Micro Focus® Net Express®

ファイル処理

第 7 版
2006 年 1 月

このガイドでは、Net Express のファイル処理機能について説明します。

はじめに

第 1 章 : はじめに

ここでは、ファイル処理の概要を説明します。

概要

プログラミング言語として COBOL が持つ主要な機能の 1 つに、組み込みのファイル処理機能があります。順編成ファイル、相対ファイル、索引ファイルを、簡単な COBOL 構文で処理できます。

Micro Focus COBOL には、次のファイル処理機能も用意されています。

ファイル処理機能	用途
行順ファイル編成	COBOL プログラムから ASCII テキストファイルへのアクセス
COBOL システムライブラリルーチンのセット	ファイルとディレクトリの操作
COBOL システムライブラリルーチンのセット	バイトレベルでのファイルの処理
Fileshare	ネットワークサーバ上のファイルグループへのアクセス
変換モジュール	Btrieve ファイルに対する読み書き (Net Express のみ)

Micro Focus ファイルハンドラ

デフォルトでは、Micro Focus ファイルハンドラによって、COBOL の標準ファイル編成 (順編成ファイル、相対ファイル、および索引ファイル) のすべての COBOL 入出力操作が実行されます。また、ファイルハンドラのアプリケーションプログラミングインターフェイス (API) を使用して、COBOL プログラムから直接ファイルハンドラを呼び出せます。

はじめに　ファイル名

第 2 章 : ファイル編成

ここでは、COBOL システムがサポートするファイル編成について説明します。

概要

ファイルはデータの集合で、通常はディスクに保存されています。論理エンティティとして、ファイルではデータを意味のあるグループに分割できます。たとえば、あるファイルに会社の製品情報をすべて保存し、別のファイルに社員情報を保存できます。物理エンティティとしてのファイルの編成について考慮する必要があります。ファイル編成とは、ファイルにデータを物理的に格納する方法を意味します。またファイル編成によって、データを呼び出す方法も決まります。

この COBOL システムでは、順ファイル編成、相対ファイル編成、索引ファイルの 3 つのファイル編成をサポートしています。ファイル編成に応じて、データを呼び出す方法が次の 3 通りあります。

ファイル構成	順呼び出し	乱呼び出し	動的呼び出し
順編成	可	不可	不可
相対編成	可	可	可
索引編成	可	可	可

順ファイル

順ファイルでは、個々のレコードを順番に呼び出します。つまり、ファイルに書き込まれた順番と同じ順番でレコードを呼び出します。新しいレコードは常にファイルの末尾に追加されます。

この COBOL システムでは、3 種類の順ファイルがサポートされています。

- レコード順
- 行順
- プリンタ順

レコード順ファイル

デフォルトでは、ファイルを作成し、ファイル編成を順ファイルに指定すると、レコード順ファイルが作成されるため、レコード順ファイルは、ほとんどの場合「順ファイル」と呼ばれます。

ファイルをレコード順ファイルとして定義するには、COBOL プログラムでファイルに対して SELECT 句に ORGANIZATION IS RECORD SEQUENTIAL を指定します。たとえば、次の

ように指定します。

```
select recseq assign to "recseq.dat"
    organization is record sequential.
```

デフォルトでは、順ファイルを指定するとレコード順ファイルが設定されるため、必ずしも ORGANIZATION IS RECORD SEQUENTIAL を指定する必要はありません。SEQUENTIAL コンパイラ指令を設定していない場合は、単に ORGANIZATION IS SEQUENTIAL と指定するのみでかまいません。

行順ファイル

行順ファイル (「テキストファイル」または「ASCII ファイル」とも呼ばれる) は、主に、表示専用データに使用します。「メモ帳」などのほとんどの PC エディタでは行順ファイルが作成されます。

行順ファイルでは、ファイルの各レコード間は、レコード区切り文字によって区切られています。レコード区切り文字は、復帰文字 (x"0D") と改行文字 (x"0A") で構成され、各レコードの空白文字を除く末尾文字の直後に挿入されます。WRITE 文は、データレコードから後続の空白文字を削除し、レコード区切り文字を追加します。READ 文は、レコード区切り文字を削除し、必要に応じて、データレコードを後続の空白文字で埋め、データを読み込むプログラムが定義するレコードサイズに合わせます。

ファイルを行順ファイルとして定義するには、COBOL プログラムでファイルに対して SELECT 句に ORGANIZATION IS LINE SEQUENTIAL を指定します。たとえば、次のように指定します。

```
select lineseq assign to "lineseq.dat"
    organization is line sequential.
```

プリンタ順ファイル

プリンタ順ファイルは、直接、または、いったんディスクファイルにスプールして、プリンタに送信するファイルを指します。このファイルは、連続した印刷レコードから構成されており、レコード間に縦方向の位置を表す文字 (改行文字など) が入る場合があります。各印刷レコードは 0 個以上の印刷可能な文字で構成され、終端に復帰文字 (x"0D") が位置しています。

プリンタ順ファイルでは、OPEN 文によりファイルに x"0D" が書き込まれ、プリンタが最初の印刷レコードを印刷する前の最初の文字位置に配置されます。WRITE 文は、印刷レコードと末尾のキャリッジリターンコード (x"0D") をプリンタに書き込む前に、印刷レコードから後続空白を削除します。WRITE 文で BEFORE 句または AFTER 句を指定すると、印刷レコードの書き込み前または後に、プリンタに対して 1 つ以上の改行文字 (x"0A")、用紙送りコード (x"0C")、または縦方向のタブコード (x"0B") を送信できます。

プリンタ順ファイルを、入力ファイル (INPUT) または入出力両用ファイル (I/O) として開かないでください。

ファイルをプリンタ順ファイルとして定義するには、SELECT 句で ASSIGN TO LINE ADVANCING FILE または ASSIGN TO PRINTER と指定します。たとえば、次のように指定し

ます。

```
select printseq
    assign to line advancing file "printseq.dat".
```

相対ファイル

相対ファイルは、各レコードをファイル内の順位 (レコード 1、レコード 2 のように) によって識別するファイルです。つまり、レコードに対して、順番に呼び出すことも、ランダムに呼び出すこともできます。

- 順呼び出しの場合は、単純に READ または WRITE 文を実行して、ファイルの次のレコードを呼び出します。
- 乱呼び出しの場合は、データ項目を相対キーとして定義します。データ項目の中で、READ または WRITE を実行する必要があるレコードの序数を指定します。

相対ファイルの呼び出しは、レコードをランダムに呼び出せるため、高速化されます。

相対ファイルに対して可変長レコードを宣言することは可能ですが、システムはファイルへの WRITE 文で最大のレコード長を仮定して、使用しない文字位置まで埋めてしまうので、ディスク領域は無駄になります。このような処理を行う理由は、COBOL ファイル処理ルーチンが、ファイルでレコード番号を指定されたレコードの物理的な位置をすばやく計算するためです。

相対ファイルには、常に固定長レコードが含まれるため、データ圧縮を指定しても領域を節約することはできません。ファイルハンドラでは、相対ファイルにデータ圧縮を指定しても無効になります。

相対ファイルの各レコードの後には 2 バイトのレコードマーカーがあり、レコードの現在の状態を示します。レコードマーカーが示す状態は、次のとおりです。

x"0D0A" - レコードは存在します。

x"0D00" - レコードは削除されたか、書き込まれていません。

相対ファイルからレコードを削除しても、レコードの内容はすぐには削除されません。レコードのレコードマーカーが削除済みとして更新されます。ただし、削除されたレコードの内容は、物理的には新しいレコードが書き込まれるまでそのまま残ります。セキュリティ上の理由からデータをファイルから削除する必要がある場合は、次の手順に従ってください。

1. REWRITE を使用して、レコードを空白などで上書きします。
2. レコードを削除します。

相対ファイルを定義するには、COBOL プログラムでファイルに対して SELECT 句に ORGANIZATION IS RELATIVE を指定します。

レコードを乱呼び出しするには、次の手順も実行する必要があります。

- ファイルに対して SELECT 句で ACCESS MODE IS RANDOM または ACCESS MODE IS DYNAMIC を指定します。
- プログラムの作業場所節で相対キーを定義します。

たとえば、次のように指定します。

```
select relfil assign to "relfil.dat"
    organization is relative
    access mode is random
    relative key is relfil-key.
...
working-storage section.
01 relfil-key   pic 9(8) comp-x.
```

上記のコード例では、相対ファイルを定義しています。呼び出し法は乱呼び出しなので、相対キー relfil-key を定義します。 乱呼び出しの場合は、常に、ファイルからレコードを読み込もうとする前に、相対キーにレコード番号を指定することが必要です。

ACCESS MODE IS DYNAMIC を指定した場合は、順呼び出しと乱呼び出しの両方でファイルを呼び出せます。

索引ファイル

索引ファイルでは、各レコードが主キーを持っています。各レコードを互いに区別するために、主キーの値は各レコードに対して一意でなければなりません。レコードの主キーの値を指定すると、レコードをランダムに呼び出せます。索引ファイルのレコードは順呼び出しも可能です。

索引ファイルでは、主キーの他に、副キーと呼ばれる追加キーを指定できます。レコードの副キーの値は一意である必要はありません。

ファイルを索引ファイルとして定義するには、COBOL プログラムでファイルに対して SELECT 句に ORGANIZATION IS INDEXED を指定します。また、RECORD KEY 句を使用して主キーも指定する必要があります。

```
select idxfile assign to "idx.dat"
   organization is indexed
   record key is idxfile-record-key.
```

実際には、ほとんどの索引ファイルは、データファイル (レコードデータを含む) と索引ファイル (索引構造体を含む) という 2 つの別のファイルで構成されます。その場合、COBOL プログラムで指定した名前はデータファイル名になります。関連付けられる索引ファイルの名前は、データファイル名に .idx という拡張子が追加されて作成されます。他の用途に拡張子 .idx を使用しないでください。

索引はレコードが追加されると大きくなる逆ツリー構造として構築されます。

索引ファイルでは、ランダムに選択したレコードを検索するためのディスクアクセス回数は、主にファイルのレコード数とレコードキーの長さによって決まります。 ファイルの入

出力は、ファイルを順に読み込んでいく場合よりも高速になります。

すべての種類のファイルを定期的にバックアップすることをお奨めしますが、索引ファイルでは、2 つのファイルのうちのどちらかのみが使用できなくなる場合 (媒体の破損など) があります。索引ファイルを損失した場合は、Rebuild ユーティリティを使用して、データファイルから索引を復元して、障害復帰に必要な時間を短縮できます。詳細は、『リビルド』の章を参照してください。

主キー

索引ファイルの主キーを定義するには、SELECT 句で RECORD KEY IS 句を使用します。

```
select idxfile assign to "idx.dat"
    organization is indexed
    record key is idxfile-record-key.
```

副キー

各レコードには、主キーの他に、副キーと呼ばれる追加のキーを必要な数のみ指定できます。副キーを定義するには、SELECT 句で ALTERNATE RECORD KEY IS 句を使用します。

```
select idxfile assign to "idx.dat"
    organization is indexed
    record key is idxfile-record-key
    alternate record key is idxfile-alt-key.
```

重複キー

重複した値を持つキーを定義できます。ただし、レコードの主キーの値は一意でなければならないため、主キーは重複させることができません。

重複キーを使用する場合は、各キーに同じ値を指定できる回数が制限されることに注意してください。重複キーに同じ値を指定するたびに、キーの出現番号が 1 ずつ増えます。個々のキーで値を重複させられる最大回数は、索引ファイルの種類によって異なります。 索引ファイルの種類とその特徴の詳細なリストについては、ヘルプトピック『索引ファイルの種類』を参照してください。

COBOL システムは、重複キーレコードを作成順に読み込むために出現番号を使用します。そのため、削除したレコードの出現番号を再使用できません。 つまり、一部のキーをすでに削除した場合は、重複値の最大回数に到達する可能性があります。

索引ファイルの種類によっては、データファイルに重複レコードを持つものがあります。索引ファイルに重複レコードが含まれている場合は、データファイルの各レコードの後に、システムレコードが続きます。このシステムレコードには、そのレコードの各重複キーに対するキーの出現番号が保持されます。この番号は、ファイルの履歴で、あるキー値が使用された回数を示すカウンタに過ぎません。重複レコードが存在すると、レコードに対する REWRITE 操作と DELETE 操作がより高速になりますが、このようなファイルのデータレコ

ードは標準的なファイルのデータレコードよりも大きくなります。

副キーに重複した値を指定するには、SELECT 句の ALTERNATE RECORD KEY 句に WITH DUPLICATES を使用します。

```
file-control.
    select idxfile assign to "idx.dat"
        organization is indexed
        record key is idxfile-record-key
        alternate record key is idxfile-alt-key
                                 with duplicates.
```

スパースキー

スパースキーを設定すると、指定した値に対する索引エントリがファイルに格納されません。 たとえば、すべて空白文字であるキーをスパースキーとして定義した場合は、空白文字しかないデータ項目を持つレコードが、そのキーの索引エントリとしてファイルに格納されることはありません。

スパースキーとして指定できるのは、副キーのみです。

この機能を使用すると、索引ファイルを小さくできます。 キーが大きくなり、副キーに指定した値を持つレコードの数が増えた場合には、多くのディスク容量を節約できます。

スパースキーを有効化するには、SELECT 句 の ALTERNATE RECORD KEY 句で SUPPRESS WHEN ALL を使用します。

```
file-control.
    select idxfile assign to "idx.dat"
    organization is indexed
    record key is idxfile-record-key
    alternate record key is idxfile-alt-key
                                 with duplicates.
                                 suppress when all "A".
```

この例では、副キーの値がすべて A であるレコードが書き込まれた場合には、実際のキーの値は索引ファイルに格納されません。

索引ファイルの呼び出し

主キーと副キーの両方を使用して、直接 (乱呼び出し)、またはキーの順番で (順呼び出し)、索引ファイルからレコードを読み込むことができます。 呼び出し法は、次のとおりです。

- SEQUENTIAL

  昇順または降順のレコードキー値の順序でレコードを呼び出します (デフォルト)。

- RANDOM

  レコードキーの値に従ってレコードを呼び出します。

- DYNAMIC

  適切な形式の入出力文を使用して、順呼び出しと乱呼び出しを切り替えます。

索引ファイルの呼び出し法は、SELECT 句で ACCESS MODE IS 句を使用して定義します。たとえば、次のように指定します。

```
file-control.
    select idxfile assign to "idx.dat"
        organization is indexed
        access mode is dynamic
        record key is idxfile-record-key
        alternate record key is idxfile-alt-key.
```

固定長レコードと可変長レコード

ファイルには次のレコードを含めることができます。

- 固定長レコード - すべてのレコードがまったく同じ長さのレコード
- 可変長レコード - レコードごとに長さが異なるレコード

可変長レコードを使用すると、ディスク領域を節約できることがあります。 固定長レコードを使用する場合は、固定レコード長を最長のレコード長に合わせる必要があります。 アプリケーションが生成するレコードのうち、短いレコードが多く、長いレコードが少ない場合は、固定長レコードでは大量のディスク領域を無駄にするため、可変長レコードの方がより適しています。

レコードの種類は、次のように指定します。

使用するレコード	指定する句
可変長レコード	RECORDING MODE IS V
固定長レコード	RECORDING MODE IS F

または

使用するレコード	指定する句
可変長レコード	RECORD IS VARYING
固定長レコード	RECORD CONTAINS *n* CHARACTERS

または

使用するレコード	指定するコンパイラ指令
可変長レコード	RECMODE"V"
固定長レコード	RECMODE"F"

または、可変長レコードを使用する場合は、RECMODE"OSVS" コンパイラ指令と次のどちらかを指定します。

- RECORD CONTAINS n TO m CHARACTERS
- それぞれの長さが異なる複数のレコード領域

ファイルヘッダー

ファイルヘッダーは、ファイルの先頭にある 128 バイトのブロックです。次のファイル構造のファイルにはファイルヘッダーが付いています。

- 索引ファイル
- 可変長レコードを持つレコード順ファイル
- 可変長レコードを持つ相対ファイル

さらに、これらのファイルでは、各レコードの先頭に 2 バイトまたは 4 バイトのレコードヘッダーが付きます。

ファイルヘッダー、レコードヘッダー、およびヘッダーを持つファイルの構造の詳細については、ヘルプトピック
『ヘッダーをもつファイル』を参照してください。

ファイル編成 ファイル状態

第 3 章 : ファイル名

ここでは、次の事項について説明します。

- ファイル命名規則
- ファイル名の割り当て
- ファイル名のマッピング
- 動的に割り当てられたファイル、または、外部で割り当てられたファイルを検索するためにランタイムシステムが採用する検索順序

ファイル命名規則

この COBOL システムと、この COBOL システムを使用して作成したアプリケーションでは、システムを実行するオペレーティングシステムの標準的なファイル命名規則を採用します。

また、New Technology File System (NTFS) のファイル命名規則もサポートしています。NTFS では、ファイル名やディレクトリ名に最大 254 文字まで使用できます。この中には、空白文字、その他の特殊文字および任意の数のピリオド (.) を含めることができます。この COBOL システムでは、最後のピリオドの後のテキストを拡張子とみなし (または最後がピリオドの場合は、空白文字を拡張子とみなします)、COBOL システムが名前を作成するときには拡張子を削除することもあります。

新しいファイル命名規則を使用する場合は、次の点に注意してください。

- インターネットアプリケーションを作成する場合は、% # ( ) @ の文字は使用しないでください。
- UNIX で利用可能な Micro Focus COBOL 製品には、長いファイル名、および空白文字を含むファイル名やフォルダ名を処理しないコンポーネントがあります。UNIX へパブリッシュされるファイルには、長い名前および空白文字を含む名前は使用しないでください。
- Pro*Cobol の Oracle 1.8 バージョンでは、空白文字を含むファイル名はサポートされていません。

ファイル名の割り当て

物理的なファイル名、または、実行時に物理的な名前にマップできる論理的なファイル名を指定するには、SELECT 句で ASSIGN 句を使用します。

ファイル名の割り当てには、次のような 3 種類の方法があります。

- STATIC

  ASSIGN 句でファイル名を文字列定数として指定します。

- DYNAMIC

  ASSIGN 句でファイル名をデータ項目として指定するので、実行時にプログラムで変更できます。

- EXTERNAL

  ASSIGN 句でファイル名を EXTERNAL として指定し、実行時に決定します。

これら 3 種類すべてのファイル名の割り当て方法でファイル名の実行時マッピングを使用できます。

ファイル名の静的割り当て

ファイル名の静的割り当てでは、ファイル名を SELECT 句で定数として指定します。

```
select filename
    assign to literal.
```

作成している物理ファイルのファイル名に空白文字を使用すると、そのファイル名は自動的に引用符で囲まれます。

次の例では、stockfile を開くと、b: ドライブの現在のディレクトリにある warehs.buy ファイルが開きます。

```
select stockfile
    assign to "b:warehs.buy".
```

次の例では、WRITE 文を使用すると、最初のパラレルプリンタである prn: にデータが出力されます。

```
select printfile
    assign to "prn:".
```

次の例では、input-file を開くと、現在のディレクトリに相対的な data ディレクトリの prog ファイルが開きます。

```
select input-file
    assign to "data¥prog".
```

ファイル名の動的割り当て

ファイル名の動的割り当てでは、ファイル名を SELECT 句で COBOL データ項目として指定します。

```
select filename
    assign to dynamic data-item
```

パラメータの内容は、次のとおりです。

filename	割り当てるファイルのファイル名
data-item	COBOL データ項目の名前。プログラムでデータ項目が明示的に宣言されていない場合は、PICTURE 句 PIC X(255) を使用して、コンパイラがデータ項目を自動的に作成します。ファイルに対して OPEN 文を実行する前に、プログラムでデータ項目の値を指定する必要があります。

作成している物理ファイルのファイル名に空白文字を含める場合は、ファイル名を引用符で囲む必要があります (下記の例 2 を参照)。

例 **1**

次の例では、input.dat というファイルが現在のディレクトリに作成されます。

```
   ...
select fd-in-name
    assign to dynamic ws-in-file.
      ...
working-storage section.
01 ws-in-file     pic x(30).
      ...
    move "input.dat" to ws-in-file.
      ...
    open output fd-in-name.
```

例 **2**

この例では、引用符を使用して、ファイル名に空白文字が含まれるファイル spacey filename.dat を作成しています。

```
select f1
     assign to dynamic f1-name.
      ...
working-storage section.
01 f1     pic x(30).
      ...
    move """spacey  filename.dat""" to f1-name
      ....
    open output f1.
```

注：ASSIGN"DYNAMIC" コンパイラ指令を使用する場合は、ASSIGN 句の DYNAMIC という語を省略できます。

ファイル名の外部割り当て

ファイル名の外部割り当てでは、ファイル名を SELECT 句で次のように指定します。

```
select filename
    assign to external external-file-reference
```

パラメータの内容は、次のとおりです。

filename	割り当てるファイルのファイル名
external-file-reference	外部環境に指定されたファイルでさらにマッピングが可能かどうかを識別する COBOL 語。*external-file-reference* に 1 つ以上のハイフン (-) が含まれる場合は、最後のハイフンまでのすべての文字が無視されます。

ランタイムシステムのファイル名マッピングの詳細については、『ファイル名のマッピング』の項を参照してください。

パスのライブラリ名

開くファイルの名前には、ライブラリ名を含めることができます。ファイル名は次の形式で指定します。

```
device¥library-name.lbr¥filename.ext
```

例

```
c:¥appdir¥applib.lbr¥app.dat
```

このタイプのファイル名は、OPEN INPUT 構文か CBL_OPEN_FILE バイトストリームルーチンを使用して、読み込み専用モードでデータファイルを開く場合のみ有効です。指定したライブラリが開いていない場合、この呼び出しによって指定されたデバイス上でライブラリが開かれます。呼び出しが終了しても、ライブラリは閉じられません。指定したライブラリが存在しない場合や、ライブラリ内に指定した filename.ext が存在しない場合には、ファイルが見つからない旨のエラーが返されます。

ファイル名のマッピング

この COBOL システムには、プログラムで ASSIGN 句を使用して指定したファイル名を別の名前にマップする方法が複数あるので、実行時に柔軟に対応できます。これらの方法では環境変数を使用します。

次の説明で使用する「環境変数」には、外部ファイルマッパーを通して有効化された Micro Focus の拡張環境変数も含まれます (『外部ファイルマッパー (Mfextmap)』の項を参照)。

プログラムを実行する前に、オペレーティングシステムの SET コマンドを実行して、使用する環境変数に適切な値を指定する必要があります。次に例を示します。

```
set dir=d2
```

ファイルハンドラにファイル名 (定数、データ項目の内容、または ASSIGN TO EXTERNAL 構文を使用した場合は外部参照として) が指定されている場合は、次のように実行されま

す。

1. ファイル名の最初の要素を分離します。つまり、最初のバックスラッシュ文字 (\) より前にあるすべてのテキストを分離します。バックスラッシュ文字が含まれない場合は、テキスト全体を分離します。

2. 最初の要素が 2 個のドル記号 ($$) で始まる (ファイルが Fileshare サーバ上にあることを示す) 場合、検索処理をその Fileshare サーバに切り替え、次の要素に対してファイル名のマッピング処理を続行します。

3. 先頭にドル記号 ($) が存在する場合は、これを削除し、最初の要素に「dd_」という文字を追加して、この名前を持つ環境変数を検索します。

4. この環境変数が見つからず、ASSIGN EXTERNAL 構文が使用されているか、またはファイル名がドル記号で始まる場合、この最初の要素 (ドル記号が先頭にある場合にはそのドル記号は除く) と同じ名前を持つ環境変数が検索されます。

5. ファイル名がドル記号で始まり、少なくとも 1 個のバックスラッシュ文字を含む場合以外は、検索に失敗するとファイル名は変更されません。この場合には、最初の要素全体と最初のバックスラッシュが名前から削除されます。

この処理はファイル名の次の要素に対して行われ、名前のすべての要素が処理されるまで繰り返されます。この処理結果を物理ファイルのファイル名にします。

例

ASSIGN 句のファイル名	検索対象の環境変数	環境変数の内容	物理ファイルのファイル名
dir¥file1	**dd_dir**	**d2**	**d2¥file1**
$dir¥file1	**dd_dir;dir**	**d2¥d4**	**d2¥d4¥file1**
dir1¥dir2¥file1	**dd_dir1**	**d4**	**d4¥dir2¥file1**
$dir1¥$dir2¥file1	**1** 回目の繰り返し： **dd_dir1; dir1**、**2** 回目の繰り返し： **dd_dir2;dir2**	**dd_dir1** または **dir1: d2¥d4 dd_dir2** または **dir2: d3**	**d2¥d4¥d3¥file1**
file1	**dd_file1**	**d2**	**d2**

複数パス

ファイル名のマッピングに使用する環境変数は、複数のパス名を指します。そのため、環境変数が指定した最初のパスに対して「ファイルが見つかりません。」という状態が返されると、システムは次のファイルを検索します。

次のような dd_dir という名前の環境変数があると仮定します。

```
dd_dir=¥a¥b;¥c¥d;
```

このときに、\a\b に対して「ファイルが見つかりません。」という状態が返されると、シ

ステムは、割り当てられたファイルを \c\d で検索します。

注：

- OPEN INPUT、OPEN I-O、OPEN EXTEND、または CBL_OPEN_FILE のバイトストリームルーチンで既存ファイルを開く場合、環境変数は、対象のファイルが検出された最初のパスに展開されます。
- OPEN OUTPUT、OPEN I-O、OPEN EXTEND、または CBL_CREATE_FILE のバイトストリームルーチンで新しいファイルを作成する場合には、環境変数は定義されている最初のパスに展開されます。 OPEN OUTPUT で開いたファイルについては、SEARCHONCREATE ファイルハンドラ構成オプションを ON に設定することで、この動作を無効にすることができます。
- 環境変数のパスリストの各パスの長さは、パスの許容最大文字数以下でなければなりません。

ライブラリ名

複数のパス環境変数に、ライブラリ名を含めることができます。 たとえば、次のようなコマンドを実行したとします。

```
set test=c:¥apdir;d:¥aplbr¥aplbr.lbr;d:¥aplbr¥aplbr1.lbr
```

c:\apdir\app.dat が存在しない場合には、ファイル名 $test\app.dat はd:\aplbr\aplbr.lbr \app.dat に解決されます。

ライブラリは、環境変数に定義されている順序で検索されます。

プログラム呼び出しで検索されるパスの一部として、ライブラリ名を使用することもできます。 たとえば、次のようなコマンドを実行したとします。

```
set appath=d:¥apdir¥progs1.lbr;d:¥appdir¥progs2.lbr
```

次の形式を使用できます。

```
call "$appath¥prog"
```

COBOL 開発システムは、ブログラム prog1で使用するライブラリとして、progs1.lbr と progs2.lbr を検索します。ライブラリは、環境変数に定義されている順序で検索されます。検索したライブラリは、実行中のアプリケーションによって明示的に閉じられるまで開いたままです。

ファイル名の割り当て

プリンタに直接レポートを送信する、または通信ポートを通してデータを転送するための COBOL プログラムを作成できます。そのためには、COBOL ファイル名にデバイス名を割り当てる必要があります。

次に示すデバイス名は、ファイル名の静的割当て、動的割当て、または外部割当てを使用して指定できます。

デバイス名	説明
CON	コンソールキーボードまたは画面
PRN	最初のパラレルプリンタ
LPT1	最初のパラレルプリンタ
LPT2	2 番目のパラレルプリンタ
LPT3	3 番目のパラレルプリンタ
COM1	最初の非同期通信ポート
COM2	2 番目の非同期通信ポート

これらのデバイス名を指定するときに、オプションで末尾にコロン (:) を指定できます。

次の例では、`fd-name` への読み書き操作を行うと、コンソール画面でデータの読み書きが行われます。

```
select fd-name
   assign to "con".
```

この例では、`fd-name` へ書き込み操作を行うと、データが最初のパラレルプリンタである `lpt1:` へ出力されます。

```
select fd-name
   assign to dynamic ws-filename.
   ...
   move "lpt1:" to ws-filename.
```

パイプの設定

COBOL ファイル構文を使用して、別のプロセス (dir コマンドなど) を起動し、そのプロセスの標準入力にデータを書き込んだり、そのプロセスの標準出力からのデータを読み込んだりできます。 この場合には、COBOL ファイル編成は、行順またはレコード順のどちらかである必要があります。

出力パイプ

プロセスを起動し、データを標準入力に書き込むためには、ファイル名の > 記号の後にコマンド名を続ける必要があります。 ファイルは、出力ファイルとして開く必要があります。

次に例を示します。

```
 select output-file
     assign to ">cmd /c sort"
     organization is line sequential.
 fd output-file.
 01 output-file-record pic x(10).
 procedure division.
```

```
    open output output-file
    write output-file-record from "Charles"
    write output-file-record from "Bill"
    write output-file-record from "Alan"
    close output-file.
```

入力パイプ

プロセスを起動し、標準出力からデータを読み込むには、ファイル名の "lt" 記号の後にコマンド名を続ける必要があります。 ファイルは、入力ファイルとして開く必要があります。

次に例を示します。

```
 select input-file
     assign to "<cmd /c dir"
     organization is line sequential.
  ...
     open input input-file
     read input-file
```

この例では、プログラムは dir プロセスを起動し、そのプロセスが標準出力に書き込む最初の行を読み込みます。

双方向パイプ

双方向パイプは、入力パイプと出力パイプの機能を組み合わせたものです。 双方向パイプを使用するには、ファイル名のパイプ記号 (¦) の後にコマンド名を続ける必要があります。 ファイルは、入出力両用ファイルとして開く必要があります。

次に例を示します。

```
 select i-o-file
     assign to "| cmd /c sort"
     organization is line sequential.
 fd i-o-file.
 01 i-o-file-record pic x(20).
 procedure division.
     open i-o i-o-file
     write i-o-file-record from "Hello world"
     write i-o-file-record from all "A"
     write i-o-file-record from all "Z"
     write i-o-file-record from x"1a"
     perform until exit
         read i-o-file
          at end
             exit perform
         end-read
         display i-o-file-record
     end-perform
```

```
close i-o-file
```

この例では、プログラムは sort プロセスを起動して、3 行目までとその後ろのファイルの終わりマーカーを標準出力に渡します。その後に、sort プロセスの標準出力から 3 行すべてを読み込みます。

外部ファイルマッパー (Mfextmap)

外部ファイルマッパー (Mfextmap) を使用すると、外部でファイル名を割り当てることができます。 この方法では、ファイル名のマッピングをテキストファイル (マッパーファイル) で処理できるため、COBOL プログラムで使用するファイル名を物理的なファイル名に柔軟にマップできます。 ファイル名のマッピングは、後でファイルを編集するだけで変更できます。

```
select filename
   assign to external assigned-name
```

パラメータの内容は、次のとおりです。

filename	プログラムで使用する論理 (内部) ファイル名
assigned-name	マッパーファイルのエントリ

外部ファイルマッパーを使用すると、オペレーティングシステムの環境変数でファイル名の割り当てを決定する場合より、環境領域に必要なメモリ量を削減できます。

マッパーファイルの構造

外部ファイルマッパーを使用するには、通常のテキストファイルを作成し (メモ帳などのテキストエディタを使用して作成)、mfextmap.dat という名前を付ける必要があります。

このファイルの各行に、次のような形式で割り当てるファイル名を記述します。

assigned-name physical-name

パラメータの内容は、次のとおりです。

assigned-name	プログラムで使用される割り当てファイル名
physical-name	物理ファイルの実際の名前。パス名を含む。

このファイルでは、次の点に注意してください。

- 割り当ては1 行に 1 つのみです。
- 割り当てるファイル名と物理的なファイル名の間は、少なくとも 1 つの空白文字で区切る必要があります。
- 割り当てるファイル名の前と物理的なファイル名の後には、無制限に空白文字を入れることができます。

プログラムの実行中にマッパーファイルの内容を変更できます。 使用されるマッピング情報は、常に、現在のバージョンのマッパーファイルに保存された情報です。

マッパーファイルの場所

マッパーファイルを作成する場合は、このファイルを mfextmap.dat と命名し、次のどこかに格納する必要があります。

- 現在のディレクトリ (プログラムが実行されているディレクトリ)
- COBOL システムパス上のディレクトリ (環境変数 COBDIR により示されます)
- 環境変数 MFEXTMAP により指定されたパス上のディレクトリ

MFEXTMAP と COBDIR は、どちらも複数のパスを定義できます。

MFEXTMAP が設定されていても、指定するディレクトリにマッパーファイルが存在しない場合には、システムは、プログラムで使用する割り当てファイル名と同じ名前の環境変数を検索し、ファイル名を決定しようとします。

MFEXTMAP が設定されていない場合には、システムは、まず現在のディレクトリでマッパーファイルを検索します。ここで見つからないと、次に COBDIR パスに従って検索します。 マッパーファイルが見つからない場合には、システムはプログラムで使用する割り当てファイル名と同じ名前の環境変数を検索し、ファイル名を決定しようとします。

外部ファイルマッパーの有効化

外部ファイルマッパーを使用するには、次を実行する必要があります。

- プログラムを、共有ランタイムシステムにリンクする。
- 有効なマッパーファイルを作成する。

次に、外部ファイルマッパーを有効化するために、ランタイムチューナー environment_mapper を TRUE に設定します。 次のように設定します。

```
set environment_mapper=TRUE
```

COBCONFIG_ 環境変数で指定された構成ファイルに追加します。

ランタイムチューナーの詳細については、ヘルプトピック 『ランタイムチューナー』を参照してください。

外部ファイルマッパーの無効化

外部ファイルマッパーを無効化すると、ディスクにアクセスして現在のバージョンのマッパーファイルを検索する必要がなくなるため、システムの処理速度が向上します。

外部ファイルマッパーを無効化するには、ランタイムチューナー environment_mapper を

FALSE に設定してください。

ファイル名　　ファイルの共有

第 4 章：ファイル状態

ファイル状態コードは、この COBOL システムでプログラムが実行したファイル操作の成否を示すために使用します。

ファイル状態コードの詳細なリストについては、ヘルプトピック『ファイル状態コード』を参照してください。

ファイル状態の意味

ファイル状態は、ファイル操作結果 (成功、または、エラーの発生など) を示す 2 バイトのコードです。 エラーが発生した場合は、ファイル状態にはエラー原因も表示されます。

ファイルにファイル状態データ項目が定義されている場合は、ファイルに入出力操作 (OPEN、CLOSE、READ、WRITE、REWRITE、START、および DELETE) を行うたびに、ランタイムシステムがファイル状態のデータ項目を更新し、操作結果を示します。

ファイル状態データ項目を定義するかどうかは選択できます。ファイル状態データ項目が宣言されていない場合に、重大なファイルエラーが発生すると、COBOL ランタイムシステムはエラーメッセージを表示し、プログラムを中断します。

入出力操作が行われるたびに、ファイル状態データ項目を調べ、操作が成功したかどうかを確認してください。 たとえば、プログラムがディスクに書き込みを行っているときには、WRITE 操作を完了できる十分なディスク領域がない場合もあります。ファイル状態データ項目を定義していない場合に、ディスク容量が不足すると、ランタイムシステムはエラー番号を表示し、プログラムを中断します。(書き込み中のファイルについて) ファイル状態のデータ項目を定義してある場合には、ファイル状態データ項目が更新され、プログラムの実行が継続されます。プログラムはファイル状態データ項目を調べ、ディスクがいっぱいであると判断すると、適切な措置を講じることができます。

ファイル状態データ項目の定義

ファイルに対して SELECT 句で FILE STATUS IS を指定し、作業場所節でファイル状態のデータ項目を定義すると、プログラムの各ファイルに、関連付けられたファイル状態のデータ項目を設定できます。

```
select in-file
    assign to "user.dat"
    file status is ws-file-status.
     ...
working-storage section.
    01 ws-file-status        pic xx.
```

各ファイルで別個のデータ項目を使用することも、複数のファイルで 1 つのデータ項目を使用することも可能です。

ファイル状態は、2 バイトのコードです。拡張ファイル状態コード以外のファイル状態の規則では、データ項目は英数字 2 文字 (PIC XX) で指定します。 拡張ファイル状態コードの場合は、2 番目のバイトにはバイナリの数字が格納されます。

ファイル状態データ項目に格納された最初のバイトは、状態キー 1 と呼ばれ、入出力操作の結果として次のどれかの状態を示します。

値	状態
0	操作の成功
1	ファイルの終わり
2	無効なキー
3	永続的なエラー
4	論理エラー (入出力操作の順序が不適切)
9	COBOL ランタイムシステムエラーメッセージ

ファイル状態データ項目の最初のバイトが 9 以外である場合には、2 番目のバイト (状態キー 2 と呼ばれる) は英数字になります。

ファイル状態データ項目の最初のバイトが 9 である場合には、2 番目のバイトは、ランタイムシステムエラーコードを含むバイナリフィールドになります。

次のコード例は、ファイル状態の確認方法を示します。まず、最初のバイト (状態キー 1) を調べ、より詳細な情報が必要な場合に 2 番目のバイト (状態キー 2) を確認します。

```
     select recseq
         assign to "recseq.dat"
         file status is ws-file-status
         organization is record sequential.
...
file section.
fd recseq
    record contains 80 characters.
01 recseq-fd-record    pic x(80).
...
working-storage section.
01 ws-file-status.
    05 status-key-1     pic x.
    05 status-key-2     pic x.
    05 binary-status redefines status-key-2  pic 99 comp-x.
...
procedure division.
    ...
    perform check-status.
    ...
check-status.
    evaluate status-key-1
     when "0"
```

```
           next sentence
        when "1"
           display "ファイルの末尾に到達しました。"
           perform check-eof-status
        when "2"
           display "無効なキー"
           perform check-inv-key-status
        when "3"
           display "永続的エラー"
           perform check-perm-err-status
        when "4"
           display "論理エラー"
        when "9"
           display "ランタイムシステムエラー"
           perform check-mf-error-message
    end-evaluate.
    ...
check-eof-status.
    if status-key-2 = "0"
        display "論理レコードがありません。"
    end-if.
    ...
check-inv-key-status.
    evaluate status-key-2
       when "2" display "重複キーを作成しようとしています。"
       when "3" display "レコードが見つかりません。"
    end-evaluate.
    ...
check-perm-err-status.
    if status-key-2 = "5"
        display "ファイルが見つかりません。"
    end-if.
    ...
check-mf-error-message.
    evaluate binary-status
     when 002 display "ファイルを開けません。"
     when 007 display "ディスク領域が不足しています。"
     when 013 display "ファイルが見つかりません。"
     when 024 display "ディスクエラー"
     when 065 display "ファイルがロックされています。"
     when 068 display "レコードがロックされています。"
     when 039 display "レコードが一致しません。"
     when 146 display "現在のレコードがありません。"
```

```
      when 180 display "ファイル形式が正しくありません。"
      when 208 display "ネットワークエラー"
      when 213 display "ロックが多すぎます。"
      when other
         display "エラー状態ではありません。"
         display binary-status
    end-evaluate.
```

ファイル状態の規則

この COBOL システムでは、ファイル状態の規則を数多くサポートしています。 各規則では、コードを個別に定義していますが、規則の間で重複するものもあります。

サポートされている規則を次に示します。

- ANSI'85 ファイル状態
- ANSI'74 ファイル状態
- Ryan MacFarland COBOL
- IBM メインフレーム COBOL
- Microsoft COBOL V2 ファイル状態

ANSI'85 ファイル状態

ANSI'85 の操作用に読み込まれた標準システムを使用している場合は、デフォルトでは、ANSI'85 ファイル状態コードが作成されます。

ANSI'74 ファイル状態

コンパイラ指令 NOANS85 でプログラムをコンパイルする場合は、ANSI'74 ファイル状態コードが作成されます。

ANSI'74 ファイル状態コードが作成されている場合に ANSI'85 構文を使用するには、NOANS85 コンパイラ指令を ANS85"SYNTAX" に置き換えてください。

ファイル状態の規則

エミュレーションを行うために、ANSI'85 と ANSI'74 以外のファイル状態コードが返されるように設定できます。この設定を行うには、次の手順を実行します。

1. CONVERTSTATUS ファイルハンドラ構成オプションを使用して、返される状態値をマップするために呼び出されるプログラムの名前を指定します。 このプログラム名は、コンパイラのタイプによって異なります。

コンパイラ	CONVERTSTATUS=
Ryan MacFarland COBOL	rmstat
IBM メインフレーム COBOL	hoststat
Microsoft COBOL V2	msstat

CONVERTSTATUS パラメータは、ファイルハンドラ構成ファイル extfh.cfg に追加できます。詳細については、『ファイルハンドラの構成』の章を参照してください。

2. リンク付きのアプリケーションを作成している場合は、オブジェクトをアプリケーションにリンクします。 リンクするオブジェクトは、コンパイラによって異なります。

コンパイラ	オブジェクト
Ryan MacFarland COBOL	rmstat.obj
IBM メインフレーム COBOL	hoststat.obj
Microsoft COBOL V2	msstat.obj

3. コンパイルする際には、次のコンパイラ指令を使用します。
    - NOANS85
    - COBFSTATCONV

注：Microsoft COBOL V2 ファイル状態コードのリストについては、ヘルプトピック『Microsoft COBOL V2 のファイル状態コード』を参照してください。

拡張ファイル状態コード

ANSI'74 と ANSI'85 のファイル状態規則は、拡張ファイル状態コードで拡張されます。拡張ファイル状態コードでは、ファイル状態の最初のバイトが "9" になります。 2 番目のバイトはバイナリ (COMP-X) の数字で、ランタイムシステムエラー番号と同じになります。

拡張ファイル状態コードは、9/nnn のように表示されます。nnn は 2 番目のバイトのバイナリの数字を指します。

たとえば、ディスクにファイルを書き込んでいるときにディスク容量が足りなくなった場合は、ANSI'74 ファイル状態は "30" になります。これは、次に示すエラーメッセージに変換されます。

永続的エラー。これ以外の情報はありません。

このエラーメッセージは非常に一般的です。"永続的エラー" は、ディスクに障害がある、またはディスクドライブのふたが開いていることを示します。この COBOL システムでは、包括的なファイル状態を返すのではなく、拡張ファイル状態 9/007 を返します。最初のバイトの文字 "9" と 2 番目のバイトのバイナリ値 7 を合わせると、「ディスクがいっぱいで

す。」という意味になります。

ANSI'74 または ANSI'85 のファイル状態コードを使用している場合に、より具体的な拡張ファイル状態があると、ランタイムシステムは拡張ファイル状態コードを返します。詳細については、ヘルプトピック
『拡張ファイル状態コード』を参照してください。

次のコード例は、標準のファイル状態を拡張ファイル状態として使用できるように再定義する方法を示します。この例では、入力ファイルが存在しないことが想定されています。そのため、OPEN INPUT 文を実行すると、9/013 (「ファイルが見つかりません。」) というファイル状態が返されます。

```
     select in-file
         assign to "user.dat".
         file status is file-status.
        ...
 working-storage section.
 01 file-status.
     05 status-key-1             pic x.
     05 status-key-2             pic x.
     05 status-key-2-binary redefines status-key-2
                                 pic 99 comp-x.
     ...
 procedure division.
     open input in-file
     if file-status not = "00"
         if status-key-1 = "9"
             if status-key-2-binary = 13
                 display "ファイルが見つかりません。"
        ...
```

拡張ファイル状態コードを表示するには、最大値 255 を格納できる大きさの表示フィールドにファイル状態データ項目の 2 番目のバイトを移動する必要があります。

```
     select in-file
         assign to "user.dat"
         file status is file-status.
 ...
 working-storage section.
 01 ans74-file-status.
     05 status-key-1             pic x.
     05 status-key-2             pic x.
     05 status-key-2-binary redefines status-key-2
                                 pic 99 comp-x.
 01 display-ext-status
     05 filler          pic xx value "9/"
     05 display-key 2            pic 999
     ...
```

```
procedure division.
    open input in-file
    if file-status not = "00"
        display "エラー。ファイル状態 =" with no advancing
        if status-key-1 = "9"
            move status-key-2-binary to display-key-2
            display display-ext-status
        else
            display file-status
        end-if
    end-if
```

ファイル状態　ファイルハンドラの構成

第 5 章：ファイルの共有

ここでは、ファイルロックとレコードロックを使用して、マルチユーザ環境で複数のユーザが同じデータを同時に変更できないようにする方法について説明します。

共有モード

共有モードは、特定のファイルについてファイルの共有とレコードのロックを行う場合に指定します。

共有モードには 3 種類あります。

共有モード	説明
SHARING WITH NO OTHER	このモードを設定したファイルを他のユーザと共有できません。
SHARING WITH READ ONLY	ファイルを入力ファイルとして開く場合は、他のユーザとの共有が可能です。
SHARING WITH ALL OTHER	他のすべてのユーザとの共有が可能です。

各共有モードでの OPEN 操作の結果を示す詳細な表については、次のヘルプトピックにある『別のファイル結合子で現在開いている利用可能な共有ファイルを開く』の表を参照してください。

『OPEN 文』

これらの共有モードを指定するには、OPEN 文で、または、SELECT 句の一部として SHARING 指定を使用します。OPEN 文で SHARING 指定を使用すると、SELECT 句の SHARING 指定は無効になります。

SELECT 句または OPEN 文のどちらにも SHARING 指定をしない場合には、共有モードは、次のどれかの条件が最初に満たされるときに決定されます。

指定する条件	共有モード
OPEN 文で WITH LOCK 指定をした場合	SHARING WITH NO OTHER
SELECT 句で LOCK MODE IS EXCLUSIVE 指定をした場合	SHARING WITH NO OTHER
SELECT 句で LOCK MODE IS MANUAL または LOCK MODE IS AUTOMATIC 指定をした場合	SHARING WITH ALL OTHER
OPEN 文のモードが OUTPUT、I-O または EXTEND の場合	SHARING WITH NO OTHER
OPEN 文のモードが INPUT で構成オプション OPENINPUTSHARED が OFF に設定されている場合	SHARING WITH READ ONLY

OPEN 文のモードが INPUT で構成オプション OPENINPUTSHARED が ON に設定されている場合	SHARING WITH ALL OTHER

レコードロック

ファイルの共有が有効である場合は、個々のレコードにロックを設定できます。 レコードにロックを設定すると、他のユーザは、ロックされたレコードを呼び出せず、ロック解除されているレコードを呼び出すようになります。

注：行順ファイルや入力ファイルとして開いたファイルでは、レコードをロックできません。

手動レコードロックと自動レコードロック

レコードをロックするには、手動または自動のどちらかで行います。

手動レコードロック

手動レコードロックを使用する場合は、READ 文を使用してレコードロックを取得してください。

手動ロックを使用するには、次のように指定します。

1. ファイルに対して SELECT 句で LOCK MODE IS MANUAL 句を指定します。
2. READ 文と WITH LOCK 句を指定します。

例

```
select fd-name
    assign to "muser.dat"
    lock mode is manual
  ...
    read fd-name with lock
```

自動レコードロック

自動レコードロックを使用する場合には、READ 文はレコードロックを自動的に取得します。 レコードをロックしない場合は、READ 文で WITH NO LOCK 句を指定してください。

自動ロックを指定するには、ファイルに対して SELECT 句で LOCK MODE IS AUTOMATIC 句を使用してください。

```
select fd-name
    assign to "muser.dat"
    lock mode is automatic
  ...
```

```
read fd-name
```

単一および複数のレコードロック

ロックするレコード数を、単一または複数のどちらかに指定できます。

単一レコードロック

単一レコードロックを使用する場合は、1 回に 1 つのレコードロックしか保持できません。 ファイルで新しく入出力操作を行うたびに、前のロックが解放されます。

単一のレコードロックを指定するには、LOCK MODE IS MANUAL 句、または LOCK MODE IS AUTOMATIC 句の後に WITH LOCK ON MULTIPLE RECORDS 句を使用しません。

複数レコードロック

複数レコードロックでは、同時に多くのレコードロックを保持できます。

複数レコードロックを指定するには、LOCK MODE IS MANUAL 句または LOCK MODE IS AUTOMATIC 句の後に WITH LOCK ON MULTIPLE RECORDS 句を続けます。 たとえば、次のようになります。

```
select fd-name
    assign to "muser.dat"
    lock mode is automatic with lock on multiple records
  ...
```

注：

- WITH LOCK ON MULTIPLE RECORDS 句を使用しない場合は、単一のレコードロックが有効になります。
- ファイルに複数のレコードロックを指定し、プログラムをコンパイルするときに WRITELOCK コンパイラ指令を使用する場合には、プログラムは、WRITE 文や REWRITE 文でファイルにアクセスするたびに、レコードロックを取得します。

レコードロックの処理

デフォルトでは、読み取り処理中にロックされたレコードを検出すると、レコードロック状態コードを返します。 この場合には、レコード領域は更新され、レコード内容が表示されます。 ただし、現在のレコードポインタはロックされたレコードと関連付けられたままであるため、この操作の後に続く READ NEXT 操作は同じレコードを呼び出します。

次の構成オプションは、上記の処理方法に影響します。

構成オプション	説明
IGNORELOCK	これを ON に設定すると、入力用に開いたファイルを読み取るときにレコードロックを無視します。

RETRYLOCK	これを ON に設定すると、ロックされたレコードへの呼び出し処理が成功するまで再試行します。 整数を設定すると、操作を中断するまでの最大再試行回数を指定できます。
LOCKTYPE	1 に設定すると、レコードロックはレコード自体の上に配置されます。この設定では、読み取り処理中にロックされたレコードを検出した場合は、そのデータレコードは返りません。 この場合は、読み取り処理後のレコード領域の内容は未定義の状態になります。
SKIPLOCK	これを ON に設定した場合は、読み取り処理中にロックされたレコードを検出すると、READ NEXT 操作では、ロックされたレコードの次にあるレコードが返されます。

注：ロックされたレコードをバイパスするために、START…KEY IS GREATER THAN.. 文も使用できます。

レコードロックの解放

ロックを取得したプログラムが次のどれかの操作を行うと、すべてのレコードロックが解放されます。

- ファイルを閉じる
- COMMIT 文を実行する
- ROLLBACK 文を実行する
- ファイルに対して UNLOCK 文を実行する
- ロックされたレコードを削除する

さらに、ロックを取得したプログラムが START 以外のファイル操作でファイルの他のレコードを呼び出した場合は、単一レコードロックが解放されます。

ファイル状態コード

プログラムをマルチユーザ環境で実行する場合は、次のファイル状態コードを常に確認することが必要です。

ファイル状態コード	説明
9/065	この状態コードは、OPEN 文により返され、ファイルがロックされていることを示します。
9/068	この状態コードは、READ 文、DELETE 文 または REWRITE 文により返され、レコードがロックされていることを示します。

9/213	この状態コードは、最大許容ロック数に達したことを示します。 最大許容ロック数は、ネットワークとオペレーティングシステムによって異なります。

ロックのアプリケーション例

ファイルロックとレコードロックのアプリケーション例が、提供されています。このアプリケーションは、メインプログラムである LOCKING.CBL と、このメインプログラムで呼び出されるその他のプログラムから構成されています。

このアプリケーションは、Examples\Net Express IDE ディレクトリの locking ディレクトリにあります。

このアプリケーションを実行するときには、最初にオプション 4 を使用してファイルを作成します。レコードロックの影響を表示するには、複数のセッションでオプション 2 と 3 を使用してください。

ファイルの共有 ファイルハンドラおよびソート API

第 6 章：ファイルハンドラの構成

ファイルハンドラの動作の一部を構成できます。 これらの動作は、ファイルハンドラ構成ファイルに値を入力して変更します。

構成ファイル

ファイルハンドラのデフォルトのファイル名は extfh.cfg です。ただし、次のように EXTFH 環境変数を設定し、別のファイル名を使用できます。 たとえば、次のように設定します。

```
set EXTFH=c:¥mydir¥test.cfg
```

上記のように記述すると、ファイル名は c:\mydir\test.cfg に設定されます。

ファイルハンドラ構成ファイルを使用して、個々のファイルまたはすべてのファイルについて、ファイルハンドラパラメータを変更できます。すべてのファイルに適用する設定は、次に示すタグの下に一覧表示されます。

```
[XFH-DEFAULT]
```

また、個々のファイルに適用する設定は、次のような個々のファイル名の下に一覧表示されます。

```
[TEST.DAT]
```

すべてのファイルに対する設定より、個々のファイルに対する設定の方が優先されます。ファイル名だけを指定する場合には、解決済みファイル名（ファイルハンドラがファイル名マッピング処理を行った後に使用するファイル名）を使用する必要があります。 または、ファイル名の前にキーワード INTERNAL を追加して、そのファイル名が SELECT 句で使用した名前であることを表すことができます。これにより、設定はファイル名マッピング処理の終わりに特定されたファイルに適用されます。たとえば、ASSIGN モードが動的割り当ての場合に、次のように記述します。

```
[INTERNAL:$myfile¥outfile.dat]
IDXFORMAT=8
```

この myfile は、そのフォルダがあるディレクトリの名前を保持する環境変数です。設定は、このディレクトリの outfile.dat に適用されます。

ASSIGN モードが外部割り当ての場合には、次のように記述します。

```
SELECT MYDATA ASSIGN TO EXTERNAL REALDATA
```

上記のように記述すると、次のように設定できます。

```
[INTERNAL:REALDATA]
```

この場合、後続のすべてのオプションが、論理名 "REALDATA" がマッピングされたファイルに適用されます。

次のように設定した場合は結果が異なります。

```
SELECT MYDATA ASSIGN TO '¥directory¥subdirectory¥datadirectory
¥myfile'
```

上記のように記述すると、extfh.cfg で INTERNAL オプションを使用できません。 extfh.cfg 内では SELECT 文の定数値は、角かっこ ([ ]) で囲んで反復する必要があります。

ファイル名マッピングの詳細については、『ファイル名』の章にある『ファイル名のマッピング』の項を参照してください。

ファイル名には、パスと DD_マッピングを使用できますが、ワイルドカードや論理名は使用できません。 ファイルハンドラ構成オプション BASENAME=ON を設定した場合には、パスを指定せずにファイル名を指定できます。 ファイル名はプログラム内のマッピングされた名前と一致する必要があります。

構成オプション

次の表は、ファイルハンドラ構成ファイルで設定できるオプションのリストです。

パラメータ	説明
BASENAME	タグで指定したファイル名をパス名なしで表すことができるようにするかどうかを指定します。
COMMITFLUSH	Fileshare を使用していないプログラムのコンテキストで、COMMIT 文や ROLLBACK 文により、レコードロックを解放するのみでなく、すべてのファイル更新をディスクに書き出すかどうかを指定します。
CONCATNAME	'+' 文字を含むファイル名を、通常のファイル名とファイル連結文字のどちらで解釈されるようにするかを指定します。
CONVERTDBSPACE	行順ファイルを読み取るときに、2 バイトの文字集合 (DBCS) の空白文字を ASCII 空白文字に変換できるようにします。
CONVERTEUCKATAKANA	行順ファイルを読み取るときに、EUC シングル幅 2 バイト文字を、等価のダブル幅のカタカナに変換できるようにします。
CONVERTSTATUS	呼び出すプログラムの名前を指定し、ファイルハンドラが処理を完了した後で、返された状態値をエミュレーション用にマップします。
DATACOMPRESS	データ圧縮を有効化するかどうかと圧縮の種類を指定します。
DATAFILE	OPEN 文に渡されたファイル名を別の名前にマップします。
EXPANDPOSITIONING	WRITE AFTER POSITIONING 文 (OS/VS COBOL 互換) を使用するときに、レコードにキャリッジ制御情報を書き込むかどうかを指定します。

EXPANDTAB	行順ファイルまたは行送りファイルで READ 操作中に検出されたタブコードを同じ数の空白文字に拡張するかどうかを指定します。
FASTREAD	索引ファイルを読み取るときに、ファイルハンドラで、データの完全性を保証するための追加確認を実行するかどうかを指定します。
FHREDIR	Fileshare を使用してリモートサーバでのファイル処理を可能にするかどうかを指定します。
FILEMAXSIZE[1]	大容量ファイルにアクセスできるように、ロックの呼び出しで使用するオフセットを保持するサイズ (バイト単位) を指定します。
FILEPOINTERSIZE	形式 8 の索引ファイルについて、ファイルポインタを格納するために使用するバイト数を指定します。
IDXDATBUF	索引ファイルのデータ部分にアクセスするときに使用するバッファのサイズを指定します。
IDXFORMAT	索引ファイルを作成するときに使用する形式を指定します。
IDXNAMETYPE	データファイルと、索引ファイル (存在する場合) の両方のファイル名タイプの形式を指定します。
IGNORELOCK	入力用に開いたファイルを読み取るときに、ロックを無視するかどうかを指定します。
INDEXCOUNT	索引ファイルについてキャッシュする索引ノードの数を指定します。
INDEXFILE	索引ファイルの名前を、個別の索引ファイルがある形式の名前にマップします。
INSERTNULL	行順ファイルや行送りファイルに WRITE 操作、または REWRITE 操作を実行するときは、印刷不可能な文字の前にヌル文字 (x"00") を挿入するかどうかを指定します。 また、READ 操作中に、これらのヌル文字を削除するかどうかを指定します。
INSERTTAB	行順ファイルや行送りファイルに WRITE 操作、または REWRITE 操作を実行するときは、タブコードを連続した空白文字に置き換えるかどうかを指定します。
KEYCHECK	アプリケーションで定義したキー定義と開こうとしている索引ファイルのキー定義が一致することをファイルハンドラで確認するかどうかを指定します。
KEYCOMPRESS	使用中のキー圧縮の種類を指定します。
LOADONTOHEAP	ファイルを開くときに、ファイルをメモリにロードするかどうかを指定します。
LOCKTYPE	使用中のレコードロックの種類を指定します。

LOG	ファイルの破損や自動ファイル回復などのファイルに関する異常な状況を、後で確認できるようにログファイルに記録するかどうかを指定します。
LOGFILENAME	LOG オプションが有効なときに、ログファイルの名前を指定します。
LSRECDELIM	行順ファイルでレコード区切り文字として使用する文字を指定します。
MAINFRAMEPRINT	WRITE AFTER ADVANCING または WRITE BEFORE ADVANCING を使用しているファイルについて、メインフレームのプリンタ形式 (filetype(11)) を使用するかどうかを指定します。
NAMEOPTIONS	OPEN 文に渡す名前に、角かっこ ([ ]) で囲まれたファイルハンドラ指令を含めてよいかどうかを指定します。 このパラメータは、古い Micro Focus 製品との互換性を確保するために提供されています。Micro Focus Workbench V4.0 との互換性が必要な場合のみに使用してください。
NFSFILELOCK	UNIX NFS ファイルシステムのレコードロックとファイルロックをアプリケーションで検出できるようにします。
NODESIZE	索引ファイルに使用する索引ノードのサイズを指定します。
NOSEQCHECK	ACCESS SEQUENTIAL が指定された索引ファイルを出力または拡張用に開く場合に、書き込まれる各新規レコードの主キーが、以前のキーよりも大きいかどうかを確認するかどうかを指定します。
OPENINPUTSHARED	入力用に開いたファイルに LOCK MODE 句を指定していない場合に、このファイルを他のユーザと共有可能にするかどうかを指定します。
OSVSREWRITE	出力用に開いた順ファイルに対して WRITE 文を使用可能にするかどうかを指定します。 WRITE 文を有効化した場合は、REWRITE 文とまったく同じ動作を行います。
READSEMA	ファイル修正以外の目的で入出力操作を実行する場合に、システムに共有ファイルのセマフォを獲得させるかどうかを指定します。
RELDATBUF	相対ファイルにアクセスするときに使用するバッファのサイズを指定します。
RELRECDELIM	固定長形式の相対ファイルでレコード区切り文字として使用する文字を指定します。
RETRYLOCK	ある操作でロックされたレコードにアクセスしようとしたときに、この操作を再試行するかどうかを指定します。
RETRYOPEN	ある操作でロックされたファイルにアクセスしようとしたときに、この操作を再試行するかどうかを指定します。

RETRYTIME	RETRYLOCK オプションまたは RETRYOPEN オプションに設定された整数が、試行回数と秒数のどちらを表すかを指定します。
RUNITLOCKDETECT	同じ実行ユニットで実行された別の OPEN 文によりレコードがロックされている場合に、これを検知できるようにするかどうかを指定します。
SEARCHONCREATE	ファイルハンドラが、作成中のファイルと同じ名前をもつファイルを探すため、複数ディレクトリパス上の特定のディレクトリを検索するかどうかを指定します。
SEQDATBUF	順ファイルにアクセスするときに使用するバッファのサイズを指定します。
SKIPLOCK	順読み取り中にロックされたレコードが検出された場合に、現在のレコードポインタを進めるかどうかを指定します。
SPACEFILL	行順ファイルや行送りファイルに READ 操作を実行する場合に、読み取ったデータ以外のレコード領域を空白文字で埋めるかどうかを指定します。
STARTUNLOCK	START 操作で、単一レコードのロックが指定されている (WITH LOCK ON MULTIPLE RECORDS 指定が設定されていない) ファイルの既存のロックを解除するかどうかを指定します。
STRIPSPACE	行順ファイルや行送りファイルに WRITE 操作または REWRITE 操作を実行する場合に、後続の空白文字を削除するかどうかを指定します。
SUPPRESSADV	レコード順ファイルの場合に、WRITE 文の ADVANCING 指定を無視するかどうかを指定します。
TRACE	ファイルハンドラのトレースを有効にするかどうかを指定します。
TRACEFILEEXTEND	既存のトレースファイルがある場合に、トレース情報を現行のトレースファイルの末尾に追加するかどうかを指定します。
TRACEFILENAME	TRACE オプションが有効なときに、作成するトレースファイルの名前を指定します。
WRITELINE	メインフレームまたは PC の動作をレコードの書き込みに使用するかどうかを指定します。
WRITETHRU	ファイルの修正をすぐにディスクに書き出すかどうかを指定します。 または、COBOL システムやオペレーティングシステムにより、変更を内部的にバッファに書き込み、後でディスクに書き出すこともできます。

脚注

1.

次の警告を参照してください。

警告：FILEMAXSIZE オプションでは、ファイルのオフセットを保存するために内部で使用されるバイト数を指定します。 これは、内部のロック機構にも影響します。 同じファイルを共有するすべてのプログラムで同じ FILEMAXSIZE 設定を使用して、セマフォとレコードロックが適切に処理されるようにし、共有ファイルでのファイルの破損を防ぐ必要があります。

FILEMAXSIZE オプションのデフォルト値は 4 です。この場合には、ファイルのアドレス指定は、Micro Focus システムの初期のバージョンと互換する 32 ビット値に制限されます。FILEMAXSIZE オプションを 8 に設定することもできます。この値を設定すると、NTFS ファイルにアクセスするときに、Windows プラットフォームでのファイルのアドレス指定に最大 64 ビット値を使用できますが、基になるレコードロック機構が変更されてしまいます。

次の場合は、FILEMAXSIZE の値を 8 に設定しないでください。

- 64 ビットのファイルシステムをサポートしないオペレーティングシステムでアプリケーションを実行している場合
- FAT ファイルシステムにアクセスするアプリケーションを実行している場合。FAT ファイルシステムは、32 ビットを超えるファイルのアドレス指定をサポートしません
- 32 ビットのファイルシステムの共有ファイルにアクセスする必要のあるアクセスプログラムがある場合
- 32 ビットのファイルアクセスのみに対応した旧バージョンの Micro Focus COBOL を使用するアプリケーションとの間でファイルを共有している場合

64 ビットおよび32 ビットのファイルシステムの詳細については、付録の『ファイルシステム』を参照してください。

これらの各オプションの有効な設定値とデフォルト値の詳細については、ヘルプトピック『構成ファイルオプション』を参照してください。

構成ファイルの例については、ヘルプトピック『構成ファイルの例』を参照してください。

構成可能な動作

ファイルハンドラ構成ファイルで構成できるファイルハンドラの動作は、次のとおりです。ここでは、オンラインヘルプよりも詳しく説明します。

大容量索引ファイル

デフォルトでは、ファイルハンドラは、最大 2 GB のファイルを処理できます。 ただし、2

GB より大きい索引ファイルを作成する必要がある場合は、ファイルハンドラ構成ファイルで IDXFORMAT パラメータを 8 に設定し、IDXFORMAT(8) ファイルを作成します。通常、索引ファイルは .idx ファイルと .dat ファイルに分かれているのに対して、これらのファイルは、単一のファイルとして構成され、最大 32 TB のサイズにすることができます。

多くのオペレーティングシステムでは、ファイルサイズに制限があります (たとえば、Windows 95 では 2 GB)。 これらのシステムで大容量ファイルを作成できるようにするには、ストライプ化オプションを使用します。 ストライプ化は大容量論理ファイルを必要な数の物理的なファイルに分割します (最大数 256)。

ファイルのストライプ化

ファイルストライプ化オプションを使用すると、最大 512 GB の論理ファイルを作成できます。 これらの論理ファイルは、最大 256 の物理ファイルで構成することができます。

これらの大容量の論理ファイルは複数の物理ファイルで構成されるため、ファイルストライプ化をグローバルに行うと、オペレーティングシステムでファイルハンドルが足りなくなる可能性があります。 そのため、ファイルストライプ化をグローバルに行うことはできません。

ファイルをストライプ化するには、ファイルハンドラ構成ファイルでストライプ化する各ファイルに STRIPING=ON オプションを設定します。

ストライプ化ファイルのヘッダー

ストライプ化したファイルでは、個々のストライプのヘッダーの長さは 128 バイトになります。ヘッダーには、次の形式の ASCII 文字列が含まれます。

ファイルのストライプ番号 *nnn* : *filename*

この *nnn* には、ストライプ番号を表す 3 桁の数値を指定します。*filename* は、関連付けられたファイルの名前です。

文字列は、128 バイトのヘッダーの終わりまで空白文字で埋められます。

オプション

ストライプ化オプションでは、大文字小文字を区別しません。

注：

- ストライプ化したファイルを使用する場合には、ファイルを開くたびに、ファイルハンドラ構成ファイルで必要なストライプ化オプションが指定されている必要があります。
- ファイルをストライプ化している場合に、ストライプの 1 つを削除すると (または、ディスク障害などの理由で使用不能になった場合)、ファイル全体が使用できなくなります。

STRIPING

ファイルをストライプ化するかどうかを指定します。 デフォルトの設定は OFF です。

構文：

```
STRIPING=[ON/OFF]
```

STRIPENAMETYPE

このオプションでは、ストライプに名前を付けるためのファイル命名規則を指定します。 デフォルトの設定は 0 です。

構文：

```
STRIPENAMETYPE=[0/1]
```

注釈:

StripeNameType を 0 に設定すると、ストライプファイルは基本ファイルと同じ名前になりますが、ファイル拡張子の前にある基本名に番号が追加されます。たとえば、ファイル test.dat に 3 個のストライプがある場合には、基本ファイル名とそのストライプは次のように表記されます。

ファイル	説明
test.dat	基本ファイル
test01.dat	ストライプ番号 1
test02.dat	ストライプ番号 2
test03.dat	ストライプ番号 3

基本名が長く、そのままではストライプファイル番号を入れることができない場合は、基本名の右端から文字が削除されます。たとえば、ファイル testfile.dat に 3 個のストライプがある場合には、ファイルとそのストライプは次のように表記されます。

ファイル	説明
testfile.dat	基本ファイル
testfi01.dat	ストライプ番号 1
testfi02.dat	ストライプ番号 2
testfi03.dat	ストライプ番号 3

StripeNameType を 1 に設定すると、ストライプファイル名は基本ファイルと同じ名前になりますが、ファイル名のファイル拡張子の後にストライプ番号が追加されます。 基本ファイル名はストライプ 0 に変更されることに注意してください。たとえば、ファイル test.dat に 3 個のストライプがある場合には、基本ファイル名とそのストライプは次のように表記されます。

ファイル	説明
test.dat.00	基本ファイル

test.dat.01	ストライプ番号 1
test.dat.02	ストライプ番号 2
test.dat.03	ストライプ番号 3

MAXSTRIPESIZE

個々のストライプの最大サイズをバイト数で指定します。n のデフォルト値は 1 GB (1,073,741,824 バイト) です。 最小値は 65,536 バイトです。 最大値は現在のところ 2 GB (2,147,483,648 バイト) です。

構文：

`MAXSTRIPESIZE=`*n*

MAXSTRIPEFILES

基本ファイル以外のストライプファイル数を指定します。 デフォルトでは、17 ファイルが作成されます。つまり、基本ファイルと 16 個のストライプです。n の最小値は 1 で、最大値は 255 です。

構文：

`MAXSTRIPEFILES=`*n*

MAXSTRIPEDIGITS

ストライプに名前を付けるときに、基本ファイルの名前に追加する桁数を指定します。 たとえば、STRIPETYPENAME が 0 の場合、2 を指定すると test01.dat、5 を指定するとオペレーティングシステムのファイル名の制限により test00001.dat か tes0001.dat になります。n のデフォルト値は 2 です。最小値は 1 で、最大値は 5 です。

構文：

`MAXSTRIPEDIGITS=`*n*

STRIPE-X

ストライプファイルのパスとサイズを指定します。x はストライプ番号、path はストライプがある場所、n はストライプのサイズを示します。n を省略すると、ストライプのサイズは前のストライプと同じになります。

STRIPE-X オプションを一度設定すると、もう一度 STRIPE-X オプションで変更するまで、すべてのストライプに新しいパスとサイズが適用されます。 Stripe- の後に続く数字はストライプ番号で、0 は基本ファイルを表します。 基本ファイル以外のストライプには、0 という文字を指定しないでください。

構文：

```
STRIPE-X=path[,n]
```

例

ファイルのストライプ化を指定する方法をいくつか例を挙げて説明します。

例 **1**

ファイル test.dat を、同じサイズの 2 つのストライプに分割します。

```
[test.dat]
Striping=on
Stripe-1=dir2
Stripe-2=dir3
```

この結果、test.dat は現在のディレクトリに格納されます。test01.dat は dir2 に、test02.dat は dir3 に格納されます。

例 **2**

ファイル test.dat を 5 つのストライプに分割します。

```
[test.dat]
Striping=on
Stripe-0=dir1
Stripe-4=dir2
```

この結果、test.dat、test01.dat、 test02.dat、test03.dat は dir1 に格納されます。test04.dat と test05.dat は dir2 に格納されます。

例 **3**

ファイル test.dat を 3 つのストライプに分割します。

```
[test.dat]
Striping=on
Stripe-0=dir1,100000000
Stripe-1=dir2,200000000
```

この結果、test.dat は、ファイルサイズが 100,000,000 バイトになり、dir1 に格納されます。また、test01.dat は、ファイルサイズが 200,000,000 バイトになり、dir2 に格納されます。

パフォーマンス

いくつかの手順を実行して、ファイルハンドラのパフォーマンスを向上できます。 これらのすべての手順で、さまざまな構成オプションを使用します。 これらの手順について次に詳しく説明します。

- 索引ファイルのノードを、ディスクに格納するのではなく、キャッシュすることで、ノードの数を減らすことができます。 INDEXCOUNT 構成オプションを使用してノードの数を指定します。ノードの数は 32 に設定することをお奨めしますが、最大で 64 まで指定できます。ただし、64 に設定すると、メモリの使用量が増えてしまいます。

- ファイルを変更しない入出力操作を実行し、ファイルを共有している場合は、ファイルハンドラでセマフォを獲得しないようにすることができます。 セマフォは、ファイル内の予約された場所に適用されるロックで、複数のユーザがファイルにアクセスすることを禁止します。 この追加の呼び出しを行うと、パフォーマンスが低下します。 ファイルハンドラでセマフォを獲得するのを停止するには、READSEMA 構成オプションを OFF に設定します。

- レコードロックの使用を調整できます。 ファイルハンドラは、デフォルトで次のような動作を行います。

    - ロックされたレコードを検出すると、ロックレコードファイル状態を返しますが、このときにレコード領域のデータも一緒に返します。

    - ロックの必要がない読み取り操作の実行時に、ファイルロックが設定されているかどうかを確認します。

  ロックが設定されているかどうかを確認すると、パフォーマンスが大幅に低下します。 この手順が行われないようにするには、次に示す構成オプションのどれかを使用します。

    - LOCKTYPE。 このオプションを 1 に設定し、ロックされたレコードにアクセスできないようにします。

    - IGNORELOCK。このオプションを ON に設定し、ロックを必要としない読み取り操作の実行時には、ロックが設定されているかどうかをファイルハンドラで確認しないようにします。

  LOCKTYPE を 1 に設定すると、IGNORELOCK の設定が無効になります。

- ファイルを開くときにファイル全体をメモリにロードし、すべてのファイル操作をメモリで実行するようにして、ファイルを閉じるときのみにディスクに書き込みを行うように指定できます。 この動作を実行するためには LOADONTOHEAP を ON に設定します。 この設定は、非共有のファイルに対してのみ実行できます。 この設定は十分に注意して使用し、容量の小さいファイルのみで実行してください。

ファイルハンドラの構成　mfsort ユーティリティ

第 7 章 : ファイルハンドラおよびソート API

ここでは、プログラムでファイルハンドラとソートモジュールを明示的に呼び出す方法を説明します。

ファイルハンドラの呼び出し

ファイルハンドラを呼び出すと、ファイルを開いたり、レコードを読み取ったりするといった一般的なアクションを効率的で一貫性のある方法で実行できます。

概要

次の構文を使用すると、プログラムで呼び出し可能ファイルハンドラを明示的に呼び出せます。

```
call "EXTFH" using opcode fcd
```

パラメータの内容は、次のとおりです。

パラメータ	説明
"EXTFH"	ファイルハンドラインターフェイスのモジュール名です。
opcode	ファイルハンドラの操作コードです。後述の『操作コード』の項を参照してください。
fcd	ファイルハンドラがアクセスするファイルの詳細を保持する FCD (ファイル制御記述) と呼ばれるデータ領域です。後述の『データ構造体』の項を参照してください。

最初の呼び出しの前に、次の手順を行う必要があります。

1. ファイル制御記述 (FCD)、レコード領域、ファイル名領域、キー定義ブロック (索引ファイルの場合) にデータ領域を割り当てます。
2. ファイルハンドラが無効な値を受け取らないように、すべてのデータ領域をバイナリのゼロに初期化します。
3. FCD に次の領域へのポインタを設定します。
    - レコード領域
    - ファイル名領域
    - キー定義ブロック (索引ファイルの場合のみ)

その後で、各ファイルハンドラ操作に対して次の手順を行います。

1. 選択した操作コードに対応する FCD のフィールドに入力を行います。
2. ファイルハンドラを呼び出します。
3. ファイル状態を確認して、ファイルの入出力操作が正常に行われたかどうかを判断します。
4. FCD フィールドやレコード領域のデータを処理します。

データ構造体

ファイルハンドラでは、ファイルの入出力操作中に次の 4 つのデータ構造体を使用します。

データ構造体	説明
ファイル制御記述 (FCD)	100 バイトのデータ領域で、使用中のファイルに関する情報が記述されます。
レコード領域	レコード領域は、レコードの読み取りや書き出しを行うデータ領域です。
ファイル名領域	ファイル名領域は、使用中のファイルの名前を記述するデータ領域で、オペレーティングシステムにより認識されます。
キー定義ブロック	ファイルハンドラが索引ファイルでの操作中に索引キー情報を記述します。

ファイル制御記述 **(FCD)**

ファイル制御記述 (FCD:File Control Description) は、使用中のファイルに関する情報を記述するデータ領域です。FCD には 2 つのバージョンがあり、どちらを使用するかは COBOL 開発システムを 32 ビットと 64 ビットのどちらのオペレーティングシステムで実行するかによって決まります。これを次の表に示します。

COBOL 開発システム	使用する **FCD**
メインフレームの Express	FCD2
32 ビット Net Express	FCD2
64 ビット Net Express	FCD3
.NET サポートおよび .NET 対応 Net Express	FCD3
32 ビット Server Express	FCD2 または FCD3
64 ビット Server Express	FCD3

FCD2 フォーマットと FCD3 フォーマットは、どちらも 32 ビットコンパイルで使用できます。 プログラムをコンパイラ指令 P64 でコンパイルすると、コピーファイル XFHFCD.CPY で、FCD3 レコードが定義されます。 プログラムを P64 指令でコンパイルしない場合には、XFHFCD.CPY で FCD2 レコードが定義されます。 FCD3 レコードは xfhfcd3.cpy ファ

イルで直接、無条件に定義され、FCD2 レコードは xfhfcd2.cpy ファイルで直接、無条件に定義されます。

ファイルハンドラを使用するには、プログラムで FCD を設定し、該当するフィールドへの入力 (操作コードによってフィールドが異なります) とファイルハンドラの呼び出しを行います。 FCD の該当フィールドに入力を行うと、ファイルハンドラがプログラムに情報を返します。

FCD で未使用の領域や予約済みの領域は、バイナリゼロに設定する必要があります。

ファイルオープン呼び出しに使用した FCD は、その後このファイルに対してアクセスするときに必ず使用する必要があります。 1 つのファイルに対して複数のオープン操作を実行できますが、オープン操作ごとに別の FCD を使用する必要があります。

FCD には、レコード領域、ファイル名領域、キー定義ブロックへのポインタ (USAGE POINTER データ項目) を記述します。

FCD 構造体の詳細については、ヘルプトピック 『ファイル制御記述 (FCD)』を参照してください。

レコード領域

レコード領域は、レコードの読み取りと書き出しを行うデータ領域です。

レコード領域のサイズは、ファイルの最大レコードよりも 4 バイト大きい必要があります。

レコード領域を指す FCD の設定例を次に示します。

```
01 RECORD-AREA      PIC X(85).
   ...
     SET FCD-RECORD-ADDRESS TO ADDRESS OF RECORD-AREA
       ...
```

ファイル名領域

ファイル名領域は、使用中のファイルの名前を含むデータ領域で、オペレーティングシステムにより認識されます。

ドライブやパスの情報のみではなく、ファイルの実際の名前も記述できます。 名前の最後は空白文字でなければなりません。

このファイル名領域は、ファイルに対する最初の操作より前に記述する必要があります。

ファイル名領域を指す FCD の設定例を次に示します。

```
01 FILENAME-AREA PIC X(65) VALUE "master.dat".
   ...
     MOVE 65 TO FCD-NAME-LENGTH
     SET FCD-FILENAME-ADDRESS TO ADDRESS OF FILENAME-AREA
       ...
```

キー定義ブロック

キー定義ブロックは、索引ファイルを開くときに、ファイルハンドラに索引キー情報を渡すために使用します。 キー定義ブロックは、次に示す 3 つのデータ領域で構成されます。

- グローバル情報領域
- キー定義領域
- 構成要素定義領域

グローバル情報領域

グローバル情報領域には、キー定義領域のサイズとファイルに含むキーの数を記述します。 詳細なレイアウトについては、ヘルプトピック 『グローバル情報領域』を参照してください。

キー定義領域

キー定義領域には、索引ファイルで使用するキーを記述します。

キー定義領域は、グローバル情報領域の次に配置され、ファイルにある各キーにつきキー定義を 1 つ記述します。

すべてのキーは、構成要素より前に定義する必要があります。

キー位置の順序に基づいてキーを識別するため、キーを定義する順序は重要です。 たとえば、1 つの主キーと 2 つの副キーを持つ索引ファイルの場合、キー定義領域には、3 つのキー定義が記述されます。つまり、主キーはキー 0、1 つ目の副キーはキー 1、2 つ目の副キーはキー 2 となります。

キー定義領域の詳細なレイアウトについては、ヘルプトピック 『キー定義領域』を参照してください。

構成要素定義領域

構成要素定義領域は、キー定義領域の後にあります。

この領域では、各キーの構成要素を定義します。分割キー以外の各キーは、1 つの構成要素から成り立ち、各構成要素をそれぞれ定義する必要があります。

構成要素の定義では、キー構成要素の位置と長さを指定します。

数字キーの場合は、構成要素定義領域を使用して、数値の型を指定できます。 その後で、IXNUMKEY コンパイラ指令を使用すると、数値の型に従ってキーを順序付けることができます (デフォルトでは、キーは英数字順に並びます)。

構成要素定義領域の詳細なレイアウトについては、ヘルプトピック 『構成要素定義領域』を参照してください。

FCD へのアクセス

プログラムで通常の COBOL 構文を使用してファイルの入出力操作を行う場合は、各ファイ

ルに対してファイル制御記述 (FCD) が自動的に作成されます。 この自動的に作成された FCD にアクセスするには、FCDREG コンパイラ指令でプログラムをコンパイルします。 これにより、FCD の読み取りや変更、FCD を使用したファイルハンドラの明示的な呼び出しなどが可能になります。

FCD にアクセスするには、プログラムの連絡節に FCD 定義を設定する必要があります。 FCD 定義の例は、コピーブックの XFHFCD.CPYに保存されています。 プログラムで次の SET 文を使用すると、読み取りや変更を行う FCD にこの定義をマップできます。

```
set address of fcd to address of fh--fcd of file
```

パラメータの内容は、次のとおりです。

パラメータ	説明
fcd	プログラムの連絡節に記述する FCD 定義の名前。
fh--fcd	ファイルにアクセスするために、この COBOL システムで使用する FCD (ハイフンが 2 つあることに注意してください)。
file	ファイルの FD 名

この SET 操作に続けて、連絡節で設定した FCD のデータ項目にアクセスすると、ファイルの FCD へアクセスできます。

同様に、次のように指定すると、キー定義ブロックにアクセスすることができます。

```
set address of kdb to address of fh--keydef of file
```

パラメータの内容は、次のとおりです。

パラメータ	説明
kdb	プログラムの連絡節で記述するキー定義ブロックの名前。
fh--keydef	ファイルにアクセスするために、この COBOL システムで使用するキー定義 (ハイフンが 2 つあることに注意してください)。
file	ファイルの FD 名

操作コード

ファイルハンドラが実行するファイル操作は、2 バイトの操作コードで識別します。

操作コードには 2 つの種類があります。

操作コードの種類	説明
標準操作コード	このコードでは、MSB に x"FA" が格納されます。LSB は、特殊操作を示します。
特殊操作コード	このコードでは、MSB に x"00" が格納されます。LSB は、特殊操作を示します。

標準操作コードと特殊操作コードの詳細なリストと詳細情報へのリンクは、ヘルプトピック 『標準の操作コード - 概要』および 『特殊な操作コード - 概要』を参照してください。

相対バイトアドレス指定

レコードの相対バイトアドレスは、その特定のレコードに対するすべてのファイルハンドラの操作で FCD の fcd-reladdr-offset (または fcd-config-flags にビット 4 を設定した場合は fcd-reladdr-big) に格納されます。 レコードの相対バイトアドレスを使用するには、READ 操作の後でこのフィールドの内容を保存するのみです。

相対バイトアドレスを使用すると、高速にレコードへアクセスできます。ただし、次のような制限があります。

- 相対バイトアドレス操作は、現在のレコードポインタに影響しません。 そのため、相対バイトアドレス操作の後の READ (順) 操作では、通常の呼び出し方法で最後に呼び出したレコードの次のレコードを返します。 相対バイトアドレス指定を使用して呼び出したレコードの次のレコードを返すわけではありません。ただし、相対バイトアドレス操作を実行するためのファイルハンドラを呼び出す前に、fcd-config-flags にビット 5 を設定すると、現在のレコードポインタを更新し、相対バイトアドレス指定を使用して呼び出したレコードを示すことができます。
- レコードの再書き込みや削除が行われると、相対バイトアドレス操作は索引を更新します。
- レコードの相対バイトアドレスの取得後にそのレコードを削除すると、その相対バイトアドレスを使用した読み書きは通常失敗します。 レコードを別のレコードに置き換えることは可能ですが、このレコードを検出できないことがあります。

相対バイトアドレス操作では、レコードロックもサポートされています。

レコードの相対バイトアドレスを取得すると、それを使用して次の操作を実行できます。

- レコードの読み取り
- レコードの再書き込み
- レコードの削除

レコードの読み取り

相対バイトアドレスを使用してファイルから具体的なレコードを読み取る方法には 2 種類あります。

- FCD の fcd-reladdr-offse t または fcd-reladdr-big にアドレスを格納し、FCD の fcd-config-flags にビット 6を設定します。 現在のレコードポインタを更新する場合は、fcd-config-flags にビット 5 を設定します。
- READ (ランダム) WITH NO LOCK、READ (ランダム) WITH LOCK、READ (ランダム) WITH KEPT LOCK、または READ (ランダム) 操作をファイルに実行します。

READ (直接) 操作も同様に機能しますが、FCD にビットを設定する必要はありません。 READ (直接) 操作では、常に FCD の相対バイトアドレスフィールドで指定されたアドレスのレコードを返し、現在のレコードポインタを更新します。

上記のどちらの方法でも、現在の参照キーを別のキーに切り替えることができます。 たとえば、主キーにより READ (順) 操作を行っている場合に、次の手順を使用すると、現在のレコードの最初の副キーにより読み取りを開始できます。

1. ファイルの次のレコードを読み取ります (このレコードのアドレスは、FCD の相対バイトアドレスフィールドで指定します)。
2. FCD の fcd-key-id に新しい参照キーを格納します
3. READ (直接) 操作を実行して、このアドレスのレコードを返します。これで、参照キーは新しいキーに変更されます。
4. ファイルの次のレコードを読み取ります。 このレコードは、新しい参照キーの索引にある次のレコードです。

レコードの再書き込み

FCD の fcd-reladdr-offset または fcd-reladdr-big にアドレスを格納し、FCD の fcd-config-flags にビット 6 を設定すると、レコードを特定のアドレスに再度書き込むことができます。

現在のレコードポインタを更新する場合は、FCD の fcd-config-flags にビット 5 を設定します。

次に、REWRITE 操作を実行します。

レコードの削除

FCD の fcd-reladdr-offset または fcd-reladdr-big にアドレスを格納し、FCD の fcd-config-flags にビット 6 を設定すると、特定のアドレスのレコードを削除できます。

次に DELETE 操作を実行します。

ユーザ独自のファイルハンドラの作成

独自のカスタマイズされたファイルハンドラを作成して、COBOL システムで提供されているファイルハンドラの代わりに使用できます。

プログラムでファイルハンドラを使用して COBOL 入出力構文を処理するには、CALLFH コンパイラ指令を使用します。 たとえば、USERFH というファイルハンドラを使用する場合は、次のようなコンパイラ指令でプログラムをコンパイルします。

```
CALLFH"USERFH"
```

これにより、すべての COBOL 入出力操作が USERFH の呼び出しにコンパイルされます。

注：CBL_EXIT_PROC を使用する場合は、ファイルハンドラを呼び出しているアプリケーションの終了時に、ファイルハンドラが適切にシャットダウンされるように、優先順位を 200 に設定する必要があります。

新しい索引ファイルの作成

特殊操作コードを使用すると、次のように、既存の索引ファイルのデータファイル部から索引ファイルを再作成できます。

1. CREATE INDEX FILE (特殊操作コード x"07") を使用して、新しい索引ファイルを作成します。
2. GET NEXT RECORD (特殊操作コード x"08") を使用して、データファイルから各レコードを読み取ります。
3. レコードの各キーの値については、ADD KEY VALUE (特殊操作コード x"09") を使用して索引ファイルにキー値を追加します。

次の例では、既存のデータファイルから新しい索引ファイルを作成する方法を示しています。

```
 78 close-file                 value x"fa80"
 78 open-new-index             value x"0007".
 78 get-next-rec               value x"0008".
 78 add-key-value              value x"0009".
   ...
     move open-new-index to fh-opcode
     perform extfh-op
     move get-next-rec to fh-opcode
     perform extfh-op
     perform until fcd-status (1:1) not = "0"
         perform varying fcd-key-id from 0 by 1
                   until fcd-key-id = key-count
                      or fcd-status (1:1) not = "0"
             move add-key-value to fh-opcode
             perform extfh-op
         end-perform
         move get-next-rec to fh-opcode
         perform extfh-op
     end-perform
     move close-file to fh-opcode
     perform extfh-op
     ...
 extfh-op.
     call "EXTFH" using fh-opcode, fcd
     if fcd-status of fcd (1:1) = "1"
```

```
        move 1 to return-code
    end-if.
```

圧縮ルーチン

ユーザ独自のアプリケーションで、ファイルハンドラがデータとキーを圧縮するために使用する圧縮ルーチンを呼び出すことができます。 詳細については、『データとキーの圧縮』の章の『圧縮ルーチン』の項を参照してください。

ソートモジュール

ソートモジュールは、データファイルのソートとマージを可能にするスタンドアロンのソートモジュールです。 このモジュールは、COBOL プログラムで SORT 文または MERGE 文を使用すると、暗黙的に呼び出されます。

データファイルに SORT 操作を実行すると、重複キーをもつレコードは、SORT 文で DUPLICATES IN ORDER 指定をしたかどうかにかかわらず、元の順番で返されます。

呼び出し可能ソートモジュールを明示的に呼び出してファイルのソートやマージを行うには、プログラムに次の呼び出し文を記述します。

```
call "EXTSM" using function-code, sort-fcd
```

パラメータの内容は、次のとおりです。

パラメータ	説明
function-code	実行する操作の種類を示す 2 バイトのコード。詳細については、ヘルプトピック 『呼び出し可能ソートモジュールの機能コード』を参照してください。
sort-fcd	プログラムのデータ部で指定したソートファイル制御記述 (Sort FCD) の名前。 FCD には、ファイルのレコード領域、ファイル名、文字の大小順序、キー定義ブロック、およびファイル定義ブロックへのポインタを記述します。 プログラムのデータ部で、SORT 操作に必要な各ファイルに対して Sort FCD を指定する必要があります。詳細については、ヘルプトピック 『ソートファイル制御記述』を参照してください。

Sort FCD の機能コードの詳細なリストと詳細な記述を含む詳細情報については、ヘルプトピック 『呼び出し可能ソートモジュール』を参照してください。

第 8 章 : mfsort ユーティリティ

ここでは、mfsort 機能の使用法について説明します。

はじめに

この COBOL システムでは、次の方法を使用してファイルのソートとマージを行うことができます。

ソート方法	説明
ランタイムシステム COBOL ソートモジュール	COBOL プログラムで SORT 文を実行するデフォルトのモジュール。 このモジュールは、CALL 文を使用して直接呼び出すこともできます。 詳細については、『ファイルハンドラとソート *API*』の章の『ソートモジュール』の項を参照してください。
mfsort ユーティリティ	コマンド行から呼び出すことができるユーティリティ。データファイルのソートとマージが実行できます。

mfsort では、データファイルのソートとマージを実行できます。 mfsort は、IBM の Dfsort 製品リリース 14 をほぼ完全にエミュレートし、次の機能をサポートします。

- INCLUDE/OMIT - 選択したレコードのインクルードと省略
- SUM - 重複キー値を持つレコードの選択されたフィールド値の累計
- OUTREC - 出力レコードの編集
- OUTFIL - 複数の出力での複雑な編集とレポート処理
- 各種 Y2K 日付フィールドでの Y2K 順序付け
- 2 桁の年フィールドから 4 桁への拡張 (OUTFIL を使用)

DFSORT の詳細については、IBM の DFSORT Web サイトを参照してください。

mfsort の起動

mfsort は、コマンド行から次の方法の 1 つを使用して起動できます。

```
mfsort instructions
```

```
mfsort take filename
```

パラメータの内容は、次のとおりです。

パラメータ	説明
instructions	**mfsort** 命令。詳細については、『命令』の項を参照してください。 コマンド行で命令を指定するときは、オペレーティングシステムが定めたコマンド行の最大長に注意してください。
filename	**mfsort** 命令を含むテキストファイル。『命令』の項を参照してください。 この方法は、多数の命令を指定する必要がある場合に使用してください。

命令

次に、有効な mfsort 命令のリストを示します。

命令	説明
*	行の残りの部分はコメントとして扱われます。 この命令は、ファイルに各命令の目的を説明するコメントを追加できるため、テキストファイルで命令を指定する場合に便利です。
CHAR-EBCDIC	EBCDIC データ。 CHAR-EBCDIC は SORT 命令、MERGE 命令、USE 命令、GIVE 命令のすべてより先に指定する必要があります。
SIGN-EBCDIC	符号付き数字 DISPLAY 項目は EBCDIC 規則に従って解釈されます。CHAR-EBCDIC が指定されている場合には SIGN-EBCDIC は不要ですが、データを作成したプログラムが SIGN"EBCDIC" コンパイラ指令でコンパイルされている場合のように、ASCII 以外のデータには SIGN-EBCDIC を設定する必要があります。 SIGN-EBCDIC は、SORT 命令、MERGE 命令、USE 命令、GIVE 命令のすべてより先に指定する必要があります。
SORT-EBCDIC	レコード順 ASCII ファイルは EBCDIC 順にソートされます。
SORT/MERGE	これらの命令は、使用するフィールドを指定する FIELD 命令の前にソートまたはマージオプションを設定する必要があります。 FIELD 命令の次に、オプションで、作業ファイルのレコードサイズと形式を指定する RECORD 命令を設定する場合があります。 SORT 命令と MERGE 命令は相互排他的に機能します。
FIELDS (*instructions*)	ファイルがソートまたはマージされるフィールド。『*FIELDS 命令*』の項を参照してください。
RECORD *definition*	レコードサイズと形式。 RECORD 命令は、作業ファイル、入力ファイル、出力ファイルの詳細を指定するために使用します。『*RECORD 命令*』の項を参照してください。
USE *input-file*	各 USE 命令で、入力ファイルを 1 つ指定します。 すべての USE 命令は GIVE 命令の前に指定する必要があります。 『入力ファイルと出力ファイルの定義』の項を参照してください。
GIVE *output-file*	各 GIVE 命令で、出力ファイルを 1 つ指定します。『入力ファイルと出力ファイルの定義』の項を参照してください。

INCLUDE/OMIT	ソートプロセスからレコードをインクルードまたは省略する条件を指定します。詳細については、IBM のマニュアル『Using DFSORT Program Control Statements』を参照してください。 INCLUDE 命令と OMIT 命令は相互排他的に機能します。
INREC	SORT/MERGE プロセスを行う前にレコードを再フォーマットします。
OUTREC	SORT/MERGE プロセスを行った後にレコードを再フォーマットします。
MODS	SORT/MERGE プロセスにレコードがリリースされるか SORT/MERGE プロセスからレコードが返されるたびに実行される外部プロシージャ (ユーザ出口) を指定します。この実装は、E15 と E35 のユーザ出口をサポートします。
SUM	同じキー値を持つレコードを単一レコードとして返すことを指定します。オプションで、1 つのフィールドを指定して同じキーを持つすべてのレコードの合計値を累積できます。 SUM 操作は、次の制御フィールド型で実行できます。 BI バイナリ FI 固定小数点整数 FL 浮動小数点 PD パック 10 進数 ZD ゾーン 10 進数
OUTFIL	この命令は、1 つ以上の出力ファイルに対する複雑な編集およびレポート処理を指定する場合に使用します。 各出力ファイルは、GIVE コマンドを使用して指定する必要があります。出力ファイルを指定しない場合には、OUTFIL は IBM の マニュアル『Using DFSORT Program Control Statements』で説明されているように動作します。
OPTION	この命令は、さまざまなオプションを指定する場合に使用します。これらのオプションの 1 つに COPY があります。このオプションでは、レコードはソートされず、出力ファイルにコピーされます。

FIELDS 命令

SORT 命令や MERGE 命令の次には、入力ファイルのソートやマージを行うフィールドを指定する FIELDS 命令を続ける必要があります。

FIELDS 命令の形式を次に示します。

```
fields({start,length,type,order},...)
```

パラメータの内容は、次のとおりです。

パラメータ	説明
start	レコードのフィールドの開始位置。1 からバイトで数えます。
length	フィールド (バイト) の長さ。
type	フィールドのデータ型。『フィールドの型』の項を参照してください。
order	出力の順序。どちらかを選択できます。 A - 昇順 D - 降順

パラメータセット (start、length、type および order) を繰り返すことにより、最大 100 フィールドを指定できます。 パラメータとパラメータセットはコンマで区切る必要があります。

フィールドの型

次に、使用可能なフィールドの型の一部を示します。

フィールドの型	定義
AQ	代替文字の大小順序を持つ文字。
BI	COMP
C5	COMP-5
C6	COMP-6
CH	PIC X DISPLAY
CX	COMP-X
FL	浮動小数点数、符号付き。
FS/CSF	符号付き数字。オプションで先頭に浮動符号を使用できます。
LI/OL/CLO	PIC S9 LEADING INCLUDED
LS/CSL	PIC S9 LEADING SEPARATE
NU	PIC 9 DISPLAY
PD	PIC S9 COMP-3
PD0	パック 10 進数。最初の半バイトと符号半バイトは無視されます。
SB/FI	PIC S9 COMP
S5	S9 COMP-5
SS	サブ文字列。 条件でのみ使用されます。
TS/CST	PIC S9 TRAILING SEPARATE
TI/ZD/OT/CTO	PIC S9 TRAILING INCLUDED
Y2B	2 桁。1 バイトバイナリの年データ。
Y2C/Y2Z	2 桁。2 バイトの年データ。オプションで、後ろに符号を含めることができます。 PIC 99 または PIC S99。
Y2D	2 桁。2 バイトパック 10 進数の年データ。 PIC 99 COMP-6。
Y2P	2 桁。2 バイトパック 10 進数の年データ。 PIC 99 COMP-3。

Y2S	2 桁。2 バイト文字の年データ。特殊なインジケータを含みます。 バイナリのゼロ、空白およびバイナリ文字は、特殊なケースとして扱われます。
Y2T	完全な日付形式 yyx...。
Y2U	完全な日付形式 yyx...、COMP-3。
Y2V	完全な日付形式 yyx...、COMP-3。最初の半バイトは無視されます。
Y2W	完全な日付形式 x...yy。
Y2X	完全な日付形式 x...yy、COMP-3。
Y2Y	完全な日付形式 x...yy、COMP-3。最初の半バイトは無視されます。

定義されているその他のフィールドの型は、IBM のマニュアル『SORT Control Statement』で参照できます。

たとえば、golf.dat は、COBOL プログラムで次のように定義した相対ファイルです。

```
 file-control.
     select members-file
         assign to "d:¥netexpress¥base¥workarea¥golf.dat"
         organization is relative
         access mode is random
         relative key is relative-key.
 data division.
 file section.
 fd members-file
     record contains 28 characters.
 01 members-record.
     03 members-number pic 9(6).
     03 members-lname pic x(10).
     03 members-fname pic x(10).
     03 members-handicap pic 9(2).
```

次の mfsort コマンドを使用すると、会員番号を含むフィールドでファイル golf.dat を昇順にソートできます。

```
mfsort sort fields(1,6,nu,a)
   use golf.dat record f,28 org rl
   give members.dat
```

ソートされたファイルのバージョンは、ファイル members.dat に書き込まれます。

入力ファイルと出力ファイルの定義

入力ファイルと出力ファイルの両方を定義するには、次の命令を指定する必要があります。

ファイル	命令
入力	`USE` *`input-file`* `[record` *`definition`*`]` `[org` *`organization`*`]` `[key` *`structure`*`]`

出力

```
GIVE output-file [record
definition]
[org organization] [key structure]
```

注：

- 可変長ファイルまたは索引ファイルの入力ファイルの場合は、ファイルの特性がファイルそのものから推測されるため、RECORD、ORG または KEY コマンドを指定する必要はありません。
- 上記の命令を一部でも省略すると、前回に指定した値が使用されます。 そのため、入力ファイルと出力ファイルをすべて同じ型と形式にする場合は、最初のファイルの値を指定した後で指定し直す必要がありません。
- 最初に指定したファイルが可変長ファイルまたは索引ファイルの場合は、どの値も指定する必要はありません。
- 入出力ファイルを指定する場合には、ASSIGN 句で指定するときと同様に、埋め込みライブラリ名と環境変数を使用できます。詳細については、『ファイル名』の章の『パスのライブラリ名』と『ファイル名のマッピング』を参照してください。
- MFSORT で生成した出力ファイルを別のファイル名にリダイレクトできます。次のようになります。
  - SYSOUT 環境変数を使用しない場合は、SYSOUT という名前のファイルに書き込まれます。
  - SYSOUT 環境変数を指定する場合は、書き込むファイル名を指定します。

RECORD 命令

RECORD 命令を使用して、次のファイルのレコードの形式と長さを指定します。

- ソート作業ファイル
- この命令の次に関連する USE 命令または GIVE 命令が続く場合の入力ファイルと出力ファイル

RECORD 命令の形式は次のようになります。

```
RECORD format,rec-len,max-len
```

パラメータの内容は、次のとおりです。

パラメータ	説明
format	レコード形式。次のどちらかです。 F : 長さが *rec-len* の固定長レコード V : 最小長 *rec-len*、最大長 *max-len* の可変長レコード
rec-len	*format* が F に設定されている場合のレコード長。 *format* が V に設定されている場合の最小レコード長。
max-len	*format* が V に設定されている場合の最大レコード長。

ソート作業ファイルに RECORD 命令を指定しない場合には、形式は固定長レコード形式にデフォルト設定されます。その場合のレコードサイズは、USE または GIVE 命令で指定した最大レコードと等しくなります。

可変長ファイルまたは索引ファイルの入力ファイルの場合は、ファイルの特性がファイルそのものから推測されるため、RECORD 命令を指定する必要はありません。

レコードサイズが 256 より大きい場合は、RECORD 命令を使用する必要があります。

行順入力ファイルに max-len を指定しない場合は、mfsort では 256 を前提とします。これを超える場合には、出力ファイルは複数のレコードに分割されます。

ORG 命令

ORG 命令では、ファイル編成を指定します。ファイル編成は、次のどれかになります。

ORG 命令	ファイル編成
IX	索引
RL	相対
SQ	順 (デフォルト値)
LS	行順

可変長ファイルまたは索引ファイルの入力ファイルの場合は、ファイルの特性がファイルそのものから推測されるため、ORG 命令を指定する必要はありません。

KEY 命令

KEY 命令では、索引ファイルのキー構造を指定します。この命令は、出力ファイルが索引ファイルで、そのキー構造が索引入力ファイルと異なる場合に使用します。

KEY 命令の形式を次に示します。

```
KEY ({start,length,ixkey},...)
```

パラメータの内容は、次のとおりです。

パラメータ	説明
start	レコード内のキーの開始位置。1 からバイトで数えます。

length キーのバイト数。

ixkey 次のうちのどれかを指定します。

P	主キー (常に最初に定義します)
A	副キー
AD	重複のある副キー
C	最後に指定した主キーまたは副キーの構成要素

KEY 命令は、キー全体の構造を表すのに必要な回数のみ繰り返せます。パラメータとパラメータセット (start, length, ixkey) はコンマで区切る必要があります。

キーは、重要な順に定義します。最初の主キー、主キーが分割されている場合はすべての構成要素、最初の副キー、そのすべての構成要素という順序で定義する必要があります。

次の例では、3 つのキーを定義します。

```
KEY (4,5,p,10,5,c,20,2,ad,40,2,a,46,10,c)
```

ここでは、

4,5,p,10,5,c	分割された最初の主キーを表します。 主キーは、文字位置 4 から始まる長さが 5 バイトの最初の構成要素と、文字位置 10 から始まる長さが 5 バイトの 2 番目の構成要素で構成されます。
20,2,ad	重複が許可される 2 番目の (副) キーを表し、文字位置 20 から始まり、長さは 2 バイトです。
40,2,a,46,10,c	3 番目のキーを表します。このキーは文字位置 40 から始まる長さが 2 バイトの最初の構成要素と、文字位置 46 から始まる長さが 10 バイトの構成要素に分割される副キーです。

コマンド例

ここでは、mfsort コマンドの例とジョブストリームをいくつか紹介します。

その他の例については、IBM のマニュアル『Examples of DFSORT Job Streams』を参照してください。

複数のファイルを使用したソート

国内の東西南北の 4 地域について、国内大会で国内組織のメンバーが達成したスコアを記録した索引ファイル (north.dat、south.dat、east.dat および west.dat) がそれぞれ 1 つずつ、計 4 つあると仮定します。このうち north.dat を定義するには、次のような COBOL 構文を使用します。

```
 file-control.
     select idxfile assign to "north.dat"
         organization is indexed
         record key is member-id.
```

```
 data division.
 file section.
 fd idxfile
     record contains 39 characters.
 01 idxfile-record.
     03 member-id  pic 9(6).
     03 surname    pic x(15).
     03 first-name pic x(15).
     03 score      pic 9(3).
```

さらに、他のファイルも同様の方法で作成し、大会の結果をファイルに入力したと仮定します。 以降の例では、これらのファイルを使用します。

文字のソート **(**昇順**)**

次の mfsort コマンドでは、4 つのファイルそれぞれからすべてのレコードを取り出し、メンバーの姓で昇順にソートしてから、相対ファイル members.dat に結果を出力します。

```
mfsort sort fields(7,15,ch,a)
   use north.dat
   use south.dat
   use east.dat
   use west.dat
   give members.dat org rl
```

数字のソート **(**降順**)**

次の mfsort コマンドでは、4 つのファイルを取り出し、メンバーのスコア順 (スコアの高い順に並べる) でソートしてから、その結果を相対ファイル scores.dat に出力します。

```
mfsort sort fields(37,3,nu,d)
   use north.dat
   use south.dat
   use east.dat
   use west.dat
   give scores.dat org rl
```

レコードの省略

次の mfsort コマンドでは、4 つのファイルを取り出し、会員番号順 (主キー) でソートしてから、その結果を索引ファイル national.dat に出力します。 スコアフィールドが 20 未満のレコードはすべて省略されます。

```
mfsort sort fields(1,6,nu,a)
   use north.dat
   use south.dat
   use east.dat
   use west.dat
   give national.dat
```

```
   omit cond (37,3,nu,lt,20)
```

INCLUDE を使用した単一ファイルのソートとサブ文字列の比較

次の mfsort コマンドでは、行順ファイル sortin.dat を取り出し、そのレコードを長さが 4 バイトの位置 11 から始まる文字フィールドでソートします。結果はファイル sortout.dat に出力され、長さが 3 バイトで位置 15 から開始するサブ文字列が、文字列 'J69,L92,J82' の連続する 3 文字と同じレコードのみが含まれます。

```
mfsort sort fields=(11,4,ch,a)
   use sortin.dat org ls record (f 80)
   give sortout.dat
   include cond=(15,3,ss,eq,c'J69,L92,J82')
```

環境変数を使用した単一ファイルのソート

次の mfsort コマンドでは、テキストファイル unsorted.txt と ファイル a.1 を取り出し、昇順にソートしてから、ファイル sorted.txt に結果を出力します。

```
set $infile=unsorted.txt
set $outfile=sorted.txt

mfsort take a.1
```

ここで、unsorted. txt は、次の内容です。

```
BBBBBBBBBBBBBBBBBBB
CCCCCCCCCCCCCCCCCCC
AAAAAAAAAAAAAAAAAAA
```

a.1 は、次の内容です。

```
use $infile org LS
sort
give $outfile
```

OUTREC を使用したレコードの変換

次の mfsort コマンドでは、形式が cyymmdd のフィールドを含むレコードを、形式が yyymmdd のフィールドを含むレコードに変換します。

```
Sort C'cyymmdd'
     SORT FIELDS=(1,7,BI,A)         * sort C'cyymmdd'
     use mfs110a.in org ls record (f 40)
    * C'cyymmdd' を C'yyyymmdd' に変換
     OUTFIL OUTREC=(1,1,CHANGE=(2, * C'c' を次のように変換
                    C'0',C'19',  *   C'0' から C'19' へ
                    C'1',C'20',  *   C'1' から C'20' へ
                    C'2',C'21'), *   C'2' から C'21' へ
```

```
                    NOMATCH=(C'99')
                  2,6)           * C'yymmdd' をコピー

    give sortout.dat
```

複雑なレポート処理のための **OUTFIL** を使用したソート

次に示す例では、OUTFIL コマンドを使用して複雑なレポートを作成します。この例では、4 つの部門の 1 つに関する損益計算書レポートを作成します。入力ファイル mfs121a.dat は最初の 2 つのフィールドでソートされ、西地域からのレコードのみが出力されます。 この SECTIONS 命令では、位置が 3 から始まる長さが 10 バイトのフィールドが変更されるとページが作成されます。 次の例を示します。

- 入力データ
- mfsort コマンド
- 結果の出力

入力データ

```
Chips        San Martin     0088902203 West
Chips        Oakland        0023412432 West
Chips        San Jose       0123213335 West
Ice Cream    Marin          0054234123 West
Chips        Gilroy         0055484342 West
Ice Cream    Napa           0085734283 West
Pretzels     San Jose       0123488534 West
Ice Cream    San Francisco  0092231245 West
Chips        San Francisco  000324343q West
Chips        San Jose       0123213335 South
Ice Cream    San Martin     0100346730 West
Pretzels     Marin          0534332344 West
Chips        Gilroy         0055484342 South
Chips        Morgan Hill    0098732232 West
Pretzels     Morgan Hill    0084384340 West
Ice Cream    San Jose       000002345u West
Pretzels     Napa           0531234856 West
Chips        Oakland        0023412432 South
Pretzels     San Martin     000023438r West
Chips        Los Angeles    000223401t West
Ice Cream    Marin          0054234123 South
Pretzels     San Francisco  0541230005 West
Ice Cream    Napa           0085734283 South
Pretzels     San Jose       0123488534 South
Ice Cream    San Francisco  0092231245 South
```

```
Chips         San Francisco  000324343q South
Ice Cream     San Martin     0100346730 South
Pretzels      Marin          0534332344 South
Chips         Morgan Hill    0098732232 South
Pretzels      Morgan Hill    0084384340 South
Ice Cream     San Jose       000002345u South
Pretzels      Napa           0531234856 South
Pretzels      San Martin     000023438r South
Chips         Los Angeles    0002234014 South
Pretzels      San Francisco  0541230005 South
```

mfsort コマンド

```
SORT FIELDS=(3,10,A,16,13,A),FORMAT=CH

  use mfs121a.dat  org ls record (f 80)
  OUTFIL
   INCLUDE=(42,6,CH,EQ,C'West'),
   HEADER1=(5/,18:'    Western Region',3/,
               18:'Profit and Loss Report',3/,
               18:'     for  ',&DATE,3/,
               18:'      Page',&PAGE),
   OUTREC=(6:16,13,24:31,10,ZD,M5,LENGTH=20,75:X),
   SECTIONS=(3,10,SKIP=P,
    HEADER3=(2:'Division:  ',3,10,5X,'Page:',&PAGE,2/,
             6:'Branch Office',24:'       Profit/(Loss)',/,
             6:'-------------',24:'--------------------'),
    TRAILER3=(6:'=============',24:'====================',/,
             6:'Total',24:TOTAL=(31,10,ZD,M5,LENGTH=20),/,
             6:'Lowest',24:MIN=(31,10,ZD,M5,LENGTH=20),/,
             6:'Highest',24:MAX=(31,10,ZD,M5,LENGTH=20),/,
             6:'Average',24:AVG=(31,10,ZD,M5,LENGTH=20),/,
             3/,2:'Average for all Branch Offices so far:',
                X,SUBAVG=(31,10,ZD,M5))),
    TRAILER1=(8:'Page ',&PAGE,5X,'Date:  ',&DATE,5/,
            8:'Total Number of Branch Offices Reporting:  ',
              COUNT,2/,
            8:'Summary of Profit/(Loss) for all',
              ' Western Division Branch Offices',2/,
            12:'Total:',
                22:TOTAL=(31,10,ZD,M5,LENGTH=20),/,
            12:'Lowest:',
                22:MIN=(31,10,ZD,M5,LENGTH=20),/,
            12:'Highest:',
                22:MAX=(31,10,ZD,M5,LENGTH=20),/,
            12:'Average:',
                22:AVG=(31,10,ZD,M5,LENGTH=20))
```

```
give outfil1.dat
```

結果の出力

```
                    Western Region

                  Profit and Loss Report

                      for  11/05/95

                         Page     1
*****************************************************
  Division:  Chips          Page:     2

     Branch Office            Profit/(Loss)
     -------------     --------------------
     Gilroy                     554,843.42
     Los Angeles                (22,340.14)
     Morgan Hill                987,322.32
     Oakland                    234,124.32
     San Francisco              (32,434.31)
     San Jose                 1,232,133.35
     San Martin                 889,022.03
     =============     ====================
     Total                    3,842,670.99
     Lowest                     (32,434.31)
     Highest                  1,232,133.35
     Average                    548,952.99



 Average for all Branch Offices so far:     548,952.99
*****************************************************
  Division:  Ice Cream      Page:     3

     Branch Office            Profit/(Loss)
     -------------     --------------------
     Marin                      542,341.23
```

Napa	857,342.83
San Francisco	922,312.45
San Jose	(234.55)
San Martin	1,003,467.30
Total	3,325,229.26
Lowest	(234.55)
Highest	1,003,467.30
Average	665,045.85

Average for all Branch Offices so far: 597,325.02

****

Division: Pretzels Page: 4

Branch Office	Profit/(Loss)
Marin	5,343,323.44
Morgan Hill	843,843.40
Napa	5,312,348.56
San Francisco	5,412,300.05
San Jose	1,234,885.34
San Martin	(2,343.82)
Total	18,144,356.97
Lowest	(2,343.82)
Highest	5,412,300.05
Average	3,024,059.49

Average for all Branch Offices so far: 1,406,236.51

****

Page 5 Date: 11/05/95

Total Number of Branch Offices Reporting: 18

Summary of Profit/(Loss) for all Western Division Branch Offices

Total:	25,312,257.22
Lowest:	(32,434.31)

```
Highest:        5,412,300.05
Average:        1,406,236.51
```

作業ファイル

ソート操作中やマージ操作中に、mfsort では一時作業ファイルを使用します。 この作業ファイルは現在のディレクトリのディスクにページングされるか、または、TMP 環境変数で指定されたディレクトリがある場合はそのディレクトリにページングされます。

mfsort は、各入力ファイルからすべてのレコードを一時作業ファイルにコピーするときに、適宜、レコードの切り捨てや埋め込みを行います。 次に、キー記述に従って、作業ファイルでソートやマージを行います。 作業ファイルでのソートやマージが済むと、レコードは各出力ファイルにコピーされ、適宜、切り捨てられるか、または埋め込まれます。

この操作では、次の点に注意してください。

- レコードを切り捨てたくない場合には、作業ファイルのレコード長を、ソートする最長レコードに合わせて十分確保するようにしてください。
- 入力ファイルが可変長であり、ソート作業ファイルが可変長でない場合、ソート作業ファイルと出力ファイルのレコードは可変長でなくなります。
- 入力ファイルが固定長であり、作業ファイルが可変長である場合、作業ファイルのレコードのレコード長は、入力ファイルの固定長または作業ファイルの最大レコード長のどちらか小さい方になります。

エラーメッセージ

mfsort エラーメッセージの詳細なリストについては、ヘルプトピック『mfsort のエラーメッセージ』を参照してください。

mfsort ユーティリティ　　データとキーの圧縮

第 9 章 : Btrieve

Btrieve は Pervasive Software 社が提供するファイル処理システムです。Btrieve の詳細については、Pervasive Software 社が提供する Btrieve マニュアルを参照してください。

Btrieve ファイルを処理するために、Micro Focus ファイルハンドラ API を使用する Xfh2btr 呼び出し変換モジュールが Net Express で提供されています。

注：

- Btrieve ランタイムシステムは、Net Express では提供されていません。 Btrieve ランタイムシステムのコピー、または Btrieve 開発者キットをすでに持っている必要があります。
- Pervasive Software 社とのライセンス契約を持っていない場合は、サードパーティに Btrieve ランタイムを配布できません。 詳細は Pervasive Software 社にお問い合わせください。

Xfh2btr 呼び出し変換モジュール

Xfh2btr 呼び出し変換モジュールを使用すると、この COBOL システムから Btrieve ファイルにアクセスするための Micro Focus ファイルハンドラ API を使用できます。

Btrieve は通常、ANSI 規格に準拠しません。ただし、デフォルトでは、Xfh2btr 呼び出し変換モジュールは、ANSI 動作をエミュレートさせるために Btrieve ランタイムシステムに必要な呼び出しを行います。 非 ANSI モードで操作することもできます。

Xfh2btr は Btrieve 6.1x 以降と互換性があります。

Xfh2btr は COBOL システムの一部として提供されます。これは、リンク可能なモジュール xfh2btr.obj としても提供され、Btrieve ファイル上で入出力を行うどのアプリケーションともリンクできます。

呼び出し変換モジュールで入出力を行うために Btrieve ランタイムシステムを呼び出す必要がある場合は、モジュール _BTRV が呼出されます。このモジュールの機能は、呼び出しをフォーマットし、Btrieve マイクロカーネルインターフェイス (Windows NT および Windows 95 用の wbtrv32.dll) に呼び出しを渡すことです。モジュール _BTRV は、Btrieve API 呼び出しで使用されるのと同じパラメータを使って COBOL プログラムから直接呼び出せます (詳細は『Btrieve プログラマーズリファレンスマニュアル』を参照)。

スタンドアロンの実行ファイルにアプリケーションをリンクする場合は、_btrv.obj にリンクする必要があります。または、Btrieve 開発者キットを持っている場合は、Pervasive Software 社から提供されるモジュールを _btrv.obj のかわりにリンクできます。これは、cobrbtrv.obj と cobpbtrv.obj です。 これらのモジュールの詳細については、Btrieve 開発者キットで提供されるマニュアルを参照してください。

Xfh2btr の呼び出し

Xfh2btr 呼び出し変換モジュールを呼び出すには、次の 3 つの方法があります。

- CALLFH コンパイラ指令を使用する。
- FILETYPE コンパイラ指令を使用する。
- 外部ファイル名割り当てを使用する。

Btrieve では、サポートしないファイル操作とファイル形式がいくつかあるので注意が必要です。たとえば、WRITE AFTER や行順ファイルなどはサポートされていません。 この場合に、変換モジュールにファイル入出力を直接行おうとすると、プログラムにエラーが返されます。

CALLFH コンパイラ指令

CALLFH コンパイラ指令は、プログラムによって使用されるファイルハンドラにすべての COBOL 入出力操作を実行することを指定します。 デフォルトでは、ファイルハンドラを指定しない場合は Micro Focus ファイルハンドラが使用されます。

CALLFH コンパイラ指令を Xfh2btr 呼び出し変換モジュールを指定するために使用する場合は、プログラムはすべての入出力を Xfh2btr 呼び出し変換モジュールに渡して、COBOL 入出力操作を Btrieve 入出力操作に変換します。

次のようにして、原始プログラムの先頭に最初の $SET 文として CALLFH を置きます。

```
$set callfh"xfh2btr"
```

FILETYPE コンパイラ指令

FILETYPE コンパイラ指令は、プログラムで作成されたファイルの形式を指定します。

次のようにして、プログラムの先頭に最初の $SET文 として FILETYPE コンパイラ指令を置きます。

```
$set filetype"n"
```

このときの、n は 5 または 6 です。

FILETYPE"5" は、ファイル操作で ANSI 規格に準拠することが要求される場合に使用します。Btrieve は通常 ANSI 規格に準拠しないため、この操作モードでは Btrieve ランタイムシステムに ANSI 動作をエミュレートするように何度も呼び出しを要求します。

FILETYPE"6" は、速さが要求される場合に使用します。 このモードでは、各 Micro Focus ファイルハンドラ操作が最も近い Btrieve ランタイム呼び出しにマップされます。 ANSI 規格への準拠は行われません。

異なった形式のファイルをプログラム内に混在させることができます。次の例のように、FILETYPE コンパイラ指令を個々の SELECT 文に置きます。

```
$set filetype"0"
select testfile-1 assign to "test-1.dat"
   organization indexed
   record key prime-key
   access sequential.
$set filetype"5"
select testfile-2 assign to "test-2.dat"
   organization relative
   access sequential.
```

この例では、test-1.dat 上のすべての入出力は Micro Focus ファイルハンドラによって処理され、test-2.dat 上のすべての入出力は Xfh2btr 呼び出し変換モジュールによって処理されます。

注：索引ファイルを作成するときに使用する形式を指定するには、ファイルハンドラ構成ファイルにあるファイル設定の下に IDXFORMAT パラメータを含めます。 ANSI 準拠または非 ANSI 準拠で Btrieve を示すには、IDXFORMAT を 5 または 6 のどちらかに設定します。詳細については、『ファイルハンドラの構成』の章を参照してください。

Btrieve 環境変数

Xfh2btr 呼び出し変換モジュールの操作を切り替えるために使用する、2 つの環境変数 BTRPAGE と BTRMAXREC があります。

BTRPAGE 環境変数

BTRPAGE 環境変数には、Btrieve ファイルが作成されるときに使用されるページサイズを指定します。 次のコマンドを使用して設定できます。

```
set BTRPAGE=nnn
```

ここで nnn は、512～4096 の範囲にある 512 バイトの倍数とします。 レコードサイズより大きい値にする必要があります。

BTRPAGE を設定しない場合、または不正な値を記述した場合には、ページサイズのデフォルトは 1024 バイトになります。

注：

- ページサイズは、ファイルに定義できるキー数や、レコードに置くキーの場所に影響を与えます。
- Btrieve Record Manager は、正しいページサイズでロードする必要があります。
- BTRPAGE の値は、xfh2btr 構成ファイルを使用してファイルごとに設定できます。

BTRMAXREC 環境変数

BTRMAXREC 環境変数には、最大レコードサイズを指定します。

BTRMAXREC は、次のコマンドを使用して設定できます。

```
set BTRMAXREC=nnnn
```

ここで nnnn は最大レコード長を指定するバイト値です。

BTRMAXREC には、アクセスされる最大レコードのレコードサイズを設定します。

BTRMAXREC を設定しない場合は、デフォルト値の 32 KB が使用されます。

BTRMAXREC の値は、xfh2btr 構成ファイルを使用してファイルごとに設定できます。

Xfh2btr 構成ファイル

注：Xfh2btr 構成ファイルを使用する場合は、mfini.obj を使ってプログラムにリンクする必要があります。

Xfh2btr 構成ファイルを使用すると、すべてのファイルまたは個々のファイルに対して、ページサイズ、最大レコード長、Btrieve オープンモードを指定できます。

構成ファイルの名前は、XFH2BTR 環境変数で指定できます。この環境変数が設定されていない場合は、デフォルトの構成ファイル名 xfh2btr.cfg が使用されます。 構成ファイルはまず現在のディレクトリで検索され、次に COBDIR 環境変数で指定されたパスに沿って検索されます。

タグ [X2B-DEFAULTS] は、ページサイズ、最大レコード長、Btrieve オープンモードのデフォルト値を指定するために使用されます。一方、ファイル名 (たとえば [test.dat]) を含むタグは、特定のファイルに値を指定するために使用されます。

- ページサイズ
- 最大レコード長
- Btrieve オープンモード

属性名は空白またはコロン (:) によって、属性値と分けます。

注：構成ファイルがある場合は、Btrieve 環境変数 BTRPAGE および BTRMAXREC は無効です。

また、Xfh2btr トレースオプションをオンにして、Xfh2btr 構成ファイルでトレースファイルの名前を指定できます。

ページサイズ

Btrieve ファイルの作成時に使用するページサイズは、Xfh2btr 構成ファイルで次のように属性を設定することで指定できます。

`BTRPAGE:nnn`

ここで nnn は、512～4096 の範囲にある 512 バイトの倍数とします。

BTRPAGE を設定しない場合、または不正な値を記述した場合には、ページサイズのデフォルトは 1024 バイトになります。

最大レコードサイズ

Btrieve ファイルの最大レコードサイズは、Xfh2btr 構成ファイルで次のように属性を設定することで指定できます。

`BTRMAXREC:nnnn`

ここで nnnn はバイト値です。 BTRMAXREC には、アクセスされる最大レコードのレコードサイズを設定します。 BTRMAXREC を設定しない場合は、デフォルト値 32 KB が使用されます。

Btrieve ファイルオープンモード

Xfh2btr 構成ファイルは、ファイルを開くときに使用する Btrieve オープンモードを指定するために使用できます。

```
INPUT-EXCLUSIVE:オープンモード
INPUT-SHAREABLE:オープンモード
OUTPUT:オープンモード
EXTEND-EXCLUSIVE:オープンモード
EXTEND-SHAREABLE:オープンモード
I-O-EXCLUSIVE:オープンモード
I-O-SHAREABLE:オープンモード
```

オープンモードの値として指定できる値は、normal (通常)、accelerated (加速)、read-only (読み取り専用)、verify (確認)、exclusive (排他)です。 これらのオープンモードの詳細については、Btrieve のマニュアルを参照してください。

Xfh2btr 構成ファイルの例

```
[X2B-DEFAULTS]
output:accelerated
i-o-shareable:accelerated
i-o-exclusive:accelerated
btrpage:2048
btrmaxrec:2000
[test1.dat]
```

```
output:exclusive
i-o-shareable:normal
btrpage:4096
```

上記の Xfh2btr 構成ファイル例では、次の内容が指定されています。

1. OUTPUT で開かれたファイルに対するデフォルトのオープンモードは accelerated です。
2. 共有 I/O で開かれたファイルに対するデフォルトのオープンモードは accelerated です。
3. 排他 I/O で開かれたファイルに対するデフォルトのオープンモードは accelerated です。
4. ファイルの作成時に使用するデフォルトのページサイズは 2048 バイトです。
5. デフォルトの最大レコード長は 2000 バイトです。
6. OPEN OUTPUT で開かれた test1.dat に対するオープンモードは exclusive です。
7. 共有 OPEN I/O で開かれた test1.dat に対するオープンモードは normal です。
8. test1.dat 作成時に使用されるページサイズは 4096 バイトです。

トレースオプション

Xfh2btr トレースオプションを設定すると、Xfh2btr 呼び出し変換モジュールによって実行される各操作および Btrieve Record Manager によって返される各エラーコードを、画面に表示するかまたはトレースファイルに記録できます。

Btrieve Record Manager からエラーが返された場合は、Xfh2btr 呼び出し変換モジュールは Btrieve エラーを、プログラムに返す COBOL ファイル状態コードにマップします。Btrieve エラーがマップされた COBOL ファイル状態コードも、トレースファイルに記録されます。

トレースで Btrieve エラーが示されないのに COBOL ファイル状態コードが返された場合には、このエラーは Xfh2btr 呼び出し変換モジュール内で生成されたものであることを意味します。 たとえば、入力用として開かれたファイルに書き込もうとすると、このようなエラーが生成されます。

トレースオプションに影響を与える属性として、TRACE-FILE と TRACE があります。

TRACE-FILE:ファイル名

これは、トレース情報を画面ではなくファイルに出力するように指定します。ファイル名が指定されない場合は、デフォルトのファイル名 xfh2btr.lst が使用されます。トレースファイルの属性は、[X2B-DEFAULTS] タグの下のみで指定できます。 複数のクライアントアプリケーションが同じトレースファイルにトレース情報を書き込んでいる場合には、ファイルにこの情報が記録される順序は特定できません。

注：トレースファイル名を指定しても、TRACE 機能はオンになりません。

TRACE

TRACE 属性はトレースをオンにします。これは [X2B-DEFAULTS] タグの下に指定するか (この場合すべてのファイル上の操作がトレースされます)、またはファイル名のタグの下に指定します (この場合そのファイル上の操作のみがトレースされます)。

Btrieve トレースオプションの設定例

Xfh2btr 構成ファイルで次のように指定すると、

```
[X2B-DEFAULTS]
trace-file:x2btrace.dat
trace:
```

すべてのファイルでトレースが行われ、トレース情報が x2btrace.dat ファイルに書き込まれます。

Xfh2btr 構成ファイルで次のように指定すると、

```
[X2B-DEFAULTS]
trace-file:
[test1.dat]
trace:
```

test1.dat ファイルでトレースが行われ、トレース情報が xfh2btr.lst (デフォルトのトレースファイル名) ファイルに書き込まれます。

トレース出力の例

次の例は、トレースオプションを使用するときに生成される出力例です。

```
                                                行
Input FA01 ifile1.tmp                            1
Output Btrieve Error=+0000 COBOL Error=0/0       2
Input FAF3 ifile1.tmp                            3
Output Btrieve Error=+0000 COBOL Error=0/0       4
Input FAF3 ifile1.tmp                            5
Output Btrieve Error=+0000 COBOL Error=0/0       6
Input FADC All Files                             7
Output Btrieve Error=+0000 COBOL Error=0/0       8
Input FA80 ifile1.tmp                            9
Output Btrieve Error=+0000 COBOL Error=0/0      10
Input FA02 ifile1.tmp                           11
Output Btrieve Error=+0000 COBOL Error=0/0      12
Input FAF5 ifile1.tmp                           13
Output Btrieve Error=+0000 COBOL Error=0/0      14
Input FAF5 ifile1.tmp                           15
Output Btrieve Error=+0000 COBOL Error=0/0      16
```

```
Input FAF5 ifile1.tmp                              17
Output Btrieve Error=+0009 COBOL Error=1/0         18
Input FA80 ifile1.tmp                              19
Output Btrieve Error=+0000 COBOL Error=0/0         20
Input FA01 test.dat                                21
Output Btrieve Error=+0029 COBOL Error=9/078       22
Input FA00 ifile1.tmp                              23
Output Btrieve Error=+0000 COBOL Error=0/0         24
Input FAF3 ifile1.tmp                              25
Output Btrieve Error=+0000 COBOL Error=4/8         26
Input FA80 ifile1.tmp                              27
Output Btrieve Error=+0000 COBOL Error=0/0         28
```

1 行目は、操作を実行する操作コードやファイル名などを入力する行の例です。

2 行目は、Btrieve エラー状態や COBOL アプリケーションに返されるエラーなどを出力する行の例です。

7 行目は、開いているすべてのファイルで実行される操作 (この場合は COMMIT) の例を示しています。

22 行目は、Btrieve ランタイムシステムから返されるエラー状態 29 が状態 9/078 として COBOL アプリケーションに返されることを示しています。 Btrieve エラー状態 29 の詳細については、関連する Btrieve マニュアルを参照してください。

26 行目は、Xfh2btr モジュール内で生成されるエラーを示します。 この場合は、INPUT で開かれたファイルに書き込まれたことを示しています。

Micro Focus ファイルハンドラと Xfh2btr の相違

Micro Focus ファイルハンドラによって行われる COBOL ファイル操作の大部分は、Btrieve ファイル上で Xfh2btr によって行われるのと同じ方法で実行されます。 ただし 2 つのシステムには相違があるため、動作が異なる場合があります。 動作の異なるものを次に示します。

- キー
- ロックレコードの検出
- OPEN OUTPUT 操作
- レコード長
- 現行レコードポインタ (CRP)
- トランザクション処理
- WRITELOCK

注：ファイル内でレコードの書き込み、再書き込み、削除をした後で順読み操作を行う場合は、現行レコードポインタ (CRP) の位置付けに特に注意してください。

キー

キーについては、次のような相違があります。

- Btrieve v5.x 以前を使用する場合には、ファイルごとのキーコンポーネントの総数 (通常のキーは 1 つのコンポーネントを持ち、分割キーは複数のコンポーネントを持ちます) は 24 を超えられません。このため、24 コンポーネントに 1 キー、1 コンポーネントに 24 キー、またはこれらの組合せで行います。

- Btrieve v6.x 以降を使用する場合には、キーコンポーネントの最大数はファイルのページサイズによって変わります。 ファイルのページサイズは Btrieve 環境変数 BTRPAGE または Xfh2btr 構成ファイルで指定できます。 4096 バイトのページサイズを持つファイルでは、キーコンポーネントの最大数は 119 です。詳細については、Btrieve のマニュアルを参照してください。

- Xfh2btr モジュールは、ファイル内のキーコンポーネントの数を確認しません。使用する Btrieve のバージョンが必要なキーコンポーネントの数をサポートしていない場合は、Btrieve ランタイムシステムからエラーが返され、アプリケーションプログラムに報告されます。

- Btrieve ファイル内のキー長の合計は、255 バイトを超えることができません。

- Btrieve ファイル内のすべてのキーは、可変長ファイルの固定部分内に完全に含まれる必要があります。

- Btrieve を使用する場合は、一部のキーで START 操作を実行できません。全体のキーを使用する必要があります。

ロックレコードの検出

この COBOL システムが他のユーザによってロックされているレコードを読み取ると、レコードデータが COBOL ファイル状態 9/068 (「レコードロック」) と一緒にプログラムに返されます。

Btrieve では、レコードがロックされている場合はデータが返されません。このため、ロックされたレコードの読み取りは Btrieve ランタイムシステムへの複数の呼び出しを必要とし、デフォルトレコードの読み取りは遅くなります。

読み取り操作の速度を向上させるには、次のような方法があります。

- 排他ロックでファイルを開く。

  これを行うと、レコードロック状態を検出するための Btrieve ランタイムシステムへの呼び出しは不要になります。これはファイルが排他的にロックされるので、個々のレコードを確認する必要がないためです。

- NODETECTLOCK コンパイラ指令を使用する。

  この指令を設定すると、レコードが他のユーザによってロックされているかを確認するための Btrieve ランタイムシステムへの呼び出しが行われません。

- トランザクションからファイルにアクセスする。

トランザクションから Btrieve ファイルにアクセスすると、ファイルに一時的に排他ロックがかかるため、レコードロックを検出するための Btrieve ランタイムシステムへの呼び出しが不要になります。

OPEN OUTPUT 操作

COBOL OPEN OUTPUT 操作は、ファイルを作成し、そのファイルを排他ロックで開きます。 これは、単一のオペレーティングシステム呼び出しで行われます。 Btrieve では、この操作は Btrieve ランタイムシステムへの 2 回の呼び出しを必要とします。最初の呼び出しでファイルを作成し、2 回目の呼び出しでファイルを排他ロックで開きます。 これらの 2 回の呼び出しの間には多少の時間差があるため、2 番目のユーザはファイルが作成され開かれるまでの間にファイルにアクセスできてしまいます。これが起こると、2 番目のユーザは空のファイルを見つけてファイルを読み取ろうとし、この結果「ファイル終了」エラーになります。OPEN OUTPUT を行うユーザは、排他 OPEN 呼び出しが失敗するため、「ファイルロック」エラーを受け取ります。

レコード長

Btrieve ファイルでは、可変長レコードは 2 つの部分、固定長部分と可変長部分から構成されます。 固定長レコードは、固定長部分のみから構成されます。

可変長レコードでは、レコードの固定長部分は通常、COBOL プログラムで定義した最小レコード長によって決まります。 ただし、重複を許可するキーの数と使用されるページサイズによって、Btrieve はレコードの固定部分の最大レコード長に制限を課します。 レコードの固定部分の最大長の決定方法は次の通りです。

p - 6 - (8 * k) [- 4 可変長レコードの場合] [- 4 相対ファイルの場合]

パラメータの内容は、次のとおりです。

p　ファイルのページサイズ

k　重複を許可するキーの数

注：

- Btrieve レコードの固定部分で許された最大サイズを超えるレコード長で固定長レコードの Btrieve ファイルを作成しようとすると、ファイルは可変長 Btrieve ファイルとして作成され、その固定部分には最大サイズが設定されます。
- 固定部分で許された最大サイズを超える最小のレコード長で可変長レコードの Btrieve ファイルを作成しようとすると、固定部分には最大サイズが設定されたファイルが作成されます。
- Btrieve では、すべてのキーが Btrieve レコードの固定部分内で定義されている必要があります。
- 相対ファイルは、各レコードの前に 4 バイトの自動増分キーを付けた Btrieve キーファイルとして作成されます。 このキーの処理はユーザプログラムには見えません。 ただし、実際の Btrieve ファイルは、上記の方法で計算したよりも 4 バイト

大きい固定レコード長を持っています。

Btrieve レコード長の例

次の例では、1024 バイトのページサイズを使用します。

- 1018 バイトのレコード長を持つ固定長レコード索引ファイルは、1018 バイトの固定レコード長を持つ固定長レコード Btrieve ファイルを作成します。

  すべてのキーは、レコードの最初の 1018 バイトにある必要があります。

- 1019 バイトのレコード長を持つ固定長レコード索引ファイルは、1014 バイトの固定レコード長を持つ可変長レコード Btrieve ファイルを作成します。

  すべてのキーは、レコードの最初の 1014 バイトにある必要があります。

- 1010 バイトのレコード長と、複製を許可するキーを 1 つ持つ固定長レコード索引ファイルは、1010 バイトの固定レコード長を持つ固定長レコード Btrieve ファイルを作成します。

  すべてのキーは、レコードの最初の 1010 バイトにある必要があります。

- 1011 バイトのレコード長と、重複を許可するキーを 1 つ持つ固定長レコード索引ファイルは、1006 バイトの固定レコード長を持つ可変長レコード Btrieve ファイルを作成します。

  すべてのキーは、レコードの最初の 1006 バイトにある必要があります。

- 1014 バイトのレコード長を持つ固定長レコード相対ファイルは、1018 バイトの固定レコード長を持つ固定長レコード Btrieve ファイルを作成します。

  4 バイト分が増加しているのは、各レコードの先頭に自動増分キーが自動的に追加されるためです。 Micro Focus ファイルハンドラ呼び出しインタフェース用に操作コード 06 の動作コードを使用している場合には、返されるレコード長は 1014 バイトです。

- 1015 バイトのレコード長を持つ固定長相対ファイルは、1014 バイトの固定レコード長を持つ可変長 Btrieve ファイルを作成します。 これは、最大 1014 バイトに制限されるためです。

  この長さには、各レコードの先頭に自動的に追加された自動増分キーの 4 バイトも含まれます。 Micro Focus ファイルハンドラ呼び出しインタフェース用に操作コード 06 の動作コードを使用している場合には、返されるレコード長は 1010 バイトです。

- 1014 バイトの最小レコード長を持つ可変長レコード索引ファイルは、1014 バイトの固定レコード長を持つ可変長レコード Btrieve ファイルを作成します。

  すべてのキーは、レコードの最初の 1014 バイトにある必要があります。

- 1015 バイトの最小レコード長を持つ可変長レコード索引ファイルは、1014 バイ

トの固定レコード長を持つ可変長レコード Btrieve ファイルを作成します。

すべてのキーは、レコードの最初の 1014 バイトにある必要があります。

- 1006 バイトの最小レコード長と、重複を許可するキーを 1 つ持つ可変長レコード索引ファイルは、1006 バイトの固定レコード長を持つ可変長レコード Btrieve ファイルを作成します。

  すべてのキーは、レコードの最初の 1006 バイトにある必要があります。

- 1007 バイトの最小レコード長と、重複を許可するキーを 1 つ持つ可変長レコード索引ファイルは、1006 バイトの固定レコード長を持つ可変長レコード Btrieve ファイルを作成します。

  すべてのキーは、レコードの最初の 1006 バイトにある必要があります。

- 1010 バイトの最小レコード長を持つ可変長レコード相対ファイルは、1014 バイトの固定レコード長を持つ可変長レコード Btrieve ファイルを作成します。

  4 バイト分が増加しているのは、各レコードの先頭に自動増分キーが自動的に追加されるためです。Micro Focus ファイルハンドラ呼び出しインタフェース用に操作コード 06 の動作コードを使用している場合には、返されるレコード長は 1010 バイトです。

- 1011 バイトの最小レコード長を持つ可変長レコード相対ファイルは、1014 バイトの固定レコード長を持つ可変長レコード Btrieve ファイルを作成します。

  4 バイト分が増加しているのは、各レコードの先頭に自動増分キーが自動的に追加されるためです。Micro Focus ファイルハンドラ呼び出しインタフェース用に操作コード 06 の動作コードを使用している場合には、返されるレコード長は 1010 バイトです。

現行レコードポインタ (CRP)

この COBOL システムでは、現行レコードポインタ (CRP) は、ファイルで START、READ (順次またはランダム)、OPEN 操作が行われる場合のみに影響を受けます。

ただし、Btrieve ランタイムシステムでは、CRP は WRITE、REWRITE、DELETE 操作によって影響を受けます。 つまり、これらの操作に続いて Btrieve ファイル内の CRP を元の位置に戻し、順次 READ 文が影響を受けないようにする必要があります。

共有環境では、CRP の再位置付けは、再位置付けを行うレコードが他のユーザによって削除されている場合には失敗します。 このような場合は、順次 READ を実行しようとするとエラーが返されます。

以後に説明する CRP の再位置付けは、次の順次 READ 文のときではなく操作が完了した後で行われます。 これによって、レコードが削除される可能性は低くなり、CRP の再位置付けが成功する可能性は高くなります。

注：非 ANSI 規格モードの操作では、CRP を再位置付けしないために操作が非常に速くなります。

WRITE 操作後の順次 **READ** 操作

I/O で開いた共有ファイルが ANSI 準拠の操作モードが使用されているトランザクションにない場合、ファイルは実際に 2 回開かれます。 最初にファイルが開かれるのは、ファイルからレコードを読み取るときです。 2 回目にファイルが開かれるのは、ファイルにレコードを書き込むときです。 このため、読み取り位置は通常の WRITE 操作で影響を受けることがありません。 ただし、WRITELOCK 指令を使用した場合には、挿入したレコードはロックされて読み戻されなければならず、これによりファイル内の CRP が変更されます。

WRITE 操作後の Btrieve ファイル位置指示子の再位置付けは、WRITELOCK 指令が使用されファイルが共有の場合のみに失敗します。 WRITE 操作後に再位置付けが失敗した場合は、順次 READ が行われるときにエラーが返されます。

DELETE 操作後の順次 **READ** 操作

乱呼び出しまたは動的呼び出しの I/O で開いたファイルでは、Btrieve ファイル位置指示子を、削除するレコードの位置に移動されます。 DELETE 操作を行った後、ファイル位置指示子は元の位置に戻されますが、再位置付けするレコードが削除されている場合はこれが失敗することもあります。

この問題は順呼び出しのファイルには発生しません。ファイル位置指示子はすでに削除するレコード上にあり、再び位置付ける必要がないためです。

注：参照キーが重複を許可するキーにある場合は、順次 READ と DELETE 文を続けて行うことはできません。 これは、DELETE 操作の後で CRP を再び位置付けることができないためです。

REWRITE 操作後の順次 **READ** 操作

乱呼び出しまたは動的呼び出しの I/O で開いたファイルでは、Btrieve ファイル位置指示子は、変更するレコードの位置に移動されます。 REWRITE 操作を行った後、ファイル位置指示子は元の位置に戻されますが、再位置付けするレコードが削除されている場合はこれが失敗することもあります。

この問題は順呼び出しのファイルには発生しません。ファイル位置指示子はすでに再書き込みするレコード上にあり、再び位置付ける必要がないためです。

トランザクション処理

この COBOL システムでは、トランザクション処理は Fileshare を使用している場合のみに実行できますが、Btrieve ユーザは、トランザクション処理を実行するために Fileshare は必要ありません。

Fileshare を使ってトランザクション内のファイルにあるレコードを更新する場合は、ファイルにある個々のレコードはロックされ、他のユーザがレコードを読むことを防ぎます。 これはレコードがファイルから削除されたときに ROLLBACK 操作が行われる可能性があるためです。 ただし、トランザクション内から Xfh2btr を介して Btrieve ファイルにアクセスする場合は、他のユーザが同時にファイルにアクセスすることを防ぐために、ファイル全体

に一時的な排他ロックがかけられます。 Btrieve v6.x 以降では、並行トランザクションが使用され、トランザクション内でファイルを共有できます。

Btrieve トランザクションは、処理中のトランザクションがなく ROLLBACK を使って開かれているファイルに対して、REWRITE 、WRITE、または DELETE 操作が行われるたびに開始されます。 この時点では、Btrieve はファイルに排他ロックをかけます。このファイルロックは並行トランザクションが使用中の場合を除いて、トランザクションが終了するまで残ります (Btrieve v6.x 以降)。

注：

- Btrieve v6.x 以降では、並行 Btrieve トランザクションが使用されます。 この場合のロックは、ページレベルで実行されます (詳細については、Btrieve のマニュアルを参照してください)。
- トランザクションは COMMIT または ROLLBACK 操作が実行されたときに終了します。 トランザクションは、ROLLBACK で開かれたファイル上で再び入出力が行われたときに再開します。

CLOSE 操作は、トランザクションがアクティブでないときのみに Btrieve CLOSE 操作が行われるため、トランザクションを再開しません。

ファイルロックエラーは、ファイルが開かれたときに Xfh2btr によって返されます。これは、この COBOL システムが、ファイルを開くときのみにファイルロックエラーを返すためです。 Btrieve は、トランザクション内でファイルにアクセスするときにファイルロックエラーを返すことができます。ただし、このファイルロックエラーは、Xfh2btr によってレコードロックエラーに変換されます。これは、トランザクション内でファイルにアクセスするときにこの COBOL システムが正当にレコードロックエラーを返せるためです。

WRITELOCK コンパイラ指令

この COBOL システムでは、WRITELOCK コンパイラ指令を使って、ファイルにレコードを書き込み、書き込まれたレコードをロックできます。 ファイルに再度書き込まれたレコードにもロックをかけます。

Btrieve では、最初にレコードがファイルへ書き込みまたは再書き込みされ、その後にそのレコードをロックするために読み戻されます。 この操作には Btrieve ランタイムシステムへ何度もの呼び出しが必要で、呼び出しに多少の時間差があります。 このため最初のユーザによってレコードがロックのために読み戻される前に、2 番目のユーザがそのレコードを呼び出して最初のユーザが書き込んだレコードをロックする可能性があります。

非 **ANSI** モードでの **Btrieve** の呼び出し

Btrieve は通常、ANSI 規格に準拠しません。ただし、デフォルトでは、Xfh2btr 呼び出し変換モジュールは、ANSI 動作をエミュレートさせるために Btrieve ランタイムシステムに必要な呼び出しを行います。ただし、FILETYPE"6" へ FILETYPE コンパイラ指令を設定することにより、非 ANSI モードでの操作を選択できます。

このモードの操作では、各 Micro Focus ファイルハンドラの操作は最も近い Btrieve ランタイム呼び出しにマップされ、ANSI 規格の確認が行われなくなります。

- この COBOL システム内で設定したランタイムスイッチは、Xfh2btr 呼び出し変換モジュールによって無視されます。
- 操作は、NODETECTLOCK コンパイラ指令を設定したように行われます。
- ファイル内のレコードを削除、再書き込み、または書き込みした後は、現行レコードポインタ (CRP) の再位置付けは行われません。
- 各 COBOL 入出力呼び出しは、最も近くの関連する Btrieve ランタイムシステム呼び出しに、次のようにしてマップされます。

COBOL 入出力呼び出し	**Btrieve** ランタイムシステム呼び出し
OPEN	Open
CLOSE	Close
WRITE	Insert
READ	Get
START	Get
DELETE	Get/Delete
REWRITE	Get/Delete
UNLOCK	Unlock
ROLLBACK	Abort transaction
COMMIT	End transaction

Btrieve エラーメッセージ

Btrieve エラーメッセージの詳細なリストについては、Net Express オンラインヘルプを参照してください。([ヘルプ] メニューの [ヘルプトピック] をクリックします。次に [キーワード] タブで、[Btrieve]、[エラー] をダブルクリックします。)

Btrieve Record Manager からエラーが返された場合、Xfh2btr 呼び出し変換モジュールは Btrieve エラーを、プログラムに返す COBOL ファイル状態コードにマップします。 これらのマッピングについても、Net Express オンラインヘルプに記載されています。([ヘルプ] メニューの [ヘルプトピック] をクリックします。次に [キーワード] タブで、[Btrieve]、[状態コード] をダブルクリックします。)

第 10 章：データとキーの圧縮

ファイルハンドラを使用して作成したファイルのレコードとキーは圧縮することができるので、物理的なディスク領域の占有が少なくてすみます。 ファイルハンドラを使用してプログラムから圧縮ルーチンを呼び出して、データ圧縮を行うことができます。

データ圧縮

データ圧縮では、レコード順ファイルまたは索引ファイルのデータを圧縮することができます。 この COBOL システムでは 2 つの圧縮機構があります。ランレングスエンコーデング (タイプ 1) と拡張ランレングスエンコーデング (タイプ 3) です。

ファイルをランレングスエンコーデングで定義すると、繰り返される任意の文字列は 1 文字として格納されます。

DATACOMPRESS コンパイラ指令でデータ圧縮を実行できます。

固定長構造の順ファイルにデータ圧縮を指定すると、可変長構造の順ファイルに変換されます。 ファイル構造の詳細については、ヘルプトピック『ファイル構造』を参照してください。

ファイルのデータ圧縮は、そのファイルに対する SELECT 文の処理時に最後に処理された DATACOMPRESS 指令で決定されます。 そのため、次のような形式の行を記述することで、データ圧縮タイプを個々のファイルについて指定できます。

```
$SET DATACOMPRESS
```

これは、SELECT 文の直前に記述します。別のファイルを処理する前に $SET NODATACOMPRESS で圧縮を解除することを忘れないでください。

DATACOMPRESS コンパイラ指令の詳細については、ヘルプトピック『DATACOMPRESS』を参照してください。

注： 圧縮された順ファイルに REWRITE 文を使用することはお奨めしません。 REWRITE 処理が成功するのは、圧縮された新しいレコードの長さが圧縮された古いレコードの長さと同じ場合のみです。

キー圧縮

キー圧縮は、索引ファイルのキーに適用できる技術です。 キー圧縮には、次の 4 種類があります。

- 後続ヌル文字の圧縮
- 後続空白文字の圧縮

- 同一の先頭文字の圧縮
- 重複する副キー値の圧縮

ファイルハンドラ構成ファイルの KEYCOMPRESS オプションでキー圧縮を指定できます。圧縮の種類を示す次の整数を使用して指定します。

1	重複キーの圧縮
2	先頭文字の圧縮
4	後続空白文字の圧縮
8	後続ヌル文字の圧縮

これらの数字をいくつかまとめて追加すると、圧縮の種類を組み合わせて使用できます (ただし、相互排他的な後続ヌル文字と後続空白文字は一緒に指定できません)。

また、プログラムをコンパイルするときに、KEYCOMPRESS コンパイラ指令を使用することもできます

ファイルのキー圧縮は、そのファイルに対する SELECT 文の処理時に最後に処理された KEYCOMPRESS 指令で決定されます。そのため、プログラムで次のような形式の行を記述すると、個々のファイルのデータ圧縮を指定できます。

```
$SET KEYCOMPRESS"8"
```

この行はプログラムの SELECT 文の直前に記述してください。他のファイルを処理する前に、$SET NOKEYCOMPRESS を指定すると、キー圧縮を解除できます。

KEYCOMPRESS コンパイラ指令の詳細については、ヘルプトピック『KEYCOMPRESS』を参照してください。

後続ヌル文字の圧縮

キーに後続ヌル文字の圧縮を定義すると、キー値の後続ヌル文字はファイルに格納されません。

たとえば、30 文字の長さの主キーまたは副キーの最初の 10 文字のみを使用し、残りがヌル文字であるレコードを書き込むとします。圧縮をしない場合は、キーの 30 文字すべてが格納されるため、30 バイトが必要になります。後続ヌル文字の圧縮を行うと、11 バイトのみですみます (10 バイトを最初の 10 文字用に、1 バイトを後続ヌル文字用に使用します)。

後続の空白文字の圧縮

キーに後続空白文字の圧縮を定義すると、キー値の後続空白文字はファイルに格納されません。 情報は格納されるので、キーは適切に配置されます。

たとえば、30 文字の長さの主キーまたは副キーで最初の 10 文字のみを使用し、残りが空白文字であるレコードを書き込むとします。圧縮をしない場合は、キーの 30 文字すべてが格納されます。 後続空白文字の圧縮を行うと、索引ファイルの 11 バイトを占有するのみですみます (10 バイトを最初の 10 文字用に、1 バイトを後続空白文字用に使用します)。

先頭文字の圧縮

キーに先頭文字の圧縮を定義すると、前のキーの先頭文字と一致する先頭文字はすべて索引ファイルに格納されません。 情報は格納されるので、キーを適切に再構築できます。

たとえば、先頭文字を圧縮するようにキーを定義し、次のようなキー値のあるレコードを書き込むとします。

```
AXYZBBB BBCDEFG BBCXYZA BBCXYEF BEFGHIJ CABCDEF
```

索引ファイルに実際に格納されるキーは、次のようになります。

```
AXYZBBB BBCDEFG XYZA    EF      EFGHIJ  CABCDEF
```

重複キーの圧縮

副キーに重複キーの圧縮を定義すると、最初の重複キーのみがファイルに含められます。残りは格納されませんが、情報は格納されるので、キーは正しく再構築できます。

たとえば、"ABC" という副キー値をもつレコードを書き込むとします。重複キーの圧縮を有効化し、同じキー値を持つ別のレコードを書き込んだ場合には、ファイルハンドラは重複キー値を物理的に索引ファイルに格納しません。 ただし、レコードはなおも副キーパスで使用可能です。

データ圧縮とキー圧縮の例

次のプログラムでは、transfile にデータ圧縮を指定します。masterfile に対してはデータ圧縮を指定しません。キー圧縮については、キー `t-rec-key` と `m-rec-key` に対して前のキーと同じ後続空白と先行文字の抑制を指定します。`m-alt-key-1` と `m-alt-key-2` に対しては重複キーの繰り返しも抑制します。

```
$set callfh"extfh"
$set datacompress"1"
$set keycompress"6"
     select transfile
         assign to ...
         key is t-rec-key.
$set nokeycompress
$set nodatacompress

     select masterfile
         assign to ...
         organization is indexed
$set keycompress"6"
         record key is m-rec-key

$set keycompress"7"
         alternate key is m-alt-key-1 with duplicates
         alternate key is m-alt-key-2.
```

```
$set nokeycompress
```

圧縮ルーチン

ファイルハンドラがデータを圧縮するために使用するルーチンは、スタンドアロンモジュールです。 このため、この圧縮ルーチンをアプリケーションで使用することができます。また、独自に作成したデータ圧縮ルーチンをファイルハンドラで使用することができます。

Micro Focus の圧縮ルーチンとユーザ独自の圧縮ルーチンは、どちらも最大 127 個まで使用できます。

Micro Focus 圧縮ルーチン

Micro Focus のルーチンは、CBLDCnnn というモジュールに格納されています。nnn は 001～127 の 数字です。Micro Focus の圧縮ルーチンを使用するには、FCD の fcd-data-compress に 001～127 の値を設定します。

Micro Focus 圧縮ルーチン CBLDC001 および CBLDC101

圧縮ルーチン CBLDC001 と CBLDC101 では、ランレングスエンコーディング (RLC) 形式を使用します。RLC は、同じ文字の文字列 (実行) を検出し、その文字列を文字の識別子、カウントまたは 1 つのオカレンスとして減らす圧縮方法です。

注：2 バイト文字 (2 バイトの空白文字を含む) は圧縮されないため、これらのルーチンは 2 バイトのオカレンスを格納するファイルでは無効です。

CBLDC001 と CBLDC101は、特に空白文字、バイナリのゼロと文字のゼロ (1 文字に減らすことができる)、印刷可能文字 (カウント数とその後に続く反復文字の計 2 文字に減らすことができる) に対して実行することを目的としています。

CBLDC001 と CBLDC101 の違いは、CBLDC101 が 256 KB までの長さの文字列を処理できる一方で、CBLDC001 は 65535 バイトまでの文字列しか処理できないということです。

圧縮ファイルでは、バイトは次の意味を持っています(16 進値表示)。

20-7F	ほとんどの印刷可能な文字。通常の ASCII 文字です。
80-9F	それぞれ 1～32 文字までの空白文字
A0-BF	それぞれ 1～32 文字までのバイナリのゼロ
C0-DF	それぞれ 1～32 文字までの文字のゼロ
E0-FF	後に続く文字の 1～32 回までのオカレンス数
00-1F	後に続く文字の 1～32 回までのオカレンス数。圧縮コードとしてではなく、文字どおり解釈します。 元のデータに 00-1F、80-9F、A0-BF、C0-DF、E0-FF のどちらかの範囲の文字が存在するときに使用されます。(このような文字の 1 文字は 2 バイトに拡張されます。それ以外の場合には、圧縮により発生する弊害はありません。)

Micro Focus 圧縮ルーチン CBLDC003 および CBLDC103

CBLDC001 と CBLDC101 同様、これらのルーチンではランレングスエンコーディング (RLC) を使用しますが、1 バイトと 2 バイトの文字列 (実行) を検出します。これらのルーチンは DBCS 文字に適していますが、CBLDC001 のかわりに使用することもできます。

CBLDC003 と CBLDC103 の違いは、CBLDC103 が256 KB までの長さの文字列を処理できる一方で、CBLDC003 は 65535 バイトまでの文字列しか処理できないということです。

圧縮形式は、ヘッダーバイト 2 個の後に 1 つ以上の文字が続きます。ヘッダーバイトのビットの内容は次のとおりです。

ビット 15	設定解除 - 1 文字
ビット 14	設定 - 圧縮シーケンス 設定解除 - 非圧縮シーケンス
ビット 0〜13	圧縮した文字または圧縮しない文字数

文字列の長さはヘッダーのビットによって異なります。

ビット 14 および 15 を設定した場合	2 つの反復文字
ビット 14 のみを設定した場合	1 つの反復文字
その他	1〜63 文字までの非圧縮文字

Micro Focus の圧縮ルーチンの呼び出し

データファイルを圧縮する場合には、ファイルハンドラは DATACOMPRESS コンパイラ指令で指定する圧縮ルーチンを呼び出します。

Micro Focus の圧縮ルーチンを呼び出すには、次の構文を使用します。

COBOL:

```
call "CBLDCnnn" using input-buffer,
input-buffer-size,
output-buffer,
output-buffer-size,
compression-type
```

C:

```
cbldcnnn(input_buffer, &input_buffer_size,
         output_buffer, &output_buffer_size,
&compression-type);
```

ルーチン CBLDC001 と CBLDC003 のパラメータの内容は、次のとおりです。

nnn	001〜127 のデータ圧縮ルーチン。

input_buffer	PIC X(サイズ) データ項目。 ルーチンの入口で圧縮または圧縮解除するデータを記述する必要があります。最大サイズは 65,535 バイトです。
input_buffer_size	2 バイトの (C では int、COBOL では PIC XX COMP-5) データ項目。 入口で入力バッファのデータ長を記述する必要があります。
output_buffer	PIC X(サイズ) データ項目。 終了時に、結果データを格納します。
output_buffer_size	2 バイトの (C では int、COBOL では PIC XX COMP-5) データ項目。 入口で使用可能な出力バッファのサイズを記述する必要があり、終了時にはバッファのデータ長を格納します。
compression-type	1 バイトの (C では char、COBOL では PIC X COMP-X) データ項目。 入口で入力データを圧縮するか圧縮解除するかを指定する必要があります。 0 - 圧縮、1 - 圧縮解除

ルーチン CBLDC101 と CBLDC103 のパラメータの内容、次のとおりです。

nnn	001～127 のデータ圧縮ルーチン。
input_buffer	PIC X(サイズ) データ項目。 ルーチンの入口で、圧縮または圧縮解除するデータを記述する必要があります。最大サイズは 256 バイトです。
input_buffer_size	4 バイトの (C では int、COBOL では PIC XXXX COMP-5) データ項目。 入口で入力バッファのデータ長を記述する必要があります。
output_buffer	PIC X(サイズ) データ項目。 終了時に、結果データを格納します。
output_buffer_size	4 バイトの (C では int、COBOL では PIC XXXX COMP-5) データ項目。 入口で使用可能な出力バッファのサイズを記述する必要があり、終了時にはバッファのデータ長を格納します。
compression-type	1 バイトの (C では char、COBOL では PIC X COMP-X) データ項目。 入口で入力データを圧縮するか圧縮解除するかを指定する必要があります。 0 - 圧縮、1 - 圧縮解除

RETURN-CODE 特殊レジスタは、操作が成功したかどうかを示します。圧縮または圧縮解除は、出力バッファが小さく結果を受け付けられない場合のみに失敗します。 0 は成功、1 は失敗を示します。

ユーザ独自の圧縮ルーチン

ユーザ独自の圧縮ルーチンは、USRDCnnn というモジュールに格納する必要があります。nnn は 128～255 までの数字です。

ユーザ独自のルーチンを呼び出すには、Micro Focus のルーチンを呼び出すのと同じ構文で、CBLDCnnnの代わりに、ファイル名 USRDCnnn を使用します。nnn は 128～255 までの数字です。

圧縮ルーチンをシステムで使用するには、必要なときに呼び出せる共有オブジェクトを作成する必要があります。

cob オプションを使用して、前の UNIX COBOL システムからのプログラムのデータ圧縮ルーチンの呼び出しを、新しい呼び出しへマップできます。次のように記述します。

```
-m CBL_DATA_COMPRESS_nnn=CBLDCnnn
```

注：

- 圧縮ルーチンでファイルハンドラを呼び出さないでください。ループが発生する可能性があります。 ファイルアクセスが必要な場合、バイトストリームのファイル入出力を使用します。
- 一度ファイルに対してデータ圧縮を有効化すると、そのファイルには常に同じ種類の圧縮を指定する必要があります。同じ種類の圧縮を指定しない場合は、ファイルを開くときに、ランタイムシステムエラーを受け取ります。
- データ圧縮は索引ファイルまたはレコード順ファイル以外のファイルには影響しません。サポートしていないファイルではコンパイル時に無視されます。
- データ圧縮された固定長順ファルを読み込む場合は、ファイルが圧縮されていることをプログラムで指定する必要があります。 DATACOMPRESS コンパイラ指令を指定してこれを行います。

データとキーの圧縮　　データファイル処理の概要

第 11 章 : ファイル操作のトレース

ここでは、ファイル操作のトレースのために用意された機能の使用法について説明します。これらの機能は、ファイル処理の問題での調査に役立ちます。

はじめに

ファイルハンドラの機能のファイル操作のトレースでは、次の操作を行うことができます。

- ファイルハンドラ構成パラメータを使用して、トレースのオン/オフを切り替える。
- ファイルハンドラ構成パラメータを使用して、ログと呼ばれるさらに高度なトレースを設定する。
- ライブラリルーチンを使用して、COBOL プログラムから動的にトレースのオン/オフを切り替える。

トレースとログの構成

ファイルハンドラ構成パラメータの TRACE、TRACEFILEEXTEND、TRACEFILENAME、LOG および LOGFILENAME を使用して、トレースとログのアクティビティを次のように制御します。

- TRACE=ON に設定すると、トレースがオンになります。
- TRACEFILEEXTEND=ON に設定すると、トレースファイルが存在する場合、現行のトレースファイルの最後にトレース情報が書き込まれます。
- TRACEFILENAME=filename を設定すると、トレースファイルにデフォルト以外の名前をつけることができます。デフォルト名は xfhtrace.xfh です。
- LOG=ON に設定すると、ファイルに関連した特殊状況 (ファイルの破損や自動ファイル回復など) がログファイルに逐次出力され、後から必要に応じて調査できます。
- LOGFILENAME=filename を設定すると、ログファイルにデフォルト以外の名前をつけることができます。デフォルト名は xfhlog です。

TRACE と LOG は、グローバルにまたは個別のファイルに設定することができます。TRACEFILEEXTEND、TRACEFILENAME、および LOGFILENAME は、グローバルにしか設定できません。

構成オプションをグローバルに設定する方法や個別ファイルに設定する方法の詳細について

は、『ファイルハンドラの構成』の章の『構成ファイル』の項を参照してください。

次に構成ファイルの例を示します。

```
[XFH-DEFAULT]
TRACE=OFF
TRACEFILENAME=c:¥trace¥mftracefile.xfh
TRACEFILEEXTEND=ON
LOG=ON
LOGFILENAME=c:¥temp¥mflogfile.log

[datafile1.dat]
TRACE=ON

[datafile2.dat]
LOG=OFF
```

この例では、ファイルトレースはグローバルにはオフ、datafile1.dat データファイルに対してはオンです。新しいトレースレコードは、既存のトレースファイル c:\trace\mftracefile.xfh がある場合、このファイルに追加されます。 ログは、グローバルではオン、データファイル datafile2.datに対してはオフです。

注： トレースファイルのサイズが 4 GBを超える場合は、構成オプションの FILEMAXSIZE=8 および IDXFORMAT=8 も指定する必要があります。

動的トレース

FCD に適切なフラグを設定するか、ライブラリルーチンを使用するかで、COBOL プログラム内からトレースのオン/オフを切り替えることができます。 これは、アプリケーションの動作のサブセットで、ファイル処理の問題に対処したい場合に便利です。 FCD フラグを設定する場合は、特定のファイルに対してこれを行います。 ライブラリルーチンを使用する場合は、トレース設定は、その時点以降のプログラムで実行されるすべてのファイル操作に影響します。

動的トレースを実行する場合には、2 つのフラグを同時に使用します。 これは、READ や WRITE などのファイル操作をトレースするには、OPEN 操作がトレースされている必要があるためです。 トレースをオフにしてプログラムを起動し、ファイルを開いた後に動的にトレースをオンにした場合、トレースの仕組みでは、後続の操作をトレースすることはできません。 最も良い方法は、プログラムの始めに OPEN と CLOSE の動的トレースをオンにしてから、すべての操作のオン/オフを必要に応じて切り替えることです。 OPEN 操作と CLOSE 操作のトレース自体もまた役に立つことがあります。

FCD を使用したトレースの制御

FCD フラグを使用するには、FCD にアクセスする必要があります。『ファイルハンドラとソート API』の章の『FCD へのアクセス』の項を参照してください。

ファイルのトレースは FCD 内の 2 つのフラグビットの設定によって制御されます。

- fcd-flags-1 のビット 1 は、すべてのファイル操作をトレースするかどうかを制御します。
- fcd-fs-flags のビット 3 は、OPEN 操作および CLOSE 操作のみをトレースするかを制御します。

これらのビット設定は、個別のファイルのトレースを制御するために必要に応じて変更できます。

ライブラリルーチンを使用したトレースの制御

ライブラリルーチンでは、FCD を持っていない場合でも、FCD フラグを設定するのと同様にトレースを操作することができます。 ライブラリルーチンを使用して変更したトレース動作はすべて、これ以降にアクセスするすべてのファイルに影響します。

トレースを制御するには、MFFH_MODIFY_TRACE ライブラリルーチンを呼び出します。次の設定ができます。

- 主トレースフラグのオン/オフを切り替える。
- OPEN/CLOSE トレースフラグのオン/オフを切り替える。
- 以前に設定した変更を無効にする。

設定したすべての変更を無効にするには、ライブラリルーチンの MFFH_MODIFY_DISABLE を呼び出します。

詳細については、ヘルプトピック 『MFFH_MODIFY_TRACE』および 『MFFH_MODIFY_DISABLE』を参照してください。

ファイル操作のトレース　　データファイルの編集

第 12 章：データファイル処理の概要

ここでは、PC でのアプリケーションデータファイルの作成および維持に役立つ、Net Express 付属のツールの概要を説明します。

ここでは次の内容を説明します。

- 使用可能なツール
- 特定の設定を変更してツールの動作を制御する方法

はじめに

Net Express には、データファイルの作成、維持、変換に役立つ次のツールが用意されています。

- データファイルエディタ

  このツールでは、さまざまなタイプのデータファイルを作成できます。 データファイルを作成したら、「データファイルエディタ」ウィンドウを使用して、レコードの作成や削除、およびデータの表示、編集、印刷ができます。 検索ツールも提供されます。詳細については、『データファイルの編集』の章を参照してください。

- データファイルコンバータ

  このツールでは、ファイル編成と形式の変換ができます。 EBCDIC と ANSI 間の変換を行うことができます。これを行う場合は、浮動小数点形式も変換できます。レコード長を変換できます。 データを再構築できます (順編成から索引編成へなど)。詳細については、『データファイルの変換』の章を参照してください。

- レコードレイアウトエディタ

  フォーマットされたデータの編集や変換が実行できるレコードレイアウトを作成、維持できます。 1 つのデフォルトレコードレイアウトと 1 つ以上の条件付きレコードレイアウトを定義できます。 また、レコードレイアウトを使用して、ANSI と EBCDIC 間で変換を行う場合に変換するデータ項目を指定することもできます。詳細については、『データファイルでのレコードレイアウトの使用』の章を参照してください。

- ファイル索引の修正

  破損した索引ファイルの索引の復元手順。 詳細については、ヘルプトピックの『ファイル索引の修正 - 概要』を参照してください。

これらのツールを総称してデータツールと呼びます。データツールの使用法の概要について

は、『入門書』の『データファイルの維持と作成』の章を参照することをお奨めします。

データツールの使用法の詳細については、Net Express オンラインヘルプを参照してください。([ヘルプ] メニューの [ヘルプトピック] をクリックします。[目次] タブで、[開発環境]、[データファイルの処理] をダブルクリックします。)

注：Btrieve ファイルではデータツールを使用できません。

データツールの構成

Net Express には、データツールの動作方法を制御できる一般的な設定が用意されています。設定を変更または表示する場合は、[オプション] メニューの [データツール] をクリックします。

次の設定ができます。

- デフォルトの文字集合として、EBCDIC と ANSI のどちらかを選択できます。
- ヌルや空白文字など、EBCDIC および ANSI のレコードの埋込みに使用する内容を指定できます。 EBCDIC と ANSI に対しては別の埋込みバイトを使用できます。

データファイルエディタに特に影響する設定については、『データファイルの編集』の章の『データファイルエディタの構成』の項を参照してください。

文字符号系の構成

Codecomp ユーティリティを使用すると、EBCDIC/ANSI 文字集合変換のためのカスタマイズされたマッピングの表を作成できます。 Net Express には多くの文字集合のサポートが組み込まれています。 詳細については、Net Express オンラインヘルプを参照してください。([ヘルプ] メニューの [ヘルプトピック] をクリックします。[キーワード] タブで、[codecomp] を検索します。)

データファイル処理の概要　　データファイルでのレコードレイアウトの使用

第 13 章 : データファイルの編集

ここでは、PC でデータファイルを作成、変換する際に使用する、Net Express のデータファイルエディタについて説明します。

データファイルエディタは、入力データファイルを、プログラムのテスト用、出力ファイルの表示用、プログラムが正常に動作しているかの検証用に調整するためのデータツールです。 最小限の設定作業でデータを、プログラム仕様に従ってフォーマットしたレコード形式に編集および表示できます。

データファイルエディタの使用に関する詳細については、ヘルプの『データファイルの編集』を参照してください。

概要

データファイルエディタには、データファイルの編集を行うためのグラフィカルインターフェイスが用意されています。 データファイルエディタでは、次のことが実行できます。

- データファイル内のレコードとフィールドの表示と編集
- 新しいデータファイルの作成
- ANSI または EBCDIC 文字集合を使用したデータの参照
- データファイル内での文字列の検索
- レコードの作成
- レコードの削除
- データの印刷

注：フォーマットされたデータを表示および編集する場合は、最初にデータファイルのレコードレイアウトをすべて定義する必要があります。 これを実行するために、Net Express にはレコードレイアウトエディタが用意されています。レコードレイアウトエディタの詳細については、『データファイルでのレコードレイアウトの使用』の章を参照してください。

以後に説明する内容は、レコードレイアウトを読み込んでデータをフォーマットしていることを前提にしています。

「データファイルエディタ」ウィンドウ

データファイルエディタは、新しいデータファイルを作成するとき、または既存のデータファイルを開くときに開きます。 データファイルエディタにはメインウィンドウが 1 つあります。

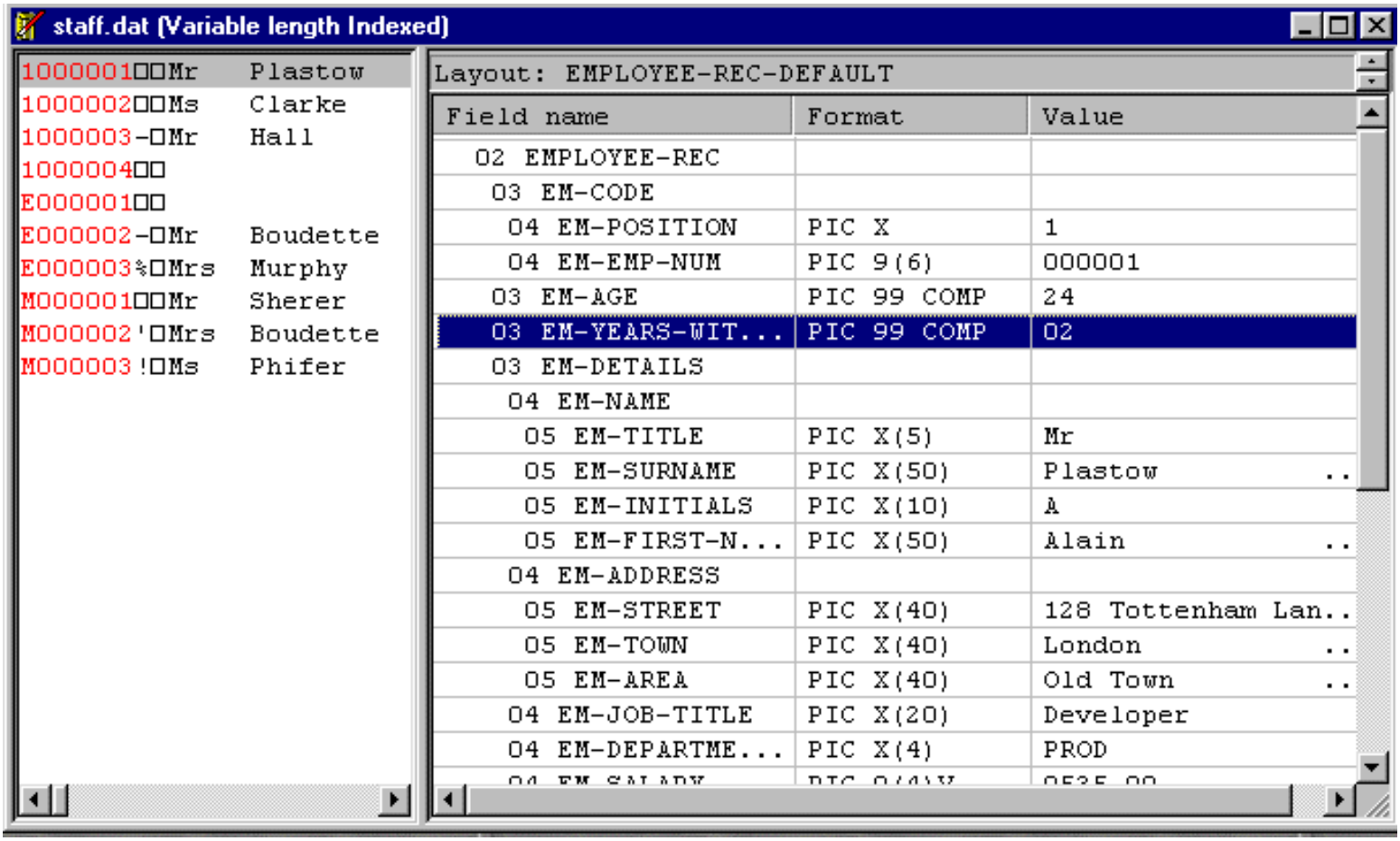

図 13-1：「データファイルエディタ」ウィンドウ

このウィンドウは、左側のペインと右側のペインに分かれています。ウィンドウの下部にはデータファイルステータスバーがあります。このウィンドウの上部にはデータファイルツールバーがあります。

16 進のデータは左側のペインと右側のペインの両方で表示および編集できます。 各バイトの 16 進の値が、各ペインの下の 2 行に表示されます。

注：データファイルにレコードレイアウトを定義していない場合には、右側のペインにはデータが表示されません。レコードレイアウトの詳細については、『データファイルでのレコードレイアウトの使用』の章を参照してください。

ヒント：「データファイルエディタ」ウィンドウで右クリックすると、状況に応じたポップアップメニューが表示され、小さなメニューオプションサブセットが示されます。 たとえば、レコードを挿入および削除できます。

左側のペイン

「データファイルエディタ」ウィンドウの左側のペインは、非フォーマット表示と呼ばれます。 左側のペインには、各行が 1 レコードを持つレコードデータが表示されます。

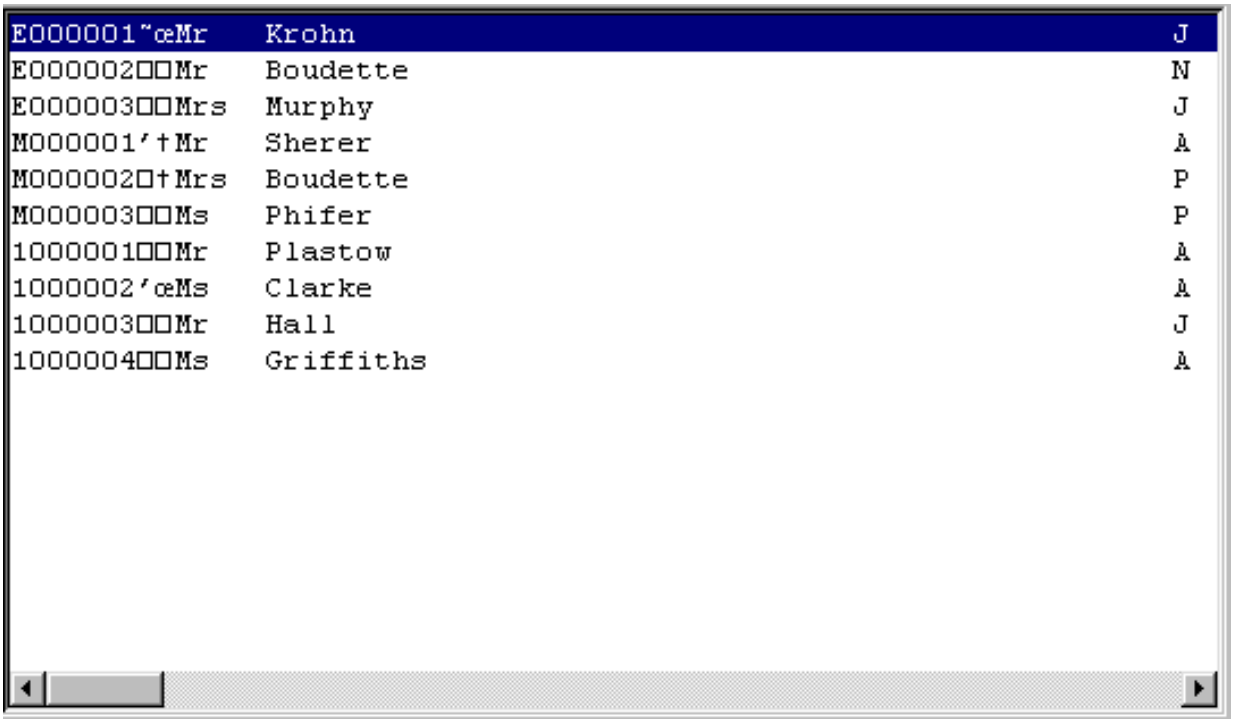

図 13-2： 左側ペインのフォーマットされていないレコードデータ

左側のペインでは、索引ファイル内のキー領域が赤で強調表示されています。

レコードをスクロールする場合は、水平スクロールバーを使用します。 レコードを上下にスクロールする場合は、カーソルの上矢印キーと下矢印キーを使用します。 現在強調表示されているレコードは現在のレコードで編集できます。 また、レコードをクリックして、編集するレコードを選択することもできます。

レコードのどちらかが使用可能なレコードレイアウトと一致しない場合や、無効な OCCURS DEOENDING ON RANGE が含まれている場合は、右側のペインおよびデータファイルステータスバーに警告メッセージが表示されます。

ヒント：レコード内にカーソルを置き、[表示] メニューの [データツール > 表示の同期] をクリックして右側のペインの同じ位置にカーソルを移動させることができます。

右側のペイン

「データファイルエディタ」ウィンドウの右側のペインは、フォーマット表示と呼ばれます。 右側のペインは、現在のレコードにレコードレイアウトが関連付けられていない場合は空になります。レコードレイアウトの詳細については、『データファイルでのレコードレイアウトの使用』の章を参照してください。

右側のペインには、上から順番に次の領域が表示されます。

- レイアウトステータスバー

  レイアウトステータスバーには、現在のレコードで使用中のレコードレイアウトの名前が表示されます。

- フィールドレイアウトビュー

  フィールドレイアウトビューでは、COBOL レコードレイアウトを使用して、レコード内の各フィールド名、PICTURE 句および値を表示します。

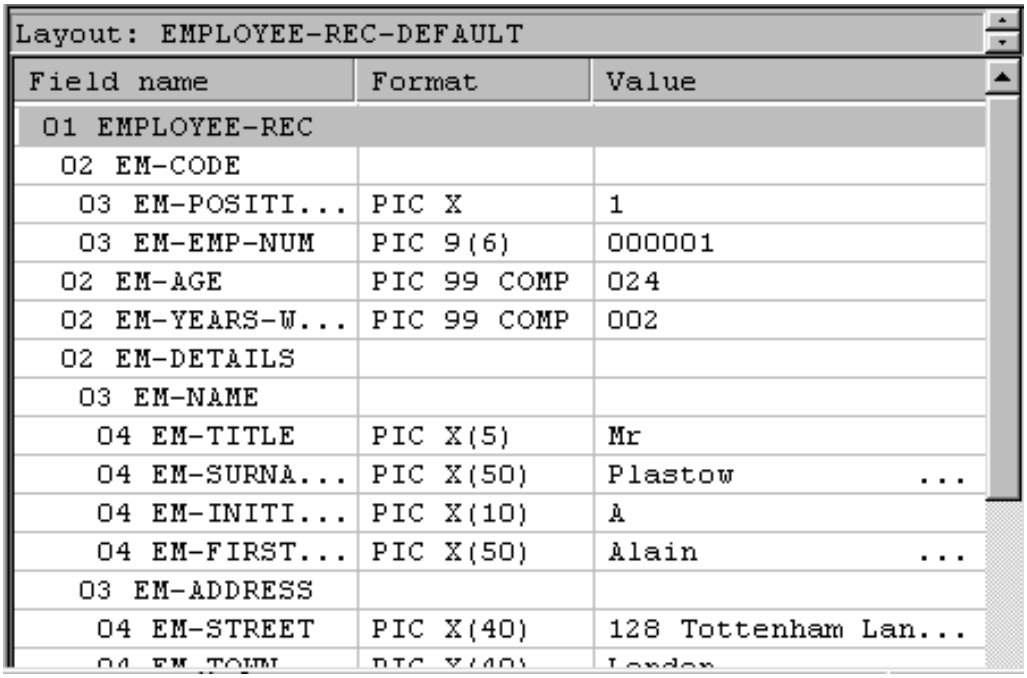

図 13-3： 右側のペインのフィールドレイアウトビュー

フィールドレイアウトビューを選択するには、右側のペイン内でクリックします。 カラムのサイズを変更するには、カラムヘッダー行上のカラム区切りをドラッグします。

各フィールドは、個々の行に表示されます。 キーボードのカーソル上矢印と下矢印キーを使用すると、フィールド内を上下にスクロールできます。最初のフィールドを選択するためには、Home キーを押します。最後のフィールドを選択するためには、End キーを押します。 現在選択されているフィールドは強調表示されます。

フィールドの内容は、カーソルをカラム「値」に合わせることで編集できます。カーソルをフィールドの先頭に移動するには、Home キーを押します。カーソルをフィールドの末尾に移動するには、End キーを押します。

ヒント：

- レコード間を移動するには、「フィールドレイアウトビュー」ウィンドウの右側のペインの右上部にある小さい上下方向の矢印ボタンを使用する、または左側のペインで新しいレコードを選択します。
- フィールドをすばやく検索するには、[検索] メニューの [データツール > フィールドへの移動] を使用します。
- フィールド内にカーソルを置き、[表示] メニューの [データツール > 表示の同期] をクリックして左側のペインの同じ位置にカーソルを移動させることができます。

データファイルツールバー

データファイルツールバーは、文字集合を変更したり、索引ファイルに別の参照キーを選択したり、データファイルに関する情報を検索したりする場合のショートカットとして使用します。

図 13-4： データファイルツールバー

データファイルエディタでは、すべてのデータを ANSI または EBCDIC のどちらかで表示します。 データファイルツールバーには、現在使用されている文字集合が表示されます。 ファイル表示形式が正しいかどう

かを確認するには、表示形式情報で空白文字を探します。EBCDIC 形式のファイルを ANSI 形式で表示すると、空白文字 (EBCDIC コード h"40") は「@」として表示されます。一方、ANSI 形式のファイルを EBCDIC 形式で表示すると、空白文字 (ANSI コード h"20") は英数字以外の文字として表示されます。この場合には、実際に表示される文字はオペレーティングシステムの文字集合によって異なります。

データファイルツールバーのオンとオフを切り替えるには、[表示] メニューの [ドッキングできるウィンドウ] をクリックします。

データファイルステータスバー

Net Express ウィンドウの下部にあるステータスバーには、左から順に次の情報が表示されます。

- 相対レコード番号 (相対ファイルの場合のみ)。相対ファイル以外の場合は、「N/A」と表示されます。
- 現在選択されているレコードの長さと次の情報
    - 可変長レコードの場合

      最大レコード長と最小レコード長 (最小レコード長から最大レコード長)
    - 固定長レコードの場合

      「固定」
- レコード数またはフィールド数
    - 左側のペイン

      ファイル (順ファイルの場合のみ) 内のレコード数 (つまり、行数)。順ファイル以外のファイルは適宜編集され、ファイル内のレコード数は不明であるため「N/A」と表示されます。詳細については、『順ファイル以外のファイルのナビゲート』の項を参照してください。
    - 右側のペイン

      レコードのフィールド数
- カーソル位置またはフィールド番号
    - 左側のペイン

      カーソルの現在の位置 (行番号とカラム位置を含む)。 行番号とカラム位置はどちらも 0 から始まります。
    - 右側のペイン

      現在選択されているフィールドの番号。
- 編集モード
    - 左側のペイン

      「OVR」 (上書きモードの場合)、または「空白」 (挿入モードの場合) が表示されます。これらのモードは、Insert キーを押すと切り替えることができます。
    - 右側のペイン

      この部分は空です。 数字フィールドは常に上書きモードで編集し、英数字フィールドは常に挿入モードで編集します。

データファイルエディタの構成

データファイルエディタは、次の方法で構成します。

- レコードを編集、削除するとき、および索引ファイルや相対ファイルを開くと表示される警告プロンプトをオフにします。詳細については、『警告プロンプトをオフに設定』の項を参照してください。
- 次を実行するかどうかを指定します。
    - データファイルを開くたびにそのファイルをバックアップするかどうか。
    - データファイルを読み込み専用と編集モードのどちらで開くか。
    - 左側ペインでのデータの編集時に上書きモードと挿入モードのどちらを使用するか。

  これらの 3 つの切り替えは、現在のセッションで次にデータファイルを開いたときに有効になります。 現在開いているデータファイルには影響しません。

これらの構成を変更するには、[オプション] メニューの [データツール] をクリックしてから、「データツールオプション」ダイアログボックスの [データファイルエディタ] タブをクリックします。

注：ここで説明するスイッチや警告は特にデータファイルエディタに適用されます。 データツールには他にも変更可能な一般的な設定があります。詳細については、『データファイル処理の概要』の章にある『データツールの構成』の項を参照してください。

警告プロンプトをオフに設定

データファイルエディタを使用する前に、レコードを編集または削除しようとするたびにプロンプトを表示するかどうか考慮する場合があります。 データファイルエディタでは、デフォルトでどちらの操作についても、続行するかどうかを確認するプロンプトが表示されます。「データツールオプション」ダイアログボックスの「更新の警告」および「削除の警告」をオフにすると、データファイルエディタではこれらのプロンプトは表示されません。

データファイルエディタでは、索引ファイルまたは相対ファイルを開くたびにプロンプトが表示されます。これらのファイルタイプの編集はすぐに適用されるためです。 順ファイルでは、データを編集しようとするとプロンプトが表示されます。「情報メッセージを表示」フィールドを選択解除して、これらのプロンプトを表示する必要がないことを指定できます。

これらの設定に行った変更は、次にデータファイルを開いたときに有効になります。 プロンプトのオンとオフを切り替える場合は、PC でデータファイルエディタを使用する他のユーザに必ず通知してください。

注：警告メッセージボックスで 「このメッセージは今後表示しない」をオンにして、現在のセッションで警告メッセージを今後表示しないようにすることができます。

バックアップの取得

「データツールオプション」ダイアログボックスの「編集する前にファイルをバックアップする」フィールドでは、データファイルエディタでデータファイルを開くたびにバックアップを取得するかどうかを指定できます。 デフォルトでは、自動的にバックアップを取得します。 バックアップを使用すると、ファイルを編集前の状態に復元できます。

バックアップファイルはデータファイルと同じフォルダに格納されます。データファイルは filename.dbk にコピーされます。データファイルが索引ファイルである場合には、索引は filename.ibk にコピーされます。

データファイルを読み込み専用で開く

「データツールオプション」ダイアログボックスの「読み込み専用で開く」フィールドでは、データファイルを読み込み専用モードと編集モードのどちらで開くかを指定できます。

- 読み込み専用モード

  他のプロセスもファイルを読み取ることができます。ただし、ファイルの内容は変更できません。

- 編集モード

  ファイル内のデータのみ編集できます。

デフォルトでは編集モードが使用されます。 ただし、編集モードを選択した場合でも、Windows の読み込み専用属性が設定されているファイルは編集できません。

挿入モードまたは上書きモードでの編集

「データツールオプション」ダイアログボックスの「常に非フォーマット表示で上書き」フィールドでは、「データファイルエディタ」ウィンドウの左側のペインでのデフォルトの編集モードを挿入モードと上書きモードのどちらにするかを指定します。 デフォルトの編集モードは上書きモードです。

データを挿入または削除すると、すべてのデータフィールドがずれる可能性があります。 このような問題を避けるためには、常に上書きモードを使用するようにしてください。上書きモードで Insert キーを押すと、挿入モードに切り替えてよいかどうか確認するメッセージボックスが表示されます。

この設定は右側のペインには影響しません。右側のペインの数字フィールドは常に上書きモードで編集し、英数字フィールドは挿入モードで編集します。

データファイルの作成

データファイルを作成するには、[ファイル] メニューの [新規作成] をクリックします。「新規作成」ダイアログボックスで[データファイル]をクリックしてから、[OK] をクリックします。

Net Express で、「ファイルの作成」ダイアログボックスが開きます。

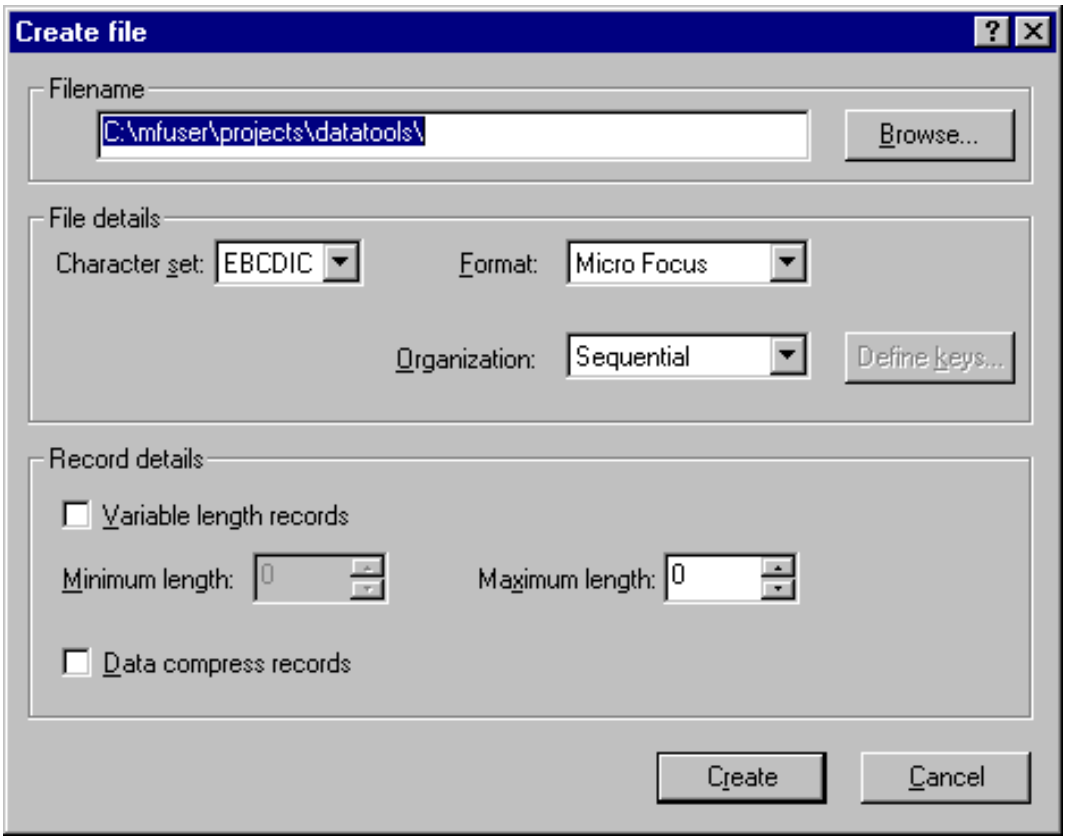

図 13-5：「ファイルの作成」ダイアログボックス

レコードの形式が固定長である場合は、「最大の長さ」フィールドで固定長を指定します。レコードの形式が可変長である場合は、「最小の長さ」フィールドと「最大の長さ」フィールドでそれぞれ最小のレコード長と最大のレコード長を指定します。

索引ファイルを作成する場合は、それらのキーを定義する必要があります。詳細については、『索引ファイルのキーの定義』の項を参照してください。

ファイル形式とレコード形式

次の表に、ファイルとそれに対応するレコード形式を示しています。 この表には、キーを定義する必要がある、または最小レコード長を指定する必要があるかどうかも示します。 最小レコード長は、可変レコード形式にのみ適用されます。

ファイル形式	レコード形式	キーを定義するかどうか	最小レコード長を指定するかどうか
Micro Focus 順ファイル	固定、可変	指定しない	レコード形式が可変の場合は指定する。
Micro Focus 行順ファイル	可変	指定しない	指定しない。
Micro Focus 索引ファイル	固定、可変	指定する	レコード形式が可変の場合は指定する。最小長は、このファイルに定義された最大オフセットとキー長の合計より長い必要があります。
Micro Focus 相対ファイル	固定、可変	指定しない	レコード形式が可変の場合は指定する。
IDXFORMAT(4)	固定、可変	指定する	レコード形式が可変の場合は指定する。最小長は、このファイルに定義された最大オフセットとキー長の合計より長い必要があります。
IDXFORMAT(8)	固定、可変	指定する	レコード形式が可変の場合は指定する。最小長は、このファイルに定義された最大オフセットとキー長の合計より長い必要があります。

ファイルヘッダー

行順編成、固定長順編成、および固定長相対ファイルには、ファイルヘッダーがありません。そのため、これらのタイプのファイルを作成する場合は、データファイルエディタでヘッダー情報を保存するプロファイル (.pro) ファイルを作成します。 デフォルトでは、プロファイルファイルは、データファイルと同じフォルダに格納され、同じファイル基本名を使用します。

プロファイルファイルのない行順ファイル、固定長順ファイル、または固定長相対ファイルを開こうとすると、ファイルヘッダーの詳細を入力するように求められます。

IDXFORMAT(8) データファイル

データツールは、大容量ファイルを処理できる Windows NT などのオペレーティングシステムで 2 GB を超える索引ファイルをサポートします。 このような大容量ファイルには、IDXFORMAT(8) 形式を使用します。

他の索引ファイルとは異なり、IDXFORMAT(8) ファイルでは、個々のファイルにその索引が保存されません。 かわりに、各 IDXFORMAT(8) ファイルにはデータと索引情報の両方が保存されます。

1 GB を超えるファイルにアクセスできるようにするには、ファイルハンドラ構成ファイル extfh.cfg に FILEMAXSIZE=8 を追加します。このファイルの詳細については、『ファイルハンドラの構成』の章を参照してください。

データファイルエディタのロック動作では、FILEMAXSIZE=4 を使用した 1 GB を超えるサイズのデータファイルをサポートします。つまり、他のプログラムは、データファイルエディタで編集するために開いたデータファイルにはアクセスできません。これは、「データツールオプション」ダイアログボックスで「読み込み専用で開く」フィールドを選択している場合には適用されません。 データファイルがロックされている場合は、データファイルエディタで、ファイルを読み込み専用で開くように求められます。

ファイルのサポートは、最大で約 21 億個のレコードを持つデータファイルに制限されます。 データファイルエディタで順編成データファイルを開く場合は、各順編成レコードにつき 25 バイトの仮想メモリが必要であるため、制限は少なくなります。

警告：

- Windows 95 または Windows 98 クライアント用のアプリケーションを作成している場合は、2 GB を超えるファイルを開かないでください。また、ファイルハンドラ構成ファイル extfh.cfg では FILEMAXSIZE の値を 8 に設定しないでください。
- データファイルエディタ、データファイルコンバータ、および索引ファイルの修正では、ストライプ化されていないファイルの場合に限り IDXFORMAT(8) をサポートします。 ストライプ化されたファイルのストライプでこれらのツールを使用すると、データが破損する可能性があります。

索引ファイルのキーの定義

索引ファイルを作成しているときは、[キーの定義] ボタンが有効になります。[キーの定義] をクリックして、「キー情報」ダイアログボックスで次のことが実行できます。

- キーを 1 つ以上定義する。
- 各キーに対して、構成要素を 1 つ以上定義する。
- 構成要素ごとに、構成要素の長さを指定する。 この値は、1,016 バイト未満または最大レコード長未満で、どちらか小さい方の値にする必要があります。
- 各構成要素について、オフセットを指定する。この値は、最小レコード長未満または 63,488 バイト未満で、どちらか小さい方の値にする必要があります。

さらに、次のような操作も可能です。

- 固有でないキーを使用する。つまり、キーの値が重複していてもかまいません。
- 1 つ以上の追加のキーを指定し、現在のキーの前後に追加する。 「キーの情報」ダイアログボックスのキーリストにある最初のキーが常に主キーとなります。
- 現在の構成要素の前後に新しく 1 つ以上の構成要素を追加する。
- キーまたは構成要素を削除する。
- キー圧縮を適用することを指定する。
- キーをスパースキーに指定する。この場合、キー内にスパース文字しか含まれないレコードに対して索引エントリは書き込まれません。

データファイルを開く

データファイルを開く場合は、Net Express でファイルの種類とレコード長を識別できるようにする必要があります。 この情報はデータファイルのヘッダー、またはプロファイル (.pro) ファイルに保存されています。ヘッダー情報の詳細については、『ファイルヘッダー』の項を参照してください。

順ファイルを開くと、データファイルエディタは、データ修正を行うためのメモリにファイルを読み込みます。 データファイルは、閉じるまでディスクで更新されません。 変更した内容を適用するかどうかを確認するメッセージが表示されます。

索引ファイルか相対ファイルを開く場合には、このタイプのファイルは非常に大きくなることがあるため、メモリにロードされることはありません。 これらのファイルのデータを修正し、レコード外に移動すると、データファイルエディタはディスクのレコードを更新し、その確認を示すメッセージを表示します。

注：

- バックアップを指定しても、バックアップファイルを保存する十分な領域がない場合は、データファイルを開くことができません。このオプションの詳細については、『バックアップの取得』の項を参照してください。
- データを誤って変更しないようにするために、ファイルを読み込み専用で開くことができます。詳細については、『データファイルを読み込み専用で開く』の項を参照してください。

ファイルヘッダー

データファイルを開くときに、Net Express ではデータファイルのヘッダー情報を使用して、ファイルの種類やレコード長を識別します。

- 索引ファイル、可変長順ファイル、および可変長相対ファイル

  これらのファイルタイプには、この情報を含むヘッダーレコードが含まれます。

- 行順ファイル、固定長順ファイルおよび固定長相対ファイル

  これらのファイルタイプにはファイルヘッダーは含まれません。ヘッダー情報は、プロファイル (.pro) ファイルで個別に維持されます。 プロファイルファイルにはデータファイルと同じファイル名を付け、データファイルと同じフォルダ内に保存する必要があります。

ヘッダー情報が利用できない場合は、ファイルタイプとレコード長の入力が求められます。 ソースコードを検査すると、ファイルの種類とレコード長を調べることができます。ファイルタイプとレコード長の識別方法については、『ファイルタイプの識別』および『レコード長の識別』の項を参照してください。

ファイルタイプの識別

行順ファイル、固定長順ファイル、および固定長相対ファイルを識別するには、SELECT 文の ORGANIZATION 指定の入出力節 (Input-Output Section) を調べます。

- 行順ファイルは、ORGANIZATION IS LINE SEQUENTIAL で表されます。
- 固定長順ファイルは、ORGANIZATION IS RECORD SEQUENTIAL で表されます。
- 固定長相対ファイルは、ORGANIZATION IS RELATIVE で表されます。

注：

- 次の条件の 1 つが満たされる場合は行順ファイルになります。
  - ORGANIZATION 指定が ORGANIZATION IS LINE SEQUENTIAL である場合。

- ORGANIZATION 指定が、修飾語 LINE または RECORD のない ORGANIZATION IS SEQUENTIAL の場合。
- すべておよびデフォルトのファイルタイプで ORGANIZAION 指定がない場合。ORGANIZATION SEQUENTIAL として暗黙または明示的に定義されたファイルは LINE になります。

それ以外の場合には、ファイルはレコード順になります。

- ORGANIZATION IS は、付随的な用語なので、プログラムでは SEQUENTIAL、LINE SEQUENTIAL または RELATIVE というキーワードしか記述されない場合があります。 詳細については、Net Express オンラインヘルプを参照してください。([ヘルプ] メニューの [ヘルプトピック] をクリックします。[目次] タブで、[リファレンス]、[COBOL 言語リファレンス]、[COBOL プログラム定義]、[環境部]、[入出力節] をダブルクリックします。)

レコード長の識別

レコード長を確認するためには、ファイル節の FD エントリとして記述されたレコードレイアウトを調べてください。 レコード長は、FD 集団項目を構成する基本データ項目の長さの合計です。

PIC X と PIC 9 の各項目では、1 つの文字位置が 1 バイトを占めます。 計算用フィールド (COMP-1、COMP-2、COMP-3、COMP-4、COMP-5 および COMP-X) の長さの計算方法を確認するには、Net Express オンラインヘルプを参照してください。([ヘルプ] メニューの [ヘルプトピック] をクリックします。[目次] タブで、[リファレンス]、[COBOL 言語リファレンス]、[COBOL 言語の概念]、[文字の表現と基数の選択]、[概要 - 文字の表現と基数の選択] をダブルクリックします。)

データファイルに関する情報の取得

データファイルツールバーにある  をクリックしてから、次のタブのどちらかをクリックして、開いているデータファイルに関する詳細情報を参照します。

- [PC ファイル]

  データファイルに関する情報 (ディスク上の位置、ファイル編成、レコード形式、レコード長など)。

- [一般]

  レコード数、文字集合、ファイルが読み込み専用で開かれたかどうか、バックアップファイルを使用しているかどうかなどの情報。

- [レコードレイアウト]

  データファイルにレコードレイアウトファイルがある場合は、現在のレコードで使用中のレコードレイアウトに関する情報。 このウィンドウの上部には、レコードレイアウトファイルの名前と位置が表示されます。「レイアウト名」の下にあるレコード 1 つをクリックすると、そのタイプのレイアウトに設定されている条件を確認できます。

データファイルのナビゲート

どのタイプのデータファイルについても、行番号とカラム位置 (どちらも 0 から開始) を定義すればファイル内の特定の位置に直接ジャンプできます。 現在のレコードにレコードレイアウトがある場合は、フィールドにも直接ジャンプできます。

特定のレコードまたは文字列を検索できます。詳細については、『データファイルの検索』の項を参照してください。

「データファイルエディタ」ウィンドウをナビゲートする方法については、『「データファイルエディタ」ウィンドウ』の項を参照してください。

順ファイルのナビゲート

COBOL プログラムを使用してファイルを連続して読み込むように、ファイル内を順番に移動できます。

順ファイル以外のファイルのナビゲート

ファイル内で移動するには、次の方法を使用します。

- レコードの追加 (索引ファイルと相対ファイル)
- キーの変更 (索引ファイルのみ)
- キーの検索 (索引ファイルのみ)
- 現在の参照キーの変更 (索引ファイルのみ)

これらのどちらかを使用してファイル内を移動すると、データファイルステータスバーの行番号が 0 にリセットされます。 これは、これらのタイプのファイルの行番号が、特定の位置に関してのみ決定されるからです。

この場合には、ファイル内の移動に合わせて行番号は変わります。

- ファイル内の前方レコードに移動すると、各レコードが移動するたびに行番号が 1 ずつ減り、負の番号として表示されます。
- ファイル内の後方レコードに移動すると、各レコードが移動するたびに行番号が 1 ずつ増えます。

行番号の整合性を維持するために、カラム番号も 0 から開始されます。

索引ファイルでのキーの変更

デフォルトでは、データファイルエディタは、主キーの順番で索引ファイルのレコードを表示します。 ファイルに副キーがある場合には、別の参照キーを選択することにより表示順序を変更できます。 索引ファイルの表示に副キーを選択するには、現在選択されているキーと文字集合を表示するデータファイルツールバーを使用します。

データファイルの編集

データファイルエディタでは、次のことが実行できます。

- レコード内の非フォーマットデータの編集

  「データファイルエディタ」ウィンドウの左側のペインを使用します。詳細については、『左側のペインでのデータ編集』の項を参照してください。

- 個々のフィールド内のフォーマットされたデータの編集

  「データファイルエディタ」ウィンドウの右側のペインを使用します。詳細については、『右側のペインでのデータ編集』の項を参照してください。

- 16 進データの編集

16 進値が表示されている場合は、この値を直接編集できます。 これにより、ファイルにテスト用の無効なデータを入れることができます。

- フィールドおよびレコードの初期化

  詳細については、『データの初期化』の項を参照してください。

- 可変長ファイルのレコード長の変更

  レコード長を短くすると、レコードは新しい長さまで切り捨てられます。 レコード長を長くすると、追加のバイトが現在の埋込みバイトで埋められます。

左側のペインでのデータ編集

非フォーマット表示でデータを挿入または削除すると、すべてのデータがずれる可能性があります。そのため、挿入または削除の後にフィールドがずれるのを防ぐために、「データツールオプション」ダイアログボックスの「常に非フォーマットビューで上書き」フィールドを必ず選択してください。詳細については、『挿入モードまたは上書きモードでの編集』の項を参照してください。

右側のペインで行われた編集内容を確認するには、[表示] メニューの [データツール > フォーマット済ペインの更新] をクリックします。

右側のペインでのデータ編集

右側のペインの数字フィールドは常に上書きモードで編集し、英数字フィールドは挿入モードで編集します。

数字フィールドでは、数字データのみ使用できます。 数字を削除すると、各桁 が 0 にセットされます。

フィールドに空白文字がある場合のみに、新しい文字を英数字フィールドに挿入できます。 たとえば、PIC X(4) フィールドにすでに 4 文字入力されている場合は、これらを削除し、新しいデータに置き換える必要があります。 英数字を削除すると、フィールドの右端に空白文字が挿入されます。

無効なデータを含むフィールドはアスタリスク (*) で表示されます。 そのようなフィールドを選択すると、メッセージボックスが表示され、フィールドを初期化するかどうか確認が行われます。

警告：索引ファイルを編集している場合は、キー値の変更について次の点に注意する必要があります。

- 重複キー値が許可されている場合を除き、1 つのレコードのキーを別のキーと同じ値に変更しないでください。 ファイル内で主キーを重複させることはできません。
- キーを含むレコード領域を編集すると、ファイルのレコードの論理的な位置が変更されている可能性があります。

データの初期化

単一のフィールドまたはレコード内のすべてのフィールドを初期化できます。

- 英数字フィールドを空白文字に初期化
- 数字フィールドを 0 に初期化

条件付きレコードレイアウトを作成した場合は、条件のテスト用に使用されたフィールドは、レコードを初期化しても影響を受けません。(条件付きレコードレイアウトの詳細については、『データファイルでのレコードレイアウトの使用』の章にある『レコードレイアウトの種類』の項を参照。)

ファイルが索引ファイルである場合には、レコードのキーはレコードの初期化により影響を受けることはありません。

データの復元

レコードを選択すると、データファイルエディタでは、レコードに行った編集内容を元に戻せるようにデータが保存されます。編集するレコードが現在のレコードである間は、[編集] メニューの [データツール > レコードの編集を元に戻す] を使用すると、そのレコードの保存されているデータを復元できます。

「データツールオプション」ダイアログボックスで「編集する前にファイルをバックアップする」フィールドを選択している場合は、バックアップからファイル全体を復元できます (『バックアップの取得』の項を参照)。バックアップファイルは、データファイルと同じ基本名で、拡張子が .dbk です。 バックアップを取り込むには、編集ウィンドウを閉じる必要があります。編集ウィンドウを閉じたら、エクスプローラなどのファイル操作ツールを使用して、バックアップ .dbk ファイルを .dat ファイルにコピーできます。

ファイルが索引ファイルである場合は、個々の索引ファイルに索引が格納されます。この索引ファイルは、データファイルと同じ基本名で、拡張子が .idx です。索引のバックアップファイルは、データファイルと同じ基本名で、拡張子が .ibk です。索引とデータを維持するためには、バックアップ .ibk ファイルを .idx ファイルにコピーする必要があります。

レコードの追加と削除

レコードを作成する方法は、順ファイル、相対ファイルおよび索引ファイルでそれぞれ異なります。順ファイルにはキーがなく、相対ファイルと索引ファイルにはそれぞれ別のタイプのキーがあります。VSAM ES ファイルには、特に配慮が必要です。

レコードレイアウトファイルを読み込んでいる場合には、この中にある既存のレコードレイアウトの 1 つを基にして新規レコードを作成するか、または新規レコードに埋込みバイトを書き込むかを選択できます。レコードレイアウトの詳細については、『データファイルでのレコードレイアウトの使用』の章を参照してください。 条件付きレコードレイアウトを作成し、新規レコードをその 1 つとする場合は、条件フィールドが適切に設定されます。

順ファイルへのレコードの追加

現在、選択されているレコードの前後に新しいレコードを挿入できます。

新規レコードは、次に示す方法でも作成できます。

- 既存のレコードを 1 回コピーする。
- 既存のレコードを何回もコピーする。 この方法は、複数反復と呼ばれます。 順ファイルでテストデータを設定する場合は、この方法を使用すると最も迅速に処理できます。

相対ファイルへのレコードの追加

新しいレコードを作成するか、または既存のレコードをコピーして、新しいレコードを相対ファイルに追加できます。

新規レコードを作成するときには、相対レコード番号を指定して、データファイルエディタで新しいレコードをファイルに配置できるようにする必要があります。データファイルステータスバーに表示される行番号は 0 にリセットされます (『順ファイル以外のファイルのナビゲート』の項を参照)。

レコードレイアウトを読み込んでいない場合は、データファイルエディタでは適切な埋め込みバイトを使用してレコードを次の値に設定します。

- 可変長ファイルの場合は、最小レコード長
- 固定長ファイルの場合は、最大レコード長

索引ファイルへのレコードの追加

新しいレコードを作成するか、または既存のレコードをコピーして、新しいレコードを索引ファイルに追加できます。データファイルステータスバーに表示される行番号は 0 にリセットされます (『順ファイル以外のファイルのナビゲート』の項を参照)。

索引ファイルレコードを追加するときは、必ず新しいレコードにすべて一意のキーを指定します。 主キーは常に一意のキーである必要があります。 場合によっては、ファイルに 1 つ以上の一意の副キーがあることもあります。

レコードレイアウトを読み込んでレコードを挿入する場合には、挿入処理中に入力されたすべてのキー情報は、選択したレコードレイアウトに関連付けられた条件付きデータよりも優先されます。レコードレイアウトの詳細については、『データファイルでのレコードレイアウトの使用』の章を参照してください。

VSAM ES ファイルへのレコードの追加

VSAM ES ファイル (ESDS ファイルとも呼ぶ) に新規レコードを作成するには、空のレコードをファイルの末尾に追加する方法しかありません。 ファイルに副索引がある場合には、一意のキー値も指定する必要があります。

新規レコードは物理的に常にファイルの末尾に置かれます。 ファイルに副索引がある場合は、編集ウィンドウの、参照キーで指定されたファイルの位置に空のレコードが表示されます。詳細については、『順ファイル以外のファイルのナビゲート』の項を参照してください。

レコードが追加されたら、編集セッションでフィールド値を指定します。

レコードの削除

VSAM ES ファイルを除き、すべての種類のデータファイルから 1 つ以上のレコードを削除できます。

不適切なレコードがある場合でも、複数削除の手順を実行できます。 削除処理は、ファイルの末尾まで継続されます。ファイルの末尾に到達すると、削除されたレコード数を表すメッセージボックスが表示されます。

注：誤って削除処理をしないように、「データツールオプション」ダイアログボックスの [削除の警告] を必ず選択してください。詳細については、『警告プロンプトをオフに設定』の項を参照してください。

データの切り取り、コピー、および貼り付け

データファイルのデータを切り取り、コピーし、貼り付けることができます。 クリップボードからデータを貼り付けることもできます。 データファイルエディタは、クリップボードの情報を、次の 2 つの形式で保存します。

- バイナリ : レコードデータ内で使用できる全文字集合をサポートします。
- テキスト : テキスト形式のブロックを受け付けるツールで貼り付け操作ができるように、ヌル文字を空白文字に置き換えます。

データファイルエディタでは、次の方法でデータファイルの情報を表示できます。

- レコードの表示
- レコードの 16 進表示
- フィールドの表示
- フィールドの 16 進表示

切り取り、コピー、および貼り付けの操作法は、次のように表示によって異なります。

- レコードの表示 : 切り取り、コピー、または貼り付けを行うレコードをクリックして選択してから右クリックすると、ポップアップメニューが表示されます。そこから該当する操作をクリックできます。
- レコードの 16 進表示およびフィールドの 16 進表示 : 切り取り、コピー、および貼り付け操作をする文字を選択してから右クリックすると、ポップアップメニューが表示されます。そこから該当する操作をクリックできます。
- フィールドの表示 : 編集のためフィールドを開いている場合は、切り取り、コピー、および貼り付け操作をする文字を選択してから右クリックすると、ポップアップメニューが表示されます。そこから該当する操作をクリックできます。 フィールドが COMP 項目の場合は、そのフィールドを表すバイナリ文字がクリップボードにコピーされるため、フィールドの選択のみを行うことができます。

索引ファイルにレコードを貼り付ける場合に、そのファイル内に同じ キーが存在するときには、一意のレコードキーを指定する必要があります。 相対ファイルにレコードを貼り付ける場合に、同じ相対レコード番号が存在するときには、一意の相対レコード番号を指定する必要があります。

データファイルの検索

検索オプションを使用して、データファイルの特定の文字列を検索できます。 データファイル内を前方にも後方にも検索できます。

また、ファイルの中で特定の文字列を検索し、検出した文字列を指定した別の文字列に置き換えることができます。 検索文字列よりも短い置換文字列を入力すると、データファイルエディタでは、空のバイトに埋め込み文字が入力されます。 検索文字列よりも長い置換文字列を入力すると、長い文字列は切り捨てられます。 これは、レコード内のフィールドの位置の一貫性を維持するために行われます。

索引ファイルのみ、現在の参照キーを検索できます。

注：COMPUTATIONAL 形式の数字キーを検索する場合は、16 進値を指定する必要があります。 データファイルエディタは、レコードレイアウトファイルが存在する場合でも、「キーの検索」ダイアログボックスに指定されたデータをフォーマットしません。 これは、キーが、それぞれ異なるフォーマットをもつ可能性がある複数のフィールドにまたがることがあるためです。

データファイルの印刷

左側または右側どちらのペインからでもデータを印刷できます。 また、データをプレビューすることもできます。

左側のペインからのレコードの印刷

左側のビューからレコードを印刷またはプレビューしている場合は、次のデータを印刷できます。

- 現在のレコードまたはすべてのレコード。 一連のレコード (レコード番号は 0 から開始) を選択することもできます。
- 各レコード全体または各レコードの一部。 バイトは 0 から始まります。
- 次のどちらか 1 つ以上の項目
    - ヘッダー
    - フッター
    - ページ番号

- バイト位置を示すルーラー
- レコード番号
- 16 進値

印刷プレビューを表示すると、データファイルステータスバーにページ番号が表示されます。 表示の倍率を変更できます。

警告：使用可能なメモリにより、印刷プレビューを行えるサイズが制限されます。 印刷プレビューメモリを準備している間にメモリが不足すると、エラーメッセージが表示されます。 ファイルの印刷では、そのような制限がありません。

右側のペインからのフィールドデータの印刷

フォーマット表示からの印刷またはプレビューは、非フォーマット表示と同じです。ただし、次の点が異なります。

- フォーマットされたレコードに対してレコードレイアウトファイルを読み込んでいる場合は、フィールドレベルでしかプレビューまたは印刷することができません。詳細については、『データファイルでのレコードレイアウトの使用』の章を参照してください。
- 一致する使用可能なレイアウトがないレコード、または、無効な OCCURS DEPENDING ON RANGE を含むレコードが検出された場合は、出力にエラーメッセージが含まれます。
- 次の項目は選択できません。
  - ルーラー
  - レコード番号
  - 16 進データ
  - レコードの一部

データファイルの編集　　データファイルの変換

第 14 章 : データファイルでのレコードレイアウトの使用

ここでは、データファイルレコードのフォーマットに使用できるレコードレイアウトエディタについて説明します。 また、データファイルエディタを使用して、データファイル内のデータを表示および編集できます。 特に、データの数値を表示できます。データファイルエディタによるデータファイルの編集方法については、『データファイルの編集』の章を参照してください。

レコードレイアウトエディタの使用に関する詳細については、Net Express オンラインヘルプを参照してください。([ヘルプ] メニューの [ヘルプトピック] をクリックします。 [目次] タブで、『Net Express の使用』、『データファイル』、『レコードレイアウトの処理』をダブルクリックします。)

概要

データファイルエディタにデータファイルを最初にロードするときに、右側のペインのデータは、表示や編集できるようにフォーマットされていません。 フォーマットされたデータを表示および編集できるようにするには、レコードレイアウトエディタを使用して、データファイル内にレコードの 1 つ以上のレイアウトを作成します。

レコードレイアウトを作成するには、レコードレイアウトエディタで、デバッグ情報 (.idy) ファイルのデータ部からデータレイアウトを抽出します(このファイルは、プロジェクト内でのデバッグ用に COBOL ソース (.cbl) ファイルをコンパイルして作成します)。 作成するレコードレイアウトには、データレコードの各フィールド名および PICTURE 句を記述します。

テストフィールドに条件を設定して、データファイルにレコードの複数のレコードレイアウトを作成できます。 データファイルエディタでは、テストフィールドを使用して、どのレコードにどのレイアウトを使用するかを決定します。

最初にデータファイルのレコードレイアウトを作成するときに、レコードレイアウト (.str) ファイルにレコードレイアウトを保存します。.str ファイルを編集して、後からより多くのレイアウトを追加できます。また、.str ファイルを開いて、不要になったレコードレイアウトを削除することもできます。

レコードレイアウトファイルは、EBCDIC と ANSI 間で変換を行うときに、データファイルコンバータで、変換するレコードを制御する場合にも使用します。

レコードレイアウトの種類

レコードレイアウトには、次の 2 つの種類があります。

- デフォルトのレコードレイアウト

  データファイルごとにデフォルトのレコードレイアウトを 1 つ定義できます。 デフォルトのレコードレイアウトは、データファイル用に作成した条件付きレコードレイアウトで指定した条件に一致しないデータをフォーマットします。

- 条件付きレコードレイアウト

  デフォルトのレコードレイアウトに加えて、データファイルでは、1 つ以上の条件付きレコードレイアウトを使用できます。 条件付きレコードレイアウトを作成する場合は、1 つ以上のフィールドに適用するテストを定義します。 条件付きレコードレイアウトを使用するには、レコードが、そのレイアウトに対して定義されたすべてのテストを満たす必要があります。

条件付きレコードレイアウトを定義しないと、データファイル内のすべてのレコードが、設定してあるデフォルトのレコードレイアウトを採用します。

レコード内のデータを変更すると、データファイルエディタによって、修正されたレコードで使用するレコードレイアウトが自動的に判断されます。

レコードレイアウトの作成

データファイルのレコードレイアウトを初めて作成する場合は、レコードレイアウトエディタで次の情報を指定する必要があります。

- レイアウト情報を抽出するデータ部を検索する場所

  COBOL プログラムの COBOL ソース (.cbl) ファイルを選択します。レコードレイアウトエディタでは、該当のデバッグ情報 (.idy) ファイルからデータ部をロードします。

  デバッグのためにプログラムをコンパイルすると、(.idy) ファイルを作成できます。 COBOL レイアウトに FILLER 項目が含まれている場合は、レコードレイアウトファイルに FILLER 項目を含ませるために、INCLUDE-FILLER コンパイラ指令でコンパイルする必要があります。INCLUDE-FILLER コンパイラ指令の詳細については、Net Express オンラインヘルプを参照してください。([ヘルプ] メニューの [ヘルプトピック] をクリックします。[目次] タブで、[リファレンス]、[コンパイラ指令] をダブルクリックします。)

- 作成する各レコードレイアウトで使用するデータ部のレイアウト

  デフォルトのレコードレイアウト 1 つと、必要な数の条件付きレコードレイアウトを作成できます。

さらに、条件付きレコードレイアウトの場合に限り、テストフィールドを選択して、テスト条件を指定する必要があります。詳細については、『テスト条件の設定』の項を参照してください。

レイアウト名

レコードレイアウト名は、データ部から取得したレイアウト名と同じです。ただし、デフォルトのレコードレイアウト名の後ろには「DEFAULT」という語が付きます。 たとえば、データ部から CUSTOMER-REC という名前のレイアウトを抽出すると、デフォルトのレイアウトレコードの場合は CUSTOMER-REC-DEFAULT、条件付きレコードレイアウトの場合は CUSTOMER-REC に変換されます。

レコードレイアウトエディタを開く

データファイルのレコードレイアウトを最初に作成する場合は、プロジェクトビューのファイルビューが表示されます。使用するデータ部を含む COBOL ファイルをコンパイルして、「ビルドタイプ」で「一般デバッグビルド」フィールドが選択されているか確認します。これによって、レコードレイアウトエディタでデータ部が抽出されるデバッグ情報 (.idy) ファイルを作成できます。その後、右側のペインの COBOL ファイル上で右クリックして、[レコードレイアウトの生成] を選択します。

レコードレイアウト (.str) ファイル内にデータファイルのレコードレイアウトを保存すると、レコードレイアウトファイルを開いてレコードレイアウトエディタを直接開くことができます。

「レコードレイアウトエディタ」ウィンドウ

レコードレイアウトエディタにはメインウィンドウが 1 つあります。

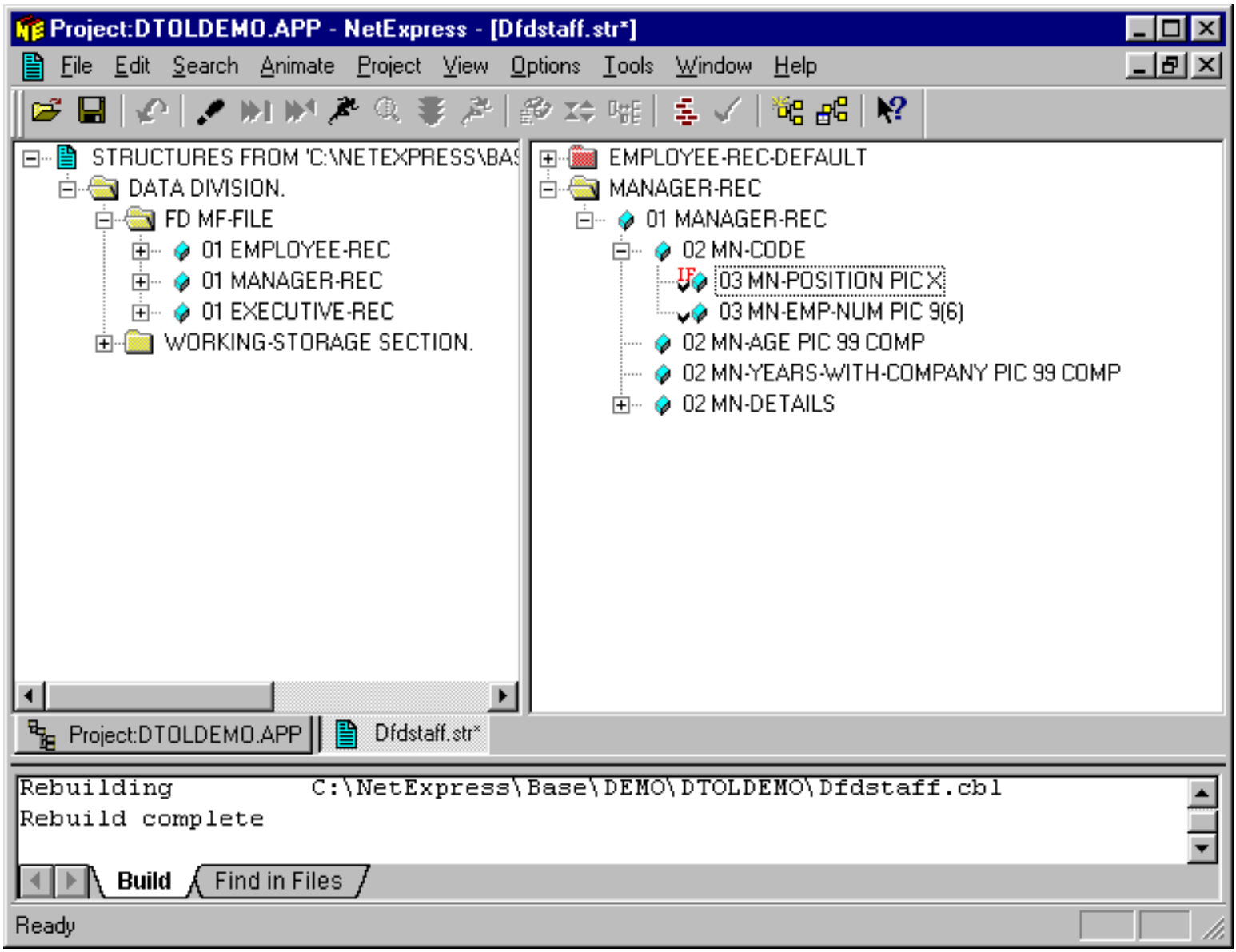

図 14-1： 「レコードレイアウトエディタ」ウィンドウ

「レコードレイアウトエディタ」ウィンドウは、次の 2 つの部分に分かれています。

- 左側のペイン には、選択した COBOL ソース (.cbl) ファイルに関連付けられたデバッグ情報 (.idy) ファイルのデータ部が表示されます。
- 右側のペイン には、レコードレイアウト (.str) ファイルのレイアウトが表示されます。

ヒント：レコードレイアウト (.str) ファイルを編集している場合には、データ部は自動的にロードされません。左側のペインにデータ部をロードするには、「レコードレイアウトエディタ」ウィンドウで右クリックし、ポップアップメニューの [データ部のロード] をクリックします。

左側のペイン

左側のペインには、選択した COBOL ソース (.cbl) ファイルに関連付けられたデバッグ情報 (.idy) ファイルからロードしたデータ部が表示されます。

「レコードレイアウトエディタ」ウィンドウの左側のペインの領域について、上から順番に説明します。

COBOL ヘッダーバー

図 14-2： COBOL ヘッダーバー

COBOL ヘッダーバーには、データ部のロード元の COBOL プログラムの名前が表示されます。

データ部ビュー

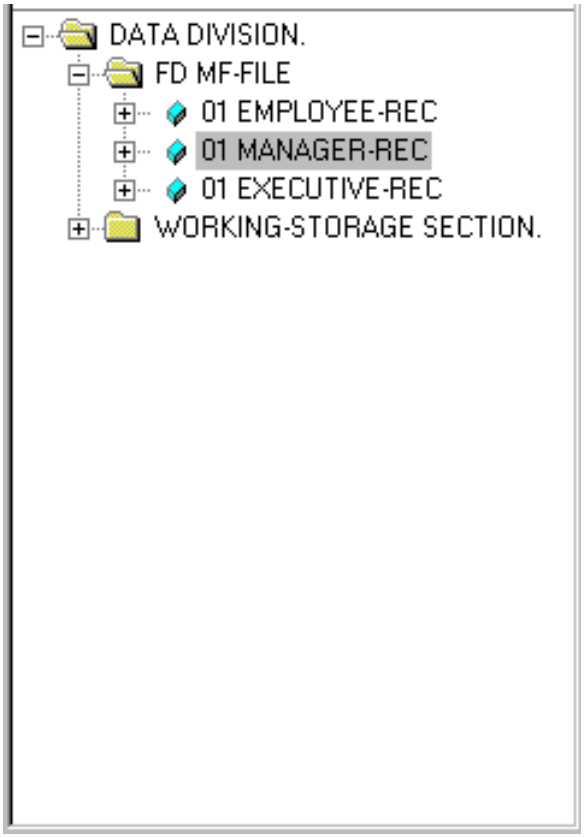

図 14-3： データ部ビュー

データ部ビューには、選択した COBOL プログラムのデータ部が表示されます。

右側のペイン

「レコードレイアウトエディタ」ウィンドウの右側のペインの領域について、上から順番に説明します。

ヘッダー情報バー

図 14-4： ヘッダー情報バー

ヘッダー情報バーには、レコードレイアウトの状態に関する情報が表示されます。このバーの構成は次のとおりです。

- レイアウトステータスアイコン

アイコン	内容

	レコードレイアウトファイルにはレコードレイアウトがあり、問題ありません。
	レコードレイアウトがロードされていません。 そのため、レコードレイアウトビューは空です。
	条件付きレコードレイアウトにテストフィールドがありません。

- レコードレイアウト数。レイアウトがない場合は、この項目には「レイアウトなし」と表示されます。
- レコードレイアウトビューで強調表示されているレコードレイアウトに関連付けられている条件数。条件がない場合は、この項目には「条件なし」と表示されます。強調表示されたレコードレイアウトがデフォルトのレコードレイアウトである場合は、この項目には「デフォルト」と表示されます。
- EBCDIC/ANSI インジケータ。

ヘッダー情報バーはオンとオフを切り替えることができます。

レコードレイアウトビュー

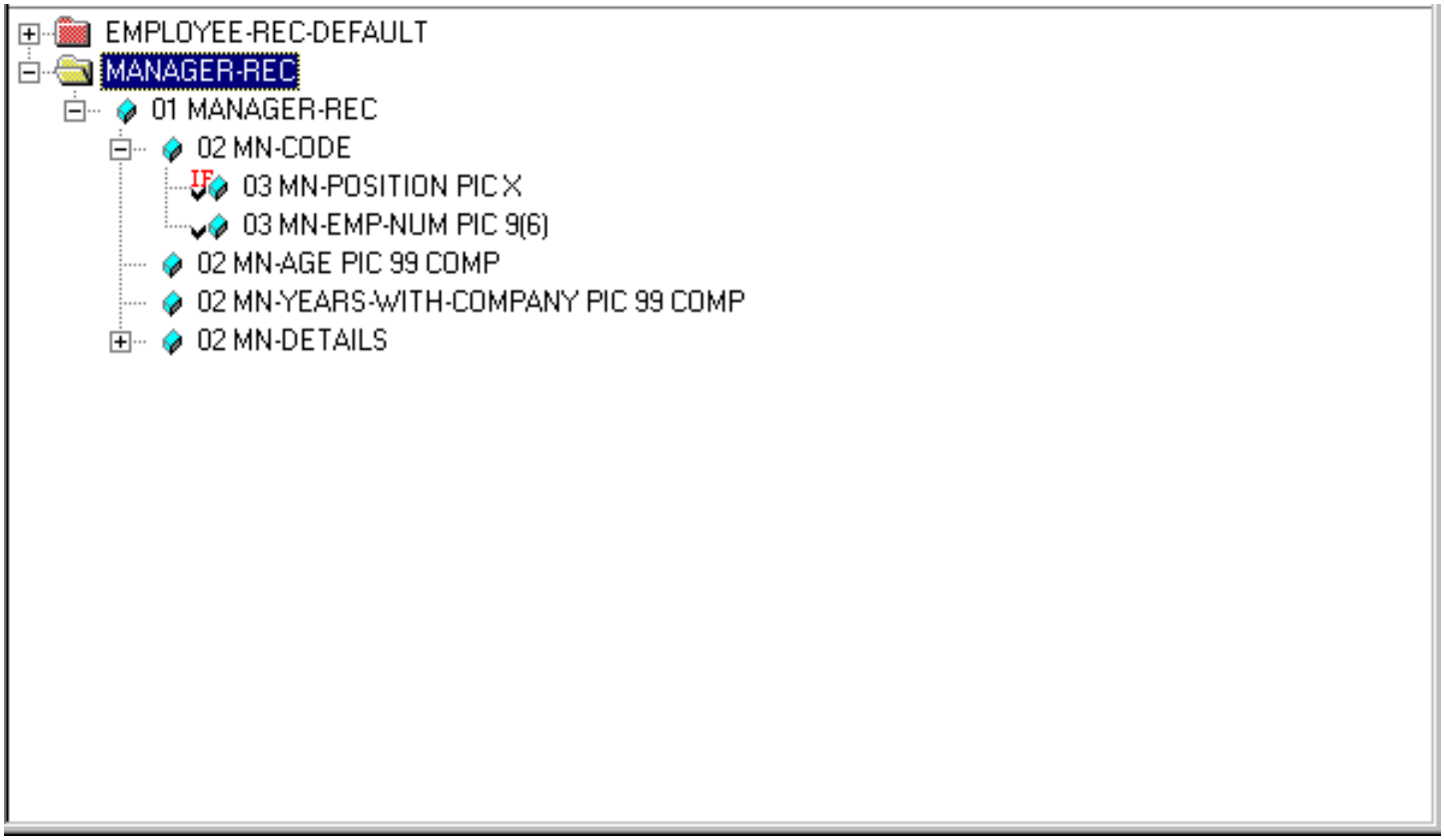

図 14-5： レコードレイアウトビュー

レコードレイアウトビューには、現在定義されているレコードレイアウトが一覧表示されます。

レイアウトアイコン	内容
	デフォルトのレコードレイアウト
	条件付きレコードレイアウト
	レイアウトの一部となるフィールド
IF	条件付きレコードレイアウトのテストフィールド

レコードレイアウトまたはグループ項目を展開するには、⊞ をクリックします。レコードレイアウトまたはグループ項目を圧縮するには、⊟ をクリックします。

レイアウトまたはフィールドを強調表示するには、レイアウトまたはフィールド上でクリックします。 ヘッダー情報バーが更新され、強調表示されたレイアウトに条件が設定されている場合は条件数が表示されます。

ヒント：

- レコードレイアウトの状態を表すツールチップを表示する方法

  レコードレイアウトビュー内のレイアウトをクリックし、ヘッダー情報バー内にある ✔ アイコンの上にカーソルを置きます。

- 条件付きレイアウトに対して定義されたすべての条件を一覧表示するツールチップを表示する方法

  レコードレイアウトビューのレコードレイアウトの隣にある 🗀 アイコンの上にカーソルを置きます。

- レコードレイアウト内のテストフィールドに設定された条件を表示するツールチップを表示する方法

  レコードレイアウトビューのレコードレイアウトのテストフィールドの隣にある IF アイコンの上にカーソルを置きます。

テスト条件の設定

条件付きレコードレイアウトごとに、レコードがそのレイアウトを採用するために満たさなければならない 1 つ以上の条件を指定する必要があります。 各条件はテストフィールドに関連付けられます。 「フィールドの属性」ダイアログボックスを使用して、条件を指定します。

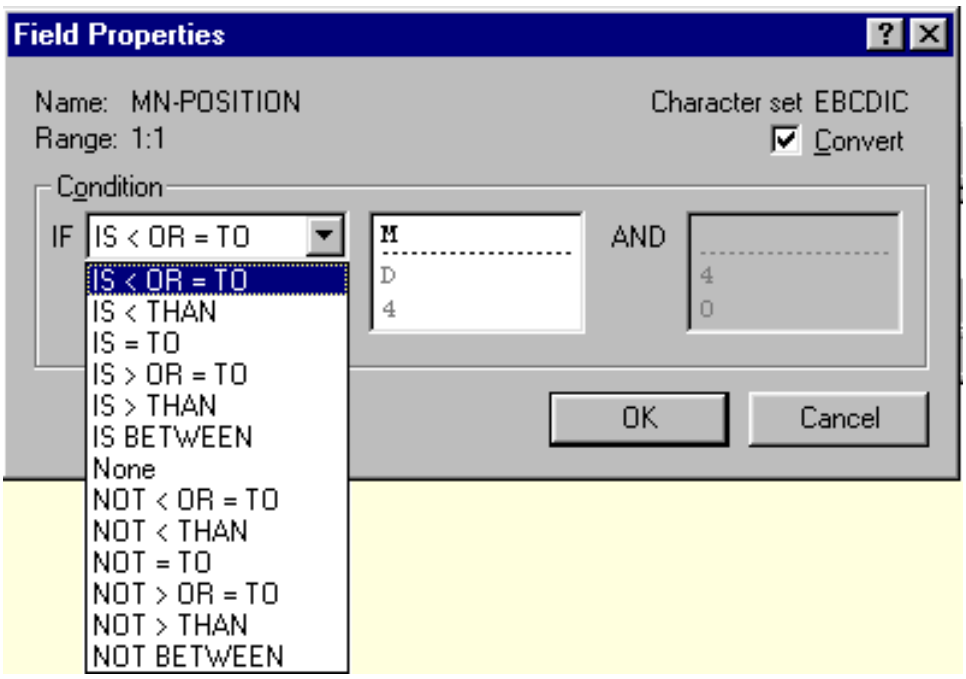

図 14-6： 「フィールドの属性」ダイアログボックスでの条件の指定

「レコードレイアウトエディタ」ウィンドウで、フィールド上で右クリックし、[属性] を選択すると、「フィールドの属性」ダイアログボックスが開きます。

条件を選択すると、右側の値ボックスの 1 つまたは両方が使用可能になります。 たとえば、条件「IS = TO」の場合は値を 1 つのみ入力する必要があります。一方、条件「IS BETWEEN」では 2 つの値を入力する必要があります。

「レコードレイアウトエディタ」ウィンドウには、各テストフィールドに対して IF が表示されます。

条件付きレコードレイアウトの作成方法の詳細については、Ne Express オンラインヘルプを参照してください。([ヘルプ] メニューの [ヘルプトピック] をクリックします。 [目次] タブで、[Net Express の使用]、[データファイル]、[レコードレイアウトの処理]、[レコードレイアウトの作成と削除] をダブルクリックします。)

EBCDIC と ANSI 間の変換

「フィールドの属性」ダイアログボックス (『テスト条件の設定』の項の図 12-6 を参照) には、EBCDIC から ANSI への変換、または ANSI から EBCDIC への変換中にフィールドを変換するかどうかを指定する「変換」チェックボックスがあります。デフォルトでは、「変換」はオンになっていますが、次のフィールドの場合はオフになります。

- 計算用のフィールド
- 再定義されたフィールド

フィールドを右クリックし、ポップアップメニューで [属性] をクリックすると、レコードレイアウト内のどのフィールドについても「変換」設定を変更できます。

警告：再定義されているフィールドに対して「変換」フィールドをオンに戻した場合は、そのフィールドを最初に定義するときに必ずオフにしてください。

EBCDIC 変換とデータファイルコンバータの詳細については、『データファイルの変換』の章を参照してください。

レコードレイアウトファイル

レコードレイアウト (.str) ファイルに 1 つのデータファイルのすべてのレコードレイアウトを保存する必要があります。

レコードレイアウトの保存

レコードレイアウトエディタを終了する前に、レコードレイアウト (.str) ファイルにレコードレイアウト情報を保存する必要があります。 データファイルエディタでデータファイルを開こうとすると、データファイルと同じディレクトリにある同じ基本名のレコードレイアウトファイルが検索されます。

データファイルを開くたびにレコードレイアウトが自動的にロードされるようにするには、レコードレイアウトファイルを保存するときに次のガイドラインに従ってください。

レコードレイアウトファイル	ガイドライン
ファイル基本名	データファイルと同じファイル基本名
ファイル拡張子	**.str**
フォルダ	データファイルと同じフォルダ

たとえば、データファイルの名前が customers.dat で、c:\data フォルダ内に保存されている場合は、レコードレイアウトファイルを c:\data\customers.str として保存する必要があります。

レコードレイアウトファイルの編集

レコードレイアウトファイル (.str) は、データファイルが閉じられている状態であればいつでも開くことができます。 「レコードレイアウトエディタ」ウィンドウが開かれ、すでに作成されているレコードレイアウトが表示されます。 レコードレイアウトエディタでは、次のことが実行できます。

- レコードレイアウトの追加
- 既存のレコードレイアウトの削除
- 条件付きレコードレイアウトに設定されているテスト条件の変更
- テスト条件の追加と既存テスト条件の削除

レコードレイアウトの取得

データファイルにレコードレイアウトファイルをロードすると、「データファイルエディタ」ウィンドウの右側のペインにフィールドレベルでフォーマットされたレコードデータが表示されます。この例については、『データファイルの編集』の章の『「データファイルエディタ」ウィンドウ』の項を参照してください。

レコードレイアウトファイルは、データファイルに自動的にロードされます。詳細については、『レコードレイアウトの保存』の項を参照してください。

自動的にロードしない場合は、[ファイル] メニューの [レコードレイアウトのロード] オプションを選択して、レコードレイアウトファイルを個別にロードできます。

データファイルでのレコードレイアウトの使用　　リビルド

第 15 章 : データファイルの変換

ここでは、PC でデータファイルを変換する際に使用する、Net Express のグラフィックなデータファイルコンバータについて説明します。

データファイルコンバータの使用に関する詳細については、Net Express オンラインヘルプを参照してください。([ヘルプ] メニューの [ヘルプトピック] をクリックします。[目次] タブで、[開発環境]、[データファイルの処理]、[データファイルの変換]をダブルクリックします。)

概要

データファイルコンバータを使用して、データファイルのファイル編成や形式を変換できます。 データファイルコンバータでは、次の操作を行えます。

- データの再構築

  ある形式から別の形式にファイルを変換すると、データを再構築できます。 たとえば、データファイルを順編成から索引編成に変換したり、追加の二次索引を追加したりできます。

- EBCDIC と ANSI 間の変換

  データファイルコンバータでは、EBCDIC と ANSI の間で簡単にデータ変換を行えるので、作成済みのデータを再作成したり、特別な変換プログラムを作成する必要がありません。 また、内部浮動小数点形式を S/370 形式と IEEE 形式間で変換することもできます。

- レコード長の変更

  可変長以外のレコードの場合は、新しいレコード長を指定できます。 可変長レコードの場合は、最小長と最大長を変更できます。

- PC にダウンロードしたメインフレームファイルの変換

  次の変換が可能です。

  - メインフレームから PC にダウンロードされたファイルの変換
  - 可変長 VRECGEN 形式から PC データファイル形式への変換
  - PC データファイル形式から、メインフレームに転送するための VRECGEN2 形式への変換
  - メインフレームの固定長レポート形式から PC 印刷形式への変換
  - PC 印刷形式からメインフレームの固定長レポート形式への変換

注：Btrieve ファイルを変換する必要がある場合は、リビルドのコマンド行バージョンをかわりに使用してください。 詳細については、Net Express オンラインヘルプを参照してください。([ヘルプ] メニューの [ヘルプトピック] をクリックします。[目次] タブで、[リファレンス]、[コマンド行リファレンス]、[索引ファイルのリビルド] をダブルクリックします。)

データファイルコンバータを開く

データファイルコンバータを開くには、[ツール] メニューの [データツール > 変換] をクリックします。 「データファイルの変換」ダイアログボックスが開きます。

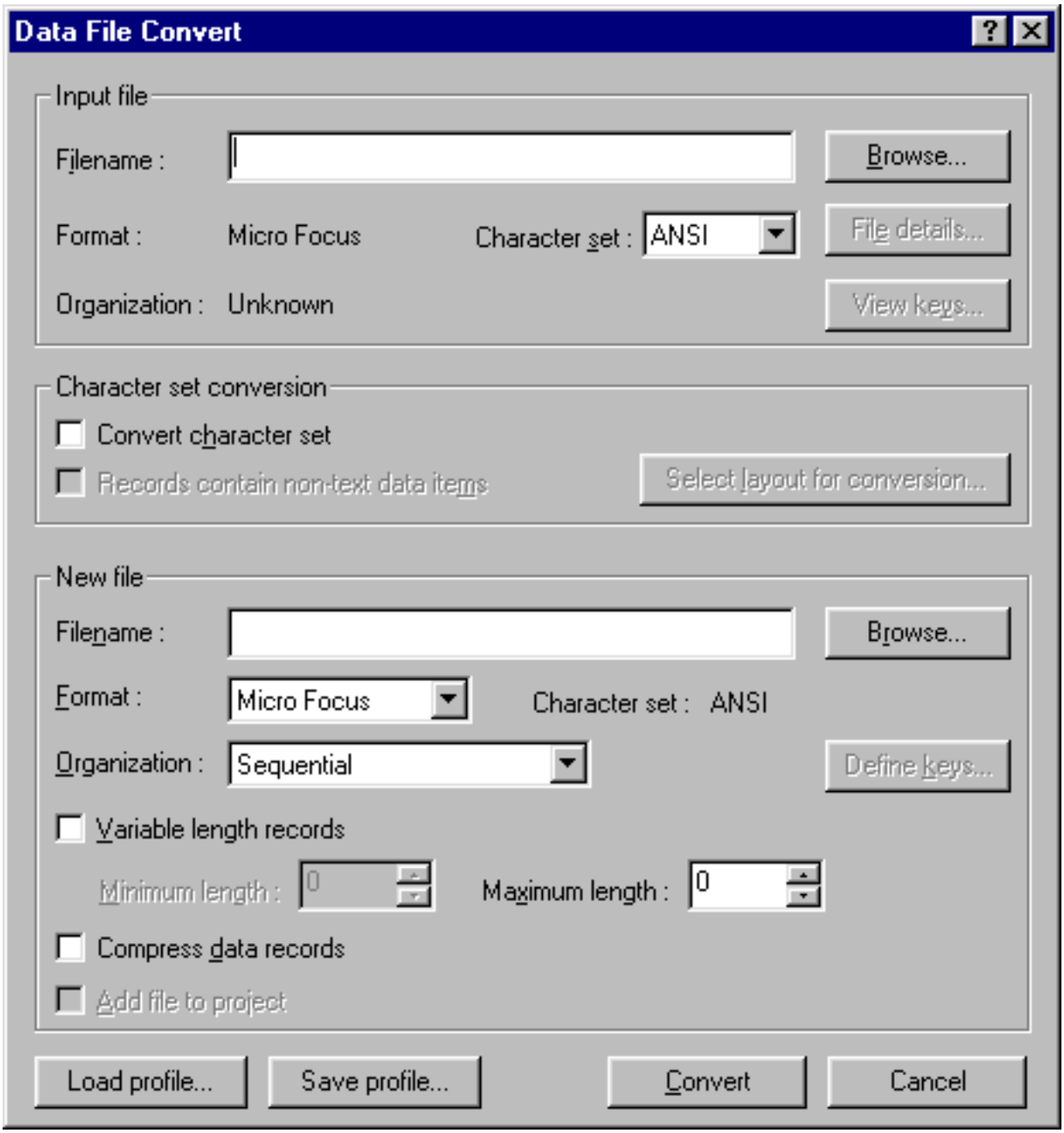

図 15-1: 「データファイルの変換」ダイアログボックス

入力ファイルの指定

最初に、入力ファイルを選択します。 Net Express で、入力ファイルの種類とレコード長を

識別できるようにする必要があります。この情報はデータファイルのヘッダーかプロファイル (.pro) ファイルに保存されます。詳細については、『データファイルの編集』の章にある『ファイルヘッダー』を参照してください。

ファイルにファイルヘッダーがない場合は、ファイル編成とレコード長を指定する必要があります。 次のファイル編成の中から 1 つを選択できます。

- ANSI メインフレームレポート
- 行順ファイル
- マシンメインフレームレポート
- PC 印刷
- 相対 - 固定長
- 順編成 - 固定長
- VRECGEN

出力ファイルの指定

出力ファイルの名前を指定する必要があります。 データファイルコンバータにより、出力ファイルに対してデフォルトの編成とレコード形式が選択されている場合は、これらの値を変更できます。 データファイルコンバータにより出力ファイルの詳細が表示されていない場合は、これらの情報を指定する必要があります。

次のファイル形式の中から 1 つを選択できます。

ファイル形式	目的
C-ISAM	古いシステムのファイルをエミュレートします。
ESDS	VSAM ES ファイルをエミュレートします。
IDXFORMAT(4)	多くの重複キーを持つ入力ファイルを変換する場合に使用します。
IDXFORMAT(8)	大容量のファイルで使用します。
Level-II	古いシステムのファイルをエミュレートします。
Micro Focus	その他すべての場合に使用します。

選択できる使用可能なファイル編成は、次のとおりです。

- 索引順編成
- 行順編成
- 相対編成
- 順編成
- VRECGEN2

出力ファイルに索引編成を指定する場合は、新しいファイルのキーも定義する必要があります。詳細については、『データファイルの編集』の章にある『索引ファイルのキーの定義』を参照してください。

デフォルトでは、データファイルコンバータによって新しいファイルが現在のプロジェクトに追加されます。新しいファイルを現在のプロジェクトに追加しない場合は、「プロジェクトへファイルを追加」をオフにします。

注：順ファイルを一意のキーを持つ索引ファイルに変換しようとしたときに、一部のレコードに重複するキー値が含まれていると、データファイルコンバータはこれらのレコードを無視し、変換しません。 変換処理の最後に、データファイルコンバータは処理しなかったレコードの数を示します。

レコード長の変更

新しいファイルの最小レコード長と最大レコード長を増減することも可能です。 増減する場合は、次のようになります。

- 最小レコード長を長くした場合に、新しいファイルの最小長よりも短いソースファイルレコードがあると、新しいファイルのレコードには、新しいファイルの最小長に達するまで適切な埋め込みバイトが書き込まれます。
- 最大レコード長を長くする場合は、新しいファイルのレコードに適切な埋め込みバイトが書き込まれます。
- 最大レコード長を短くする場合は、新しいファイルのレコードでは新しい最大長に合わせて切り捨てられます。

EBCDIC と ANSI 間の変換

EBCDIC と ANSI 間で変換を行う場合は、レコードレイアウトを使用して、変換するフィールドを指定できます。レコードレイアウトを作成するときに、「プロパティ」ダイアログボックスで「変換」フィールドの設定を変更できます。デフォルトでは、次のタイプのフィールドを除きすべてのフィールドで「変換」フィールドをオンにできます。

- 計算用のフィールド
- 再定義されたフィールド

1 つ以上のレコードについて個々のフィールドの「変換」フィールドの設定を変更できます。詳細については、『データファイルでのレコードレイアウトの使用』の章の、特に『EBCDIC と ANSI 間の変換』を参照してください。

EBCDIC と ASCII 間の変換

一般的に ASCII と EBCDIC 間で変更を行う場合は、GUI ファイルコンバータ (DFConv) に対する呼び出し可能なインターフェイスを使用する必要があります。DFCONV およびコマンド行インターフェイスの詳細については、『ファイル変換ユーティリティ』の章を参照してください。

Micro Focus COBOL プログラムには、ASCII と EBCDIC 文字列間の変換を高速で行うことができる機能も用意されています。 これは、「_CODESET」と呼ばれます。

_CODESET の形式は次のとおりです。

```
call "_codeset" using function-code, length, string
```

各パラメータの内容は次のとおりです。

function-code	PIC 9(2) COMP-X • 0=convert EBCDIC から ASCII • 1=convert ASCII から EBCDIC
length	PIC 9(9) COMP-X
string	PIC X(*n*)

「_codeset」 は、メインフレーム COBOL コンパイラの 1 つの予約語指令でコンパイルされたプログラム (OS/VS COBOL の OSVS や VS COBOL II の VSC2 など) では機能しません。「_codeset」では、メインフレームでサポートされていない COMP-X データ型を使用しているからです。

PC へのメインフレームファイルの転送

メインフレームファイルを PC に転送して、PC 形式のファイルとして処理できます。 PC へは、メインフレームファイルをバイナリ形式で転送する必要があります。メインフレームファイルが可変長の場合は、『可変長ファイル』の項を参照してください。

可変長ファイル

変換前のファイルが可変長ファイルである場合は、システム管理者に、メインフレームのプログラム VRECGEN を使用して、メインフレームで初期再フォーマットするように依頼する必要があります。 再フォーマットされたファイルは、データファイルコンバータを使用して VRECGEN 形式から変換できます。

メインフレームにファイルをアップロードしたい場合は、データファイルコンバータを使用して、ファイルを VRECGEN2 形式に変換します。 その後、システム管理者にメインフレームプログラム VRECGEN2 を使用して、メインフレームのデータを再フォーマットするように依頼する必要があります。

VRECGEN、VRECGEN2 およびコマンド行インターフェイスの詳細については、『ファイル変換ユーティリティ』の章を参照してください。

メインフレームレポートファイル

メインフレームで生成されて、PC にダウンロードするか、または PC で実行されるメインフレームアプリケーションで作成されたレポートファイルを変換しなければならない場合があります。

メインフレームレポートファイルは、PC 印刷ファイル以外に変換することができません。同様に、PC 印刷ファイルは、メインフレームレポートファイル以外に変換することができません。 メインフレームレポートファイルは、EBCDIC 形式のコードファイルでなければならず、PC 印刷ファイルは ANSI 形式のコードファイルでなければなりません。

メインフレームレポートファイルには、キャッリッジ制御文字が含まれます。データファイルコンバータは、キャッリッジ制御文字を PC で印字可能な形式に変換します。 出力ファイ

ルがメインフレームファイルである場合は、レコード長にはキャッリッジ制御文字も含まれます。 出力ファイルが PC 印刷ファイルである場合は、レコード長はデータ長のみになります。

実際には PC で作成された入力ファイルをメインフレーム出力書式ファイルとして指定した場合は、エラーメッセージ 139(「入力ファイルは、"本物の" メインフレームレポート形式ではありません。」) が表示されます。 このようなエラーを防ぐには、通常のメインフレームダイアレクトを使用して EBCDIC レポートファイルを作成します。

データファイルの変換　ファイル変換ユーティリティ

第 16 章 : リビルド

概要

リビルドは、コマンドプロンプトから呼び出すことができるファイル管理ユーティリティです。これを使用して次のことを行うことができます。

- 索引ファイルの再編成
- 破損した索引ファイルのリビルド
- ある形式から別の形式へのファイル編成の変換
- 索引ファイルの検証

リビルドがファイルに対して実行する操作は、リビルドのコマンド行で指定するオプションによって異なります。

注：

- Fileshare を使用してファイルにアクセスする場合には、Fileshare をサポートしているリビルドオプションのみを使用します。 サポートしていないオプションを実行すると、未定義の結果を受け取ることになります。 詳細については、『Fileshare でのリビルド』を参照してください。
- ストライプ化された IDXFORMAT "8" ファイルを処理する場合は、次のことを確認してください。
    - すべてのストライプが存在すること。
    - リビルドでファイルハンドラ構成ファイル extfh.cfg が選択されていること。

  これを行っていないと、予期しない結果になる場合があります。
- ファイルの再編成、リビルド、または変換を行う場合は、必ず事前にファイルのバックアップコピーを作成してください。この作業は、コマンド行を誤って指定したためにファイルが破損してしまう場合に備えて行います。索引ファイルの場合、ファイルの索引部分とデータ部分の両方を保存しておくことが必要です。
- キーオフセットは 1 から始まります。
- 一時ファイルを保持しておくディレクトリを TMPDIR で指定した場合を除いて、すべての一時ファイルはシステムの一時ファイルディレクトリに作成されます。

Net Express では、IDE に ファイル索引の修正ツールを用意しています。リビルドの代わりに、これを使用して破損した索引ファイルを修正することができます。詳細については、ヘルプトピック
『ファイル索引の修正 - 概要』を参照してください。

コマンド行

リビルドのコマンド行の構文は、次のとおりです。

```
rebuild in-file [,out-file] [options]
```

in-file と out-file に指定するファイル名には、どちらも拡張子が必要です。out-file と in-file の両方に、同じファイル名を指定することはできません。in-file や out-file の代わりに環境変数を使用してファイル名マッピングを実行することができます。

リビルドが実行する操作内容は、コマンド行で指定するオプションにより定義されます。

オプションファイル

リビルドのコマンド行で at 文字 (@) の次にファイル名を記述すると、そのファイルにリビルドコマンド行オプションが含まれていることを示します。

ファイルを使用してリビルドコマンド行オプションを用意する場合、ファイルは ASCII テキストファイルでなければなりません。

リビルドオプションファイルは、次の形式にします。

- オプションは空白文字で区切る。
- 行の終わりを空白文字として扱う。
- アスタリスク (*) からその行の終わりまでをコメントとして扱う。

次に例を示します。

```
rebuild infile.dat, outfile.dat @conv.par
```

conv.par は、リビルドオプションを含む ASCII テキストファイルです。

情報のリダイレクト

リビルドユーティリティで表示されるすべての情報は、標準的なオペレーティングシステムのリダイレクトを使用して、テキストファイルにリダイレクトすることができます。

`>`　　新規ファイルを作成します。

`>>`　　既存のファイルを拡張します。

次に例を示します。

```
rebuild oldms001.dat,ms001.dat -k:1+20<N> -i >> rebuild.prt
```

この例では、リビルドからの出力を既存のファイル rebuild.prt にリダイレクトします。

警告:出力をリダイレクトする場合、-v オプションを使用しないでください。

リビルドオプション

リビルドにより実行される処理は、コマンド行で指定したオプションで定義されます。

UNIX システムの場合、リビルドオプションには、先頭にスラッシュ (/) 文字ではなくハイフン (-) 文字を付ける必要があります。

また、次のオプションも使用できます。

-c	ファイル圧縮を指定します。
-d	索引ファイルのデータ領域から索引を再構築します。
-e	不正な重複キーを報告し、処理を続行します。
-f	索引ファイルを検証します。
-i	処理しているファイルの情報を表示します。
-k	索引ファイルのキー構造を定義します。
-m	将来のために予約されています。
-n	ファイルに関する情報を表示します (他の処理は実行されません)。
-o	入出力ファイルの構成を指定します。
-p	IDXFORMAT "8" ファイルをかわりにリビルドします。
-q	将来のために予約されています。
-r	入出力ファイルのレコード構造を定義します。
-s	入力ファイルの形式を指定します。
-t	出力ファイルの形式を指定します。
-u	前回更新に失敗したファイルの回復を試みます。
-v	ファイルの処理に伴って増加するレコードカウントを表示します。
-y	ファイルサイズの矛盾を無視してリビルドを強制実行します。
-x	索引ファイルの再編成時にデータを書き込む順序を指定します。
-z	ファイルアクセスに Fileshare サーバを使用するようにリビルドに指示します。
--q	リビルドのバナーの表示を抑制します。
--v	リビルドのバージョン番号を表示します。

オプションはコマンド行の任意の場所に指定できます。ただし、二重ハイフン (--) で始まるオプションは、一重ハイフン (-) で始まるオプションの前に配置する必要があります。オプションの形式には特に制約はありませんが、各オプションは空白文字で区切る必要があります。

-f または -v オプションのパラメータの前には、コロン (:) を使用する必要があります。 他のオプションの場合は、パラメータの前にコロンを置くかどうかは任意です。オプションのパ

ラメータ間には空白文字は挿入しないでください。

オプションの詳細については、ヘルプトピック
『REBUILD コマンド行オプションの概要』を参照してください。

索引ファイルの再編成

ファイルを再編成するためのコマンド行の形式は次のとおりです。

```
rebuild in-file,out-file [-c]  [-d] [-i]
    [-k] [-n] [-p] [-s] [-v] [-x]
```

索引ファイルは、レコードの追加、削除、変更などにより更新されるときに、索引構造体とデータ構造体に切り離されるため、処理効率が悪くなります。さらに、削除されたレコードに残された領域は常に再使用されるわけではないので、ファイルが必要以上に大きくなってしまいます。

多くの変更を行った場合、索引ファイルを再編成して、データと索引を順にリビルドし、空き領域を再利用できるようにすると便利です。この作業により、最適な性能とデータの完全性が保証されます。

ファイルを再編成する利点は他にもあります。たとえば、ファイルを順次処理する場合、ファイルを処理するためのアクセス時間は時間が経つにつれて増えます。これは、レコードが更新され、順序が変更されるためです。この状況でファイルを再編成すると、アクセス時間を短縮することができます。

3 つ目の理由は 重複キーチェインをリビルドすることです。 重複キーには重複オカレンスの数が追加され一意に識別されます。 重複キーが新たに追加されるとそのキーに、最後に追加されたキーのオカレンス値を 1 つインクリメントした数値が使用されます。 キーが削除されても、残りのキーのオカレンス値は変更されません。つまり、一部のキーをすでに削除した場合は、重複値の最大回数に到達する可能性があります。 このような状況では、ファイルを再編成することが得策です。 重複キーの詳細については、『ファイル編成』の章の『重複キー』の項を参照してください。

次に例を示します。

```
rebuild infile.dat,outfile.dat
```

この場合、リビルドでは索引ファイル (infile.idx) を使用してデータファイル (infile.dat) を読み込み、出力ファイル (outfile.dat) が作成されます。infile.dat の削除されたレコードは、出力ファイルに書き込まれません。

注：入力ファイルと出力ファイルに同じファイル名を使用することはできません。

破損した索引ファイルのリビルド

索引をリビルドするためのコマンド行の形式は次のとおりです。

```
rebuild in-file[,out-file] [-c] [-d] [-e]
    [-i] [-k] [-n] [-p] [-s] [-v]
```

out-file パラメータは指定を省略できますが、指定することをお奨めします。指定せずにリビルド操作が失敗した場合、元のファイルが失われるからです。

索引ファイルの破損理由はいくつかあります。次に例を示します。

- 索引ファイルを閉じる操作が失敗した。
- 索引ファイルが開いている間にパワーサージが発生した。
- 索引ファイルが開いている間にマシンがリブートされたか、電源がオフになった。

COBOL プログラムが破損した索引ファイルを開くときに、ランタイムシステムがファイルの破損を検出すると、拡張ファイル状態コードを返します。

リビルドを使用すると、破損したファイルを自動的にリビルドすることができます。

- ファイルのデータ部分が完全で、索引部分だけが破損している場合、リビルドはデータ ファイルから完全なデータを持つ新しい索引 (.idx) ファイルを作成することができます。
- ファイルのデータ部分が破損している場合、リビルドは、新しい索引ファイル (データ部分と索引部分の両方) をリビルドすることができます。ただし、リビルドは検出したデータレコードの破損部分を廃棄するため、データの損失が起こります。

注： リビルドを使用して既存のデータファイルから新しい索引ファイルを生成すると、ファイルは大きくなることがあります。 理由の 1 つは、元の索引ファイルに保持されているデータファイルの空き領域に関する情報が失われることです。そのため、空き領域は再使用できなくなります。 もう 1 つの理由は、-d オプションを指定した場合、索引レコードが通常の入出力操作で個別に作成され、部分的にのみ使用された索引ノードができることになるからです。この問題を解決するには、索引をリビルドした後でファイルを再編成する必要があります。

次の例では、リビルドは .idx ファイルを読み込んでキー情報を取得し、データファイル infile.dat を読み込んで新しい索引ファイルを作成します。

```
rebuild infile.dat
```

次の例では、リビルドは入力ファイルを読み込んでキー情報を取得し、物理的な順序でデータを処理し重複レコードを飛ばして、新しい索引ファイルを作成します。

```
rebuild infile.dat, outfile.dat -d -e
```

索引ファイルがない場合には、-k オプションを使用してキー構造体に関する情報を指定する必要があります。

ファイルの変換

ファイルを変換するためのコマンド行の形式は、次のとおりです。

```
rebuild in-file,out-file [-c] [-i]
[-k] [-o] [-r] [-s] [-t] [-v]
```

REBUILDを指定することによって、ファイル構成と形式を変換できます。

次の例では、リビルドは C-ISAM ファイルをシステムで使用される形式に変換し、ファイルのデータ部分を圧縮します。

```
rebuild infile.dat,outfile.dat -s:c-isam -c:d1 -t:mf
```

索引ファイルの検証

索引ファイルを検証するためのコマンド行の形式は次のとおりです。

```
rebuild in-file -f
```

リビルドでは、多くの検証を実行できます。また、すべての検証を実行するか、サブセットだけを実行するかを選択することができます。確認する内容が多いほど、検証処理にかかる時間は長くなります。

次の例では、リビルドはファイルの完全性をすべて検査します。

```
rebuild test.dat -f
```

Fileshare でのリビルド

Fileshare サーバに存在するファイルにリビルドを実行できますが、使用できるのは限られた範囲のリビルドオプションだけです。Fileshare でリビルドを実行するためのコマンド行の形式は、次のとおりです。

```
rebuild in-file[,out-file] -z:server-name {/l|/n}
```

説明

server-name	Fileshare サーバの名前。 *server-name* のすべてのオカレンスが同一の Fileshare サーバを参照する必要があります。

-z リビルドオプションを使用せずに、Fileshare サーバを指定することもできます。 コマンド行の形式は、次のとおりです。

```
rebuild $$server-name¥in-file [$$server-name¥out-file] {/l|/n}
```

UNIX システムでは、server-name のプレフィックス **$$** の代わりに **¥$¥$** を使用します。

呼び出し可能リビルド

次のように指定すると、COBOL プログラムでリビルドを呼び出せます。

```
call "callrb" using commands status
```

パラメータの内容は、次のとおりです。

commands	リビルドコマンド行を格納する PIC X(600) 項目。リビルドはオフセット 599 から 0 へ向かってコマンド行を調べるため、この項目の長さは 600 バイトになります。
status	返されたファイル状態を格納する PIC XX の COMP X 項目。この項目は、リビルドの呼び出し結果を示します。

リビルドをプログラムで呼び出すと、-v オプションを指定しない限り、通常のメッセージは表示されません。-v オプションを設定すると、処理中のレコードの合計が表示されます。

エラーが発生した場合、またはリビルドが成功しなかった場合、RETURN-CODE にはゼロ以外の値が格納され、status には返されたファイル状態が格納されます。 リビルドを呼び出した後は、常に RETURN-CODE と status を確認する必要があります。

RETURN-CODE に格納される値を次に示します。

値	説明
0	リビルドが正常に実行されたことを示します。
1	ファイルが見つからない、ファイル形式が無効である、などの入力ファイルにエラーが発生したことを示しています。状態パラメータを確認してください。
2	出力ファイルにエラーが発生したことを示します。状態パラメータを確認してください。
4	ファイルが破損しています。
9	オプションが無効である、オプションの組合わせが無効である、などのエラーがパラメータリストに含まれていることを示します。

エラーが発生した場合は、同一のオプションを使用して同じファイルのコマンド行からリビルドを実行します。画面出力では、エラー原因についてより詳細な情報が表示されます。

注： -f オプション (索引ファイルの検証) を使用している場合には、ゼロ以外の RETURN-CODE はファイルが破損していることを示します。その場合は、より詳細な情報を取得するために、同一のオプションで同じファイルのコマンド行から再度リビルドを実行してください。

次の例は、COBOL プログラムによるリビルドの呼び出しを示します。

```
01 parameters   pic x(600).
01 status       pic xx comp-x.
  ...
    move "infile.dat,outfile.dat -s:lii -c:d1" to parameters
    call "callrb" using parameters,status
    end-call
```

リビルドメッセージ

リビルドは、索引ファイルのリビルド、変換、再編成を行うときに、エラー、情報、または警告メッセージを出力する場合があります。詳細については、ヘルプトピック『エラーメッセージ』を参照してください。

リビルド例

ここでは、リビルドを使用した例を説明します。

```
rebuild infile.dat,outfile.dat -i
```

ファイルを再編成します。データは主レコードキーの順序で書き込まれ、再編成の完了時に処理情報が表示されます。

```
rebuild infile.dat,outfile.dat -x:3
```

ファイルを再編成します。データは、3 番目の副キーの順序で書き込まれます。

```
rebuild infile.dat
```

索引ファイルをリビルドします (索引 (.idx) ファイルがまだ存在していて、そこに保持されたキー情報が破壊されていないことを前提とします)。

```
rebuild infile.dat -k:1+20 -i
```

索引ファイルをリビルドします。索引 (.idx) ファイルは不要であり、存在していても無視されます。これは、指定されたキー定義を使用して完全に再作成されます (ファイル作成時に使用したキー定義に対応させる必要はありません)。リビルドが完了すると、処理情報が表示されます。

索引ファイルのない C-ISAM および Level 2 のファイルでは、-r および -s オプションを使用してレコード情報とファイル形式を指定します。

```
rebuild infile.dat,outfile.dat -s:c-isam -c:d1 -t:mf
```

C-ISAM 形式の索引ファイルをシステムで使用される形式に変換し、ファイルのデータ部分を圧縮します。

```
rebuild infile.dat,outfile.dat --o:rel,ind -k:1+2:3+10d -c:i7 -r:
f10
```

キーを圧縮して固定長の相対ファイルの索引を生成します。

```
rebuild infile.dat,outfile.dat -s:lii -t:lii
```

LEVEL II V2.5 COBOL 形式のファイルを再編成して、出力ファイルに LII 構造を保存します。同様の方法で、-s および -t を他の形式と組み合せることができます。

```
set data=/tmp.dat rebuild data -i
```

ファイル /tmp.dat を再編成して、情報を表示します。

データファイルの変換　　ファイル変換ユーティリティ

リビルド　ファイルシステム

第 17 章：ファイル変換ユーティリティ

ここでは、メインフレームと PC 間でのファイルの移動に役立つ Net Express の次のユーティリティについて説明します。

ファイル変換ユーティリティ	用途
DFCONV	ダウンロードしたデータファイルを、PC で必要な形式に変換します。
VRECGEN	メインフレームファイルを、メインフレームから PC にダウンロードするために必要な形式に変換します。
VRECGEN2	PC からアップロードされたファイルをメインフレームで必要な形式に変換します。

DFCONV バッチファイルの変換

DFCONV ユーティリティは、データファイルコンバータとも呼ばれ、データファイルを変換します。 このユーティリティは、ファイル形式、編成、データ形式の変換、およびデータファイル変換プロファイルを使用した索引ファイルの再編成を実行するためのバッチおよび呼び出しプログラムインターフェイスを提供します。

データファイルコンバータでは、次の処理を実行できます。

- メインフレームから PC へダウンロードされたデータファイルの変換
- PC データファイルから、メインフレームに転送するための VRECGEN2 形式のファイルへの変換
- メインフレームから PC 形式、またはその逆の、レポートファイルの変換
- ANSI データ形式と EBCDIC データ形式間のファイルの変換
- 浮動小数点形式の変換
- ファイル構造体の再編成
- 複数のレコードタイプの変換
- Micro Focus 形式ファイルから Btrieve 形式ファイルへ、またはその逆の変換
- バッチインターフェイスとして動作
- 呼び出しプログラムインターフェイスとして動作

注：

- テキスト以外のデータ項目 (COMPUTATIONAL データと数値データ) を持つレコード上で文字集合を変換するには、レコードレイアウトファイルが作成済みであることが前提になります。レコードレイアウトファイルの作成方法の詳細については、『入門書』の『データファイルの作成と維持』の章にある『レコードレイアウトファイルの作成』の項を参照してください。
- データファイルコンバータの GUI バージョンも利用できます。詳細については、『データファイルの変換』の章を参照してください。

操作

以後の項では、データファイルコンバータの 2 つの操作について説明します。

- コマンド行バッチインターフェイス
- 呼び出しバッチインターフェイス

コマンド行バッチインターフェイス

コマンド行は、次のとおりです。

```
run dfconv profile-fname [input-fname] [output-fname]
```

パラメータの内容は、次のとおりです。

profile-fname	実行する必要のある作業の詳細を含むプロファイルファイルの名前 (『プロファイルファイルでのファイル変換の定義』の項を参照)
input-fname	入力ファイルの名前 (オプション)
output-fname	出力ファイルの名前 (オプション)

入力ファイル名と出力ファイル名はプロファイルファイルから抽出できるため省略できます。 プロファイルファイルのみではなく、コマンド行でもファイル名を指定した場合は、コマンド行で指定した名前が優先されます。

変換中にエラーが発生した場合は、変換後にオペレーティングシステムのエラーレベル (ERRORLEVEL) は 0 以外に設定されます。 バッチファイルから変換を実行している場合は、このエラーレベルを調べ、変換後に実行するアクションを決めることができます。

呼び出しバッチインターフェイス

変換のプロファイルが存在する場合は、変換を行うために、次のインターフェースを使用してデータファイルコンバータを呼び出せます。

```
call dfconv using dfconv-params
```

この場合の dfconv-params は、次のように定義します。

```
01 dfconv-params.
    03 profile-filename    PIC X(65).
    03 input-filename      PIC X(65).
    03 output-filename     PIC X(65).
```

パラメータの内容は、次のとおりです。

profile-filename	データファイル変換の詳細を含むプロファイルファイルの名前 (『プロファイルファイルでのファイル変換の定義』の項を参照)
input-filename	入力ファイルの名前
output-filename	出力ファイルの名前

パラメータフィールドのみではなく、プロファイルファイルでファイル名を指定した場合は、パラメータフィールドで指定した名前が優先されます。

エラーが発生した場合は、データファイルコンバータから戻ったときに、RETURN-CODE にゼロ以外が設定されます。このエラー番号は、profile-filename フィールドに返されます。

データファイルコンバータの使用

以後の項では、データファイルコンバータを使用して、データファイルを変換し、ファイル索引をリビルドする方法について説明します。

ファイル形式とデータ変換

ファイルをある形式から別の形式へ変換すると、データをある形式から別の形式へ再構築できます。 たとえば、ファイル順編成から索引編成へ変換できます。 また、アプリケーションの変更の必要性に応じて、二次索引を追加することもできます。 データファイルコンバータを使用する場合は、作成済みのデータを再作成したり、特別な変換プログラムを作成する必要はありません。

多くのファイル転送パッケージではある形式から別の形式へファイルを変換しても、ファイル全体が別のデータ形式に変換されるのみです。 一般的な COBOL データファイルとデータベースレコードは、さまざまなテキストとバイナリデータで構成される複雑なレコード構造になっており、それらも変換する必要があります。 データファイルコンバータの GUI バージョンでは、レコードレイアウトの記述を使用してデータを簡単に変換できます。レコードレイアウトファイルの作成方法の詳細については、『入門書』の『データファイルの作成と維持』の章にある『レコードレイアウトファイルの作成』の項を参照してください。

データファイルコンバータでは、内部浮動小数点形式を、IEEE 形式と S/370 形式間で変換できます。

EBCDIC と **ANSI** 間の変換

メインフレームのファイルは、PC に転送するときにバイナリ形式にする必要があります。最初に、このファイルが可変長の索引ファイルである場合は、メインフレームプログラム VRECGEN を使用して、メインフレームでそのファイルの初期再フォーマットを実行してください。 再フォーマット後に、データファイルコンバータで、そのファイルをダウンロードして処理できます。

メインフレームから **PC** 印刷ファイルへの変換

メインフレームのレポート形式ファイルは、メインフレームから PC へダウンロードされたファイル、または、アプリケーションで作成され PC にダウンロードされ PC で実行されるファイルのどちらかです。

メインフレームレポート形式には、データファイルコンバータによって PC で印刷可能な形式に変換されるキャリッジ制御文字が含まれます。 メインフレームレポートファイルを定義する場合は、レコード長にキャリッジ制御文字を含め、レコードは固定長にする必要があります。 PC 印刷ファイルでは、レコード長はデータ長のみになります。

PC 印刷形式ファイルは、1 行あたり 132 文字の印刷が可能な標準の ANSI プリンタで印刷できます。 1 行あたり 80 文字しか印刷できない ANSI プリンタを使用する場合は、ファイルを圧縮モードで印刷する必要があります。

ダウンロードされたメインフレームアプリケーションを使用して PC で作成された入力ファイルをメインフレームレポート形式ファイルとして指定した場合には、エラーメッセージ 139(「入力ファイルは、"本物の" メインフレームレポート形式ではありません。」) が表示されます。 このエラーは、プリンタ出力をメインフレームレポート形式ではなく、PC 形式で作成すると発生する場合があります。 プログラムのコンパイルと実行を制御するために使用する指令をチェックしてください。

データファイルコンバータでは、メインフレームレポートファイルと PC 印刷ファイル間の変換を特別にサポートします。 これらの形式と他の形式間の変換はサポートされません。

索引ファイルの再編成

レコードを追加、削除および変更して索引ファイルを更新すると、索引構造体とデータ構造体は切り離され、処理効率が悪くなります。 さらに、削除されたレコードに残された領域は常に再使用されるわけではないので、ファイルが必要以上に大きくなってしまいます。 多数の変更を行った後は、索引ファイルを再編成することをお奨めします。 入力ファイルと出力ファイルの設定を一致させるために、データと索引を順番にリビルドして、空き領域を再利用し、ファイルのパフォーマンスとデータの完全性を最適化できます。

プロファイルファイルでのファイル変換の定義

ここでは、プロファイルファイルでファイルの変換、またはファイルをある編成から別の編成へ再構築するために必要なエントリについて説明します。 たとえば、次のどれか 1 つを実行しなければならないとします。

- 索引ファイルへの二次索引の追加
- 順ファイルまたは相対ファイルから索引ファイルへの変換
- 索引ファイルから相対ファイルへの変換

```
DFS Profile File Version  V09.B01.07

Batch-Process: Convert-file
                   ************************************************
                   *           (Twin)                (Single)     *
                   * オプション: Convert-File          Rebuild-index *
                   *           Rebuild-index-and-data  None         *
                   ************************************************
Strt-file:              drive:¥path¥filename.STR
Edit-Mode:     Quick
                   * オプション: Quick/Full                        *
-------------------------------------------------------------------------------
Input-File     drive:¥path¥inputfile.dat
   Format:         Micro-Focus
                       * オプション: Micro-Focus       IDXFORMAT(4)  *
                       *             Btrieve           C-ISAM        *
                       *             LEVEL-II          ESDS          *
   Organization:   VRECGEN
                       * オプション: Sequential   Line-Sequential     *
                       *             Indexed   Mainframe-Report-ANSI *
                       *             Relative  Mainframe-Report-mach *
                       *             VRECGEN     PC-Print            *
   Record-Format:  Variable
                       * オプション: Fixed/Variable                  *
   Character-Set:  EBCDIC
                       * オプション: ASCII/EBCDIC                    *
   Floating-Point: 370
                       * オプション: IEEE/370                        *
   Compression:    Off
                       * オプション: On/Off                          *
   Min-Rec-Length: 352
   Max-Rec-Length: 633
-------------------------------------------------------------------------------

Output-File    drive:¥path¥outputfile.dat
   Format:         Micro-Focus
   Organization:   Indexed
                       * オプション: Sequential  Line-Sequential     *
                       *             Indexed   Mainframe-Report-ANSI *
                       *             Relative    PC-Print            *
                       *             VRECGEN2                        *
```

```
Record-Format:  Variable
Character-Set:  EBCDIC
Floating-Point: 370
Compression:    Off
Min-Rec-Length: 352
Max-Rec-Length: 633
                    * キー設定                                    *
                    *        d= 重複を許可                        *
                    *        s= スパース文字集合                  *
                    *        圧縮                                 *
                    *             cd= 重複                        *
                    *             cl= 先頭余白                    *
                    *             ct= 後部余白                    *
Prime-key :   1:1, 2:6

Alt-Key  1:   8:1
              d,cl
Alt-Key  2:   9:1
              d,cd,ct
Alt-Key  3:   10:5
```

このファイル内の最初のレコードは、プロファイルファイルの形式を示しています。 DFCONV で使用されるプロファイルは、以前の Workbench で使用されているものと同じ形式です。

変換操作の指定

実行する操作は、Batch-Process オプションのパラメータとして選択します。

Batch-Process 定義されたファイルで実行される操作。

Convert-file Rebuild-index Rebuild-index-and-data

Input-File はすべての操作に対して定義され、Output-File は Convert-file および Rebuild-index-and-data で定義されます。 Rebuild-index 操作では入力データファイルのファイル索引を再作成するため、Output-File の記述は不要です。

Strt-file 入力ファイルのレコードレイアウト記述を含むファイル。

No-structure *location¥filename*.str

データファイルを EBCDIC と ANSI 間で変換するときに、レコードにテキスト以外のデータ項目が含まれる場合は、変換が必要なデータ項目と変換しない項目 (バイナリ値など) とを定義するためにレコードレイアウトの記述が必要です。 レコードレイアウトは、レコードレイアウトエディタで作成し、.str ファイル拡張子の付いたファイルに保存します。レコードレイアウトファイルの作成方法の詳細については、『入門書』の『データファイルの作成と維持』の章にある『レコードレイアウトファイルの作成』の項を参照してください。

Edit-Mode このプロファイルをデータファイルエディタで読み取る場合の編集モード。

Quick Full

このパラメータは DFCONV では使用されません。

入力ファイルパラメータの指定

最初に、入力ファイルを選択します。

Input-File	定義された操作に対する入力ファイル名。

次に、パラメータを使用して入力ファイルの説明を記述します。

パラメータ	説明
Format	入力ファイルのファイル形式。 • Micro Focus • ESDS • Btrieve • IDXFORMAT(4) • C-ISAM • LEVEL-II デフォルトの形式は Micro Focus です。これは通常ホストからダウンロードされ PC で再作成されるファイルに使用されます。 選択したファイルがメインフレームの VSAM エントリ順データセットのエミュレーションである場合は、ESDS を選択して、PC に VSAM ESDS のエミュレーションを提供します。
Organization	入力ファイルのファイル編成。 • 行順 • 順編成 • 索引順編成 • 相対編成 • VRECGEN • ANSI メインフレームレポート (ANSI キャッリッジ制御文字を含むメインフレームレポート) • マシンメインフレームレポート (マシンコード制御文字を含むメインフレームレポート) • PC 印刷
Record-Format	入力ファイルのレコード形式。 • 固定 • 可変 注：可変長メインフレームレポートファイルはサポートされません。
Character-Set	入力ファイルで使用される文字符号化方式。 • ASCII • EBCDIC 注：Net Express で使用される文字符号化方式は、EBCDIC と ANSI です。ASCII は、以前 Workbench で使用されていますが、このユーティリティでは ANSI を表します。
Floating-Point	入力ファイルで使用される浮動小数点形式のタイプ。 • IEEE • 370
Compression	データ圧縮を入力ファイルで使用するかどうかを示します。 • オン • オフ 注：データ圧縮でファイルが作成された場合には On に設定する必要があります。

Min-Rec-Length 入力ファイルの最小レコード長。

ファイルが固定長である場合は、最大レコード長と同じ値に設定する必要があります。

Max-Rec-Length 入力ファイルの最大レコード長。

作成されているファイルが索引ファイルである場合は、次のレコードを使用してキーを定義します。

Prime-key 主キーの開始位置 (カラム) と長さ。

位置:長さ (1:12 など)。位置フィールドと長さフィールドの区切りにコロンを使用します。

プロファイルファイルの 2 つ目のレコードでは、作成されている索引ファイルに必要なキーフラグが設定されます。フラグはコンマで区切ります (d,cd など)。

- d 重複レコードキーを許可
- s スパース文字 - サポートされない
- cd 重複レコードキーの圧縮
- cl レコードキーの先頭余白の圧縮
- ct レコードキーの後続空白の圧縮

出力ファイルの指定

最初に、出力ファイルを選択します。

Output-File 定義された操作に対する出力ファイル名。

次に、パラメータを使用して出力ファイルの説明を記述します。

パラメータ	説明
Format	出力ファイルのファイル形式。 • Micro Focus • ESDS • Btrieve • IDXFORMAT(4) • C-ISAM • LEVEL-II. デフォルトの形式は Micro Focus です。これは通常 PC 上で作成されたファイルに使用されます。 ファイルがメインフレームの VSAM エントリ順データセットのエミュレーションである場合は、ESDS を選択して、PC に VSAM ESDS のエミュレーションを提供します。 VSAM KSDS と RRDS の場合は、Micro Focus 形式を選択します。
Organization	出力ファイルのファイル編成。 • 行順 • 順編成 • 索引順編成 • 相対編成 • VRECGEN2 • ANSI メインフレームレポート(ANSI キャリッジ制御文字を含むメインフレームレポート) • PC 印刷

Record-Format 出力ファイルのレコード形式。

- Fixed Variable

注：可変長メインフレームレポートファイルはサポートされません。

Character-Set 出力ファイルで使用される文字符号化方式。

- ASCII EBCDIC

注：Net Express で使用される文字符号化方式は、EBCDIC と ANSI です。ASCII は、以前 Workbench で使用されていますが、このユーティリティでは ANSI を表します。

Floating-Point 出力ファイルで使用される浮動小数点形式のタイプ。

- IEEE
- 370

Compression データ圧縮を出力ファイルで使用するかどうかを示します。

- オン
- オフ

Min-Rec-Length 出力ファイルの最小レコード長。

ファイルが固定長である場合は、最大レコード長と同じ値に設定する必要があります。

Max-Rec-Length 出力ファイルの最大レコード長。

作成されているファイルが索引ファイルである場合は、次のレコードを使用してキーを定義します。

Prime-key 主キーの開始位置 (カラム) と長さ。 開始位置 (オフセット) は 1 から開始します。データファイルコンバータの GUI バージョンの「キー情報」ダイアログボックスでは、キーオフセットは 0 から開始します。

位置:長さ (1:12 など)。位置フィールドと長さフィールドの区切りにコロンを使用します。

プロファイルファイルの 2 つ目のレコードでは、作成されている索引ファイルに必要なキーフラグが設定されます。フラグはコンマで区切ります (d,cd など)。

d	重複レコードキーを許可
s	スパース文字 - サポートされない
cd	重複レコードキーの圧縮
cl	レコードキーの先頭余白の圧縮
ct	レコードキーの後続空白の圧縮

副キーは、主キーの後に定義し、最初の副キーには「Alt-Key 1:」というキーワードを使用し、2 つ目の副キーには「Alt-Key 2:」というキーワードを使用します。 主キーでは、副キーを定義しているプロファイルファイル内で 2 つのレコードの形式が定義されます。

副レコードキーは最大 63 個まで定義できます。それぞれのキーについて、レコード内の位置と長さ、および重複値が許可されるかどうかを指定します。 各キーは分割可能で、重複キーを定義できます。

分割キーは、キー定義レコード内で複数の位置と長さの定義を、コンマと空白で区切って指定します (1:12, 20:5 など)。

変換に関する一般情報

入力ファイルと出力ファイルを定義する場合は、次の点に注意する必要があります。

- 選択されたファイルタイプが索引、可変長順または可変長相対である場合

  レコード長情報はファイルヘッダーから抽出され、プロファイルファイル内の入力ファイルで定義されたレコード長情報を置き換えます。

- ファイルタイプがメインフレームレポート形式である場合

  入力レコード長と出力レコード長を設定し、メインフレーム印刷形式ファイルのレコード長が、キャリッジ制御文字を保持できる 1 文字以上の長さになるようにする必要があります。

- DFCONV を呼び出すプログラムがCHARSET"EBCDIC" でコンパイルされている場合

  データファイルコンバータではプロファイルを解釈できません。ファイルコンバータは DOS および ANSI に基づいたプログラムにのみ適用されます。 適切な処理を行うためには、ANSI に基づくプログラムを使用してデータファイルコンバータを呼び出します。

- 出力ファイルにテキスト以外のデータが含まれる場合

  出力ファイルタイプとして、行順は指定できません。 バイトに X'1B' 以下の値が含まれる場合は常に追加バイト (X'00') が ANSI 行順レコードに挿入されるため、必要以上に大きいファイルになってしまう場合があります。 追加のバイトを含めて行順ファイルに保持されているデータにヌル保護を提供しないと、これらのバイトは、このタイプのファイルで使用される制御文字として解釈されます。

ファイルのリビルド

データファイルからリビルドへのインターフェイスは、破損した索引のリビルドをサポートします。

```
DFS Profile File Version  V08.U04.01

Batch-Process: Rebuild-index
                   ***************************************************
                   *           (Twin)                  (Single)      *
                   * オプション: Convert-File          Rebuild-index *
                   *           Rebuild-index-and-data  None          *
                   ***************************************************
Strt-file:     No-structure
Edit-Mode:     Quick
                   * オプション: Quick/Full                          *
-------------------------------------------------------------------------------
Input-File     e:¥test¥testidx.dat
   Format:         Micro-Focus
                       * オプション: Micro-Focus       IDXFORMAT(4)  *
                       *             Btrieve           C-ISAM        *
                       *             LEVEL-II          ESDS          *
   Organization:   Indexed
                       * オプション: Sequential  Line-Sequential     *
                       *             Indexed   Mainframe-Report-ANSI *
                       *             Relative  Mainframe-Report-mach *
                       *             VRECGEN     PC-Print            *
   Record-Format:  Variable
                       * オプション: Fixed/Variable                  *
   Character-Set:  ASCII
                       * オプション: ASCII/EBCDIC                    *
   Floating-Point: IEEE
                       * オプション: IEEE/370                        *
```

```
Compression:    Off
                    * オプション:  On/Off                              *
Min-Rec-Length: 352
Max-Rec-Length: 633
                    * キー設定                                          *
                    *        d= 重複を許可                             *
                    *        s= スパース文字集合                       *
                    *        圧縮                                       *
                    *             cd= 重複                              *
                    *             cl= 先頭余白                          *
                    *             ct= 後部余白                          *
Prime-key :   1:1, 2:6

Alt-Key  1:   8:1
              d
Alt-Key  2:   9:1
              d
```

リビルド機能は、索引順ファイルをリビルドする場合にのみ使用できます。 他のファイル形式をリビルドするには、入力ファイルと出力ファイルに同一の設定を使用するファイル変換機能を使用します。

注:上記で説明したプロファイルファイルを使用している場合は、Prime-key フィールドで定義したオフセットは 1 から開始します。ただし、データファイルコンバータの GUI バージョンを使用する場合は、キーのオフセットは「キー情報」ダイアログボックスで 0 から開始します。データファイルコンバータの詳細については、『データファイルの変換』の章を参照してください。

破損した索引ファイルの回復

破損した索引のある索引ファイルは、このオプションを使用して回復できます。 索引が破損するのは、プログラムによって索引ファイルを開いて変更したが、ファイルを閉じていない状態で電源が切断された場合などです。 索引が破損したことを示すフラグがファイル内で検出されると、そのファイルに次にアクセスするときにレポートされます。 データの完全性が回復されるまでは、そのファイルではそれ以上の操作は実行できません。

リビルドでは、回復に必要な索引のみをリビルドします。 索引がすでに存在する場合は、リビルドはそのファイル内の情報を使用して索引を回復しようとします。

一般的なリビルド情報

リビルドを使用する場合は、次の点に注意する必要があります。

- リビルドを使用する前にリビルドまたは再編成するファイルのバックアップコピーを作成する必要があります。 リビルドでは、元のファイルが破損しないようにあらゆる予防措置が行われますが、パラメータを誤って指定すると元のファイルが破損する可能性があります。 また、元の索引が上書きされるため、後から問題が発生する場合に備えてバックアップを保持することをお奨めします。
- 索引ファイルの索引をリビルドする機能では、元のファイルに保存されていた、データファイルの空き領域に関する情報は保持されません。 そのため、この領域は再使用できず、ファイルは必要以上に大きくなってしまう場合があります。 この問題を解決するためには、索引をリビルドした後にファイルを再編成し、空き領域をすべて削除してください。 (空き領域は、データファイル内の、削除されたレコードで使用されていた領域で構成されます。 これらの領域に関する情報は通常は保持され、新しいレコードに適宜再使用されます。)
- リビルドプロセスが完了したら、ファイルの末尾に空白の空き領域ポインタレコードを追加して

データファイルが変更されます。 索引をリビルドするたびに、新しいレコードが追加されます。このため、再編成を行わずにリビルドを何度か実行すると、データファイルが大きくなってしまいます。 この問題についても、ファイルを再編成し、すべての空き領域ポインタレコードを削除し、単一の新しい空白の空き領域を追加して解決します。

データファイルコンバータのエラーメッセージ

エラーメッセージについては、Net Express オンラインヘルプに記載されています。([ヘルプ] メニューの [ヘルプトピック] をクリックします。[目次] タブで、[リファレンス]、[エラーメッセージ]、[データファイルコンバータのメッセージ] をダブルクリックします。)

VRECGEN

VRECGEN は、メインフレームユーティリティです。 VRECGEN を使用して、可変長のメインフレームファイルのコピーを作成します。 作成されたコピーでは、メインフレームから PC へダウンロードするためにデータレコードにレコード長情報を埋め込みます。Micro Focus は、VRECGEN ユーティリティのソースコード (vrecgen.cbl) を提供しています。ただし、使用前にソースコードを変更し、提供されているバージョンの vrecgen.cbl で指定されていないオプションを指定する必要があります。

VRECGEN で作成されるファイルには、各可変長データレコードの前に 2 バイトのレコード長フィールドがあります。 レコード長フィールドには、2 バイトのレコード長フィールドに可変長データレコードの長さを加えた値が含まれます。 多くのファイル転送ソフトウェア (IBM SEND/RECEIVE など) では論理レコードを処理し、各レコードの長さを含む「レコード記述ワード」を削除するため、このフィールドが必要になります。 この情報がない場合は、レコードを区切って、PC 上でデータセットをリビルドすることはできません。

VRECGEN で作成されたファイルは、特別なダウンロード形式で表されます。 このファイルをバイナリ形式で転送するには、標準の転送ユーティリティを使用する必要があります。 このファイルは、データファイルコンバータで処理できます。

注：VRECGEN のソースコード (vrecgen.cbl) は、Micro Focus がモデルとして用意しているもので、最終的なユーティリティというわけではなく、変更しないですべてのサイトのニーズを満たすものではありません。アプリケーションやサイトの要件を満たす特定の値を含めるためには vrecgen.cbl を編集する必要があります。一般的に、vrecgen.cbl を編集するのは、最大レコード長を 32,000 バイトよりも大きくするためです。

VRECGEN のインストール

可変長テストファイルをダウンロードする場合は、まずメインフレーム環境に VRECGEN プログラムをアップロードし、コンパイルする必要があります。 このプログラムは、メインフレームの可変長形式のファイルを変換するためのサンプル COBOL プログラムとして提供されます。

注：各サイトは提供されたソースコードを見直し、可変長ファイルの最大長と最小長を決定する必要があります。 これらの値に MVS 「レコード記述ワード」用の 4 を加えた値を、サンプル FD のレコード記述の OCCURS DEPENDING ON 句で使用する必要があります。

VRECGEN のインストール方法

1. VRECGEN のソースコードファイルをインストール手順の一環として PC にコピーします。ファイル名は vrecgen.cbl とし、Net Express\base\source ディレクトリにコピーします。
2. Net Express をインストールした後に、ファイル vrecgen.cbl をメインフレームにアップロードし、ANSI から EBCDIC への変換を指定します。 ファイルが 80 バイトのカードイメージ形式の固定長ファイルであることを確認します。

3. COBOL コンパイラを使用して、メインフレーム上で VRECGEN をコンパイルします。 このプログラムのコンパイル時には、次のオプションを指定します。 一部はコンパイラオプション、一部は実行時オプション、一部はリンク編集オプションです。 使用中の環境およびコンパイラに適用できるもののみを使用します。

    - NOSSRANGE または NOSUBRNGCK (サブスクリプト範囲チェックなし)
    - DATA(24)
    - NOOPT
    - AMODE=24,RMODE=24

注：VS COBOL II のリリース 3 以降を使用しているときに、ファイルステータス 39 が発生した場合は、CMPR2 コンパイルオプションを使用して VRECGEN を再コンパイルします。

VRECGEN の実行

メインフレームで VRECGEN を実行する前に、適切な JCL 情報を指定する必要があります。 正しい STEPLIB、入力データセット、および出力データセットをポイントする、次の JCL スケルトンを入力します。

```
//MYJOB   JOB (ACCOUNTING INFORMATION)
//STP1    EXEC PGM=VRECGEN
//STEPLIB DD DSN=STEPLIB,DISP=SHR
//SYSOUT  DD SYSOUT=*
//INPDD   DD DSN=INPUT.OS.DATA.SET.DISP=SHR
//OUTDD   DD DSN=OUTPUT.OS.DATA.SET,
//        DISP=(NEW,CATLG,DELETE),
//        DCB=(RECFM=VB,LRECL=LLL,BLKSIZE=BBB),
//        SPACE=(TRK,(X,Y),,RLSE),
//        UNIT=SYSDA
```

注：

- 出力ファイルの LLL (LRECL) 値と BBB (BLKSIZE) 値を設定するときは、使用中の VRECGEN 内の OUTDD の FD で指定されている値を参照してください。 プログラムでのデフォルトの LRECL と BLKSIZE は 32700 ですが、JCL、VRECGEN の FD、入力ファイルのファイルラベルが矛盾しないように変更する必要があります。
- 結果の出力ファイルは、標準的な転送機能を使用して PC に転送可能な、特別なダウンロード形式になります。 ダウンロードするときは、バイナリ転送を指定し、キャリッジ制御文字である CR (キャリッジリターン) と LF (改行文字) を使用しないでください。

VRECGEN2

VRECGEN2 は、Micro Focus 可変長レコード形式のファイルを、メインフレーム形式に変換するメインフレームユーティリティです。 このユーティリティを使用すると、これらのファイルをメインフレームにアップロードしてメインフレームで使用できます。

VRECGEN2 のインストール

Micro Focus 可変長レコード形式をメインフレーム形式に変換する場合は、まずメインフレーム環境に VRECGEN2 プログラムをアップロードし、コンパイルする必要があります。

1. VRCGEN2 のソースコードファイルは、インストール手順の一環として PC にコピーされます。 ファイル名は vrecgen2.cbl とし、 Net Express\base\source ディレクトリにコピーします。

2. 次に、CRLF と ANSI をパラメータとして使用して、VRECGEN と同様の方法で、vrecgen2.cbl ファイルをメインフレームにアップロードします。

VRECGEN2 の実行

次の手順は、100～400 バイトの長さのレコードを持つ可変長データファイルをアップロードする方法を示します。

1. データファイルコンバータを使用して、可変長 PC ファイルを VRECGEN2 入力形式に変換します。 最小レコード長と最大レコード長に、それぞれ 100 と 400 を指定します。 レコード長を含む 2 バイトをデータファイルコンバータが付加するため、実際の最小レコード長と最大レコード長は、それぞれ 102 と 402 です。 ファイルが ANSI 形式の場合は、同時に EBCDIC 形式に変換する必要もあります。
2. メインフレームで、PC からのファイルを受信するデータファイルを割り当てます。 次のパラメータを使用します。
    - FORMAT = VB
    - RECLEN = 406
    - BLKLEN = 410
3. ファイルをアップロードします。 CRLF 挿入は設定しないでください。 手順 1 で EBCDIC に変換した場合は、ANSI を指定しないでください。
4. メインフレームの vrecgen2.cbl のソースを変更し、出力ファイルの FD が、必要なメインフレームデータファイルの形式と一致するようにします。
5. vrecgen2.cbl をコンパイルし、リンクします。
6. VRECGEN2 を実行します。 JCL は、次の変更点を除き、VRECGEN と同じです。
    - プログラム名を、VRECGEN2 とする必要がある。
    - 出力ファイル定義が、VRECGEN2 内の出力 FD と一致している必要がある。
    - 次のように指定する必要がある。
        - RECFM=FB
        - LRECL=404
        - BLKSIZE=408

出力結果は、有効な可変長ファイルとなります。 次に、100 バイトのレコード長を使用した場合の、VRECGEN2 形式のファイルの例を示します。

バイト	内容
1-4	レコード記述ワード
5-6	レコード長
7-106	レコードデータ。 最大レコードサイズよりも短い場合は、レコードにヌルバイトが入力されます。

注：VRECGEN のファイル形式はメインフレームから PC へのダウンロード、VRECGEN2 のファイル形式は MicroFocus 形式ファイルのメインフレームへのアップロードにそれぞれ使用され、両者に互換性はありません。

リビルド　　ファイルシステム

付録 A 章 : ファイルシステム

オペレーティングシステムによっては、ファイルシステムでサイズの異なるアドレス指定が使用されていることがあります。 多くの現行ファイルシステムでは、32 ビットか 64 ビットのアドレス指定を採用しています。 これら 2 つのサイズを採用しているオペレーティングシステム/ファイルシステムを下記に示します。

ファイルハンドラは、32 ビットアドレス指定か 64 ビットアドレス指定のどちらかを処理するように設定します。 1 つのファイルを共有するすべてのアプリケーションで、同じ設定のファイルハンドラを使用していることが重要です。 同じでない場合、データの破損が起こる可能性があります。

64 ビットアドレス指定を使用するファイルシステムで、ファイルハンドラを、32 ビットアドレス指定用に設定することはできます。 この場合、ファイルハンドラは、すべてのファイルを安全なサイズに (32 ビットアドレス指定に) 制限します。

32 ビットファイルシステム

- Windows 95
- Windows 98
- Windows Millennium
- Windows NT (FAT)
- Windows 2000 (FAT)
- Windows XP (FAT)
- UNIX (大きなファイルシステムが有効でない)

64 ビットファイルシステム

- Windows NT (NTFS)
- Windows 2000 (NTFS)
- Windows XP (NTFS)
- UNIX (大きなファイルシステム)
- Linux

ファイルシステムと FILEMAXSIZE

基本ファイルシステムによって、このファイルハンドラの構成オプションで設定する値が決まります。

警告：

- 次の場合は、FILEMAXSIZE の値を 8 に設定しないでください。

- 64 ビットファイルシステムをサポートしていないオペレーションシステムでアプリケーションを実行する場合。
- FAT ファイルシステムにアクセスするアプリケーションを実行している場合。FAT ファイルシステムは、32 ビットを超えるファイルのアドレス指定をサポートしません。
- 32 ビットのファイルシステムの共有ファイルにアクセスする必要のあるアクセスプログラムがある場合。
- 32 ビットのファイルアクセスのみに対応した旧バージョンの Micro Focus COBOL を使用するアプリケーションとの間でファイルを共有している場合。

- あるファイルにアクセスするすべてのアプリケーションで、ファイルハンドラを同じ FILEMAXSIZE の値に設定する必要があります。
- Novell システムでは、32 ビット (4GB) サポートのみ提供されています。

詳細については、ヘルプトピック
『FILEMAXSIZE』を参照してください。
