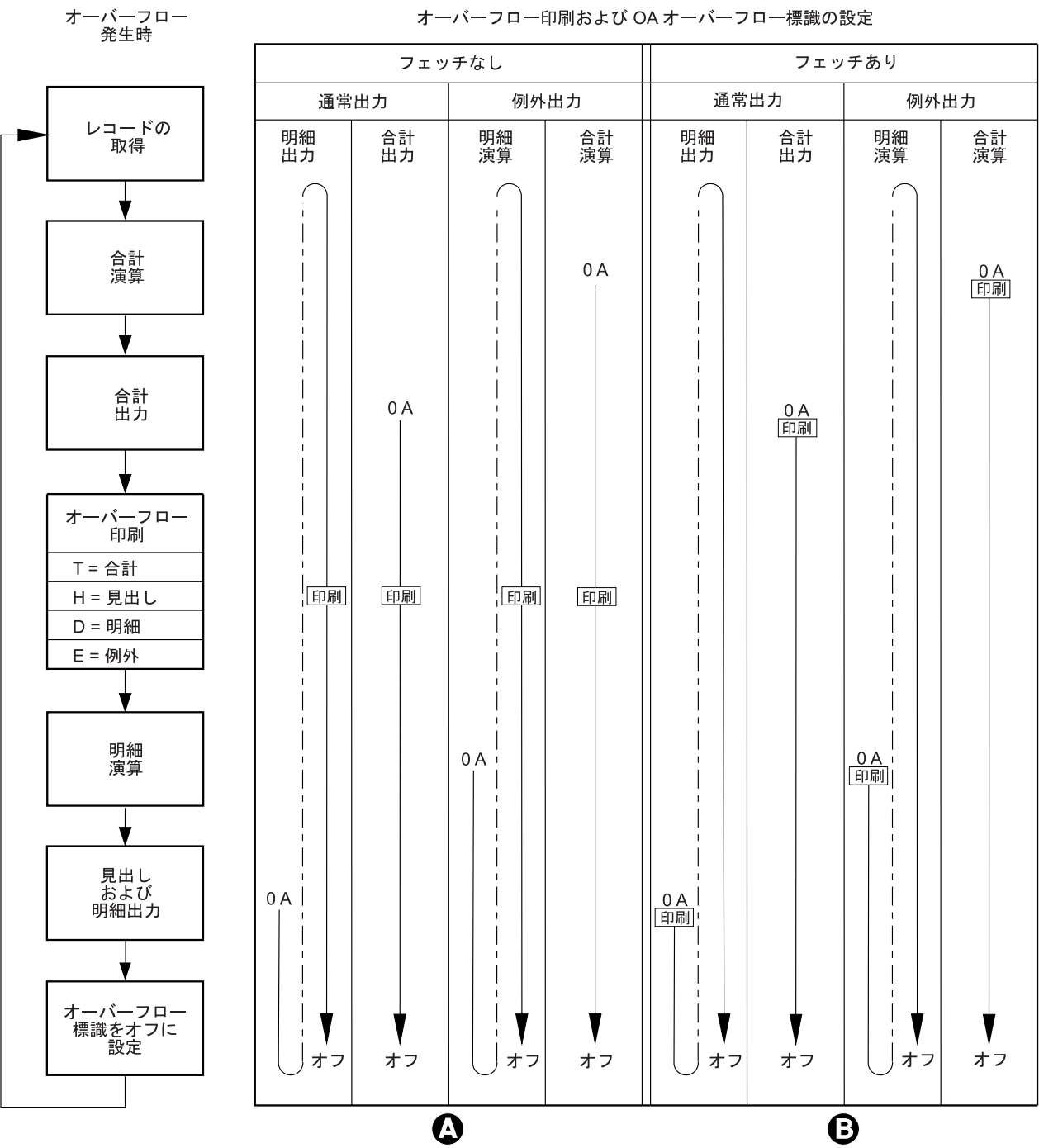


図 6-34. オーバーフロー印刷: オーバーフロー標識の設定

- 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わ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た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: 図 6-35 は、フェッチ・オーバーフローの使い方を示します。

*..	1	...	2	...	3	...	4	...	5	...	6	...	7	...	*
O	Filename++	DF..	N01N02N03	Excnam+++	B++	A++	Sb+Sa+							
O	PRINTER	H	OA						3	05					
O	N01N02N03	Field+++++	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 6-35. 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 6-9 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 表 6-10 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 6-9. 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)
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 6-10. 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: 図 6-36 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      0      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 6-36. 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。

テープ装置のアクセス

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、6-66 ページの『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、またはディスク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。6-63 ページの図 6-37 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、IBM i 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 6-11. 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： 6-63 ページの図 6-37 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 0   F  132     SEQ
*
```

図 6-37. 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、IBM i 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、またはディスクなど）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クを指定する一方、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。 |

SPECIAL ファイルの使用

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 ファイルに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。 6-65 ページの図 6-38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 6-12 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 6-12. 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 6-38 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    0    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                      MOVE      'HELLO'  FLD
C                      MOVE      '1'      FLD1          1
C                      WRITE     EXCPTN   OUTBUF
C                      MOVE      '2'      FLD1          1
C                      WRITE     EXCPTN   OUTBUF
C                      SETON
LR
```

図 6-38. SPECIAL 装置

図 6-39 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 6-39. ユーザー作成プログラム USERIO

SPECIAL ファイルの使用

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。

WORKSTN ファイルの使用

IBM i 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、IBM i 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、IBM i 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 (IBM i またはその他のオペレーティング・システム) 上の他のアプリケーション・プログラムとの通信 (データの送信およびデータの受信) を行います。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 IBM i 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 の詳細については、資料「*ICF プログラミング*」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/systems/i/infocenter/> で **IBM i Information Center** の中の『データベースおよびファイル・システム』カテゴリの『DB2 for i』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力バッファではなく、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/systems/i/infocenter/> で **IBM i Information Center** の中の『データベースおよびファイル・システム』カテゴリの『DB2 for i』の節を参照してください。

外部記述 WORKSTN ファイルの使用

図 6-40 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
A          R PROMPT          REF(DSTREF) 1
A 73N61          TEXT('Item Prompt Format')
A          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 6-40. 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、IBM i 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、IBM i 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、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、IBM i 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。

IBM i 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、IBM i 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、IBM i 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 IBM i システムに渡します。IBM i 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。

外部記述 WORKSTN ファイルの使用

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、6-76 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。(6-71 ページの図 6-41 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。6-72 ページの図 6-42 は、サブファイル・レコード様式の DDS の例を、6-73 ページの図 6-43 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/systems/i/infocenter/> で **IBM i 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 for i』の節を参照してください。

[illegible]

図 6-41. 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。

SFFILE 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/systems/i/infocenter/> で **IBM i Information Center** の中の『データベースおよびファイル・システム』カテゴリの『DB2 for i』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。

外部記述 WORKSTN ファイルの使用

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 6-42. サブファイル・レコード様式の詳細仕様書

サブファイル・レコード様式の詳細仕様書 (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れ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詳細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、6-93 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 6-43. 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 ファイルの使用

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。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。 6-76 ページの表 6-13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 IBM i 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、IBM i 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 IBM i 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/systems/i/infocenter/> で **IBM i 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 for i』の節を参照してください。

有効な WORKSTN ファイル命令

表 6-13 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 6-13. 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
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の詳細については、「IBM Rational Development Studio for i: 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/systems/i/infocenter/> で **IBM i Information Center** 中の『データベースおよびファイル・システム』カテゴリの『DB2 for i』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、資料「*ICF プログラミング*」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。資料「*ICF プログラミング*」にある、プログラム装置の送信勧誘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。資料「*ICF プログラミング*」にある、送信勧誘プログラム装置からの読み取りに関する節、および 1 プログラム装置からの読み取りに関する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 命令コードの詳細については、「*IBM Rational Development Studio for i: 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ことを示します。

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

複数装置ファイル

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。

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チ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 6-14. 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
6-79 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。

表 6-14. 対話式アプリケーション例の各モジュールの説明 (続き)

モジュール	説明
6-82 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。
6-93 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。
6-101 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

図 6-44 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 6-44. 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 (DPSIZ)、デフォルトの入力値 (CHGINPDT)、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。

メイン・メニュー照会

```
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 6-45. 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 hhdrscn;

// 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データの検索
SearchZip();

elseif srchCustKey;
// 得意先名による得意先データの検索
SearchName();
endif;
enddo;

*inlr = *on;
/end-free

```

図 6-46. 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。

メイン・メニュー照会

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。

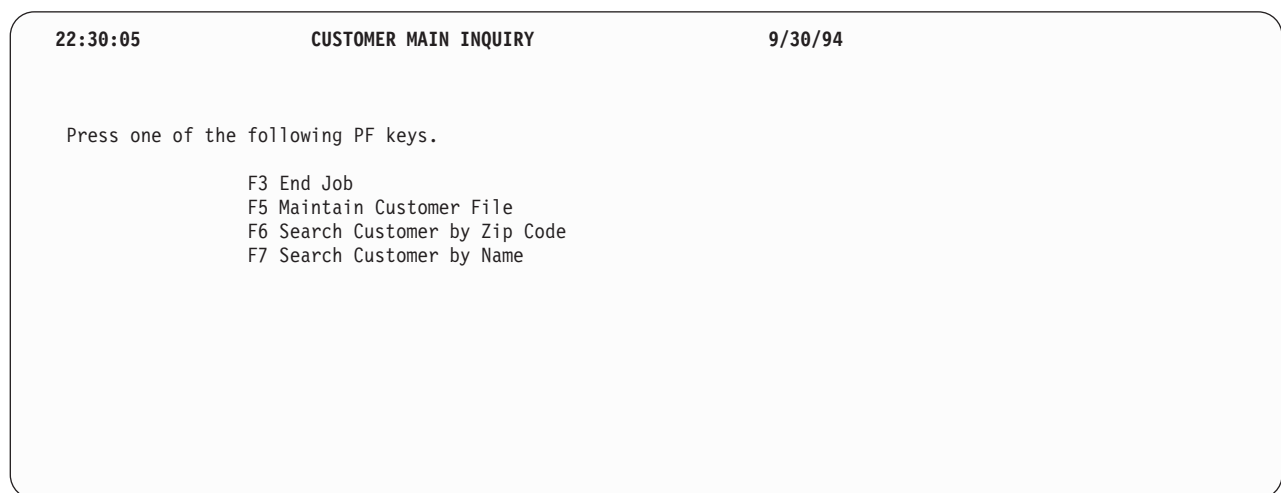
```
CRTPGM PGM(MYPROG) MODULE(CUSMAIN CUSMNT SCHZIP SCHNAM) ENTMOD(*FIRST)
```

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 6-47 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 6-47. 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 6-48. 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(QSYSPT)
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  0  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 6-49. 表示装置ファイル MNTMENU の DDS

ファイルおよび装置の処理 **6-85**

ファイルの維持

(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 6-50. モジュール CUSMNT のソース


```

//*****
//   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;

```

ファイルの維持

```
//*****
// 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(cusmst11);
// 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 *inl2;
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(cusmst11);
// 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 cusmst11;
endif;
endsr; // サブルーチン UpdateSub の終わり
```

```

//*****
// 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

```

ファイルの維持

```
//*****
//   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
```

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ファイル照会画面 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。

6-91 ページの図 6-51 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。

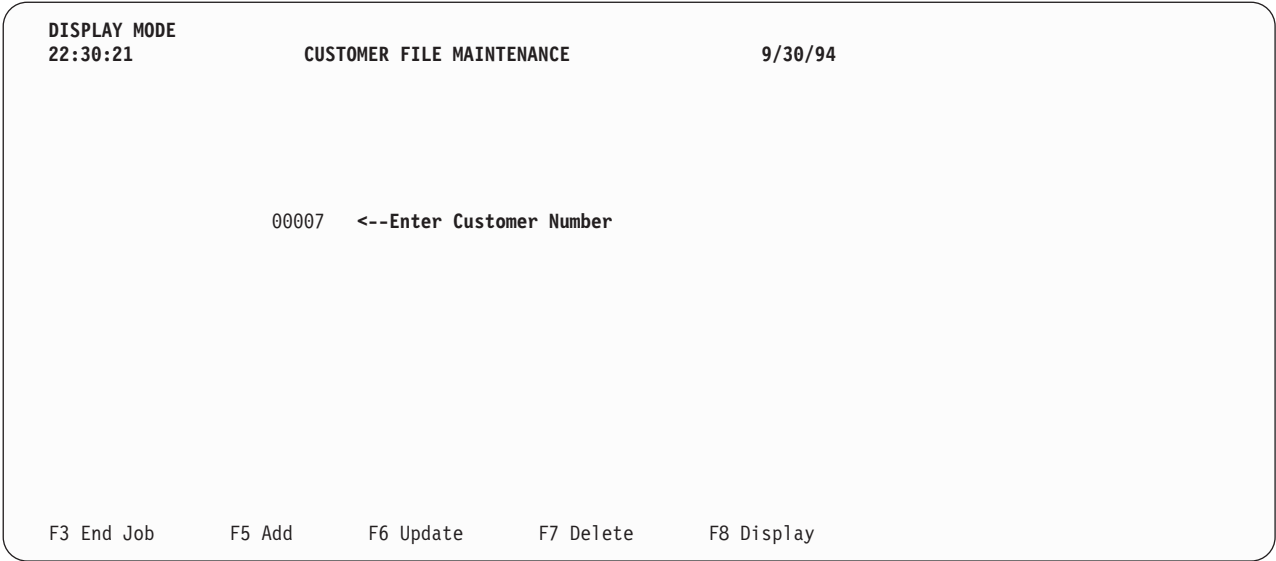


図 6-51. 'CUSTOMER FILE MAINTENANCE' (得意先ファイル維持) 表示モード・プロンプト画面

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 6-52 に示すようにデータが表示されます。

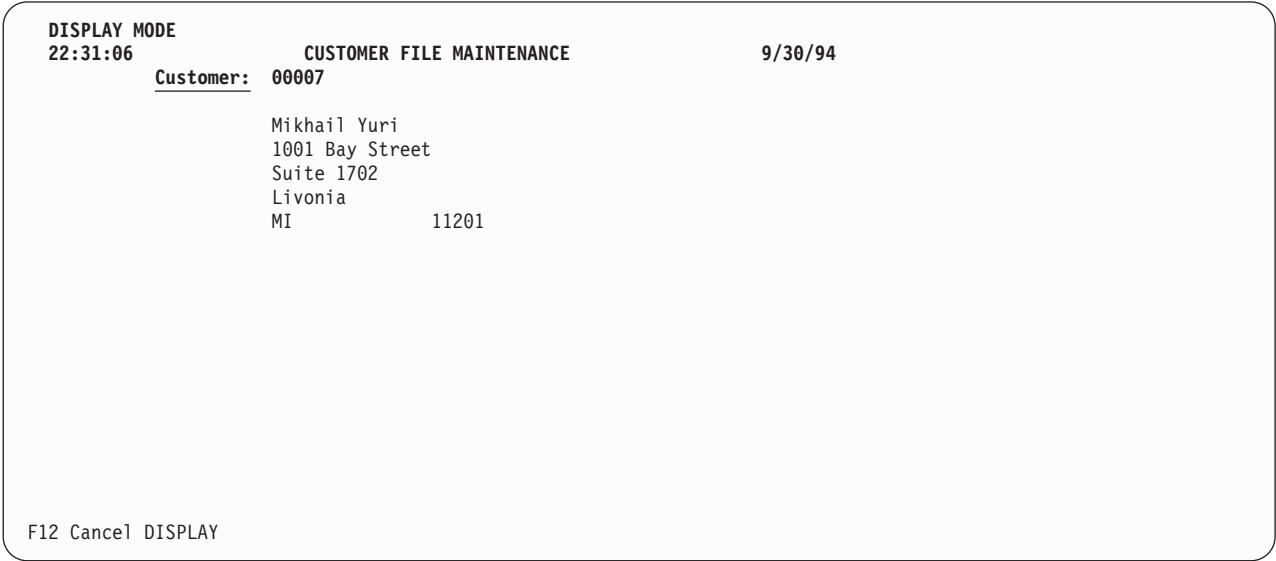


図 6-52. 'CUSTOMER FILE MAINTENANCE' (得意先ファイル維持) 表示モード画面

ワークステーション・ユーザーは、6-92 ページの図 6-53 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。

ファイルの維持

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 6-53. '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 6-54 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
<u>Customer:</u> 00012		
Name JUDAH GOULD		
Address 2074 BATHURST AVENUE		
City YORKTOWN		
State NY Zip 70068		
F12 Cancel Addition		

図 6-54. '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、6-93 ページの図 6-55 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 6-55. '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 6-56 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 6-56. '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 6-57. 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。

SCHZIPD: 表示装置ファイルの DDS

```

# A*****
# A* FILE NAME: SCHZIPD *
# A* 関連プログラム: SCHZIP *
# A* 関連ファイル: CUSMSTL2 (論理ファイル) *
# A* DESCRIPTION: THIS IS THE DISPLAY FILE SCHZIPD. IT HAS 6 *
# 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
# A NAME R 9 4
# A ARBAL R 9 27EDTCDE(J)
# A R SUBCTL
# A SFLCTL(SUBFILE)
# 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 0 4 14DSPATR(HI)
# A 7 4'Customer Name'
# A DSPATR(HI UL)
# A 7 27'A/R Balance'
# A DSPATR(HI UL)

```

図 6-58. 表示装置ファイル SCHZIPD の DDS

郵便番号による検索

SCHZIPD 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同時に、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、IBM i 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

```

# //*****
# // PROGRAM NAME:      SCHZIP                      *
# // 関連ファイル:    CUSMSTL2 (論理ファイル)        *
# //                  SCHZIPD (WORKSTN FILE)          *
# //      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 *
# //              同じ郵便番号で検索を再実行する。 *
# //              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。 *
# //*****
# Fcsmstl2  if  e          k disk
# Fschzipd  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 6-59. モジュール SCHZIP のソース

郵便番号による検索

```
//*****
//   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(cusmst12);
  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;
```



```

//*****
// サブルーチン - ProcessSubfile *
// 目的 - サブファイル进行处理し、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(cusmst12);
// 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(schzipd);
// 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(cusmst12);
// 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

```

郵便番号による検索

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 (SCHZIPD) 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 キーを押してその郵便番号による検索を再開する。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。

6-101 ページの図 6-60 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。

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 6-62. 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。

SCHNAMD: 表示装置ファイルの DDS

```
# A*****
# A*      FILE NAME:  SCHNAMD                      *
# A*関連プログラム:  SCHNAM                      *
# A* 関連ファイル :  CUSMSTL3   (論理ファイル)    *
# A*  DESCRIPTION:  THIS IS THE DISPLAY FILE SCHNAMD. IT HAS 7  *
# 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 6-63. 表示装置ファイル SCHNAMD の DDS

名前による検索と照会

```

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          0  9 54
A          CUST          R          0  9 43
A          NAME          R          0  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          0  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 00  6 35DSPATR(HI)
A          8 25'Name'
A          NAME          20A  0  8 35DSPATR(HI)
A          10 25'Address'
A          ADDR1          20A  0 10 35DSPATR(HI)
A          ADDR2          20A  0 11 35DSPATR(HI)
A          13 25'City'
A          CITY          20A  0 13 35DSPATR(HI)
A          15 25'State'
A          STATE          2A  0 15 35DSPATR(HI)
A          15 41'Zip Code'
A          ZIP          5S 00 15 50DSPATR(HI)
A          17 25'A/R Balance'
A          ARBAL          10Y 20 17 42DSPATR(HI)
A          EDTCDE(J)

```

SCHNAMD 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同時に、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、IBM i 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) *
# // SCHNAMD (WORKSTN FILE) *
# // 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、set11 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
# Fschnamd 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 namekey klist
# C kfld name
```

図 6-64. モジュール SCHNAM のソース

```

//*****
//   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、ループを出す
    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(cusmst13);
            // 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 subct1;
    enddo;

endsr; // サブルーチン ProcessSubfile の終わり

```

```

//*****
//   サブルーチン - FillSubfile                                     *
//   目的         - サブファイルを充てんする                         *
//*****
begsr FillSubfile;

    // Loop through all customer records
    recnum = 0;
    dou %eof(schnamd);
        // Read next record
        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

```

```

//*****
// 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(schnamd);
// 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 namekey 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

```

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 (SCHNAMD) 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。

名前による検索と照会

- 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 6-65 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 6-65. 'CUSTOMER SEARCH and 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

図 6-66 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 6-66. 'CUSTOMER SEARCH and 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、 6-111 ページの図 6-67 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	<u>CUSTOMER SEARCH & INQUIRY BY NAME</u>	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 6-67. 'CUSTOMER SEARCH and INQUIRY BY NAME' (名前による得意先検索および照会) 詳細情報画面

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。サポートされていない RPG III 機能を参照してください。

OPM RPG/400 と ILE RPG との相違点

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信する場合には、スタック内にプログラム入力プロシージャが存在できるように、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ことがあります。これは、ILE プロシージャがユーザー入りロプロシージャである場合、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。これはサイクル・メイン・プロシージャにだけ適用されることに注意してください。サブプロシージャに対して再帰は使用可能です。
- 7.

ILE RPG モジュールのサイクル・メイン・プロシージャが、RPG サイクルの *INLR を検査する部分に達しないうちに、プログラム・スタックから取り消されると、*TERM 処理が実行されます。

OPM RPG プログラムが、RPG サイクルの *INLR を検査する部分に達しないうちに、プログラム・スタックから取り消されると、*TERM 処理は実行されません。

*TERM 処理では以下のことが行われます。

- ・ オープンされたグローバル・ファイルがクローズされます。
- ・ プログラムがロックしたデータ域が解除されます。
- ・ 次回の呼び出しに備えてプログラム変数が更新されるようにプログラムまたはモジュールが設定されます。

ILE RPG サイクル・メイン・プロシージャが取り消されたとき *INLR がオンであった場合、*INLR はこのプロシージャの次回の呼び出しに備えてオンになることはなく、RPG サイクルは通常 *INIT から始まります。

OPM RPG プログラムが取り消されたとき *INLR がオンであった場合でも、*INLR はこのプログラムの次回の呼び出しに備えてオンになり、RPG サイクルは *DETC 部分を実行しないで *TERM に進みます。

8. ゴース 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。

9. 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、5-89 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。

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。

OPM RPG/400 と ILE RPG との相違点

2. プログラムが順次入力操作を実行し、ファイルの終わり状態になる場合、通常の操作とは、同じモジュール内の任意の後続の順次入力操作で、データベースへの物理的入力要求なしに即時にファイルの終わり状態になることです。ただし、ファイルが共有される場合は、RPG ランタイムは常時物理的入力要求をデータベースに送信し、共用ファイルを使用してファイルが別のプログラムまたはモジュールへの呼び出しで位置変更されている場合に、入力操作は成功します。
3. 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。)
4. 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。

5. 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。
6. 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。
7. 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 III 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。

8. 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。
9. 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。
10. OPM RPG/400 で参照制約エラーが起これると、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。
11. 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。
 12. 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。
 13. 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 つ以上の検索指数が一致してしまう DDS 機能を使うと、予想と異なる結果になることがあります。

例えば、ABSVAL を数字キーに使うと、ファイルの中で 1 の値を持つキーに対して、-1 と 1 の両方の検索指数が一致してしまいます。16 進の照合順序を使うと、-1 の検索指数は 1 の値を持つ実際のキーには一致しません。

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。詳しくは、6-52 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。
 14. 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。
 15. 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。
 16. RPG/400 コンパイラーは DELET 命令を出力命令と見なします。更新可能レコード様式に DELET 命令と CLEAR または RESET 命令が含まれていて、UPDAT 命令が含まれていないと、RPG/400 コンパイラーはレコード様式のフィールドをクリアまたはリセットしますが、ILE RPG コンパイラーはレコード様式のフィールドをクリアもリセットもしません。ILE RPG コンパイラーにフィールドをクリアまたはリセットさせる場合は、当該命令の演算項目 2 に *ALL を指定するか、UPDATE 命令をプログラムに追加することができます。

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 データの初めを示します。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せん。

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く、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、System/38™ 環境 RPG III のレイアウトおよび 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 RPG/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、7-9 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、7-25 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 7-1 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 7-1. 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

変換の概要

表 7-1. ソース・メンバー・タイプとその変換状況 (続き)

ソース・メンバー・タイプ	変換?	変換後のメンバー・タイプ
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、 7-17 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 図 7-1 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・リストの 1 行に収まる情報の最大量と対応しています。

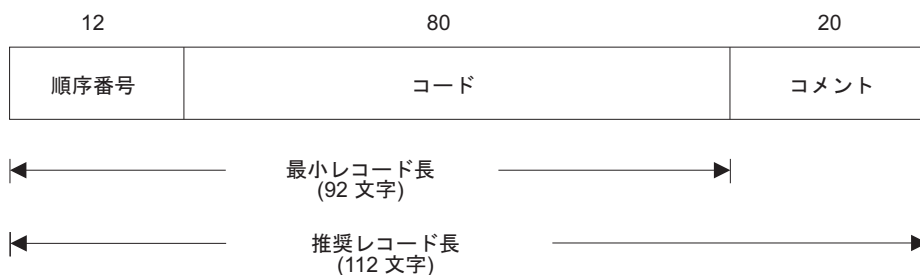


図 7-1. RPG IV レコード長の説明

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 QRPGLSRC 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 QRPGLSRC 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 QRPGL 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。7-25 ページの図 7-8 は、このファイルの DDS を示しています。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(QRPGL) 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、7-24 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 から RPG III への変換援助プログラム** を使用し、その後で RPG III から RPG IV への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。

ソースの変換

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。詳細については、「*IBM Rational Development Studio for i: 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、7-9 ページの『ログ・ファイル』および 7-24 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、7-7 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。

詳しくは、『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、7-21 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、4-10 ページの『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・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、7-25 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。7-11 ページの『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』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。

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

表 7-2. ファイル識別

パラメーター	説明
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。

表 7-3. 変換処理

パラメーター	説明
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。

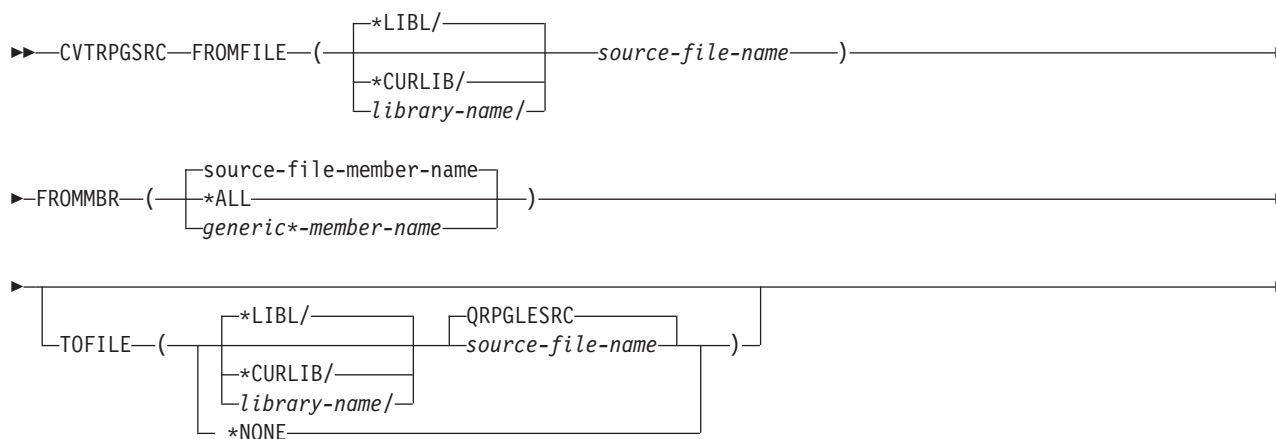
表 7-4. 変換フィードバック

パラメーター	説明
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。

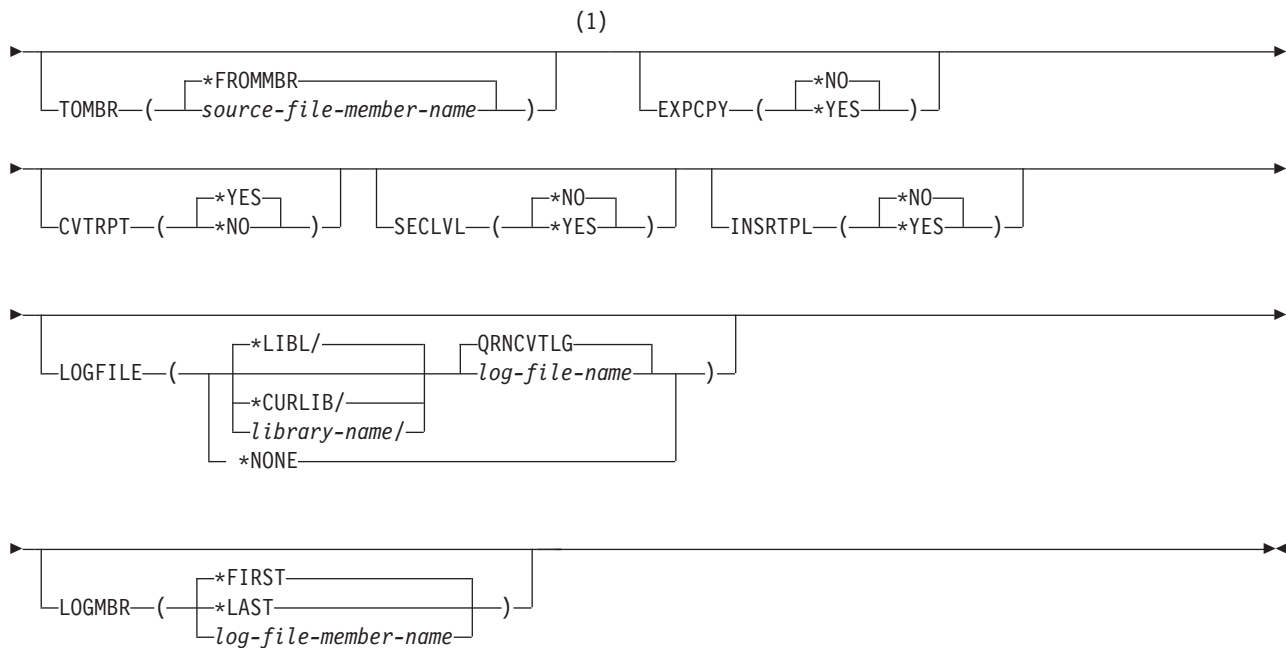
CVTRPGSRC コマンド構文

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



ソースの変換



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。

library-name

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ない、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。

library-name

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。

ソースの変換

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 コマンドに QRPGLR ライブラリーの QRNCVTLG ファイルの「From オブジェクト」、およびユーザーのライブラリーの QRNCVTLG ファイルの「New オブジェクト」を指定してログ・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。

log-file-name

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。

library-name

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。

log-file-member-name

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 QRPGLSRC 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 QRPGLSRC および QRNCVTLG は既に存在していなければなりません。

ソースの変換

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 QRPGRSRC 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 QRPGLSRC に入れたい場合には、次のように入力します。

```
CVTRPGSRC  FROMFILE(OLDRPG/QRPGRSRC)
            FROMMBR(*ALL)
            TOFILE(NEWPRG/QRPGLSRC)
            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 QRPGRSRC 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 NEWPRG のソース物理ファイル QRPGLSRC 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 (7-7 ページの表 7-1 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る、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/QRPGRSRC)
            FROMMBR(PAY*)
            TOFILE(NEWPRG/QRPGLSRC)
            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 QRPGRSRC 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 NEWPRG のソース物理ファイル QRPGLSRC 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/QRPGRSRC)
            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、7-21 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。

ソースの変換

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。(7-8 ページの図 7-1 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

次の例は、RPG IV に変換される RPG III ソース・メンバーの例を示しています。7-19 ページの図 7-2 は、RPG III バージョンのソースを示しています。


```
H
FFILE1  IF  E          DISK
FQSYSPRT 0   F      132    OF  LPRINTER
LQSYSPRT 60FL 560L
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
I          4   6 ARR1          COMM3
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 7-2. 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 7-20 ページの図 7-3 に示します。

ソース変換の例

```
1 .....H*unctions+++++Comments+++++
2      H DFTNAME(TSTPGM)
3 .....F*ilename++IPEASFRlen+LKlen+AIDevice+.Functions+++++Comments+++++
4      FFILE1      IF      E      DISK      COMM1
5      FQSYSVRT      0      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      FORMAT1      01
23     C      READ
24     C      NAME      DSPLY
25     C      SETON      LR
26     C      EXCEPT      OUTPUT
27     OQSYSVRT      E      OUTPUT      01
28     O      ARR2(3)      10
29 **CTDATA ARR1
30 123
31 **CTDATA ARR2
32 456
```

図 7-3. 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作成します。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。

変換の分析

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 7-4 は、変換例のコマンドの要約を示します。

5769WDS V5R2M0 020719 RN	IBM ILE RPG	ISERIES1	08/15/02 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		
FROM メンバー	:	REPORT		
To ファイル	:	QRPGLESRC		
ライブラリー	:	MYLIB		
To 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 7-4. 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。 7-23 ページの図 7-5 は、サンプル変換のソース・セクションを示しています。

```

5769WDS V5R2M0 020719 RN          IBM ILE RPG          ISERIES1    08/15/02 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 <----- ソースの仕様 ----->----- 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 7-5. 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 7-6 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。

```

5769WDS V5R2M0 020719 RN          IBM ILE RPG          ISERIES1    08/15/02 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・
          included in this source member have also been converted to
          RPG IV.
          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
          operation codes have been identified because the user may
          wish to:
          -- 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 7-6. 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。 7-24 ページの図 7-7 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。

変換の分析

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:		
元のレコードの読み取り数	3
変換済みレコードの書き出し数	4
出された最高重大度メッセージ	30
* * * * * 最終の要約の終わり * * * * *		
* * * * * 変換の終わり * * * * *		

図 7-7. 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。重大エラーが見つかった場合には、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 QRPGL 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。7-25 ページの図 7-8 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(QRPGL/QARNCVTLG) TOFILE(MYLIB/QRNCVTLG)
      CRTFILE(*YES)
```


A	R QARNCVTFM		
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	LGTOTFL	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 7-8. ライブラリー QRPGL 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 コンパイラ指示ステートメントが RPG III ソースに存在している

変換上の問題の分析解決

- 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、7-17 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、7-1 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。

FREE 命令コードの変換: FREE 命令の関数を置き換えるには、まず、FREE 命令が使用されていた理由を判別する必要があります。

- プログラムがそのプログラムに対する次の呼び出しで初期化されるようにするために FREE 命令が使用されていた場合、呼び出されるプログラムが特殊パラメーターで (またはパラメーターなしで) 呼び出され、単に LR 上に設定されて、戻されることを示すように、そのプログラムを変更します。次に、FREE 命令をコーディングする代わりに、特殊な "free" パラメーターでプログラムを呼び出します。

```
* RPG III coding
C          CALL 'MYPGM'
C          PARM          P1
...
C          FREE 'MYPGM'
...
C          CALL 'MYPGM'
C          PARM          P1

* Replacement RPG IV coding for the "reresolve" function of FREE

C          call          MYPGM_var
C          parm          p1
...
* Cause MYPGM to initialize on the next call
C          call          MYPGM_VAR
```

```

...
C          call      MYPGM_var
C          parm      p1

* Modified version of MYPGM.  It ends itself when it is called with no parameters.
D          SDS
D PARMS          *PARMS
C  *ENTRY      PLIST
C          PARM      NAME      10
C  PARMS      IFEQ      0
C          SETON      LR
C          RETURN
C          ENDIF
...

```

- **FREE** 命令が含まれるプログラムが、そのプログラムの次の呼び出しで再びそのプログラムに解決されるように **FREE** 命令が使用されていた場合、文字変数を使用して呼び出しを行うように、呼び出し側プログラムを変更できます。呼び出されたプログラムが再び解決されるようにするには、文字変数を使用して、**FREE** を行う場所で異なるプログラムを呼び出す必要があります。これにより、次の **CALL** 命令で文字変数を使用するときに、システムは再度プログラムに対して解決を実行します。"FREE" 関数に対して呼び出される非常に高速実行のプログラムを作成します (単純に **RETURN** 命令を持つ **ILE RPG** プログラムなど)。

```

* RPG III coding
C          CALL 'MYPGM'
C          PARM      P1
...
C          FREE 'MYPGM'
...
C          CALL 'MYPGM'
C          PARM      P1

* Replacement RPG IV coding for the "reresolve" function of FREE

D MYPGM_var      s          21a      INZ('MYPGM')
C          call      MYPGM_var
C          parm      p1
...
* Cause a reresolve to MYPGM for the next call
C          eval      MYPGM_var = 'MYLIB/FREEPGM'
C          call      MYPGM_VAR
C          reset      MYPGM_var
...
C          call      MYPGM_var
C          parm      p1

```

DEBUG 命令の関数を置換するには、対話式デバッガーを使用します。プログラム・デバッグの詳細については、5-1 ページの『プログラムのデバッグ』を参照してください。

/COPY コンパイラ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、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 7-18 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 7-19 ページの図 7-2 と 7-20 ページの図 7-3 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。

変換上の問題の分析解決

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 7-9 と 図 7-10 は、それぞれ TEST2 と COPYDS1 のソースを示します。

```
H                                     TSTPGM
FFILE1  IF  E                                     COMM1
FQSYSPRT 0  F      132      OF  LPRINTER
LQSYSPRT 60FL 560L
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 7-9. 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 7-10. 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. ILE RPG コンパイラー・リストや「IBM Rational Development Studio for i: ILE RPG 解説書」の情報をを使用して、変換後にコードを手動で修正する。

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、ファイル記述仕様書のキーワード (RAFDATA、FORMLEN、および FORMOFL) に変更されます。 /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、対応するファイル記述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

7-18 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 (7-20 ページの図 7-3) 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。

- 組み合わせら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 (7-20 ページの図 7-3) 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せられた場合には、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せられた場合には、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 7-11. 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 7-12. 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 7-29 ページの図 7-11 のソースを図 7-13 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 7-13. 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引き起こすため、ILE RPG コンパイラー・リストや「*IBM Rational Development Studio for i: ILE RPG 解説書*」の情報を使用してこれらのエラーを修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、7-31 ページの図 7-14 のフィールド ARRAY は 7-31 ページの図 7-15 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 7-31 ページの図 7-16 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、7-31 ページの

ジの図 7-17 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 7-14. 外部データ構造の DDS

E		ARRAY	10	1
IDSONE	E	DSEXTREC		
C		CHAR	DSPLY	
C		SETON		LR

図 7-15. 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 7-16. 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 7-17. 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 7-18 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。7-32 ページの図 7-19 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、7-32 ページの図 7-20 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る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 が組み合わせられたので、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 7-18. 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 7-19. 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 7-20. 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。詳細については、3-17 ページの『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

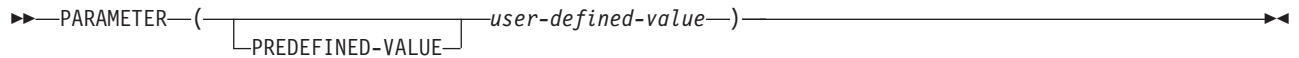
プログラム作成コマンドおよびサービス・プログラム作成コマンドについての詳細は、Web サイト <http://www.ibm.com/systems/i/infocenter/> で **IBM i Information Center** の中の『プログラミング』カテゴリの『CL および API』の節を参照してください。

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。

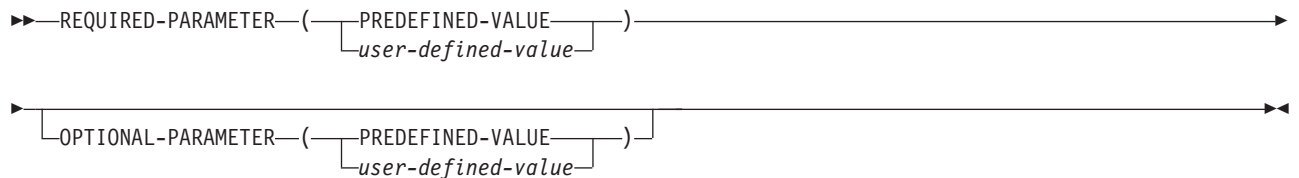
—▶ 記号は、構文図の終わりを示します。

—▶ 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。

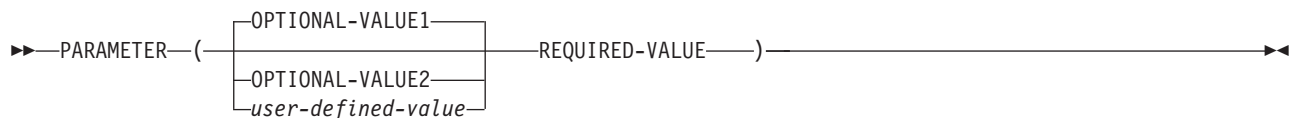
必須パラメーターは基準線に表示され、必ず入力しなければなりません。**任意指定パラメーター**は基準線の下に表示され、必ず入力する必要はありません。次のサンプルでは、**REQUIRED-PARAMETER** およびその値を入力しなければなりません、**OPTIONAL-PARAMETER** またはその値は入力の必要がありません。



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。

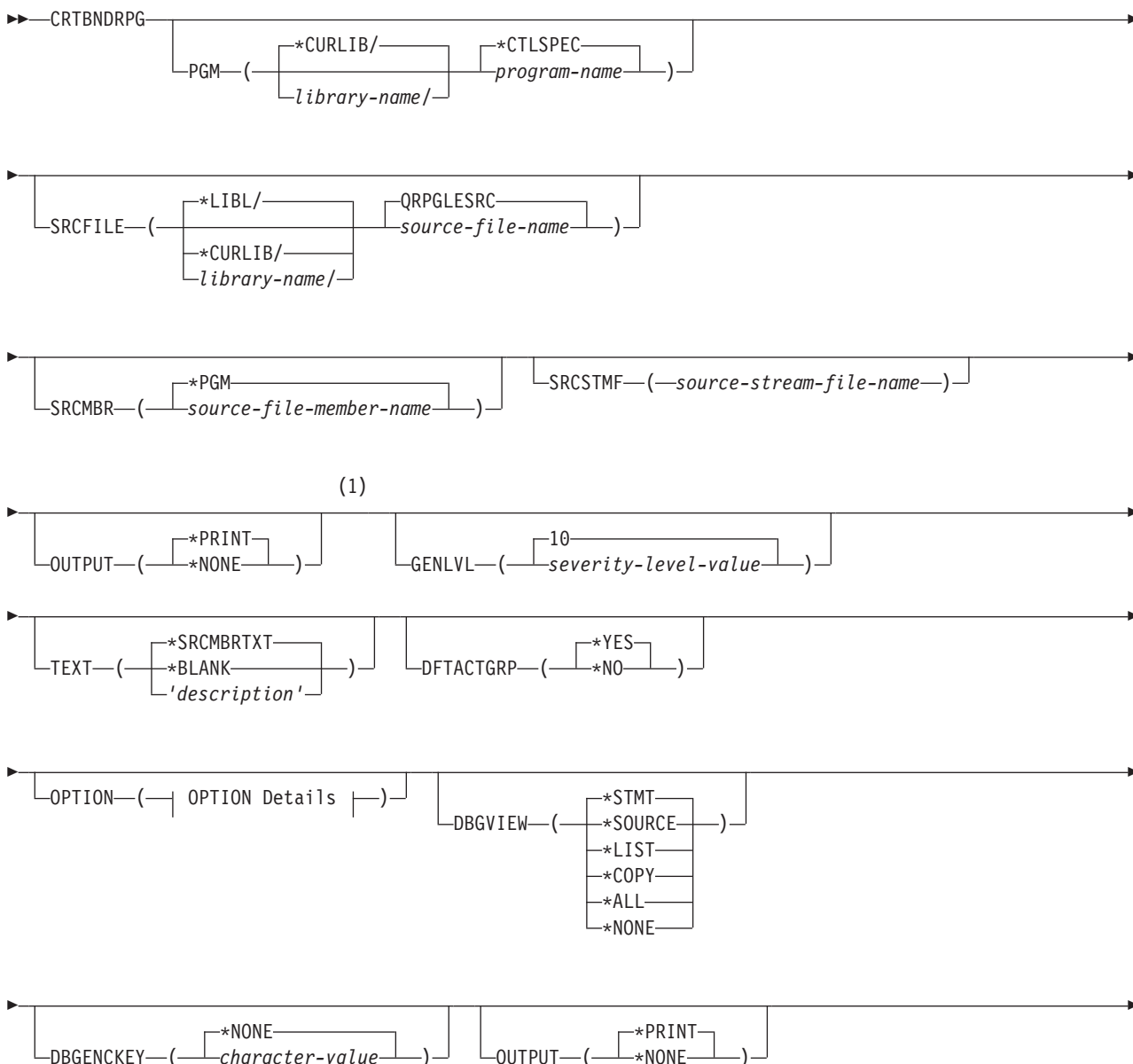
構文図の読み取り

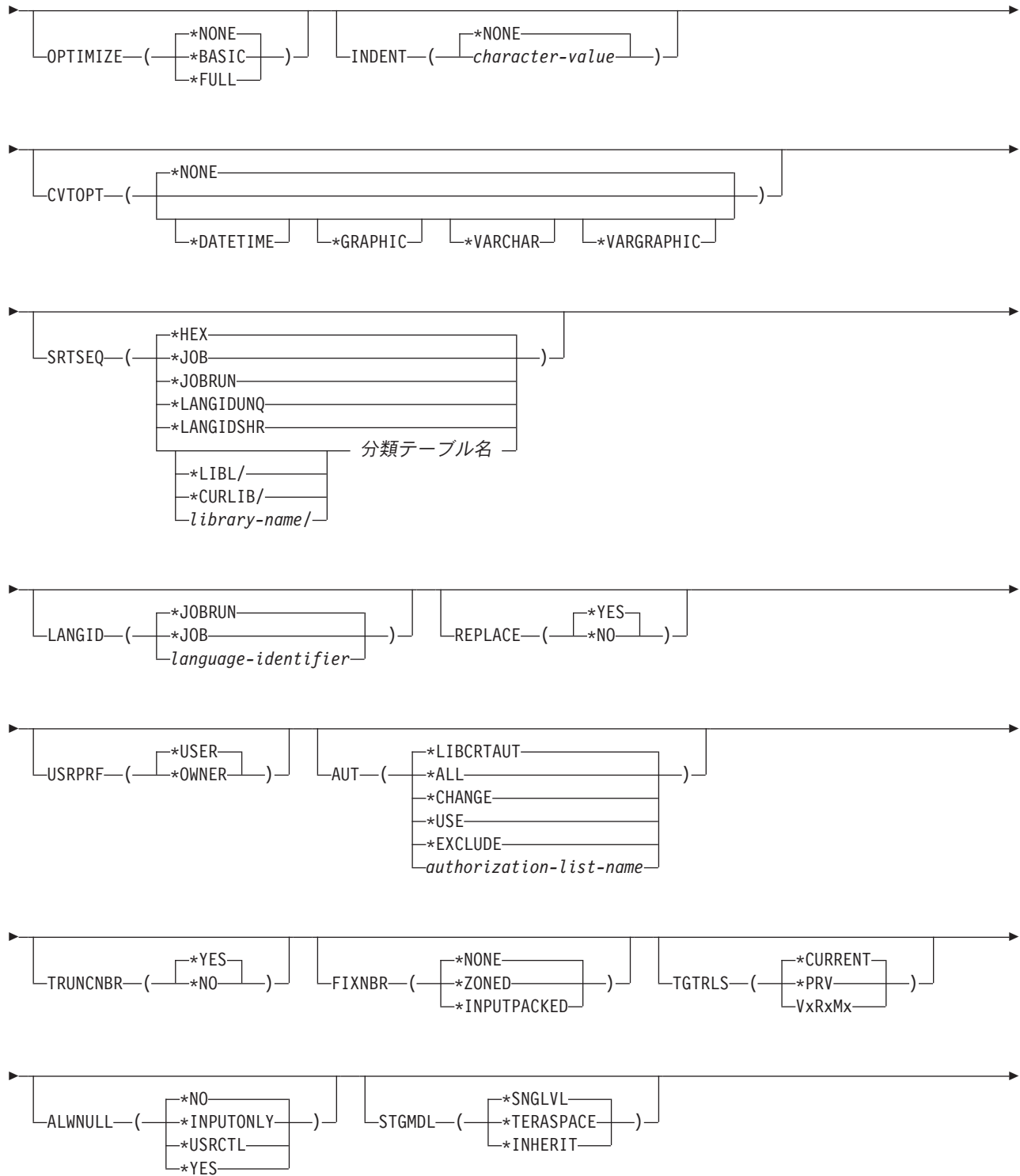


CRTBNDRPG コマンド

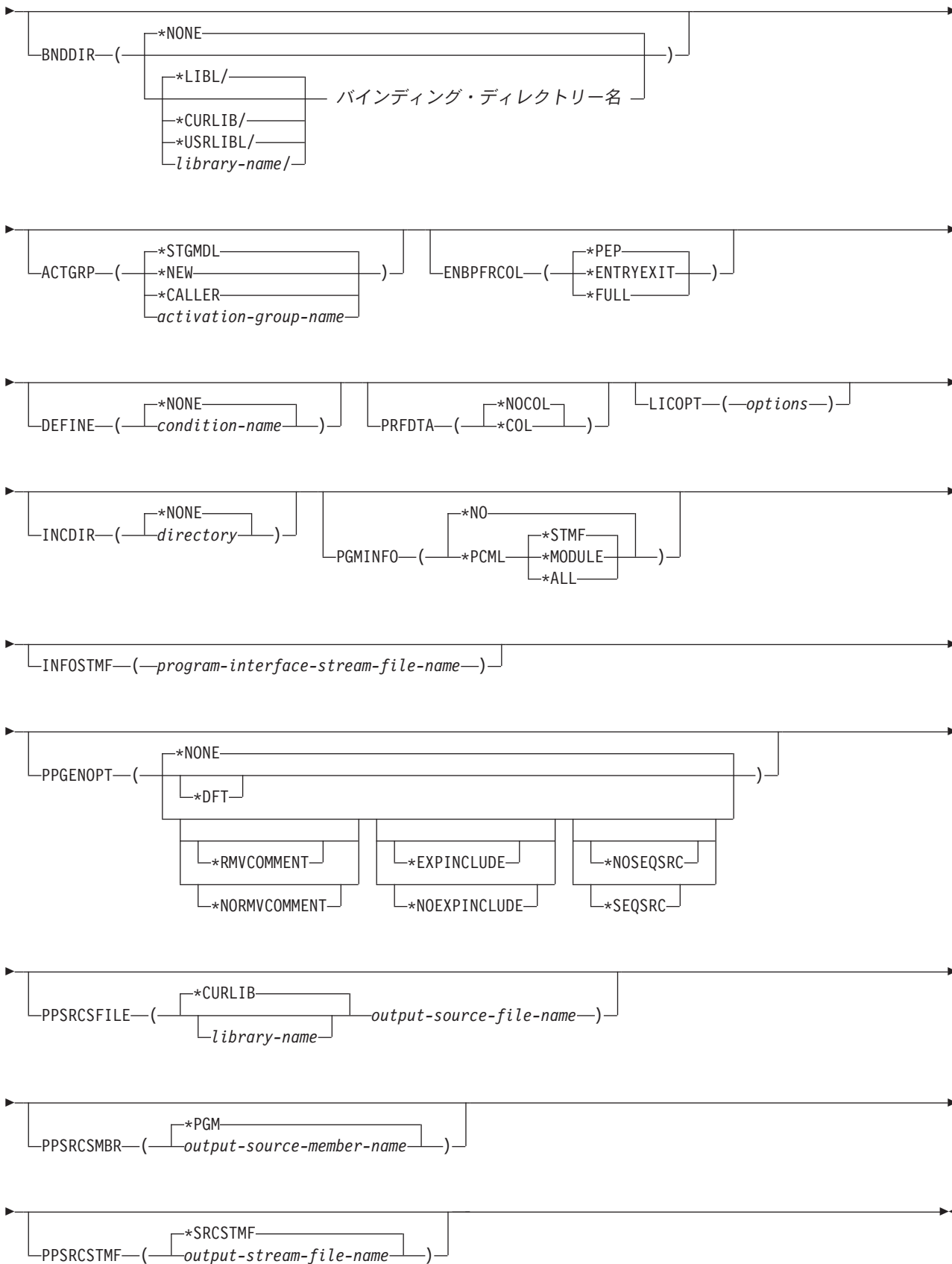
バインド RPG プログラムの作成 (CRTBNDRPG) コマンドは、ソース・コードから一時モジュール・オブジェクトを作成してから、プログラム・オブジェクトを作成することによって、RPG モジュールの作成 (CRTRPGMOD) コマンドおよびプログラムの作成 (CRTPGM) コマンドの結合されたタスクを実行します。プログラム・オブジェクトがいったん作成されると、CRTBNDRPG は作成した一時モジュールを削除します。CRTBNDRPG コマンドの構文図全体を下に示します。

ジョブ: B,I プログラム: B,I REXX: B,I EXEC





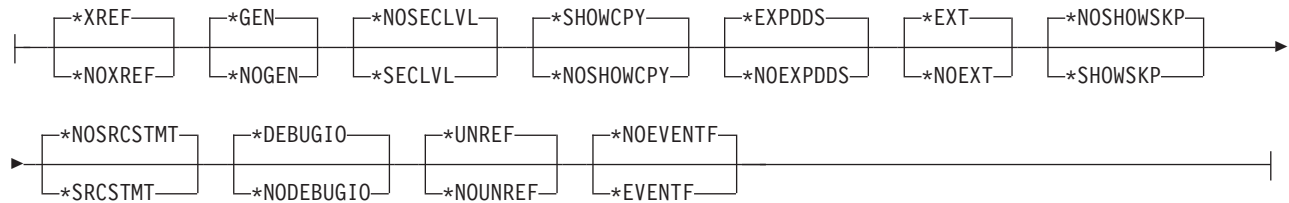
CRTBNDRPG コマンド



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 IBM i 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おらず、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。

program-name

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。

library-name

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。

CRTBNDRPG コマンド

***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。

library-name

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。

severity-level-value

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。

DFACTGRP

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。

DFACTGRP(*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 DFACTGRP(*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。

DFACTGRP(*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。

DFACTGRP(*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。

CRTBNDRPG コマンド

***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、コンパイラはステートメントを無視します。

***SHOWSKP**

コンパイラ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。

***UNREF**

非参照データ項目はコンパイルされたモジュールに含められます。

***NOUNREF**

非参照データ項目は、コンパイルされたモジュールに含められません。これによって使用ストレージの量が少なくなり、もっと大きなプログラムがコンパイルできるようになります。 *NOUNREF オプションを選択した場合は、デバッグ中に非参照データ項目を見たり、それに代入したりすることはできません。非参照データ項目は OPTION (*XREF) を指定して作成された相互参照表には表示されます。

***NOEVENTF**

Rational Developer for i で使用するイベント・ファイルを作成しません。 Rational Developer for i
は、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、Rational Developer for i 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**

Rational Developer for i 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。

Rational Developer for i は、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、Rational Developer for i 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。

CRTBNDRPG コマンド

***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。

DBGNCKEY

デバッグ・ビューに組み込まれるプログラム・ソースを暗号化するために使用される暗号鍵を指定します。

***NONE**

暗号鍵は指定されていません。

character-value

モジュール・オブジェクトに保管されるデバッグ・ビューに組み込まれるプログラム・ソースを暗号化するために使用される鍵を指定してください。鍵の長さは 1 バイトから 16 バイトまでです。1 バイトから 15 バイトまでの長さの鍵は、ブランクが埋め込まれて 16 バイトの暗号鍵となります。長さがゼロの鍵を指定することと ***NONE** を指定することは同じです。

コード・ページによって変わる文字が鍵に含まれている場合は、受動システムが起動システムと同じコード・ページを使用するようにする必要があります。そうしないと、鍵が一致せず、暗号化解除できなくなることがあります。さまざまなコード・ページを持つシステムで暗号鍵を入力しなければならない場合は、どの EBCDIC コード・ページでも変わらない文字から成る鍵を使用することをお勧めします。

OUTPUT

コンパイラ・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・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・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。

character-value

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。

CRTBNDRPG コマンド

*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。

library-name

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。

language-identifier

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/systems/i/infocenter/> で **IBM i 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。

CRTBNDRPG コマンド

*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な説明は、資料「機密保護解説」に記載されています。

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を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。

target-release

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・レベルより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 モジュールが、外部記述データベース・ファイルからのヌル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 モジュールが、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 モジュール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 モジュールが、外部記述の入力専用データベース・ファイルから、null を含むヌル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。

- ヌル可能キー・フィールドを使用する
- ヌル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ヌル可能フィールドが実際に null であるかどうかを判別する
- ヌル可能フィールドが null であるように設定する

CRTBNDRPG コマンド

*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、ヌル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ヌル可能フィールドが 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。

STGMDL

プログラムの記憶域モデル属性を指定します。

*SNGlvl

プログラムは単一レベル記憶域モデルで作成されます。単一レベル記憶域モデル・プログラムが活動化され実行されると、単一レベル記憶域が自動かつ静的な記憶域として提供されます。単一レベル記憶域プログラムは単一レベル記憶域活動化グループ内でのみ実行されます。

*TERASPACE

プログラムはテラスペース記憶域モデルで作成されます。テラスペース記憶域モデル・プログラムが活動化され実行されると、テラスペース記憶域が自動かつ静的な記憶域として提供されます。テラスペース記憶域プログラムはテラスペース記憶域活動化グループ内でのみ実行されます。

*INHERIT

プログラムは継承記憶域モデルで作成されます。このプログラムは、活動化されると、このプログラムが活動化された活動化グループの記憶域モデルを選択します。これは、その呼び出し元の記憶域モデルを継承することと同じ意味になります。 *INHERIT 記憶域モデルが選択されたときは、活動化グループ (ACTGRP) パラメーターで *CALLER を指定する必要があります。

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。

library-name

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。

***STGM DL**

STGM DL(*TERASPACE) が指定されていると、プログラムは呼び出されて QILETS 活動化グループで活動化されます。その他の場合は、このプログラムは呼び出されて QILE 活動化グループで活動化されます。

***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。

activation-group-name

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チ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・プロシーチャーではなく、そのプログラムの実際のプログラム入力プロシーチ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チ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**

すべてのプロシーチ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、外部プロシーチ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。

condition-name

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、プロシーチャー内のプロシーチ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。

CRTBNDRPG コマンド

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。

directory

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 パラメーター:

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するかどうか、およびその生成場所を指定します。以下の順序でオプション値を指定します。

generate

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

プログラム・インターフェース情報は生成されません。

*PCML

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します。PCML が生成されると、Java メソッドがこの RPG モジュール内のプロシーチャーをより簡単に呼び出せるようになり、必要な Java コードも少なくなります。

location

generate パラメーターが *PCML である場合、生成されたプログラム情報の場所を指定します。指定できる値は次のとおりです。

*STMF

プログラム情報をストリーム・ファイルに生成することを指定します。生成された情報が含まれるストリーム・ファイルの名前は、INFOSTMF オプションで指定する必要があります。

*MODULE

プログラム情報を RPG モジュールに保管することを指定します。CRTBNDRPG の場合、プログラムを作成する前の最初のステップとしてモジュールが作成されます。

*ALL

プログラム情報をストリーム・ファイルに生成し、さらにモジュールに保管することを指定します。生成された情報が含まれるストリーム・ファイルの名前は、INFOSTMF オプションで指定する必要があります。

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全体を実行します。プリプロセッサ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を実行します。プリプロセッサ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。

CRTBNDRPG コマンド

*CURLIB

プリプロセッサ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、プリプロセッサ出力ファイルは QGPL ライブラリー内に作成されます。

library-name

プリプロセッサ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。

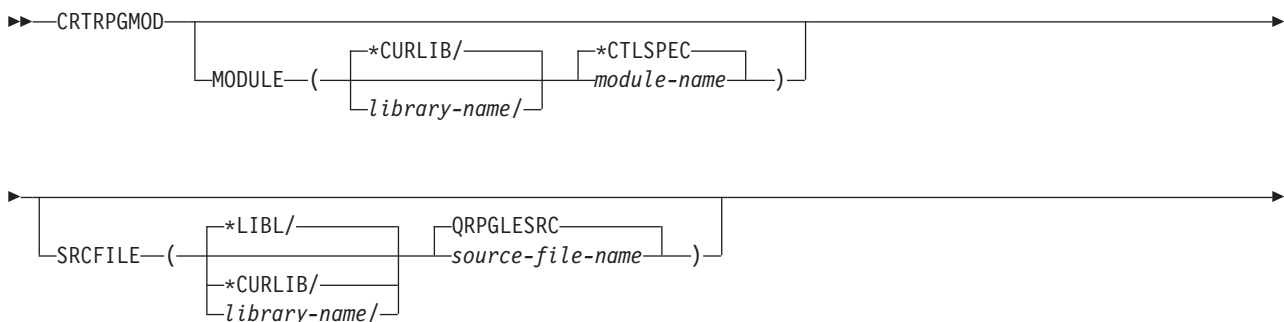
このパス名は絶対パスまたは相対パスのどちらでも構いません。絶対パス名の先頭は /、相対パス名の先頭は . 以外の文字です。

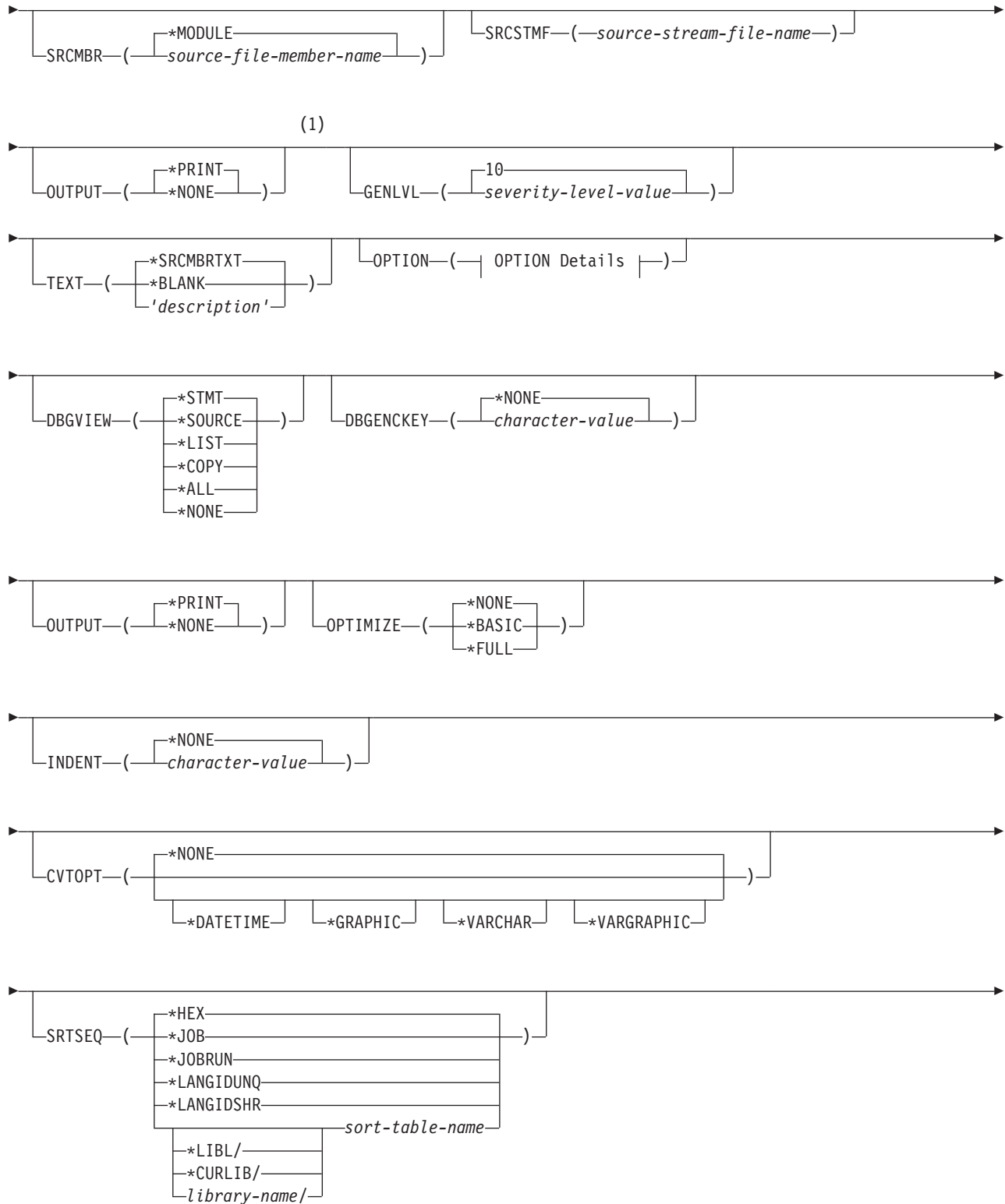
絶対修飾であれば、パス名としては完全です。相対修飾の場合、パス名にジョブの現行作業ディレクトリーを付け加えることによって、パス名が完全なものになります。

CRTRPGMOD コマンド

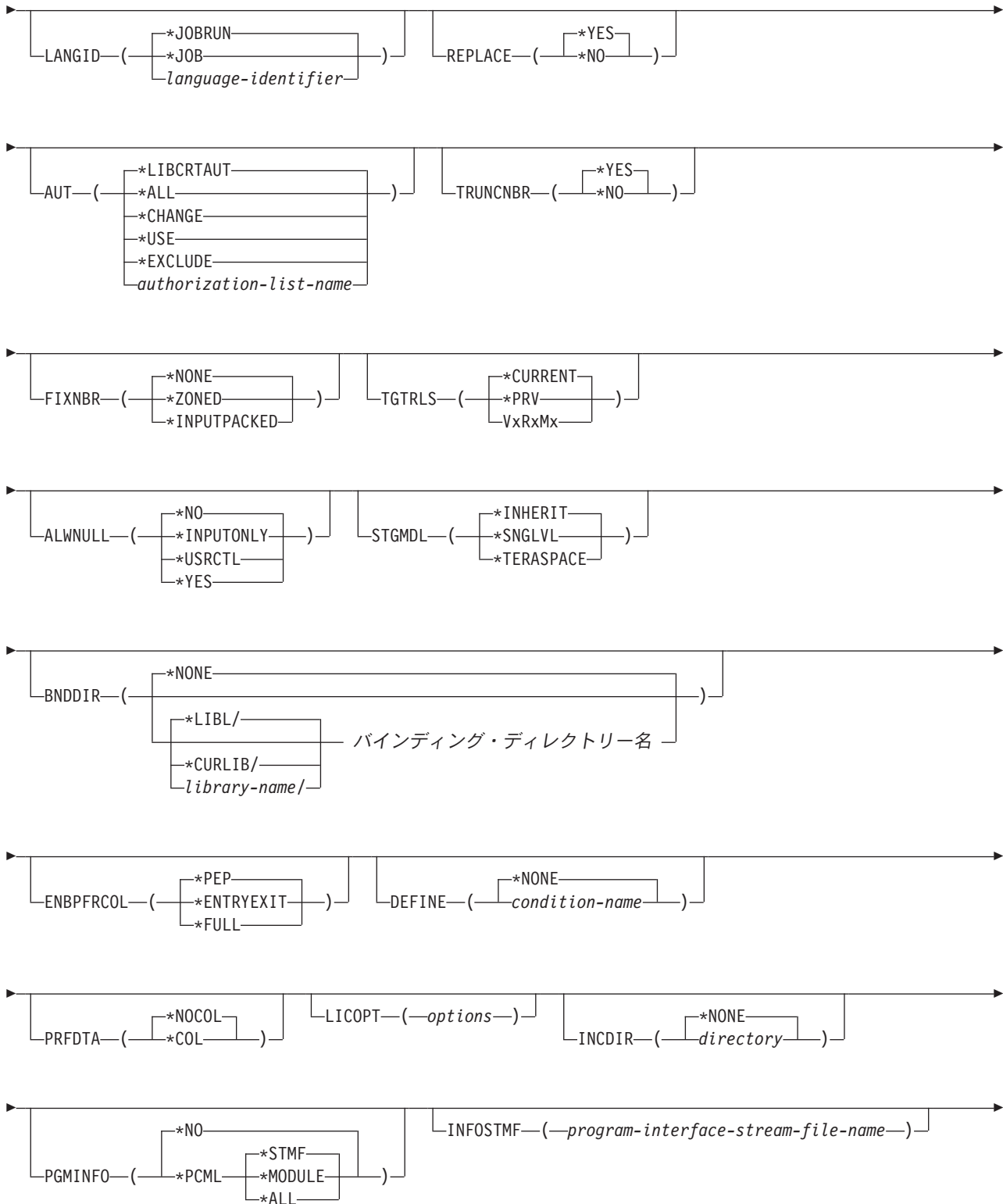
RPG モジュールの作成 (CRTRPGMOD) コマンドは ILE RPG ソース・コードをコンパイルして、モジュール・オブジェクト (*MODULE) を作成します。CRTRPGMOD コマンドの構文図全体が下に示されます。

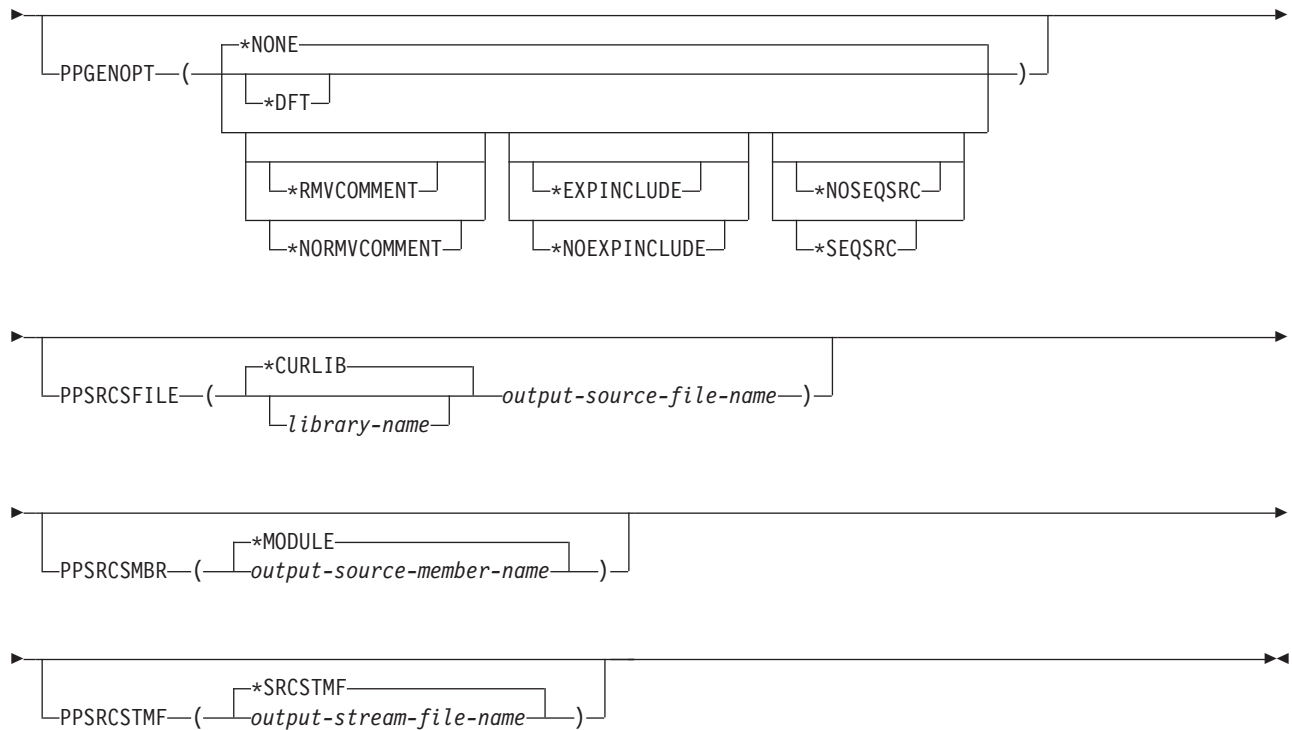
ジョブ: B,I プログラム: B,I REXX: B,I EXEC





CRTRPGMOD コマンド

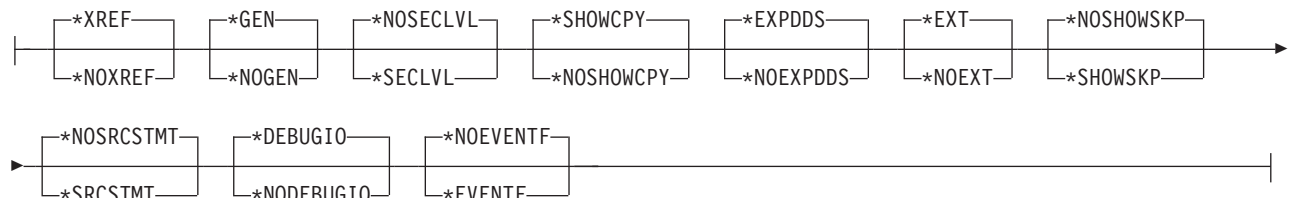




注:

- 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。)

STGMDL パラメーターの意味は、CRTRPGMOD コマンドの場合と CRTBNDRPG コマンドの場合とで異なります。

STGMDL

モジュールが使用する記憶域のタイプを指定します。

***INHERIT**

モジュールは継承記憶域モデルで作成されます。継承記憶域モデル・モジュールは、単一レベル記憶域モデル、テラスペース記憶域モデル、または継承記憶域モデルによってプログラムおよびサービス・プログラムにバインドすることができます。単一レベル記憶域モデル・プログラムおよびテ

CRTRPGMOD コマンド

テラスペース記憶域モデル・プログラムの自動かつ静的記憶域として使用される記憶域のタイプは、オブジェクトの記憶域モデルと一致します。継承記憶域モデル・オブジェクトは呼び出し元の記憶域モデルを継承します。

*SNGLVL

モジュールは単一レベル記憶域モデルで作成されます。単一レベル記憶域モデル・モジュールは、単一レベル記憶域を使用しているプログラムおよびサービス・プログラムにのみバインドすることができます。これらのプログラムとサービス・プログラムは、単一レベル記憶域を自動かつ静的記憶域として使用します。

*TERASPACE

モジュールはテラスペース記憶域モデルで作成されます。テラスペース記憶域モデル・モジュールは、テラスペース記憶域を使用しているプログラムおよびサービス・プログラムにのみバインドすることができます。これらのプログラムとサービス・プログラムは、テラスペース記憶域を自動かつ静的記憶域として使用します。

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、4-15 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 7-5 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 7-5. 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せされる長さ
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

表 7-5. 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
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
EVAL-CORR 要約	*XREF ⁴	EVAL-CORR 操作のサブフィールドの要約
外部参照	*EXT	コンパイル時参照された外部プロシージャ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
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わるかの詳細については、『*NOSRCSTMT ソース見出し』および『*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。
4. OPTION(*XREF) が指定された場合、要約は、EVAL-CORR 操作で処理されるかどうかに関係なく、すべてのサブフィールドに関する情報をリストします。OPTION(*NOXREF) が指定された場合、要約は、EVAL-CORR 操作で処理されないサブフィールドに関する情報のみリストします。EVAL-CORR 操作がない場合、EVAL-CORR 要約セクションは印刷されません。

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。7-58 ページの図 7-21 は、CRTBNDRPG コマンドを使ってコンパイルしたプログラム 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 ID	*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	*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.pcm1 8			
組み込みディレクトリー	/projects/ABC Electronics Corporation/copy files/prototypes			
	/home/mydir 9			

図 7-21. 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、4-17 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 (CRTTRPGMOD 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) 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

CRTTRPGMOD 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 ソース仕様書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、7-60 ページの図 7-22 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	ソースの仕様										注記	Src Seq	ID	番号
番号	1	2	3	4	5	6	7	8	9	10				
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 7-22. `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 7-23 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	3	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 7-23. `OPTION(*SRCSTMT)` が指定された場合のリストのセクション例

7-61 ページの図 7-24 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。

5769WDS	V5R2M0	020719	RN	IBM ILE RPG	MYLIB/MYSRC	ISERIES1	02/08/15 14:21:00	Page	2		
1a											
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	番号	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 ID : *JOB RUN										
	* 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										
	* スケール可能 : *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
	* * * * * : * * *										
	* * * * * : INFILE MYLIB/INFILE										
	* * * * * : INREC INREC										
	-----*										
8	FKEYL6	IF	E	K DISK				971104		000800	
	-----*										
	* * * * * : * * *										
	* * * * * : KEYL6 MYLIB/KEYL6										
	* * * * * : REC1 REC1										
	* * * * * : REC2 REC2										
	-----*										
9	FOutfile	O	E	DISK				971104		000900	
	-----*										
	* * * * * : * * *										
	* * * * * : OUTFILE MYLIB/OUTFILE										
	* * * * * : 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 7-24. リストのサンプル・ソース部分

コンパイラー・リスト

* データ構造 : DSEXT1									
* 接頭部 : BI : 0									
* 外部フォーマット : REC1 : MYLIB/DSEXT1									
* 様式テキスト : レコード様式の説明									

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									

* RPG レコード様式 : INREC									
* 外部フォーマット : INREC : MYLIB/INFILE									

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									

* RPG レコード様式 : REC1									
* 外部フォーマット : REC1 : MYLIB/KEYL6									

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									

* RPG レコード様式 : REC2									
* 外部フォーマット : REC2 : MYLIB/KEYL6									

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 <----- ソースの仕様 -----><----- 注記 -----> Src Seq									
番号2.....+<----- 26 - 35 ----->.....4.....+.....5.....+.....6.....+.....7.....+.....8.....+.....9.....+.....10 ID 番号									
38 C	MOVE		'123'	BI_FLD1				971104	001500
39 C/COPY MYCPY								971104	001600

* RPG メンバー名 : MYCPY									
* 外部名 : MYLIB/QRPGLESRC(MYCPY)									
* 最終変更 : 98/07/24 16:20:04									
* テキスト '記述' : Text specified on Copy Member									

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 MOVE 'x' Y 10 002700									
54 C MOVE 'x' Z 10 002800									
55 C/ELSE 971104 002900									
10 ----- C MOVE ' ' Y 10 971104 003000									
----- C MOVE ' ' Z 10 971104 003100									
56 C/ENDIF 971104 003200									

行 <----- ソースの仕様 -----><----- 注記 -----> Do Page 変更 SRC SEQ									
番号2.....+.....3.....+.....4.....+.....5.....+.....6.....+.....7.....+.....8.....+.....9.....+.....10 番号 行 日付 ID 番号									
57=OUTREC 6000001									

* RPG レコード様式 : OUTREC									
* 外部フォーマット : OUTREC : MYLIB/OUTFILE									

58=0 FLDY 100A CHAR 100 6000002									
59=0 FLDZ 132A CHAR 32 6000003									
60=0 GRAPHFLD 156G GRPH 12 835 6000004									
* * * * * ソースの終わり * * * * *									

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		ソースの仕様										注記		DO	PAGE	変更	Statement
SEQ	番号	1	2	3	4	5	6	7	8	9	10	番号	行	日付	番号		番号
...	...	1	2	3	4	5	6	7	8	9	10	番号	行	日付	番号		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 '4' 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。図 7-25 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 7-25. 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。7-65 ページの図 7-26 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 7-26. OPTION(*SRCSTMT) が指定された追加の診断メッセージ例

出力バッファ位置

出力バッファ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 7-27 は出力バッファ位置テーブルの例を示しています。

```

出力バッファ位置
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
* * * * * 出力バッファ位置の終わり * * * * *

```

図 7-27. 出力バッファ位置テーブル

/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。図 7-28 に例を示します。

```

/COPY 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   QRPGLSRC MYCPY      37  98/07/24 16:20:04
* * * * * / コピー・メンバーの終わり * * * * *

```

図 7-28. 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、7-66 ページの図 7-29 に示したように、代替照合順序

コンパイラー・リスト

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			971104	003300	
61 **	-----*				
	* 代替照合順序テーブルのデータ:				
	-----*				
62 ALTSEQ 1122ACAB4B7C36F83A657D73	971104 003400		変更	SRC SEQ	
行番号 <-----データ・レコード----->	日付 ID 番号				
.....1.....2.....3.....4.....5.....6.....7.....8.....9.....10					
-----*					
* 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 7-29. 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。7-67 ページの図 7-30 に例を示します。

キ ー ・ フ ィ ー ル ド 情 報				
ファイル レコード	内部 フィールド名	外部 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 7-30. 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力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。7-68 ページの図 7-31 は、2 つのサブプロシージャをもつモジュール 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。

コンパイラー・リスト

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	
グローバル・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			
UPDATE	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 7-31. *OPTION(*NOSRCSTMT)* が指定された相互参照テーブル例

EVAL-CORR 要約

*OPTION(*XREF)* が指定された場合、EVAL-CORR 要約は、以下を示しているソースまたはターゲット・データ構造のいずれかの、すべてのサブフィールドをリストします。

- サブフィールドが割り当てられているかどうか
- サブフィールドが割り当てられていない場合、ソースおよびターゲット・サブフィールドが対応すると見なされない理由
- 割り当てられているサブフィールドについて、配列エレメントの数の差またはサブフィールドでヌルが可能かどうかなど、割り当てに影響する可能性のある追加情報

*OPTION(*NOXREF)* が指定されている場合、EVAL-CORR 要約は、対応するサブフィールドに関する情報をリストしません。 サブフィールドは対応すると見なされないという理由で、対応しないサブフィールドのみをリストします。

EVAL-CORR 要約 1	1	13	14	19	24	2
		28				
FLD1		割り当て済み; 完全一致突き合わせ				
FLD2		割り当て済み; ターゲットとソースに互換性があります				
構成されています	3	ターゲット・サブフィールドはソース・サブフィールドよりも少ない要素で				
FLD3		割り当て済み; 完全一致突き合わせ				
		ターゲット・サブフィールドはヌル可能です; ソース・サブフィールドは				
4 *RNF7349 FLD5		ソースとターゲットでデータ・タイプが異なります				
EVAL-CORR 要約	2	22				
FLD1		割り当て済み; 完全一致突き合わせ				
SUBDS	5	割り当て済み; 完全一致突き合わせ				
SUBF1		ターゲット・サブフィールドは OVERLAY で定義されています				
FLD2		割り当て済み; 完全一致突き合わせ				
*RNF7341 FLD3		ターゲット内のみ				

図 7-32. EVAL-CORR 要約

1 EVAL-CORR 要約番号

『追加の診断』セクションのメッセージは、関連する EVAL-CORR 要約を番号別に参照します。

2 EVAL-CORR ステートメント番号

同じ (同一か LIKEDS または LIKEREK で関連している) ソースおよびターゲット・データ構造を持つ EVAL-CORR 操作は、同じ EVAL-CORR 要約を共有します。この例では、1 組のデータ構造定義を持つ 5 つの EVAL-CORR 操作、および別の 1 組を持つ 1 つの EVAL-CORR 操作があります。

3 サブフィールドの追加情報

サブフィールドが割り当てられています。追加情報が別の行にリストされます。

4 サブフィールドが割り当てられていないことを示すメッセージ

サブフィールドが割り当てられていません。エラー・メッセージおよびテキストが、サブフィールドが対応していると見なされない理由を示します。

5 データ構造サブフィールド

サブフィールドがデータ構造である場合、そのサブフィールドはインデントでリストされます。

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、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部にはプロシージャ名と、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。7-70 ページの図 7-33 に例を示します。

外 部 参 照			
静的結合プロシージャ:			
プロシージャ		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 7-33. 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 7-34 に例を示します。

メ ャ セ ャ ジ の 要 約			
MSG ID	SV 番号	メ ャ セ ャ 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 : レコードの各読み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 ャ セ ャ ジ の 要 約 の 終 わ り * * * * *			

図 7-34. 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。 7-71 ページの図 7-35 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 7-35. 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 外部プロシージャおよびフィールドの場所を指定できないことです。

付録 E. プリプロセッサ・プロバイダーの情報

メイン・ソース・ファイルとコピー・ファイルを新しいソース・メンバーにマージする RPG プリプロセッサは、特殊ディレクティブ ****FREE** を含むコードに関連した、次のルールを守る必要があります。

プリプロセッサは特殊ディレクティブ ****END-FREE** を使用して、メイン・ソース・ファイルとコピー・ファイルを単一のソース・ファイルにマージするプリプロセスされたソース・メンバー内で、完全自由コードの終わりを示すことができます。

- プリプロセッサがコピー・ファイルを、プリプロセスされた出力にマージする場合、次のようになります。
 - コピー・ファイルが完全自由形式モードで、/COPY ディレクティブまたは /INCLUDE ディレクティブを含むファイルが完全自由形式モードでなかった場合、プリプロセッサは、プリプロセッサ出力の、コピー・ファイルの最後の部分に ****END-FREE** を挿入する必要があります。
 - ****FREE** が挿入されるのはプリプロセッサ・コードが桁制限ソース・モードの時のみであり、****END-FREE** が挿入されるのはプリプロセッサ・コードが完全自由形式ソース・モードの時のみであるならば、プリプロセッサは、都合に応じて、追加の ****FREE** 行および ****END-FREE** 行を挿入することができます。
 - コピー・ファイルが完全自由形式モードでなく、/COPY ディレクティブまたは /INCLUDE ディレクティブを含むファイルが完全自由形式モードだった場合、プリプロセッサは、プリプロセッサ出力の、コピー・ファイルの最後の部分に ****FREE** を挿入する必要があります。

プリプロセッサ・プロバイダーの情報

- **FREE または **END-FREE を含む、挿入された行の残りの部分はブランクでなければなりません。
- **FREE 特殊ディレクティブと **END-FREE 特殊ディレクティブは、出力ソース内で均衡を保っている必要があります。ただし、ソースが完全自由モードで終了している場合、ソースは **END-FREE で終了している必要はありません。
- 特殊ディレクティブは次の場合に受け入れられます。
 - **FREE および **END-FREE は、ステートメント間にのみコーディング可能である。
 - **FREE は、ソースが完全自由形式モードの場合指定できない。
 - **END-FREE は、ソースが桁制限モードの場合指定できない。
 - **FREE は、コピー・ファイルの 1 行目にのみ指定可能である。
 - **END-FREE は、コピー・ファイルには指定できない。
- 特殊ディレクティブが受け入れられず、プリプロセッサが前処理の続行を選択した場合、プリプロセッサは、無効な特殊ディレクティブをプリプロセッサの出力にコピーしてはならない。

参考文献

IBM i 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 の資料を参照してください。

- 「ADTS/400: Programming Development Manager (SC09-1771-00)」には、コピー、削除、名前変更などの操作を容易にするために、プログラム開発管理機能 (PDM) を使用した、ライブラリー、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のリストの操作に関する情報が記載されています。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。
- 「ADTS for AS/400: Source Entry Utility (SC09-2605-00)」には、ソース・メンバーを作成および編集するための Application Development ToolSet 原始ステートメント入力ユーティリティー (SEU) の使用に関する情報が記載されています。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。
- 「アプリケーション表示プログラミング (SC88-4031-01)」には、以下に関する情報が記載されています。
 - 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 (SD88-5008-10)」には、以下のもののセットアップおよび管理に関する情報が記載されています。
 - 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、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 (SD88-5038-06)」には、オブジェクトとライブラリー、CL プログラミング、フローの制御とプログラム間の通信、CL プログラムでのオブジェクトの操作、および CL プログラムの作成に関する一般的な説明を含む、IBM i プログラミングのトピックの広範囲な説明が記載されています。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)」には、通信環境における実行管理機能、通信状況、通信問題の追跡と診断、エラー処理と回復、パフォーマンス、特定の回線速度とサブシステム記憶域に関する情報が記載されています。

- 「*GDDM Programming Guide (SC41-0536-00)*」には、IBM i 図形データ表示管理プログラム (GDDM) を使用したグラフィックス・アプリケーション・プログラムの作成に関する情報が記載されています。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)*」には、IBM i 図形データ表示管理プログラム (GDDM) を使用したグラフィックス・アプリケーション・プログラムの作成に関する情報が記載されています。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)*」には、IBM i 通信および IBM i システム間通信機能 (IBM i-ICF) を使用するアプリケーション・プログラムを作成するために必要な情報が記載されています。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)*」では、IBM i 対話式データ定義ユーティリティ (IDDU) を使用してデータ・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
- 「*IBM Rational Development Studio for i: ILE C/C++ プログラマーの手引き (SC88-4024-02)*」には、ILE C 言語を使用したアプリケーションの開発方法に関する情報が記載されています。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。
- 「*IBM Rational Development Studio for i: ILE COBOL プログラマーの手引き (SD88-5045-07)*」には、IBM i 上で ILE COBOL プログラムを作成、コンパイル、バインド、実行、デバッグ、および保守する方法に関する情報が記載されています。これには、他の ILE COBOL プログラムと非 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を説明します。
- 「*ILE 概念 (SD88-5033-09)*」では、IBM i 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を説明します。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。
- 「*IBM Rational Development Studio for i: ILE RPG 解説書 (SD88-5043-07)*」には、ILE RPG プログラミング言語に関する情報が記載されています。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 (SD88-5073-03)*」には、印刷を理解し制御するために役立つ情報が記載されています。IBM i の印刷要素およびその概念の特定情報、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、スプール出力ファイルの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 IBM i 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。IBM i で使用できるフォントも示されています。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。

- 「機密保護解説書 (SD88-5027-11)」では、システム機密保護サポートを使用して、適切な権限を持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たり、データを故意または偶発的な損傷や消滅から保護したり、機密保護情報を最新の状態に維持したり、システム上で機密保護をセットアップしたりする方法を説明します。
- 「IBM i および関連ソフトウェアのインストール、アップグレードおよび削除 (SD88-5002-11)」には、IBM から入手したライセンス・プログラム、プログラム一時修正 (PTF)、および 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が記載されています。また、この解説書は、新しいリリースを導入するユーザーにも役立ちます。
- 「意外にも RPG IV でできること。システム・アクセスなどに関する魔法使いのガイド」(英語) には、RPG IV および統合化言語環境 (ILE) を最大限に活用したい IBM i プログラマーのためのヒントが記載されています。これは、次の IBM Redbooks® Web サイトで利用できます。

<http://www.redbooks.ibm.com/>

特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒103-8510

東京都中央区日本橋箱崎町19番21号

日本アイ・ビー・エム株式会社

法務・知的財産

知的財産権ライセンス渉外

以下の保証は、国または地域の法律に沿わない場合は、適用されません。IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態を提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Corporation

Software Interoperability Coordinator, Department YBWA

3605 Highway 52 N

Rochester, MN 55901

U.S.A.

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性があります。その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があります、単に目標を示しているものです。

本書はプランニング目的としてのみ記述されています。記述内容は製品が使用可能になる前に変更になる場合があります。

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、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。これらのサンプル・プログラムは特定物として現存するままの状態を提供されるものであり、いかなる保証も提供されません。IBM は、お客様の当該サンプル・プログラムの使用から生ずるいかなる損害に対しても一切の責任を負いません。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生的創作物にも、次のように、著作権表示を入れていただく必要があります。

© (お客様の会社名) (西暦年). このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。

© Copyright IBM Corp. _年を入れる_.

プログラミング・インターフェース情報

この「ILE RPG プログラマーの手引き」の資料には、プログラムを作成するユーザーが IBM i のサービスを使用するためのプログラミング・インターフェースが記述されています。

商標

IBM、IBM ロゴおよび ibm.com は、世界の多くの国で登録された International Business Machines Corporation の商標です。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、『www.ibm.com/legal/copytrade.shtml』をご覧ください。

Adobe、Adobe ロゴ、PostScript、PostScript ロゴは、Adobe Systems Incorporated の米国およびその他の国における登録商標または商標です。

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。

Java およびすべての Java 関連の商標およびロゴは Oracle やその関連会社の米国およびその他の国における商標または登録商標です。

他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。

使用条件

これらの資料は、以下の条件に同意していただける場合に限りご使用いただけます。

個人使用: これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、非商業的な個人による使用目的に限り複製することができます。ただし、IBM の明示的な承諾をえずに、これらの資料またはその一部について、二次的著作物を作成したり、配布（頒布、送信を含む）または表示（上映を含む）することはできません。

商業的使用: これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、お客様の企業内に限り、複製、配布、および表示することができます。ただし、IBM の明示的な承諾をえずにこれらの資料の二次的著作物を作成したり、お客様の企業外で資料またはその一部を複製、配布、または表示することはできません。

ここで明示的に許可されているもの以外に、資料や資料内に含まれる情報、データ、ソフトウェア、またはその他の知的所有権に対するいかなる許可、ライセンス、または権利を明示的にも黙示的にも付与するものではありません。

資料の使用が IBM の利益を損なうと判断された場合や、上記の条件が適切に守られていないと判断された場合、IBM はいつでも自らの判断により、ここで与えた許可を撤回できるものとさせていただきます。

お客様がこの情報をダウンロード、輸出、または再輸出する際には、米国のすべての輸出入関連法規を含む、すべての関連法規を遵守するものとします。

IBM は、これらの資料の内容についていかなる保証もしません。これらの資料は、特定物として現存するままの状態を提供され、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なしで提供されます。

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]

アクセス・パス

外部記述ディスク・ファイル 6-21

索引付きファイルの場合 6-27

例 6-28

圧縮、オブジェクトの 4-40

圧縮解除、オブジェクトの 4-40

アプリケーション・プログラミング・インターフェース (API)

バインド不能の呼び出し 4-77

メッセージ検索 (QMHRTVM) API 4-106

QMHSNDPM 7-2

一時変更、ファイル

概説 6-43

一時変更、ファイルの 6-8

概説 6-13

コンパイラー・リストに示された 7-58

例 6-14

イベント・ファイル 7-40

入り口モジュール 3-22, 4-34

印刷コマンド・キー 6-69

エスケープ・メッセージ

定義 5-58

未処理 5-64

エラー

コンパイルの訂正 4-19

実行時、Java 4-157

実行時の訂正 4-21

ファイル 5-61

プログラム 5-61

RPG から Java を呼び出したとき 4-147

エラー処理サブルーチン

使用 5-70

ファイル・エラー 5-71

プログラム 5-73

ループの防止 5-77

エラー処理サブルーチンでのループの防止 5-77

エラー標識

指定 5-67

演算仕様書

概要説明 3-1

プログラム記述 WORKSTN ファイル 6-75

オーバーフロー

標識 6-55

ページ 6-54

オーバーフロー標識

有無 6-55

概要説明 6-55

条件付け出力 6-55

設定 6-57

フェッチ・オーバーフロー・ルーチン論理 6-57

プログラム・サイクルとの関係 6-57

例 6-56, 6-57

PRINTER ファイルによる 6-53

オープン・データ・パス

共用 6-17

応答、実行時照会メッセージに対する 4-56

応答リスト、メッセージの

追加 4-56

変更 4-57

応答リスト項目処理 (WRKRPLYE) コマンド

システム応答リストの変更 4-57

応答リスト項目追加 (ADDRPLYE) コマンド

システム応答リストへの追加 4-56

置き換え、プログラム内のモジュールの 4-39

オンライン情報

作成コマンド 7-37

ILE ソースの 5-4

[カ行]

開始、コミットメント制御の 6-46

開始、ILE ソースの 5-8

解除、ロック・レコードの 6-16

外部記述ファイル

アクセス・パス 6-21

一時変更 6-8

外部記述への追加 6-5

出力仕様書 6-9

仕様書 6-5

定義 6-3

ファイル仕様書 6-5

フィールド名の変更 6-6

物理および論理ファイル 6-20

プログラム記述として 6-4

利点 6-1

レコード様式仕様書 6-21

レコード様式の名前変更 6-6

WORKSTN ファイルとして 6-66, 6-69

外部参照リスト、コンパイラー・リストの 7-69

解放、ILE プログラムの資源の 4-60

活動化、プログラムの 4-57

活動化グループ

管理 4-57

削除 4-59

識別 4-34, 4-58

活動化グループ (続き)

- 指定の 4-34
 - 削除 4-58
 - 指定 4-58
- 定義 4-57
- 例外処理での役割 5-59
- OPM デフォルト 4-59
- QILE 4-34, 4-58
- *CALLER 4-58
 - 指定 4-58
 - OPM デフォルトでの実行 4-59
- *NEW 4-34, 4-104
 - 指定 4-58
 - 終了 4-58

活動化グループ再利用 (RCLACTGRP) コマンド

- 活動化グループの削除 4-59
- 名前付き活動化グループ 4-58

可変長形式

- デバッグ中の表示 5-43

可変長レコード 6-62

画面設計機能 (SDA) 4-54

監査ファイル 7-9

管理、活動化グループの 4-57

管理、実行時の記憶域の 4-60

管理、動的に割り振られた記憶域の 4-60

管理、プログラムの 3-13

管理、RPG 命令を使用したデフォルト・ヒープの 4-62

キー

- 複合 6-25
- 部分 6-25
 - レコードまたはファイルの場合 6-24

キー順アクセス・パス 6-21

キー順処理 6-51

- アクセス・パス 6-21
- 限界内順次 6-40
- 索引付きファイル 6-27
- 予期しない結果 6-51
- レコード・アドレス限界値ファイル 6-30

キーによらない処理 6-43

キーによる順次処理

- 概説 6-33
- 例 6-33

キーによるランダム処理

- 概説 6-38
- 例 6-39

キーワード

- 継続記入行の場合 6-20
 - CLEAR 6-69
 - HELP 6-69
 - HOME 6-69
 - PRINT 6-69
 - ROLLDOWN 6-69
 - ROLLUP 6-69
- 表示装置ファイルの
 - CLEAR 6-69
 - HELP 6-69

キーワード (続き)

表示装置ファイルの (続き)

- HOME 6-69
- PRINT 6-69
- ROLLDOWN 6-69
- ROLLUP 6-69
- DDS 6-20
- EXPORT 4-36
- NOOPT 4-39, 5-66
- *OMIT 4-89

記憶域解放 (CEEFRST) バインド可能 API 3-15

記憶域管理

- 実行時の管理 4-60
- 実行時の割り振り 4-68
- 動的記憶域 4-60

記憶域再割り振り (CEEZST) バインド可能 API 3-15

記憶域モデル 4-107

- 継承記憶域モデル 4-107
- 推奨事項 4-108
- 単一レベル記憶域モデル 4-107
- テラスペース記憶域モデル 4-107

記述子、操作の

- 定義 4-87
- 例 4-44

機能キー

- 標識 6-68
- WORKSTN ファイルで 6-68

機能チェック

- 定義 5-58
- 未処理 5-65

共用、ファイルのオープン・データ・パスの 6-17

組み込み関数

- %ADDR 4-89

組み込みソース・ビューの作成 5-6

グラフィック形式

- グラフィック CCSID
 - コンパイラ・リストに示された 7-60
- EVAL を使用して値を割り当てるための規則 5-48
- NLSS デバッグに関する考慮事項 5-22
- グラフィックス・ルーチンの呼び出し 4-106
- グラフィック表示ルーチン (PGR) 4-106
- グラフィック・サポート 4-106
- グラフィック・データ表示管理プログラム (GDDM) 4-106
- 結果標識 (01-99、H1-H9、OA-OG、OV、L1-L9、LR、U1-U8、KA-KN、KP-KY、RT)
 - エラー標識として 5-67

限界、レコードの 6-22

限界内順次処理

- 概説 6-40
- 例 6-41

権限、コマンドに対する 1-1

言語、ILE 3-11

言語間呼び出し 4-97

検査、レベル 6-10

検査、渡されるパラメーター数の 4-90

検索指数

外部記述ファイル

説明 6-24

部分キーの参照 6-25

有効な 6-25

プログラム記述ファイル 6-28

原始ステートメント入力ユーティリティー (SEU) 4-1

コンパイラー・リストのブラウズ 4-20

ソース・ステートメントの入力 4-2

検出、プログラム内のエラーの 5-1

コード生成エラー、コンパイラー・リストの 7-71

コード変換の制約 7-25

構造化命令

字下げ 4-18

構文図

解釈 7-33

CRTBNDRPG コマンド 7-34

CRTRPGMOD コマンド 7-52

CVTRPGSRC コマンド 7-11

互換性の相違点、OPM RPG/400 と ILE RPG との 7-1

コマンド機能 (CF) キー 6-67

コマンド定義 4-56

コマンド・アテンション (CA) キー 6-67

コミットメント制御 6-45

開始および終了 6-46

条件付き 6-49

ファイルの指定 6-47

プログラム・サイクルでの 6-50

有効範囲 6-47

例 6-48

ロック 6-46

COMMIT 命令 6-48

コンパイラー指示

リスト見出しの変更 4-17

コンパイラー・リスト

インライン診断メッセージ 4-19

形式の指定 4-17

構造化命令の字下げ 4-18

コンパイル・エラーの訂正 4-19

サンプル・リスト 7-57

実行時エラーの訂正 4-21

使用 4-15

セクション 4-16, 7-57

追加の診断メッセージ 4-20

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 4-21

デフォルトの情報 4-16

入手 4-16

文書化としての使用 4-21

読み取り 7-56

SEU を使用したブラウズ 4-20

コンパイラー・リストにおける構造化命令の字下げ 4-18

コンパイラー・リストの形式の指定 4-17

コンパイラー・リストのキー・フィールド情報 7-66

コンパイラー・リストの作成 4-16

コンパイラー・リストのソース・セクション 7-59

コンパイラー・リストの追加の診断メッセージ・セクション
7-64

コンパイラー・リストのプロローグ・セクション 7-57

コンパイル

モジュールの作成 4-24

CRTBNDRPG コマンドの使用 4-10

ILE RPG と OPM RPG/400 との相違点 7-1

ILE における 3-12

コンパイル時の配列およびテーブル

コンパイラー・リストのセクション 7-65

コンパイル・エラーの訂正 4-19

[サ行]

サービス・プログラム

拡張アプリケーション・プログラムにおける 3-24

関連する CL コマンド 4-43

更新 4-49

作成 4-41

作成の方針 4-42

サンプル・バインダー・リスト 4-50

資源再利用 4-60

使用の理由 4-41

デバッグ・セッションへの追加 5-11

バインダー言語 4-47

変更 4-43

例 4-44

CRTBNDRPG によるバインド 4-13

サービス・プログラムの更新 4-49

サービス・プログラムの作成

概要 4-41

方針 4-42

サービス・プログラムの作成 (CRTSRVPGM) コマンド

および ILE 3-13

パラメーター 4-43

例 4-47

サービス・プログラム表示 (DSPSRVPGM) コマンド 4-41

サービス・プログラム変更 (CHGSRVPGM) コマンド 4-49

再開点 5-79

再帰

再帰呼び出し 3-39, 4-78

条件処理プログラムの呼び出し 5-80

サイクル、プログラム

概要説明 3-2

コミットメント制御 6-50

最後のサイクル 3-3

フェッチ・オーバーフロー論理 6-57

サイクル・フリー・モジュール 4-27

最終要約、コンパイラー・リストの 7-70

最終レコード (LR) 標識

プログラム / プロシージャーの終了に使用 4-102, 4-103

最適化

定義 4-39

デバッグ時のフィールドに対する影響 5-2

例外の処理に関する考慮事項 5-66

最適化 (続き)

レベル

オブジェクトの変更 4-39

チェック 4-40

最適化レベルの変更

プログラムまたはモジュールの 4-39

再バインド 4-38

索引付きファイル

アクセス・パス 6-27

概要説明 6-27

有効な検索指数 6-28

削減、オブジェクト・サイズの 4-40, 5-5

削除、活動化グループの 4-59

作成、サービス・プログラムの

概要 4-41

方針 4-42

作成、ソース物理ファイルの 4-1

作成、デバッグ・ビューの

ステートメント 5-8

リスト 5-7

ルート・ソース 5-6

COPY 5-6

作成、モジュールの

概説 4-24

CRTRPGMOD の使用 4-25

CRTRPGMOD のデフォルトの値の使用 4-27

作成方針、ILE プログラムの 3-17

サブストリング、文字または図形リテラルの

ILE デバッグ組み込み %SUBSTR 5-47

サブファイル

概要説明 6-70, 6-71

記述 6-70

使用 6-72

使用可能なファイル命令コード 6-71

制御レコード様式 6-70

例 6-72

レコード様式 6-70

サブフィールド

ファイル情報データ構造の場合 5-93, 5-95

プログラム状況データ構造の場合 5-92

PRTCTL の場合 6-60

サブプロシージャ

概要 3-27

コーディング上の考慮事項 3-39

コンパイラー・リストの情報 7-69

ステップイン 5-33

ステップオーバー 5-32

ダンプ・リストの内部データ 5-98

デバッグ 5-35

ファイルの有効範囲 4-32

戻り 4-104

例 3-8

論理フロー 3-4

サブプロシージャからの戻り 4-104

サブルーチン

エラー 5-70

サブルーチン (続き)

ファイル・エラー (INFSR) 5-71

プログラム・エラー処理 (*PSSR) 5-73

ループの防止 5-77

SUBR ルーチンの呼び出し 4-106

参考文献 8-1

式

戻り値 4-83

識別、活動化グループの 4-58

識別情報の除去 4-40

資源再利用 (RCLRSC) コマンド

記憶域を解放するため 4-60

ILE プログラム 3-20

OPM 互換プログラム 3-18

試行変換、実行 7-16

試行変換の実行 7-16

システム応答リスト

追加 4-56

変更 4-57

システム間通信機能 (ICF) 6-66

システム機能

スプーリング 6-18

実行時エラー、コンパイラー・リストで訂正 4-21

実行時エラーの訂正 4-21

実行時照会メッセージに対する応答 4-56

実行時の記憶域の管理 4-60

実行時配列

実行時の記憶域の割り振り 4-68

指定、エラー標識の 5-67

指定、活動化グループの 4-58

指定、コンパイラー・リストの形式の 4-17

指定変更、外部記述の 6-8

指定変更、ファイルの

概要説明 6-3

定義 6-3

終了、コミットメント制御の 6-46

終了、プログラムまたはプロシージャの

異常終了 4-103

システム呼び出し後 4-57

正常終了 4-102

バインド可能 API の使用 4-104

未終了の戻り 4-103

戻りの概要 4-101

出力

仕様書

プログラム記述 WORKSTN ファイル 6-74

出力仕様書

外部記述での 6-9

概要説明 3-1

プログラム記述 WORKSTN ファイル 6-74

例 3-7

出力スプーリング 6-19

出力バッファー位置、コンパイラー・リストでの 7-65

出力ファイル 6-75

出力レコード

ブロック化 6-26

順次のみの処理 6-32, 6-33
順次ファイル 6-29
順序検査
 入力仕様書 6-11
照会、呼び出されたプログラム / プロシージャーの名前の
 4-99
照会メッセージ
 応答 4-56
 リスト 4-56
状況コード
 データ管理エラー 7-4
消去コマンド 6-69
条件処理プログラム 5-57
 概要 5-80
 再帰呼び出し 5-80
 例 5-80
 例外のパーコレート 5-81
 レジスター 5-80
条件付きコミットメント制御、指定 6-49
条件付き停止点
 ジョブの設定および除去 5-20
 ステートメント番号の使用 5-23
 スレッドの設定および除去 5-25
 設定 5-21
 定義 5-16
条件付け出力
 オーバーフロー標識 6-55
仕様書
 外部記述ファイル 6-5
 順序 3-1
 説明 3-1
 タイプ 3-1
 ファイル記述 6-5
 レコード様式 6-21
仕様書テンプレート、挿入 7-18
小数点以下の桁数
 入力仕様書
 外部記述での 6-8
 プログラム記述 WORKSTN ファイル 6-74
省略されたパラメーター 4-88
 *OMIT 4-89
省略された引数の検査 (CEETSTA) 4-89
除去、停止点の
 概要 5-16
 条件付きジョブ・ブレイクポイント 5-20
 条件付きスレッド・ブレイクポイント 5-25
 ステートメント番号の使用 5-23
 すべて 5-26
 無条件ジョブ・ブレイクポイント 5-17
 無条件スレッド・ブレイクポイント 5-19
除去、プログラム内のエラーの 5-1
処理、例外 / エラー
 一般的な考慮事項 5-63
 エラー / 例外処理サブルーチンの概要 5-70
 エラー標識 5-67
 概要 5-58

処理、例外 / エラー (続き)
 最適化に関する考慮事項 5-66
 条件処理プログラム 5-80
 タイプ 5-57
 取り消し処理プログラム 5-86
 パーコレーション 5-59
 ファイル・エラー / 例外 (INFSR) サブルーチン 5-71
 未処理 5-64
 戻り点の指定 5-79
 ループの防止 5-77
 ILE RPG と OPM RPG/400 との相違点 5-63, 7-3
 NOOPT キーワード 5-66
 RPG 特有 5-61
 *PSSR エラー処理サブルーチン 5-73
 'E' 拡張の使用 5-67
処理方式
 キーによらない 6-43
 キーによる順次 6-33
 キーによるランダム処理 6-38
 限界内順次 6-40
 順次のみ 6-33, 6-43
 相対レコード番号 6-42
 連続 6-32
 DISK ファイルの場合 6-31
 WORKSTN ファイル 6-69, 6-76
診断、プログラム内のエラーの 5-1
スタック、呼び出し 4-78, 5-58
ステートメント・ビュー
 作成 5-8
 デバッグのための使用 5-23
ステップ、デバッグの
 プログラムまたはプロシージャーに対する 5-32
 プログラムまたはプロシージャーの中へ 5-33
 プログラムを通じた 5-31
ストリング引数に関する記述情報の取得 (CEESGI) 4-88
スプリング 6-18
スレッド・アプリケーション
 概要 3-15
 コーディング上の考慮事項 4-108
 デバッグ 5-16
 プロシージャーのロックとアンロック 4-110
制御境界 5-59
制御言語 (CL) プログラム
 よく使用されるコマンド 3-10
 ILE RPG で使用されるコマンド 3-10
 ILE プログラム内のモジュールとして 3-21
 OPM 互換アプリケーション・プログラム 3-17
制御仕様書
 概要説明 3-1
 変換に関する考慮事項 7-10
 例 3-6
制御仕様書キーワード
 コンパイル・オプション・キーワード
 コンパイラー・リストの例 7-57
制御中断
 例 6-56

制御レコード様式、サブファイル 6-70
正常プログラム / プロシーチャー 終了 4-102
整数形式
 TRUNCNBR パラメーター 7-46
静的プロシーチャー呼び出し 4-77
静的呼び出し 3-14, 4-77
設定、停止点の
 概要 5-16
 条件付きジョブ・ブレイクポイント 5-20
 条件付きスレッド・ブレイクポイント 5-25
 ステートメント番号の使用 5-23
 無条件ジョブ・ブレイクポイント 5-17
 無条件スレッド・ブレイクポイント 5-19
 例 5-18, 5-21
説明、パラメーターの
 CRTBNDRPG コマンド 7-37
 CRTRPGMOD コマンド 7-55
 CVTRPGSRC コマンド 7-12
前ページ・コマンド・キー 6-69
ソースの、データ・ファイルからの、変換 7-18
ソースの編集 (STRSEU) コマンド 4-2
ソース物理ファイル、作成 4-1
ソース・デバッガーの各国言語サポート (NLS) 5-52
ソース・デバッグ
 ウォッチ条件の設定 5-26
 開始、ソースの 5-8
 概説 5-1
 概要 3-15
 可変長形式でのフィールドの表示 5-43
 簡略名の表示 5-52
 国別言語サポート 5-52
 組み込み関数
 概説 5-46
 例 5-47
 %ADDR 5-46
 %INDEX 5-46
 %SUBSTR 5-46
 %SUBSTR を使用した値の変更 5-48
 %VARs 5-46
最適化の影響 4-39, 5-2
ステップスルー 5-31
セッションからのオブジェクトの除去 5-11, 5-12
セッションへのオブジェクトの追加 5-11
ソースの表示 5-13
属性の表示 5-50
データおよび式の表示 5-36
テーブルの内容の表示 5-39
定様式ダンプの入手 5-91
デバッグ中のモジュールの変更 5-13
デバッグ用のプログラムの作成 4-12
デバッグ・オプションの設定 5-10
デバッグ・セッションでの OPM プログラムの限界 5-11
配列の内容の表示 5-39
標識の表示 5-42
フィールド値の変更 5-48
複数発生のデータ構造の表示 5-40

ソース・デバッグ (続き)
 ブレイクポイントの設定と除去 5-16
 プログラムの準備 5-4
 プロダクション・ファイルの更新 5-10
 ポインターがアドレス指定するデータの表示 5-43
 文字形式でのフィールドの表示 5-42
 予期しない結果 5-38
 リスト出力オプションによる調整 4-21
 16 進値としてのフィールドの表示 5-42
 EVAL を使用して値を割り当てるための規則 5-48
 ILE RPG と OPM RPG/400 との相違点 7-3
 NLSS に関する考慮事項 5-22
 UCS-2 形式でのフィールドの表示 5-43
ソース・プログラム
 いくつかのメンバーの変換 7-16
 システムへの入力 4-1
 すべてのメンバーの変換 7-16
 変換時のソース・メンバー・タイプ 7-7
 変換時のファイル名およびメンバー名 7-8
 変換時のレコード長 7-8
 報告書簡易作成機能ソース・メンバーの変換 7-17
 ILE RPG への変換 7-9
 SQL ステートメントの入力 4-5
ソース・メンバー・タイプ、変換 7-7
相互参照表 7-67
操作記述子
 定義 4-87
 例 4-44
操作記述子情報の検索 (CEEDOD) 4-44
 操作記述子 4-88
 例 4-88
相対レコード番号 6-30
相対レコード番号による処理 6-42
装置
 WORKSTN 6-66
装置ファイル
 概説 6-52
 装置依存性 6-1
 装置独立性 6-1
 複数装置 6-76
 ワークステーション・ファイル 6-66
 DISK ファイル 6-20
 PRINTER ファイル 6-53
 SEQ ファイル 6-62
装置名、機能 6-2
挿入、仕様書テンプレートの 7-18
即時変換の実行 7-15

[タ行]

代替照合順序
 デバッグに関する考慮事項 5-22
ダンプ、定様式 5-91
逐次化されたプロシーチャー 4-112
通信
 他のプログラムおよびシステムへのアクセス 6-66

次ページ・コマンド・キー 6-69

データ域

RETURNCODE 4-22

データ管理命令 6-11

データ構造

サブフィールド

デバッグ中の表示 5-40

変換上の問題 7-30

複数回繰り返し

デバッグ中の表示 5-40

EVAL デバッグ・コマンドの使用 5-40

データベース・データ

デバッグ中の更新 5-10

データベース・ファイル

概説 6-20

ソース・メンバー 6-20

データ・ファイル 6-20

フィールド・レベル記述 6-20

物理および論理ファイル 6-20

レコード・レベル記述 6-20

データ・ファイル、ソースの変換 7-18

テーブル

デバッグ中の表示 5-39

テーブル、パラメーターの

CRTBND RPG コマンド 4-11

CRTRPGMOD コマンド 4-25

CVTRPGSRC コマンド 7-10

テープ・ファイル 6-29

定義仕様書

概要説明 3-1

停止 (H1-H9) 標識

プログラム / プロシーチャーの終了に使用 4-102, 4-103

停止点

条件付き

ジョブの設定および除去 5-20

スレッドの設定および除去 5-23

ステートメント番号を使用した設定 5-23

すべての除去 5-26

設定と除去 5-16

テスト 5-17

無条件

ジョブの設定および除去 5-17

スレッドの設定および除去 5-19

定様式ダンプ 5-91

テスト・ライブラリー、使用法 5-10

手操作のコード変換 7-25

デバッグ

ウォッチ条件の設定 5-26

開始、ソースの 5-8

概説 5-1

概要 3-15

可変長形式でのフィールドの表示 5-43

簡略名の表示 5-52

国別言語サポート 5-52

組み込み関数

概説 5-46

デバッグ (続き)

組み込み関数 (続き)

例 5-47

%ADDR 5-46

%INDEX 5-46

%SUBSTR 5-46

%SUBSTR を使用した値の変更 5-48

%VARS 5-46

最適化の影響 4-39, 5-2

ステップスルー 5-31

セッションからのオブジェクトの除去 5-11, 5-12

セッションへのオブジェクトの追加 5-11

ソースの表示 5-13

属性の表示 5-50

データおよび式の表示 5-36

テーブルの内容の表示 5-39

定様式ダンプの入手 5-91

デバッグ中のモジュールの変更 5-13

デバッグ用のプログラムの作成 4-12

デバッグ・オプションの設定 5-10

デバッグ・セッションでの OPM プログラムの限界 5-11

配列の内容の表示 5-39

標識の表示 5-42

フィールド値の変更 5-48

複数発生 of データ構造の表示 5-40

ブレークポイントの設定と除去 5-16

プログラムの準備 5-4

プロダクション・ファイルの更新 5-10

ポインターがアドレス指定するデータの表示 5-43

文字形式でのフィールドの表示 5-42

予期しない結果 5-38

リスト出力オプションによる調整 4-21

16 進値としてのフィールドの表示 5-42

EVAL を使用して値を割り当てるための規則 5-48

ILE RPG と OPM RPG/400 との相違点 7-3

NLSS に関する考慮事項 5-22

UCS-2 形式でのフィールドの表示 5-43

デバッグ、ビューの

コピー・ソース 5-6

ステートメント 5-8

定義 5-4

デバッグ中の変更 5-15

デフォルト 5-8

リスト 5-7

ルート・ソース 5-6

デバッグ開始 (STRDBG) コマンド 5-8

プロダクション・ファイルの更新 (UPDPROD) パラメータ
ー 5-10

デバッグ終了 (ENDDBG) コマンド 5-8

デバッグ中のソースの表示 5-13

デバッグ中のデータおよび式の表示 5-36

デバッグ中のフィールド値の変更 5-48

デバッグのためのプログラムの準備 5-4

デバッグ・オプションの設定 5-10

デバッグ・コマンド

概説 5-3

デバッグ・コマンド (続き)

デバッグ中の名前との等値化 5-51

ATTR 5-50

CLEAR 5-18

DISPLAY 5-14

EQUATE 5-51

EVAL 5-36, 5-48

STEP 5-31, 5-32

STEP INTO 5-33

STEP OVER 5-32

WATCH 5-26

デバッグ・セッションからのオブジェクトの除去 5-11

デバッグ・セッションへのオブジェクトの追加 5-11

デバッグ・データ

オブジェクト・サイズに対する影響 5-5

作成 5-4

なし 5-5

モジュールからの除去 4-40

デバッグ・データなし 5-5

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 4-21

デフォルトの活動化グループ 3-17, 3-25, 4-59

実行 4-59

デフォルトの例外処理プログラム、RPG 5-61

デフォルト・ヒープ 4-62

テラスペース 4-107

テンプレート、仕様書テンプレートの挿入 7-18

統合化言語環境 (ILE)

影響

CRTBNDRPG を使用するプログラム 3-20

OPM 互換プログラム 3-18

概要 3-11

言語間呼び出し 4-97

言語間呼び出しに関する考慮事項 4-98

プログラムの管理 3-13

プログラムの作成 3-12

プログラムの作成方針 3-17, 3-19, 3-21

プログラムの内部構造 4-33

プログラムの呼び出し 3-14

ILE コンパイラーのファミリー 3-11

ILE プログラムの終了 4-57

動作、バインドされた ILE RPG モジュールの 4-32

動作上の相違点、OPM RPG/400 と ILE RPG との 7-1

到着順アクセス・パス 6-21

動的記憶域 4-60

動的配列

実行時の記憶域の割り振り 4-68

動的呼び出し 3-14, 4-77

特殊なコマンド・キー 6-69

特殊なルーチン、呼び出し 4-106

取り消し処理プログラム 5-57

使用 5-86

例 5-87

CEERTX (呼び出しスタック項目終了ユーザー出口プロシージャの登録) 5-86

取り消し処理プログラム (続き)

CEEUTX (呼び出しスタック項目終了ユーザー出口プロシージャ) 5-86

[ナ行]

長さ、ファイルのレコードの、変換に関する考慮事項 7-8

名前付き活動化グループ 4-58

名前変更、フィールドの 6-6

名前変更、レコード様式名の 6-6

入出力共用ファイル 6-75

入力

ファイル 6-75

入力仕様書

概要説明 3-1

入力レコード

非ブロック化 6-26

ネストされた例外 5-64

[ハ行]

配列

実行時前配列 7-32

デバッグ中の表示 5-39

変換上の問題 7-30

ロード 7-32

バインダー言語

使用の理由 4-42

例 4-47

バインダー・リストの作成 4-37

バインディング・エラー、コンパイラー・リストの 7-71

バインド

定義 4-33

プログラム内のモジュール 4-33

プログラムに対するサービス・プログラム 4-48

モジュールの変更後 4-38

バインド RPG プログラムの作成 (CRTBNDRPG) コマンド

および ILE 3-13

機能ごとにグループ化されたパラメーター 4-11

構文図 7-34

使用 4-10

デバッグ・ビューによるリスト出力オプションの調整 4-21

デフォルトのパラメーター値 4-11

パラメーターの説明 7-37

プログラムの作成 4-10

プログラムの作成方針 3-17, 3-19

例

静的バインドによるプログラム 4-13

ソース・デバッグ用のプログラム 4-12

OPM 互換プログラム 4-14

RETURNCODE データ域 4-22

バインド可能な API

概要 3-15

記憶域解放 (CEEFRST) 3-15

記憶域再割り振り (CEECZST) 3-15

バインド可能な API (続き)

- サンプル・コーディング 4-105
- ストリング引数に関する記述情報の取得 (CEESGI) 4-88
- 説明 4-105
- 操作記述子情報の検索 (CEEDOD) 4-88
- 操作記述子の受け渡し 4-87
- ヒープ記憶域取得 (CEEGTST) 3-15, 4-69
- ヒープ作成 (CEECRHP) 3-15, 4-69
- ヒープ廃棄 (CEEDSHP) 3-15, 4-69
- プロシージャーからの戻り 4-104
- 呼び出し規則 4-105
- CEE4ABN 4-104
- CEECRHP (ヒープ作成) 3-15, 4-69
- CEEZST (記憶域再割り振り) 3-15
- CEEDSHP (ヒープ廃棄) 3-15, 4-69
- CEEFRST (記憶域解放) 3-15
- CEEGTST (ヒープ記憶域取得) 3-15, 4-69
- CEEHDLR (ILE 条件処理プログラム登録) 5-80
- CEEHDLU (ILE 条件処理プログラム登録解除) 5-80
- CEERTX (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の登録) 5-86
- CEETREC 4-104
- CEETSTA (省略された引数の検査) 4-89
- CEEUTX (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) 5-86

パフォーマンス上のヒント

- プログラムの呼び出し 4-103
- LR オンの呼び出し 7-3

パフォーマンスに関する考慮事項

- サブルーチン対サブプロシージャー 4-44

パラメーター

- 受け渡し 4-80
- 指定 4-100
- 省略された 4-88
- 操作記述子 4-87
- データ・タイプ一致要件 4-87
- 渡される数の検査 4-90
- CL CALL コマンドを使用した受け渡し 4-52

パラメーターの受け渡し

- 値による 4-85
- 概要 4-80
- 参照による 4-85
- 省略されたパラメーター 4-88
- 少ないデータの受け渡し 4-95
- 操作記述子 4-87
- データ・タイプ一致要件 4-87
- 読み取り専用参照による 4-85
- 例 4-52
- 渡される数の検査 4-90
- CL CALL コマンドの使用 4-52
- ILE 言語の受け渡し方法 4-97
- PARM の使用 4-100
- PLIST の使用 4-101

パラメーターの説明

- CRTBNDRPG コマンド 7-37
- CRTRPGMOD コマンド 7-55

パラメーターの説明 (続き)

- CVTRPGSRC コマンド 7-12

パラメーター・テーブル

- CRTBNDRPG コマンド 4-11
- CRTRPGMOD コマンド 4-25
- CVTRPGSRC コマンド 7-10

パラメーター・リスト

- 識別 4-80
- 指定に関する規則 4-101
- PARM により作成される 4-101

ヒープ

- 定義 4-60
- デフォルト・ヒープ 4-62
- 例 4-68

ヒープ記憶域取得 (CEEGTST) バインド可能 API 3-15, 4-69

ヒープ作成 (CEECRHP) バインド可能 API 3-15, 4-69

ヒープ廃棄 (CEEDSHP) バインド可能 API 3-15, 4-69

引き渡し、ファイル・パラメーターの 4-96

ビューのデバッグ

- コピー・ソース 5-6
- ステートメント 5-8
- 定義 5-4
- デバッグ中の変更 5-15
- デフォルト 5-8
- リスト 5-7
- ルート・ソース 5-6

評価の順序

- プロトタイプ呼び出しで 4-96

標識

- エラー 5-67
- エラー標識として 5-67
- オーバーフロー
 - 有無 6-55
 - 概要説明 6-53
 - 設定 6-57
 - フェッチ・オーバーフロー論理 6-57
 - プログラム・サイクルとの関係 6-57
 - 例 6-56, 6-57
- PRINTER ファイルによる 6-53
- 機能キー (KA ~ KN, KP ~ KY)
 - WORKSTN ファイルで 6-68

最終レコード (LR)

- 概要説明 3-4
- プログラム / プロシージャーの終了に使用 4-102, 4-103
- 使用 3-4
- 停止 (H1-H9)
 - プログラム / プロシージャーの終了に使用 4-102, 4-103
- デバッグ中の表示 5-42
- 戻り (RT)
 - プログラム / プロシージャーの終了に使用 4-102, 4-103

ファイル

- 一時変更 6-8
- 一般的な考慮事項 6-1

ファイル (続き)

- オープン・オプション 6-17
- 外部記述 6-1
- 外部記述ディスク 6-21
- 共用 6-17
- 索引付き 6-27
- 指定変更 6-3
- 処理図
 - 順次ファイル 6-62
 - SPECIAL ファイル 6-64
 - WORKSTN ファイル 6-76
- 装置依存性 6-1
- 装置独立性 6-1
- 名前
 - 一時変更 6-8
 - 外部記述 6-1
 - プログラム記述 6-11
- プログラム記述 6-1, 6-11
- 命名規則 6-3
- 有効なキー 6-24
- ロック 6-15
- DISK 6-20
- ILE RPG と OPM RPG/400 との相違点 7-3
- PRINTER 6-53
- SEQ 6-29, 6-62
- WORKSTN 6-66
- ファイル一時変更 6-8
 - 概説 6-13, 6-43
 - コンパイラー・リストに示された 7-58
 - 例 6-14
- ファイル仕様書
 - 外部記述ファイルの場合 6-5
 - 概要説明 3-1
 - コミットメント制御 6-47
- ファイル仕様書 (補足 E)
 - 変換上の問題 7-21, 7-29
- ファイル情報データ構造
 - エラー処理サブルーチンでの使用 5-71
 - 例 5-71
- ファイルの共用 6-17
- ファイルのレコード長、変換に関する考慮事項 7-8
- ファイルのロック 6-15
- ファイル命令
 - 順次ファイルに使用できる 6-62
 - DISK ファイルに使用できる 6-44
 - PRINTER ファイルに使用できる 6-53
 - SPECIAL ファイルに使用できる 6-64
 - WORKSTN ファイルで使用可能 6-76
- ファイル例外 / エラー
 - 定義 5-61
 - 例 5-71
 - INFSR サブルーチンの使用 5-71
- ファイル例外 / エラー処理サブルーチン (INFSR)
 - 仕様書 5-71
 - 説明 5-71
 - 例 5-71

フィールド

- デバッグ中の値の変更 5-48
- デバッグ中の現在の値のメンテナンス 5-2
- デバッグ中の属性の表示 5-50
- デバッグ中の名前との等値化 5-51
- デバッグ中の表示
 - 可変長 形式での 5-43
 - 文字形式で 5-42
 - 16 進値として 5-42
 - EVAL の使用 5-36
 - UCS-2 形式での 5-43
- フィールド、式、またはコマンドと名前の等値化 5-51
- フィールド参照ファイル、例 6-22
- フィールドの属性の表示 5-50
- フィールド名の変更 6-6
- フェッチ・オーバーフロー
 - 概要説明 6-57
 - 論理 6-57
- 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る場合 6-48
- 複数装置ファイル
 - WORKSTN 6-76
- 複数モジュールのバインド 4-36
- 符号のない整数形式
 - TRUNCNBR パラメーター 7-46
- 物理ファイル 6-20
- 部分キー 6-25
- ブレイクポイントのテスト 5-17
- フローチャート
 - フェッチ・オーバーフロー・ルーチン論理 6-57
- プログラミングのヒント
 - サブプロシージャー・ブレイクポイントの設定 5-34
 - NOMAIN モジュールの作成 4-42
- プログラム
 - 異常終了 4-103
 - ウォッチ条件の設定 5-26
 - 関連する CL コマンド 4-37
 - 高機能 ILE 3-24
 - 更新 4-39
 - 混合言語 3-23
 - サイズに対するデバッグ・データの影響 5-5
 - サイズの縮小 4-40
 - 最適化レベルの変更 4-39
 - 識別情報の除去 4-40
 - 式を使用する呼び出し 4-83
 - 資源の解放 4-60
 - 実行 4-51
 - 実行、OPM デフォルトの活動化グループの 4-59
 - 終了 4-57
 - ステップイン 5-33
 - ステップオーバー 5-32
 - ステップスルー 5-31
 - 正常終了 4-102
 - ソース・ステートメントの入力 4-1
 - 単一言語 3-22
 - ILE の影響 3-20

プログラム (続き)

- デバッグ中のソースの表示 5-13
- デバッグ中の変更 5-13
- デバッグの準備 5-4
- 内部構造 4-33
- パラメーターの受け渡し 4-80
- 複数モジュール
 - 一般的な作成方針 3-21
- プログラム入力プロシージャ 4-33
- 別のデバッグ・ビュー 5-15
- 変更 4-38
- 未終了の戻り 4-103
- メニュー方式アプリケーションからの実行 4-54
- モジュールのバインド 4-33
- 戻り 4-101
- ユーザー作成コマンドを使用した実行 4-56
- 呼び出し 4-75, 4-77
- 例 3-5
- CALL 命令を使用した呼び出し 4-98
- CALLP 命令を使用する呼び出し 4-83
- OPM 互換
 - 作成方法 3-17
 - プログラムの作成方針 3-17, 3-25
 - 例 3-17
 - ILE の影響 3-18
- SQL ステートメントの入力 4-5
- プログラム / プロシージャの異常終了 4-103
- プログラム / プロシージャの終了
 - 異常終了 4-103
 - システム呼び出し後 4-57
 - 正常終了 4-102
 - バインド可能 API の使用 4-104
 - 未終了の戻り 4-103
 - 戻りの概要 4-101
- プログラム / プロシージャ呼び出し
 - 概要 4-76
 - グラフィックスの呼び出し 4-106
 - 言語間呼び出し 4-97
 - 固定形式の呼び出し 4-98
 - 再帰呼び出し 4-78
 - 自由形式呼び出し 4-83
 - 正常プログラム / プロシージャ 終了 4-102
 - 静的呼び出し 4-77
 - 特殊なルーチンの呼び出し 4-106
 - バインド可能 API の呼び出し 4-105
 - パラメーターの受け渡し方法 4-84
 - プログラム / プロシージャの異常終了 4-103
 - プロシージャの呼び出し 4-77
 - 未終了の戻り 4-103
 - 戻り値 4-83
 - 呼び出されたプログラムまたはプロシージャからの戻り 4-101
 - 呼び出し側プログラム 4-77
 - 呼び出しスタック 4-78
 - CALL 命令の使用 4-98
 - CALLB 命令の使用 4-98

プログラム / プロシージャ呼び出し (続き)

- ILE における 3-14
- プログラム記述ファイル
 - 定義 6-3
 - 物理および論理ファイル 6-20
 - 有効な検索指数 6-28
 - DISK ファイルとして 6-27
 - WORKSTN ファイルとして 6-73, 6-74, 6-75
- プログラム更新 (UPDPGM) コマンド
 - 使用 4-39
- プログラム作成のワン・ステップ処理 4-10
- プログラム参照表示 (DSPPGMREF) コマンド 4-99
- プログラム識別情報 4-40
- プログラム状況データ構造
 - エラー処理サブルーチンでの使用 5-73
 - 例 4-100, 5-74
- プログラムの活動化 4-57
- プログラムの管理 3-13
- プログラムの作成
 - コーディング上の考慮事項 3-38
 - 方針 3-17
 - 避けるべき方針 3-25
 - ワン・ステップ処理の使用 4-10
 - CRTBNDRPG の使用 3-19
 - CRTPGM コマンド 4-34
 - CRTRPGMOD および CRTPGM の使用 4-24
 - CRTRPGMOD を使う ILE アプリケーション 3-21
 - OPM 互換 3-17
 - 例 4-12, 4-13, 4-14, 4-36
- OPM 互換
 - 作成 3-17
 - 避けるべき方針 3-25
- プログラムの作成 (CRTPGM) コマンド 3-21
 - および ILE 3-13
 - システムの処置 4-35
 - 使用 4-34
 - パラメーター 4-35
 - プログラムの作成 4-24
 - 例 4-48
 - 複数モジュールのバインド 4-36
- プログラムの実行
 - 概要 4-51
 - メニュー方式アプリケーションからの 4-54
 - ユーザー作成コマンドの使用 4-56
 - CL CALL コマンドの使用 4-52
 - ILE RPG と OPM RPG/400 との相違点 7-1
 - OPM デフォルト活動化グループでの 4-59
- プログラム表示 (DSPPGM) コマンド
 - 最適化レベルの決定 4-40
- プログラム変更 (CHGPGM) コマンド
 - 最適化パラメーター 4-40
 - 識別情報の除去 4-40
- プログラム名
 - *FROMMBR パラメーター 7-14
- プログラム例外 / エラー
 - 定義 5-61

プログラム例外 / エラー (続き)
ループの防止 5-77
例 5-74, 5-80
*PSSR サブルーチンの使用 5-73
プログラム例外 / エラー処理サブルーチン
説明 5-73
例 5-74
プログラム・エントリー・プロシージャー (PEP)
および呼び出しスタック 4-78
決定 4-34
定義 4-33
プログラム・サイクル
概要説明 3-2
コミットメント制御 6-50
最後のサイクル 3-3
フェッチ・オーバーフロー論理 6-57
プロシージャー
異常終了 4-103
ステップオーバー 5-32
正常終了 4-102
静的プロシージャー呼び出し 4-77
ダンプ情報 5-91
パラメーターの受け渡し 4-80
プロシージャー・ポインター呼び出し 4-77
未終了の戻り 4-103
戻り 4-101
呼び出し 4-75
プロシージャー・ポインター呼び出し 4-77
プロトタイピング、Java メソッド 4-133
プロトタイプ
使用 4-82
説明 3-28
プロトタイプ呼び出し
パラメーターの評価の順序 4-96
プロトタイプ・プログラムまたはプロシージャー
プロトタイプ呼び出し 3-28
分散データ管理 (DDM)
ファイル 6-51
文書化、プログラムの 4-21
分類順序
コンパイラー・リストの ALTSEQ テーブル 7-65
デバッグに関する考慮事項 5-22
SRTSEQ パラメーターの影響 6-19
ページ番号、PRINTER ファイル中 6-54
ページ見出し 4-17
ページ・オーバーフロー、PRINTER ファイル中 6-54
別のビュー、モジュールの 5-15
ヘルプ・コマンド・キー 6-69
変換の分析 7-21
変換報告書
使用 7-21
セクション 7-21
入手 7-17
変換報告書の入手 7-17
変更、サービス・プログラムの 4-43
変更、プログラムの 4-38

変更、モジュールの 4-38
ホーム・コマンド・キー 6-69
報告書簡易作成プログラム
ILE RPG への変換 7-17

[マ行]

未終了の戻り 4-103
未処理エスケープ・メッセージ 5-64
未処理機能チェック 5-65
未処理例外 5-64
ミシン線への印刷防止 6-57
無条件停止点
ジョブの設定および除去 5-17
ステートメント番号の使用 5-23
スレッドの設定および除去 5-19
設定 5-18
定義 5-16
命令コード 6-76
概説 3-4
順次ファイルに使用できる 6-62
DISK ファイルに使用できる 6-44
PRINTER ファイルに使用できる 6-53
SPECIAL ファイルに使用できる 6-64
メンテナンス、OPM 互換性の 4-14, 4-59
メイン・プロシージャー
概要 3-27
コーディング上の考慮事項 3-39
ファイルの有効範囲 4-32
戻り 4-102
メッセージ
インライン診断 4-19
照会
応答 4-56
追加の診断メッセージ 4-20
例外
タイプ 5-58
未処理 5-64
例 5-64
メッセージ要約、コンパイラー・リストの 7-70
メモリー管理命令
ALLOC (記憶域割り振り) 命令コード 4-62
DEALLOC (記憶域解放) 命令コード 4-62
REALLOC (新しい長さの記憶域再割り振り) 命令コード 4-62
%ALLOC 組み込み関数 4-62
%REALLOC 組み込み関数 4-62
文字形式
デバッグ中の表示 5-42
文字 CCSID
コンパイラー・リストに示された 7-60
モジュール
入り口モジュールの決定 4-34
概要 4-24
関連する CL コマンド 4-32
サイズに対するデバッグ・データの影響 5-5

モジュール (続き)

- サイズの縮小 4-40
- 最適化レベルの変更 4-39
- 作成 4-24
- 識別情報の除去 4-40
- ダンプ・リストの情報 5-91
- デバッグ中のソースの表示 5-13
- デバッグ中の変更 5-13
- デバッグの準備 5-4
- バインドされた ILE RPG の動作 4-32
- 複数のバインド 4-36
- 複数プロシージャー・モジュールの概要 3-27
- プログラムとの関係 4-33
- プログラム内の置き換え 4-39
- プログラムへのバインド 4-33
- 別のデバッグ・ビュー 5-15
- 変更と再バインド 4-38
- CRTRPGMOD コマンド 4-25
- NOMAIN モジュールの作成 4-27

モジュール作成

- 概説 4-24
- CRTRPGMOD の使用 4-25
- CRTRPGMOD のデフォルトの値の使用 4-27

モジュールの識別情報 4-40

モジュールのデバッグ・ビューの変更 5-15

モジュールの変更 (CHGMOD) コマンド 4-40

- 識別情報の除去 4-40

モジュール表示 (DSPMOD) コマンド 4-99

モジュール・ソース表示 (DSPMODSRC) コマンド 5-10, 5-11, 5-12, 5-14

戻り (RT) 標識

- プログラム / プロシージャーの終了に使用 4-102, 4-103

戻り、メイン・プロシージャーからの 4-102

戻り、呼び出されたプロシージャーからの 4-101

戻り、呼び出されたメイン・プロシージャーからの 4-101

戻り値

- 式を使用する戻り 4-83

戻り状況 パラメーター 6-64

戻り点、ENDSR での指定 5-79

戻り点の指定 5-79

[ヤ行]

ユーザー入り口プロシージャー (UEP)

- および呼び出しスタック 4-78

- プログラムでの役割 4-33

ユーザー作成コマンド、RPG プログラムの実行 4-56

有効なキー

- ファイルの場合 6-24
- レコードの場合 6-24

有効なファイル命令

- SPECIAL ファイル 6-64

有効範囲

- ファイル 4-32

要件、変換援助プログラムの 7-9

様式名 6-74

要約表

順次ファイル処理 6-62

使用可能なファイル命令コード

- 順次 6-62

- DISK 6-43

- PRINTER 6-53

- SPECIAL 6-64

- WORKSTN 6-76

SPECIAL ファイル処理 6-64

呼び出し

- Java からの RPG 4-144

- RPG からの Java 4-136

呼び出し、プログラム / プロシージャーの

- 概要 4-76

- グラフィックスの呼び出し 4-106

- 言語間呼び出し 4-97

- 固定形式の呼び出し 4-98

- 再帰呼び出し 4-78

- 自由形式呼び出し 4-83

- 正常プログラム / プロシージャー 終了 4-102

- 静的呼び出し 4-77

- 特殊なルーチンの呼び出し 4-106

- バインド可能 API の呼び出し 4-105

- パラメーターの受け渡し方法 4-84

- プログラム / プロシージャーの異常終了 4-103

- プロシージャーの呼び出し 4-77

- 未終了の戻り 4-103

- 戻り値 4-83

- 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 4-101

- 呼び出し側プログラム 4-77

- 呼び出しスタック 4-78

- CALL 命令の使用 4-98

- CALLB 命令の使用 4-98

- ILE における 3-14

呼び出しスタック 4-78, 5-58

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) 5-86

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の登録

- (CEERTX) 5-86

呼び出し命令

- 固定形式の呼び出し 4-98

- 自由形式呼び出し 4-83

- 使用 4-83

- 特殊なルーチン 4-106

- 呼び出されたプロシージャーの名前の照会 4-99

- 呼び出し側プログラム 4-98

- DSPPGMREF 4-99

読み取り、次のレコードの

- WORKSTN サブファイル 6-71

読み取り、レコードの 6-76

予約語

- *CANCL 5-79

- *DETC 5-79

- *DETL 5-79

- *GETIN 5-79

予約語 (続き)

*OFL 5-79

*TOTC 5-79

*TOTL 5-79

[ラ行]

ライブラリー、作成 4-1

ライブラリーの作成 4-1

リスト、コンパイラー

インライン診断メッセージ 4-19

形式の指定 4-17

構造化命令の字下げ 4-18

コンパイル・エラーの訂正 4-19

サンプル・リスト 7-57

実行時エラーの訂正 4-21

使用 4-15

セクション 4-16, 7-57

追加の診断メッセージ 4-20

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 4-21

デフォルトの情報 4-16

入手 4-16

文書化としての使用 4-21

読み取り 7-56

SEU を使用したブラウズ 4-20

リスト、バインド・プログラム

基本 4-50

サービス・プログラム内のエクスポートの決定 4-41

作成 4-37

セクション 4-37

保守資源として 4-38

リスト・ビューの作成 5-7

ルート・ソース・ビューの作成 5-6

ループ、エラー処理サブルーチンでの防止 5-77

例

コンパイル

サービス・プログラム 4-44

サンプル・バインダー・リスト 4-50

静的バインドによるプログラム 4-13

ソース・デバッグ用のプログラム 4-12

複数モジュールのバインド 4-36

OPM 互換プログラム 4-14

サブプロシージャ 3-32

NOMAIN モジュールの作成 4-27

サンプル ILE RPG プログラム 3-5

対話式アプリケーション 6-78

デバッグ

可変長形式でのフィールドの表示 5-43

条件式ブレークポイントの設定 5-21

セッションからのプログラムの除去 5-12

セッションへのサービス・プログラムの追加 5-11

テーブルの内容の表示 5-39

デバッグ用ソースの例 5-52

デバッグ・オプションの設定 5-10

デバッグ・セッションでの別のモジュールの表示 5-13

例 (続き)

デバッグ (続き)

配列の内容の表示 5-39

標識の表示 5-42

フィールド値の変更 5-49

フィールド値を表示するための %SUBSTR の使用 5-47

フィールドの属性の表示 5-50

複数発生データの構造の表示 5-40

ポインターがアドレス指定するデータの表示 5-43

無条件ブレークポイントの設定 5-18

文字形式でのフィールドの表示 5-42

モジュールのデバッグ・ビューの変更 5-15

16 進値としてのフィールドの表示 5-42

UCS-2 形式でのフィールドの表示 5-43

独自ヒープの管理 4-68

入出力

サブファイル処理 6-93

照会プログラム 6-79

データの維持 6-82

郵便番号による照会と名前による検索 6-101

複数プロシージャのあるモジュール 3-34

プログラム / プロシージャ呼び出し

省略されたパラメーターの使用 4-44

渡されるパラメーターの数の検査 4-90

例外の処理

エラー処理サブルーチンでのループの防止 5-77

条件処理プログラムの使用 5-80

取り消し処理プログラム 5-86

取り消し処理プログラムの使用 5-87

ファイル・エラー処理サブルーチン 5-71

未処理エスケープ・メッセージ 5-64

未処理機能チェック 5-65

*PSSR エラー処理サブルーチン 5-74

CL CALL コマンドを使用したパラメーターの受け渡し
4-52

RPG IV への変換

試行変換の実行 7-16

変換のサンプル 7-18

1 ファイル中のいくつかのメンバー 7-16

1 ファイル中のすべてのメンバー 7-16

例外

実行時のモニター 4-57

ネストされた 5-64

例外 / エラー処理

一般的な考慮事項 5-63

エラー / 例外処理サブルーチンの概要 5-70

エラー標識 5-67

概要 5-58

最適化に関する考慮事項 5-66

条件処理プログラム 5-80

タイプ 5-57

取り消し処理プログラム 5-86

パーコレーション 5-59

ファイル・エラー / 例外 (INFSR) サブルーチン 5-71

未処理 5-64

戻り点の指定 5-79

例外 / エラー処理 (続き)
ループの防止 5-77
例外 4-156
ILE RPG と OPM RPG/400 との相違点 5-63, 7-3
MONITOR グループ 5-68
NOOPT キーワード 5-66
RPG 特有 5-61
*PSSR エラー処理サブルーチン 5-73
'E' 拡張の使用 5-67
例外処理プログラム
優先順位 5-64
RPG 特有 5-61, 5-67
例外処理プログラムのタイプ 5-57
例外のパーコレーション 5-59
例外のパーコレート
条件処理プログラムの使用 5-81
例外メッセージ
タイプ 5-58
パーコレーション 5-59
未処理 5-64
CL MONMSG による予期しない処理 5-89
レコード
解除 6-16
限界 6-30
有効なキー 6-24
ロック 6-16
レコード長、ファイルの、変換に関する考慮事項 7-8
レコードの非ブロック化 / ブロック化 6-26
レコードのブロック化 / 非ブロック化 6-26
レコードのロック 6-16
レコード様式
外部記述ファイルの仕様書 6-21
サブファイル用 6-70
名前変更 6-6
無視 6-7
レコード様式の無視 6-7
レコード・アドレス・ファイル
限界値レコードによる 6-30
限界内順次 6-30
相対レコード番号 6-30
相対レコード番号による 6-30
変換上の問題 7-21, 7-29
レコード・ロック・タイムアウトで再試行 6-16
レベル検査 6-10
連続処理 6-32
ローカル変数
定様式ダンプ 5-98
ログ・ファイル
概要 7-9
使用 7-24
DDS 7-24
ロック
コミットメント制御の下 6-46
タイムアウトで再試行 6-16
独立型 6-16
ファイル 6-15

ロック (続き)
レコード・ロック待ち時間 6-16
ロックなしの読み取り 6-16
UNLOCK 6-16
ロング・ネーム
コンパイラ・リスト内の 7-67
論理ファイル
概要説明 6-20
複数様式 6-20

[ワ行]

割り振り、実行時配列の記憶域の 4-68

[数字]

01-99 標識
サンプルの定様式ダンプ中 5-98
デバッグ中の表示 5-42
16 進値、デバッグ中に表示される 5-42
2 ステップ処理、プログラム作成の 4-24
2 バイト文字セット
NLSS デバッグに関する考慮事項 5-22
RPG IV における文字フィールド 7-6

A

ACTGRP パラメーター
指定 4-58
使用 4-13
CRTBNDRPG コマンド 4-11, 7-48
CRTPGM コマンド 4-34
ALLOC (記憶域割り振り) 命令コード 4-62
all- thread 静的変数 4-111
ALWNULL パラメーター
CRTBNDRPG コマンド 4-11, 7-47
CRTRPGMOD コマンド 4-25
AS/400 SQL 用 DB2
SQL ステートメントの入力 4-5
ATTR デバッグ・コマンド
使用 5-50
定義 5-3
例 5-50
AUT パラメーター
CRTBNDRPG コマンド 4-11, 7-45
CRTRPGMOD コマンド 4-25

B

BIND プログラム・リスト
基本 4-50
サービス・プログラム内のエクスポートの決定 4-41
作成 4-37
セクション 4-37
保守資源として 4-38

BNDDIR パラメーター、CRTBNDRPG の
静的バインド 4-13
CRTBNDRPG コマンド 4-11, 7-48
CRTRPGMOD コマンド 4-25
BREAK デバッグ・コマンド
使用 5-18, 5-20, 5-24
定義 5-3
例 5-21

C

CALL CL コマンド
パラメーターの受け渡し 4-52
パラメーターの受け渡し例 4-52
プログラムの実行 4-52
CALL (プログラム呼び出し) 命令コード
使用 4-98
変換報告書 7-22
CALLB (バインド・プロシージャーの呼び出し) 命令コード
使用 4-98
呼び出し側プログラム 4-98
CALLP (プロトタイプ・プログラムまたはプロシージャーの呼
び出し) 命令コード
使用 4-83
CCSID
コンパイラー・リストに示された 7-60
CEE4ABN 4-104
CEECRHP (ヒープ作成) バインド可能 API 3-15, 4-69
CEE CZST (記憶域再割り振り) バインド可能 API 3-15
CEEDOD (操作記述子情報の検索) 4-44
操作記述子 4-88
例 4-88
CEEDSHP (ヒープ廃棄) バインド可能 API 3-15, 4-69
CEEFRST (記憶域解放) バインド可能 API 3-15
CEE GTST (ヒープ記憶域取得) バインド可能 API 3-15, 4-69
CEEHDLR (ILE 条件処理プログラム登録) 5-80
CEEHDLU (ILE 条件処理プログラム登録解除) 5-80
CEERTX (呼び出しスタック項目終了ユーザー出口プロシ
ジャーの登録) 5-86
CEESGI (ストリング引数に関する記述情報の取得) 4-88
CEETREC 4-104
CEETSTA (省略された引数の検査) 4-89
CEEUTX (呼び出しスタック項目終了ユーザー出口プロシ
ジャー) 5-86
CL コマンド
権限 1-1
構文図の読み取り 7-33
使用 7-32
追加のサービス・プログラム・コマンド 4-43
デバッグ開始 (STRDBG) 5-8, 5-10
デバッグ終了 (ENDDBG) 5-8
プログラム関連の 4-37
プログラム除去 (RMVPGM) 5-11
プログラム追加 (ADDPGM) 5-11
モジュール関連の 4-32
モジュールの変更 (CHGMOD) 4-40

CL コマンド (続き)
モジュール・ソースの表示 (DSPMODSRC) 5-10, 5-11,
5-12, 5-14
よく使用されるコマンド 3-10
ADDRPLYE 4-56
CALL 4-52
CHGPGM 4-40
CRTPGM コマンド 4-34
CRTRPGMOD 4-25
CVTRPGSRC 7-10
DSPMOD 4-99
DSPPGMREF 4-99
MONMSG 5-89
RCLACTGR 4-58
RCLRSC 4-60
UPDPGM 4-39
WRKRPLYE 4-57
CLEAR デバッグ・コマンド
使用 5-18, 5-21, 5-25
すべての除去 5-26
定義 5-3
COMMIT (コミット) 命令コード
コミットメント制御 6-48
システムに関する考慮事項 6-48
複数装置をもつ 6-48
CPF417C
一時変更による 6-13
CRTBNDRPG コマンド 7-38
CRTBNDRPG コマンドによるプログラムの作成 4-10
CRTRPTPGM (報告書簡易作成プログラム作成) コマンド
報告書簡易作成機能メンバーの変換 7-17
CVTOPT パラメーター
CRTBNDRPG コマンド 4-11, 7-43
CRTRPGMOD コマンド 4-25
CVTRPGSRC (RPG ソースの変換) コマンド
構文図 7-11
コマンドのデフォルトの値の使用 7-15
デフォルトのパラメーター値 7-10
パラメーターの説明 7-12
例 7-16
CVTRPT パラメーター 7-14, 7-17, 7-21

D

DBCS
NLSS デバッグに関する考慮事項 5-22
RPG IV における文字フィールド 7-6
DBGVIEW パラメーター
使用 4-12
ソースの表示のための値 5-13
デバッグのためのプログラムの準備 5-4
リスト出力オプションによる調整 4-21
CRTBNDRPG コマンド 4-11, 7-41
CRTRPGMOD コマンド 4-25
DEALLOC (記憶域解放) 命令コード 4-62

DEFINE パラメーター
 CRTBNDRPG コマンド 4-11, 7-49
 CRTRPGMOD コマンド 4-25
DETAIL パラメーター
 バインダー・リストの作成 4-37
DETC 5-79
DETL 5-79
DFTACTGRP パラメーター、CRTBNDRPG の
 使用 4-10, 4-13, 4-14
 説明 7-39
 CRTBNDRPG コマンド 4-11
 OPM デフォルトでの実行 4-59
DISK ファイル
 外部記述
 アクセス・パス 6-21
 概要説明 6-21
 プログラム記述として 6-4
 例 6-22
 レコード様式仕様書 6-21
 概要説明 6-20
 使用可能なファイル命令コード
 キー順処理方式 6-44
 キーによらない処理方式 6-44
 処理方式
 概要 6-31
 キーによる順次処理 6-33
 キーによるランダム処理 6-39
 限界内順次処理 6-40
 相対レコード番号による処理 6-42
 連続処理 6-32
 プログラム記述
 索引付きファイル 6-27
 順次ファイル 6-29
 処理 6-31
 レコード・アドレス・ファイル 6-30
 レコード様式仕様書 6-21
DISPLAY デバッグ・コマンド
 簡略名の表示 5-52
 使用 5-14
 定義 5-3
DUMP (プログラム・ダンプ) 命令コード
 使用 5-92
 定様式ダンプの入手 5-91

E

ENBPFCOL パラメーター
 CRTBNDRPG コマンド 4-11, 7-49
 CRTRPGMOD コマンド 4-25
ENDSR (サブルーチンの終わり) 命令コード
 戻り点の指定 5-79
ENTMOD パラメーター 4-34
EQUATE デバッグ・コマンド
 使用 5-51
 定義 5-3
 例 5-51

EVAL デバッグ・コマンド
 値の変更 5-48
 値を割り当てるための規則 5-48
 可変長 形式での 5-43
 使用 5-36
 データ構造の表示 5-40
 テーブルの内容 5-39
 定義 5-3
 配列の内容 5-39
 標識 5-42
 文字形式で 5-42
 例 5-37, 5-49
 UCS-2 形式での 5-43
EXFMT (様式の書き出し / 読み取り) 命令コード 6-76
EXPCPY パラメーター 7-14
EXPORT キーワード
 重複した名前 4-36

F

FIND デバッグ・コマンド 5-4
FIXNBR パラメーター
 CRTBNDRPG コマンド 4-11, 7-46
 CRTRPGMOD コマンド 4-25
FREE (プログラムの非活動化) 命令コード 7-24
FROMFILE パラメーター 7-12
FROMMBR パラメーター 7-13, 7-16

G

GDDM 4-106
GENLVL パラメーター
 CRTBNDRPG コマンド 4-11, 7-38
 CRTRPGMOD コマンド 4-25

I

ICF 通信ファイル 6-66
IGNORE キーワード 6-7
ILE C
 拡張アプリケーション・プログラムにおける 3-24
 混合言語のアプリケーション・プログラムにおける 3-23
 デバッグのモジュールのソース例 5-57
 パラメーターの受け渡し方法 4-97
 ILE 言語として 3-11
ILE CL
 拡張アプリケーション・プログラムにおける 3-24
 混合言語のアプリケーション・プログラムにおける 3-23
 状況と通知例外の予期しない処理 5-89
 パラメーターの受け渡し方法 4-97
 ILE RPG プログラムの呼び出し 3-22
 ILE 言語として 3-11
 ILE プログラム内のモジュールとして 3-21
 RPG プログラムの呼び出し 3-19

ILE COBOL

パラメーターの受け渡し方法 4-97

ILE 言語として 3-11

ILE RPG

サポートされている装置タイプ 6-52

データ管理命令 6-11

バインドされたモジュールの動作 4-32

プログラム例 3-5

変換 7-6

例外処理の概要 5-61

論理図 3-3

OPM RPG/400 の動作上の相違点 7-1

RPG IV 言語の概要 3-1

ILE RPG と OPM RPG/400 との入出力の相違点 7-3

ILE RPG プログラムの一部 3-5

ILE RPG への変換 7-6

ILE 条件処理プログラム登録 (CEEHDLR) API 5-80

ILE 条件処理プログラム登録解除 (CEEHDLU) API 5-80

ILE ソース・デバッガー

開始 5-8

説明 5-2

デバッグ・コマンド 5-3

ILE バインド可能 API を使う戻り 4-104

INDENT パラメーター 5-7

CRTBNDRPG コマンド 4-11, 7-42, 7-43

CRTRPGMOD コマンド 4-25

INSRTPL パラメーター 7-14, 7-18

J

Java

コーディング・エラー 4-147

実行時エラー 4-157

ネイティブ・メソッド 4-144

プロトタイピング 4-133

呼び出し、PCML の使用による RPG プログラムの 4-164

Java 仮想マシン (JVM) 4-149

Java からの RPG の呼び出し 4-144

Java メソッドの呼び出し 4-136

RPG からの Java の呼び出し 4-136

JNI 機能、ラッパー 4-149

L

LANGID パラメーター

CRTBNDRPG コマンド 4-11, 7-44

CRTRPGMOD コマンド 4-25

LICOPT パラメーター

CRTBNDRPG コマンド 7-50

LOGFILE パラメーター 7-15

LOGMBR パラメーター 7-15

M

MCH3601 7-3

MODULE パラメーター 4-34

CRTBNDRPG コマンド 7-37

CRTRPGMOD コマンド 4-25

MONITOR グループ 5-68

MQSeries 4-132

N

NOMAIN モジュール

コーディング上の考慮事項 3-38

作成 4-27

NOOPT キーワード

および例外の処理 5-66

デバッグ中の現在の値のメンテナンス 5-2

プログラムの最適化レベル 4-39

NOT

ILE RPG と RPG/400 との動作上の相違点 7-1

null サポート

ヌル可能フィールドの表示 5-45

O

OFL 5-79

ON-ERROR グループ 5-68

OPM 互換性、メンテナンス 4-14, 4-59

OPM デフォルトの活動化グループ 3-17, 3-25

実行 4-59

OPM と ILE RPG の相違

動作上の相違点 7-1

例外処理 5-63

OPTIMIZE パラメーター

CRTBNDRPG コマンド 4-11, 7-42

CRTRPGMOD コマンド 4-25

OPTION パラメーター

使用 4-16, 4-21

デバッグ・ビュー・オプションによる調整 4-21

リスト・ビュー・オプションとデバッグ・ビュー・オプションの調整 5-7

CRTBNDRPG コマンド 4-11, 7-39

CRTRPGMOD コマンド 4-25

OPTIONS キーワード

*NOPASS 4-89

*OMIT 4-89

OUTPUT パラメーター

使用 4-16

CRTBNDRPG コマンド 4-11, 7-42

CRTRPGMOD コマンド 4-25

P

PARM (パラメーターの識別) 命令コード 4-52

指定に関する規則 4-101

PARM (パラメーターの識別) 命令コード (続き)

使用 4-100

*OMIT 4-88, 4-89

PCML

プログラム呼び出しマークアップ言語を参照 4-164

PGM パラメーター

CRTBNDRPG コマンド 4-11

PLIST (パラメーター・リストの識別) 命令コード 4-52

使用 4-101

*ENTRY PLIST 4-101

PREFIX キーワード 6-6

PRFDTA パラメーター

識別情報の除去 4-40

CRTBNDRPG コマンド 4-11, 7-49

CRTRPGMOD コマンド 4-25

PRINTER ファイル

オーバーフロー標識 6-53

現在行の値へのアクセス 6-60

使用可能なファイル命令コード 6-53

フェッチ・オーバーフロー・ルーチン論理 6-57

プログラムで使用できるファイルの最大数 6-53

ページ・オーバーフロー 6-54

用紙制御の変更 6-60

PRTCTL (プリンター制御) 6-60

PRTCTL (プリンター制御)

一般情報 6-60

例 6-61

Q

QUAL デバッグ・コマンド

定義 5-3

ILE RPG 5-48

R

Rational Developer for i のイベント・ファイル 7-40

REALLOC (新しい長さの記憶域再割り振り) 命令コード 4-62

RECNO キーワード

相対レコード番号による処理 6-42

RENAME キーワード 6-6

REPLACE パラメーター

CRTBNDRPG コマンド 4-11, 7-44

CRTRPGMOD コマンド 4-25

RETURN (呼び出し元への戻り) 命令コード

異常終了での役割 4-103

正常終了での役割 4-102

未終了の戻り 4-103

RETURNCODE データ域 4-22

RPG IV

概要 3-1

サポートされていない RPG III 機能 7-26

変換 3-17, 3-19, 7-6

RPG III の動作上の相違点 7-1

RPG IV への変換

いくつかのファイル・メンバーの変換 7-16

概要 7-7

組み込み SQL をもつソース・メンバーの変換 7-18

試行変換の実行 7-16

すべてのファイル・メンバーの変換 7-16

制約 7-9

データ・ファイルからのソースの変換 7-18

ファイルに関する考慮事項 7-7

ファイルのレコード長 7-8

ファイル名およびメンバー名 7-8

変換 7-9

変換上の問題 7-25

変換エラー報告書の使用 7-21

変換の分析 7-21

変換報告書の入手 7-17

報告書簡易作成機能ソース・メンバーの変換 7-17

有効なソース・メンバー・タイプ 7-7

要件 7-9

例 7-18

ログ・ファイル 7-9

ログ・ファイルの使用 7-24

CVTRPGSRC コマンド 7-10

RPG モジュールの作成 (CRTRPGMOD) コマンド

および ILE 3-13

機能ごとにグループ化されたパラメーター 4-25

構文図 7-52

使用 4-25

デフォルトの値 4-27

パラメーターの説明 7-55

パラメーターのデフォルト値 4-25

プログラムの作成方針 3-21

例 4-47, 4-48

S

SECLVL パラメーター 7-14

SEQ ファイル

概要説明 6-62

可変長 6-62

使用可能なファイル命令コード 6-62

処理図 6-62

制約事項 6-62

例 6-62

SET デバッグ・コマンド

定義 5-3

SETLL

例外 MCH3601 7-3

SEU を使用したコンパイラー・リストのブラウズ 4-20

SPECIAL PLIST の区域パラメーター 6-64

SPECIAL ファイル

概説 6-63, 6-65

有効なファイル命令 6-64

レコードの削除 6-64

SRCFILE パラメーター

CRTBNDRPG コマンド 4-11, 7-37

SRCFILE パラメーター (続き)

CRTRPGMOD コマンド 4-25

SRCMBR パラメーター

CRTBNDRPG コマンド 4-11, 7-38

CRTRPGMOD コマンド 4-25

SRTSEQ パラメーター

キーの比較での影響 6-19

デバッグに関する考慮事項 5-22

CRTBNDRPG コマンド 4-11, 7-43

CRTRPGMOD コマンド 4-25

STEP デバッグ・コマンド

オーバー 5-32

定義 5-3

中へ 5-33

STGMDL パラメーター

CRTBNDRPG コマンド 7-48

CRTRPGMOD コマンド 7-55

STRSEU (ソースの 編集) コマンド 4-2

SUBR23R3 (メッセージ検索) 4-106

SUBR40R3 (2 バイト文字変数の操作) 4-106

SUBR41R3 (2 バイト文字変数の操作) 4-106

T

TBREAK デバッグ・コマンド

使用 5-19, 5-25

定義 5-3

TEXT パラメーター

CRTBNDRPG コマンド 4-11, 7-38

CRTRPGMOD コマンド 4-25

TGTRLS パラメーター

CRTBNDRPG コマンド 4-11, 7-46

CRTRPGMOD コマンド 4-25

THREAD デバッグ・コマンド

使用 5-19

定義 5-4

TOFILE パラメーター 7-13, 7-16

TOMBR パラメーター 7-14, 7-16

TOTC 5-79

TOTL 5-79

TRUNCNBR パラメーター

CRTBNDRPG コマンド 4-11, 7-46

CRTRPGMOD コマンド 4-25

U

UCS-2 形式

デバッグ中の表示 5-43

UCS-2 CCSID

コンパイラ・リストに示された 7-60

USRPRF パラメーター、CRTBNDRPG の

CRTBNDRPG コマンド 4-11, 7-45

W

WATCH デバッグ・コマンド

条件の設定 5-26

定義 5-4

例 5-30

WORKSTN ファイル

外部記述 6-66

処理 6-69

機能キー標識 6-68

サブファイル

使用 6-72

制御レコード様式 6-70

表示装置ファイルの場合 6-70

例 6-72

レコード様式 6-70

サンプル照会プログラム 6-79

サンプル照会・検索プログラム 6-101

サンプル・サブファイル処理プログラム 6-93

サンプル・データ維持プログラム 6-82

使用 6-66

使用可能なファイル命令コード 6-76

処理 6-76

定義 6-66

複数装置 6-76

プログラム記述

演算仕様書 6-75

概要説明 6-73

考慮事項 6-75

出力仕様書 6-74

出力ファイル 6-75

入出力共用ファイル 6-75

入力仕様書 6-74

入力ファイル 6-75

様式名あり 6-74

様式名なし 6-75

例 6-78

X

XML 4-126

XML 文書の処理 4-127

XML パーサーの制限 4-130

XML パーサー・エラー・コード 4-128

XML パーサーの制限 4-130

XML パーサー・エラー・コード 4-128

XML 文書の処理 4-127

XML パーサーの制限 4-130

XML パーサー・エラー・コード 4-128

[特殊文字]

*CALLER 4-58

*CANCL 5-79

*DETC 5-79

*DETL 5-79

*ENTRY PLIST 4-101

*EXTDFT
コンパイラー・リスト内の 7-64
例 7-60

*GETIN 5-79

*JOB
分類順序、SRTSEQ 7-43

*JOBRUN
言語識別コード、LANGID 7-44
分類順序、SRTSEQ 7-44

*NEW 4-58

*OFL 5-79

*OMIT 4-88, 4-89

*TOTC 5-79

*TOTL 5-79

*USER
ユーザー・プロファイル、USRPRF 7-45

/COPY ステートメント
コンパイラー・リストのテーブル 7-65
ソース・ファイルの使用 4-7
変換上の問題 7-17, 7-27
変換報告書 7-22
COPY デバッグ・ビュー 5-6

%ADDR デバッグ組み込み 5-46

%ADDR (変数のアドレスの検索)
省略されたパラメーター 4-89

%ALLOC 組み込み関数 4-62

%INDEX デバッグ組み込み 5-46

%PARMS (パラメーター数の戻り)
パラメーターの数の検査 4-90

%REALLOC 組み込み関数 4-62

%SUBSTR デバッグ組み込み
値の変更 5-48
例 5-47

%VARS デバッグ組み込み 5-46



プログラム番号: 5770-WDS

Printed in Japan

日本アイ・ビー・エム株式会社

〒103-8510 東京都中央区日本橋箱崎町19-21