



Btrieve API Guide

Zen v16

Activate Your Data™



Copyright © 2026 Actian Corporation. All Rights Reserved.

このドキュメントはエンドユーザーへの情報提供のみを目的としており、Actian Corporation (“Actian”)によりいつでも変更または撤回される場合があります。このドキュメントは Actian の専有情報であり、著作権に関するアメリカ合衆国国内法及び国際条約により保護されています。本ソフトウェアは、使用許諾契約書に基づいて提供されるものであり、当契約書の条件に従って使用またはコピーすることが許諾されます。いかなる目的であっても、Actian の明示的な書面による許可なしに、このドキュメントの内容の一部または全部を複製、送信することは、複写および記録を含む電子的または機械的のいかなる形式、手段を問わず禁止されています。Actian は、適用法の許す範囲内で、このドキュメントを現状有姿で提供し、如何なる保証も付しません。また、Actian は、明示的暗示的法的に関わらず、黙示的商品性の保証、特定目的使用への適合保証、第三者の有する権利への侵害等による如何なる保証及び条件から免責されます。Actian は、如何なる場合も、お客様や第三者に対して、たとえ Actian が当該損害に関してアドバイスを提供していたとしても、逸失利益、事業中断、のれん、データの喪失等による直接的間接的損害に関する如何なる責任も負いません。

このドキュメントは Actian Corporation により作成されています。

米国政府機関のお客様に対しては、このドキュメントは、48 C.F.R 第 12.212 条、48 C.F.R 第 52.227 条第 19(c)(1) 及び (2) 項、DFARS 第 252.227-7013 条または適用され得るこれらの後継的条項により限定された権利をもって提供されます。

Actian、Actian DataCloud、Actian DataConnect、Actian X、Avalanche、Versant、PSQL、Actian Zen、Actian Director、Actian Vector、DataFlow、Ingres、OpenROAD、および Vectorwise は、Actian Corporation およびその子会社の商標または登録商標です。本資料で記載される、その他すべての商標、名称、サービスマークおよびロゴは、所有各社に属します。

目次

このドキュメントについて	xv
このドキュメントの読者	xv
Btrieve API の紹介	1
Btrieve API 関数	1
BTRV	2
BTRVID	3
BTRCALL	3
BTRCALLID	3
BTRVEX	3
BTRVEXID	4
旧バージョンの関数	5
Btrieve API 関数のパラメーター	5
オペレーション コード	6
ステータス コード	6
ポジションブロック	7
データ バッファ	8
データ バッファ 長	8
キー バッファ	9
キー番号	10
クライアント ID	11
キー長	12
Btrieve API オペレーションの要約	13
セッション固有のオペレーション	13
ファイル固有のオペレーション	14
ファイル アクセス および 情報	14
データ 取得	15
データ 操作	17
サポート されない オペレーション	18
Btrieve API オペレーションの実行における一連のイベント	18
Btrieve API オペレーション	21
Abort Transaction (21)	24

パラメーター	24
前提条件.....	24
手順.....	24
結果.....	24
ポジショニング.....	25
Begin Transaction (19 または 1019)	26
パラメーター	26
前提条件.....	26
手順.....	26
結果.....	27
ポジショニング.....	27
Clear Owner (30)	28
パラメーター	28
前提条件.....	28
手順.....	28
結果.....	28
ポジショニング.....	29
Close (1)	30
パラメーター	30
前提条件.....	30
手順.....	30
結果.....	31
ポジショニング.....	31
Continuous Operation (42)	32
パラメーター	32
手順.....	33
詳細.....	33
結果.....	35
ポジショニング.....	36
Create (14)	37
パラメーター	37
前提条件.....	37
手順.....	37
詳細.....	38
結果.....	58
ポジショニング.....	59

Create Index (31)	60
パラメーター	60
前提条件	60
手順	61
詳細	63
結果	64
ポジショニング	65
Delete (4)	66
パラメーター	66
前提条件	66
手順	66
詳細	66
結果	67
ポジショニング	68
Drop Index (32)	69
パラメーター	69
前提条件	69
手順	69
詳細	69
結果	70
ポジショニング	71
End Transaction (20)	72
パラメーター	72
前提条件	72
手順	72
結果	72
ポジショニング	73
Find Percentage (45)	74
パラメーター	74
前提条件	74
手順	74
詳細	75
結果	77
ポジショニング	77
Get By Percentage (44)	78
パラメーター	78

前提条件.....	78
手順.....	78
詳細.....	79
結果.....	81
ポジショニング.....	81
Get Direct/Chunk (23)	83
パラメーター	83
前提条件.....	83
手順.....	83
詳細.....	84
結果.....	90
ポジショニング.....	92
Get Direct/Record (23)	93
パラメーター	93
前提条件.....	93
手順.....	94
結果.....	94
ポジショニング.....	95
Get Directory (18)	96
パラメーター	96
前提条件.....	96
手順.....	96
結果.....	96
ポジショニング.....	96
Get Equal (5)	97
パラメーター	97
前提条件.....	97
手順.....	97
結果.....	98
ポジショニング.....	98
Get First (12)	99
パラメーター	99
前提条件.....	99
手順.....	99
結果.....	100
ポジショニング.....	100

Get Greater Than (8)	101
パラメーター	101
前提条件	101
手順	101
結果	102
ポジショニング	102
Get Greater Than or Equal (9)	103
パラメーター	103
前提条件	103
手順	103
結果	104
ポジショニング	104
Get Key (+50)	105
パラメーター	105
前提条件	105
手順	105
結果	106
ポジショニング	106
Get Last (13)	108
パラメーター	108
前提条件	108
手順	108
結果	109
ポジショニング	109
Get Less Than (10)	110
パラメーター	110
前提条件	110
手順	110
結果	111
ポジショニング	111
Get Less Than or Equal (11)	112
パラメーター	112
前提条件	112
手順	112
結果	113
ポジショニング	113

Get Next (6)	114
パラメーター	114
前提条件.....	114
手順.....	114
結果.....	115
ポジショニング.....	116
Get Next Delete Extended (85)	117
パラメーター	117
前提条件.....	117
手順.....	117
詳細.....	118
結果.....	118
ポジショニング.....	120
Get Next Extended (36)	121
パラメーター	121
前提条件.....	121
手順.....	121
詳細.....	122
結果.....	131
ポジショニング.....	133
Get Position (22)	134
パラメーター	134
前提条件.....	134
手順.....	134
結果.....	134
ポジショニング.....	135
Get Previous (7)	136
パラメーター	136
前提条件.....	136
手順.....	136
結果.....	137
ポジショニング.....	138
Get Previous Delete Extended (86)	139
パラメーター	139
前提条件.....	139
手順.....	139

詳細	140
結果	140
ポジショニング	140
Get Previous Extended (37)	142
パラメーター	142
前提条件	142
手順	142
詳細	143
結果	143
ポジショニング	144
Insert (2)	145
パラメーター	145
前提条件	145
手順	145
結果	146
ポジショニング	146
Insert Extended (40)	148
パラメーター	148
前提条件	148
手順	148
詳細	149
結果	149
ポジショニング	150
Login/Logout (78)	152
パラメーター	152
前提条件	152
ログイン手順	152
ログアウト手順	152
結果	153
注記	153
ポジショニング	153
Open (0)	154
パラメーター	154
前提条件	154
手順	154
詳細	155

結果.....	158
ポジショニング.....	159
Reset (28)	160
パラメーター.....	160
前提条件.....	160
手順.....	160
結果.....	160
ポジショニング.....	161
Set Directory (17)	162
パラメーター.....	162
前提条件.....	162
手順.....	162
結果.....	162
ポジショニング.....	163
Set Owner (29)	164
パラメーター.....	164
前提条件.....	164
手順.....	164
詳細.....	165
結果.....	166
ポジショニング.....	166
Stat (15)	167
パラメーター.....	167
前提条件.....	167
手順.....	167
詳細.....	168
結果.....	169
ポジショニング.....	169
Stat Extended (65)	170
パラメーター.....	170
前提条件.....	170
手順.....	171
サブファンクション 1 : 拡張ファイル情報.....	171
サブファンクション 2 : システムデータ情報.....	172
サブファンクション 3 : 重複レコードによる競合情報.....	174
サブファンクション 4 : ファイル情報.....	175

サブファンクション 5：ゲートウェイ情報.....	179
サブファンクション 6：ロック オーナーの識別	180
サブファンクション 7：セキュリティ情報.....	183
サブファンクション 8：ステータス コード 71 の発生原因となる、テーブル名ま たはファイル名の一覧表示	187
結果	188
Step First (33)	189
パラメーター	189
前提条件	189
手順	189
結果	190
ポジショニング	190
Step Last (34)	191
パラメーター	191
前提条件	191
手順	191
結果	192
ポジショニング	192
Step Next (24)	193
パラメーター	193
前提条件	193
手順	193
結果	194
ポジショニング	194
Step Next Extended (38)	195
パラメーター	195
前提条件	195
手順	195
詳細	196
結果	196
ポジショニング	198
Step Next Delete Extended (87)	199
パラメーター	199
前提条件	199
手順	199
詳細	200

結果.....	200
ポジショニング.....	201
Step Previous (35)	202
パラメーター.....	202
前提条件.....	202
手順.....	202
結果.....	203
ポジショニング.....	203
Step Previous Delete Extended (88)	204
パラメーター.....	204
前提条件.....	204
手順.....	204
詳細.....	205
結果.....	205
ポジショニング.....	205
Step Previous Extended (39)	207
パラメーター.....	207
前提条件.....	207
手順.....	207
詳細.....	208
結果.....	208
ポジショニング.....	208
Stop (25)	209
パラメーター.....	209
手順.....	209
結果.....	209
ポジショニング.....	210
Unlock (27)	211
パラメーター.....	211
前提条件.....	211
手順.....	211
結果.....	212
ポジショニング.....	212
Update (3)	213
パラメーター.....	213
前提条件.....	213

手順	213
結果	214
ポジショニング	214
Update Chunk (53)	216
パラメーター	216
前提条件	216
手順	216
詳細	217
結果	224
ポジショニング	225
Version (26)	226
パラメーター	226
前提条件	226
手順	226
結果	227
ポジショニング	228

A. Btrieve オペレーションのクイック リファレンス 229

このドキュメントについて

このドキュメントは、Btrieve アプリケーションプログラミング インターフェイス (API) についての手引書です。

このドキュメントの読者

このドキュメントは、Zen に精通し、Btrieve API を使用してアプリケーションを開発するユーザーを対象としています。

Btrieve API の紹介

Zen MicroKernel エンジンは、高パフォーマンスなデータ処理とプログラミングの生産性向上を目的として設計されています。MicroKernel エンジン オペレーションを使用すると、開発アプリケーションではキー値、あるいはシーケンシャルまたはランダムアクセス方法に基づいて、レコードの取得、挿入、更新、または削除が行えるようになります。

Btrieve API は、以下のプログラミング言語および開発環境と互換性があります。

- Embarcadero C/C++
- Embarcadero Delphi
- GNU C/C++
- Micro Focus COBOL
- Microsoft Visual Basic
- Microsoft Visual C++
- Watcom C/C++

以下のセクションで API の機能について説明します。

- [Btrieve API 関数](#)
- [Btrieve API 関数のパラメーター](#)
- [Btrieve API オペレーションの要約](#)
- [Btrieve API オペレーションの実行における一連のイベント](#)

[Btrieve API オペレーションの一覧](#)または [Btrieve オペレーションのクイック リファレンス](#)へ直接移動することもできます。

Btrieve API 関数

Btrieve API は単一関数です。この API では、ほとんどのプログラム動作が関数名ではなくオペレーションコード パラメーターによって決定されます。アプリケーションで使用する API は、異なるプラットフォーム間でのコードの移植性を重視するか、特定

のプラットフォームで可能な限り最高のパフォーマンスを重視するかを基準として選択してください。

Btrieve アプリケーションでは、データ ファイルに対して絶対に標準の I/O を実行しないでください。開発するアプリケーションでは、Btrieve API 関数を使って、すべてのファイル I/O を実行する必要があります。

次の表は、オペレーション コードで使用するための Btrieve API 関数の一覧です。

関数	オペレーティングシステム	説明
BTRV BTRVID	すべて	オペレーティング システム間での完全なコード互換性を得るために使用します。大部分の開発者にとって、この利点はパフォーマンスのわずかな低下を十分に埋め合わせるものです。古いデータ バッファ レイアウトを使用します。
BTRCALL BTRCALLID	すべて	キー長の引数を使用する場合に使用します。古いデータ バッファ レイアウトを使用します。
BTRVEX BTRVEXID	すべて	長いデータ バッファが必要な場合、または新しいデータ バッファ レイアウトを使用する場合に使用します。データ バッファを正しく解釈できる限りにおいては、BTRV タイプのエントリ ポイントと混在させることができます。

Btrieve API 関数を呼び出すための言語固有の構文を調べるには、『*Zen Programmer's Guide*』の [Btrieve API プログラミング](#) を参照してください。

BTRV

BTRV によって、アプリケーションは MicroKernel エンジン呼び出しを実行できるようになります。BTRV 関数は、プログラミング インターフェイスのインストール オプションで提供されるすべての言語インターフェイス モジュールでサポートされています。場合によって、BTRV 関数は実際に BTRCALL 関数を呼び出すことがあります。しかしながら、プラットフォームに依存しないということから、BTRV 関数の方がより好ましいでしょう。

BTRVID

BTRVID によって、アプリケーションはクライアント ID パラメーターを含む単独の MicroKernel エンジン呼び出しを実行できるようになります。このパラメーターはアプリケーションで制御できます。アプリケーションでは BTRVID を使って、自分自身を MicroKernel エンジンに対する複数のクライアント ID として割り当て、ほかのクライアントの状態に影響を与えることなく、各クライアントのオペレーションを実行することができます。詳細については、[クライアント ID](#) を参照してください。

BTRCALL

Windows および Linux の場合、BTRCALL は BTRV 関数に相当します。BTRV で発生する若干のパフォーマンス低下が問題にならない限り、BTRCALL ではなく BTRV 関数を使用するようにしてください。

BTRCALLID

クライアント レベルの制御が必要で、アプリケーションが Windows、Linux、または Raspbian で動作する場合は、BTRCALLID 関数を使用します。

中間関数を呼び出さないこと以外は、この関数は BTRVID 関数に類似しています。

メモ： 従来の BTRCALLID32 関数は BTRCALLID 関数にエイリアスされました。

BTRVEX

13.0 および 16.0 形式ファイルのサイズが大きくなる可能性がある場合、およびより大きなデータバッファを使用する場合、以前の Btrieve インターフェイスが提供していたものより大きなランタイム値が必要となります。新しいエン트리ポイントである BTRVEX および BTRVEXID は、これらの要件を満たしています。これらは BTRCALL および BTRCALLID と似ていますが、いくつかの関数の引数のデータ型の幅が広がっている点と、いくつかのデータバッファのレイアウトが異なっている点が違います。宣言は `btrvexid.h` 内にあり、実装は BTRCALL と同じファイル内にあります。

BTRVEX エン트리ポイントを使用する Btrieve オペレーションの場合、データバッファで渡される一部の値は、8 バイトのレコード アドレスやレコード カウントのように幅が広がっています。8 バイト動作は BTRVEX エン트리ポイントに起因するものであり、アクセスするファイルの形式バージョンによって決まることではないことに留

意してください。BTRVEX を使用する場合、次のオペレーションはデータバッファのレイアウトの変更を必要とします。

- Create (14)、Create Index (31)
- Stat (15)
- Get Position (22)
- Get Direct (23)
- Get Next Extended (36)、Get Previous Extended (37)
- Step Next Extended (38)、Step Previous Extended (39)
- Insert Extended (40)
- Find Percentage (45)
- Stat Extended (65) サブファンクション 3 および 8
- Unlock (27)

エン트리ポイントの選択は、レコードデータには影響しません。

上述のとおり、データバッファサイズの引数は BTRCALL が 16 ビット整数を使用する箇所が BTRVEX では 32 ビット整数へのポインターになっています。このため、データバッファを 64 KB より大きくすることができます。

新しいファイル形式へ移行する場合、ポジションブロックとクライアント ID の値は BTRCALL と BTRVEX の両方で使用できるため、すべてのコードを一度に BTRVEX に変換する必要はないことを覚えておいてください。

BTRVEX のキー番号の引数は 32 ビット整数であるのに対し、BTRCALL は 8 ビットの符号付き整数を使用します。既存のコードを BTRVEX へ簡単に変換するためには、BTRVEX エントリーポイントはキー値 128 ~ 255 を -128 ~ -1 に再マップします。これは、符号付きバイト (例: -2) としてでなく符号なしバイト (例: 0xFE) として指定された定数に対応します。

BTRVEXID

BTRVID および BTRCALLID と同様、BTRVEXID は BTRVEX の利点に加えて、クライアント ID の制御が可能です。

旧バージョンの関数

次の関数は、以前のバージョンの Btrieve API に対応して書かれた古いアプリケーションとの互換性を維持するためだけにサポートされています。

- BTRCALLBACK
- BTRVINIT
- BTRVSTOP
- RQSHELLINIT
- WBRQSHELLINIT
- WBTRVINIT
- WBTRVSTOP
- BRQSHELLINIT

現在のバージョンではこれらの関数を使用しませんが、これらの関数を呼び出す古いアプリケーションは v6.15 以降の MicroKernel バージョンでも正常に実行されます。

Btrieve API 関数のパラメーター

どの関数呼び出しもすべてのパラメーターを提供する必要があります。これは、MicroKernel エンジンがすべてのオペレーションですべてのパラメーターを使用するわけではなく、場合によっては、パラメーター値を無視することがある場合にも、当てはまります。一般に、それぞれのオペレーションでは異なるパラメーターが送られ、返されます。各 Btrieve API オペレーションのパラメーターの詳細については、[Btrieve API オペレーション](#)を参照してください。

メモ： C 開発者： C 言語インターフェイスで使用されるプラットフォームに依存しないデータ型とポインターについては、[btitypes.h](#) ファイルを参照してください。

Btrieve API 関数は次のパラメーターを使用します。

- [オペレーション コード](#)
- [ステータス コード](#) (BASIC と COBOL のみ)
- [ポジション ブロック](#)
- [データ バッファー](#)

- データ バッファ長
- キー バッファ
- キー番号
- クライアント ID (BTRVID、BTRCALLID、および BTRVEXID 関数のみ)
- キー長 (BTRCALL、BTRCALLID、BTRVEX、および BTRVEXID 関数のみ)

オペレーション コード

オペレーション コード パラメーターは、Btrieve API 関数の動作を決定します。たとえば、1 つまたは複数のレコードの読み取り、書き込み、削除、更新などのオペレーションです。アプリケーションでは、すべての Btrieve API 呼び出しに対してオペレーション コードを指定する必要があります。このコードを MicroKernel エンジンが変更することはありません。オペレーション コードについては、[Btrieve API オペレーション](#)で説明しています。

メモ： C 開発者： 指定する変数のデータ型は、BTI_WORD (符号なし short integer) または BTI_INT (符号付き 32 ビット integer) のいずれかである必要があります。BTI_INT は BTRVEX および BTRVEXID でのみ使用されます。どちらの場合も、変数は値で渡されます。

ステータス コード

BASIC および COBOL アプリケーションでは、MicroKernel エンジンが符号付き整数としてステータス コードを返します。ほとんどのプログラミング環境で、ステータス コードは Btrieve API 関数呼び出しの戻り値です。ただし、一部の BASIC および COBOL 言語インターフェイスでは、ステータス コード パラメーターが必要となります。このパラメーターには、オペレーションの実行中にエラーが発生したかどうかを示す、コード化された値が格納されます。Btrieve API 呼び出しの終了後、アプリケーションではステータス変数の値を必ずチェックして、正常に終了したかどうかを判断する必要があります。

Zen コンポーネントによって、呼び出しから API へステータス コードが返されます。これらの API に書き込む場合は、以下の 3 つの状態に対して処理を行う必要があります。

- API の成功

- 予期された API の失敗
- 予期されなかった API の失敗

以下は、3 つすべての状態を処理する C コードの例です。

```
status = BTRVID(B_VERSION, posBlock1, &versionBuffer, &dataLen, keyBuf1, keyNum, (BTI_BUFFER_PTR)
&clientID);
if (status == B_NO_ERROR)
{
    /* 通常の実行を続行 */
    status = BTRVID(...);
}
else if (status == B_RECORD_MANAGER_INACTIVE)
{
    /* 予期されたエラーの処理 */
    printf("Btrieve Get Version() returned B_RECORD_MANAGER_INACTIVE\n");
}
else
{
    /* 予期されなかったエラー */
    printf("Btrieve Get Version() returned %d\n", status);
} /* 別の場合は終了 */
```

このステータスコードの処理方法に従うと、アプリケーションの将来的な安定性を確保するうえで役立ちます。

メモ： 古い BTRV 関数では、ステータスコードは 2 バイト整数でしたが、新しい BTRVEX および BTRVEXID では 4 バイト整数を返します。

ポジションブロック

ポジションブロックパラメーターは、128 バイト配列のアドレスで、MicroKernel エンジンがファイル I/O 構造体や Open (0) オペレーションに関連するポジショニング情報を格納するために使用されます。アプリケーションは、ファイルを開くたびに固有のポジションブロックを割り当てる必要があります。MicroKernel エンジンが、アプリケーションが Open オペレーションを実行する際にポジションブロックを初期化し、その後ファイルの操作中にこのポジションブロックを参照して更新します。このため、アプリケーションでは、そのファイルに対する以降すべての Btrieve API オペレーションで同じポジションブロックを指定する必要があります。

メモ： ポジションブロックへの書き込みを行ってはいけません。書き込みを行うと、ポジション喪失エラーやその他のエラー、あるいはファイルの損傷などの原因となります。

一度に複数のファイルを開く場合、MicroKernel エンジンがポジションブロックを使って、特定の呼び出しで対象となるファイルを判別します。同様に、同じファイルを複数

回開く場合、エンジンは **Open** オペレーションごとにそれぞれ異なるポジションブロックを使用します。さらに、エンジンは、同じファイルを開くクライアントごとにも別個のポジションブロックを使用します。クライアント間でポジションブロックを共有することはできません。

メモ： ポジションブロックはエン트리 ポイントにバインドされません。そのため、**BTRV** を使用してデータ ファイルを開き、**BTRVEX** を使用してデータを読み取り、**BTRCALL** を使用してファイルを閉じることが可能です。

データ バッファ

アプリケーションでファイルとデータをやり取りする場合は、データ バッファを使用します。データ バッファを使って **MicroKernel** エンジンとの間でやり取りされる情報は、実行される **Btrieve API** オペレーションによって異なります。しばしば、データ バッファには、アプリケーションとファイルの間で相互に転送される 1 つまたは複数のレコードが格納されています。しかし、**Btrieve API** オペレーションによっては、ファイル仕様やキー仕様、**MicroKernel** エンジンのバージョン情報など、その他の情報がデータ バッファに格納されることもあります。

必ず、ファイル内の最長レコードを収容できるだけの長さのデータ バッファを割り当ててください。データ バッファの割り当てサイズよりも大きな値をデータ バッファ長パラメーターに指定した場合、**MicroKernel** エンジン変更オペレーションによって、データ バッファの後に続くデータが破壊される可能性があります。

メモ： エン트리 ポイントに応じて、同じオペレーションで異なるレイアウトを使用します。**BTRV**、**BTRVID**、**BTRCALL**、および **BTRCALLID** は従来のレイアウトを使用します。**BTRVEX** および **BTRVEXID** は新しい、少し異なるレイアウトを使用します。レイアウトが異なっても、ユーザー データ レコードには影響しません。

データ バッファ長

データ バッファを必要とするオペレーションでは、アプリケーションはデータ バッファのサイズ (バイト単位) を示す変数を渡す必要があります。このデータ バッファは、オペレーションによって返されるデータを十分格納できるだけの大きさでなければなりません。

メモ： **BASIC** 開発者：データ バッファ長パラメーターとして、長整数の **ByRef** を渡す必要があります。

C、COBOL、Pascal の開発者：古い BTRV 関数の場合は、データ バッファ長パラメーターとして 2 バイト整数へのポインタを渡す必要がありますが、新しい BTRVEX および BTRVEXID 関数は 4 バイト整数を使用します。

可変長レコードを含むファイルにレコードを挿入したり、そのファイルを更新したりする場合、データ バッファ長は、ファイルを最初に作成したときに指定したレコード長に、固定長部分を超えて含まれる文字数を加算した値と等しくなければなりません。可変長レコードを取得する場合、データ バッファ長はファイル内の最長レコードに対応できる長さである必要があります。1 件のレコードが最大データ バッファ サイズより長い場合は、チャンク オペレーションを使用して、レコードの部分を操作する必要があります。

MicroKernel エンジンではデータ バッファ長パラメーターによって、データ バッファに使用可能なスペース量を判断します。割り当てたデータ バッファより長いデータ バッファ長を渡すと、MicroKernel エンジンによってメモリが上書きされる場合があります。データ バッファ長は、実際に割り当てられたデータ バッファのサイズを常に正確に表すようにしてください。

メモ：データ バッファ長は、古い BTRV 関数の場合は 2 バイト、新しい BTRVEX および BTRVEXID 関数の場合は 4 バイトです。最大データ バッファ サイズは、古い関数の場合は 64 KB、新しい関数の場合は 252 KB です。

キー バッファ

Btrieve API オペレーションでキー バッファが使用されない場合でも、アプリケーションは各 Btrieve API オペレーションにキー バッファ パラメーターを渡す必要があります。オペレーションによっては、アプリケーションがキー バッファのデータを設定する場合や、Btrieve API 関数がこれを返す場合があります。

メモ：BASIC 開発者：キー バッファ パラメーターとして文字列を渡す必要があります。キー値が整数である場合、アプリケーションでは Btrieve API 関数を呼び出す前に、MKIS ステートメントを使用してキー値を文字列に変換しておく必要があります。キーが複数のセグメントから構成されている場合は、それらを結合して 1 つの文字列変数にし、その変数をキー バッファとして渡す必要があります。

キー バッファとして渡した文字列変数が定義されたキー長より短い場合、MicroKernel エンジンからエラーが返されます。最初のアプリケーション呼び出しがキー バッファの初期化を必要としない場合は、文字列変数に SPACE\$(x) の値を割り

当てます。この x は、キーの定義されている長さを表します。アプリケーションが文字列変数に BASIC の値を割り当てるまで、その長さは 0 です。

C 開発者：キーバッファパラメーターとしてキー値を含む変数のアドレスを渡す必要があります。btitypes.h ファイルは、キーバッファを VOID ポインター (BTI_VOID_PTR) として定義しています。キーバッファのデータ型は、必要に応じてアプリケーションで定義できます。

COBOL 開発者：キーバッファパラメーターとしてレコード変数を渡す必要があります。キーが複数のセグメントから構成されている場合は、01 レベルのレコード下の個別フィールドとして、それらを正しい順序でリストします。これで、レコード全体をキーバッファとして渡すことができます。

Pascal 開発者：キーバッファパラメーターとしてキー値を含む変数を渡す必要があります。キーが複数のセグメントから構成されている場合は、レコード構造体を使ってキーに含まれる個々のフィールドを定義します。

ほとんどの環境で、アプリケーションが Btrieve API 呼び出しを実行するとき、MicroKernel エンジンではキーバッファ長を決めることができません。バッファは少なくとも、キーを作成したときに選択したキー長と同じだけの長さがあるようにしてください。そうでないと、Btrieve オペレーションにより、メモリ内でキーバッファの後に格納されているデータが破壊される可能性があります。13.0 以前の形式のファイルを使用している場合は、キーの最大長は 255 なので、キーバッファは 255 バイトにすることを勧めます。16.0 形式のファイルの場合は、1024 バイトにすることを勧めます。

キー番号

キー番号パラメーターで渡される情報は、実行しているオペレーションによって異なります。ほとんどの場合、キー番号には、特定のオペレーションが最高 119 あるキー (アクセス) パスのうちのどれに従っているかを示す値が含まれます。すべての関数で、キー番号は 0 から 118 までの範囲の値となります。

キー番号のサイズは次のように異なります。

- BTRV および BTRVID では、このパラメーターは 2 バイト整数です。
- BTRCALL、BTRCALLID、BTRCALL32、および BTRCALLID32 では、これは 1 バイトの符号付き文字 (BTI_CHAR) です。

- BTRVEX および BTRVEXID では、これは 4 バイト整数です。BTRVEX ヘコードを移行するための便宜上、キー値 128 ~ 255 は -128 ~ -1 にマップされます。これは、大きな符号なしバイト値（例：0xFF）から BTRCALL の符号付きバイト引数への変換をシミュレートします。

Btrieve API 関数がキー番号パラメーターを変更することはありません。

ファイルを開くときのモードを示す値など、その他の情報がキー番号パラメーターから渡されたり、キー番号パラメーターに返される場合もあります。

クライアント ID

クライアント ID パラメーターは、BTRVID、BTRCALLID、および BTRVEXID 関数でのみ使用されます。クライアント ID パラメーターは、MicroKernel エンジンがコンピュータ上のクライアントを区別できるようにする 16 バイト構造体のアドレスです。クライアント ID には次のような構造体を使用します。

要素	長さ (バイト単位)	説明
Filler (Filler)	12	0 に初期化します。
Service Agent ID (サービスエージェント ID)	2	MicroKernel エンジンに対するアプリケーションの各インスタンスを識別します。これは 2 文字の ASCII 値です。この ID の値は ASCII 値の AA (0x41 0x41) より大きいか等しくなければなりません。MicroKernel エンジンには次の値に特別な意味を持たせています。
		0x4140 (@A) 内部処理に使用されます。
		0xFFFF 内部処理に使用されます。
		0x4952 (RI) 内部処理に使用されます。
		0x5244 (DR) 内部処理に使用されます。
		0x4553 (SE) 0x4353 (SC) 0x4344 (DC) 0x4544 (DE) 0x5544 (DU) Scalable SQL によって生成されたクライアントを識別するために使用されます。

要素	長さ (バイト単位)	説明
		0x5257 (WR) Btrieve リクエスターによって使用されます。
Client Identifier (クライアント識別子)	2	アプリケーションの現在のインスタンス内でクライアント ID を確立します。MicroKernel エンジンでは、この一意な識別子を、並行処理およびトランザクション処理のために使用します。

キー長

キー長パラメーターは、BTRCALL、BTRCALLID、BTRVEX、および BTRVEXID でのみ使用されます。

キー長の値は次のように使用されます。

- BTRCALL および BTRCALLID の場合は、キー長は符号なし文字の型 BTI_BYTE として、割り当てられたキー バッファの長さの値を渡します。指定できる最大長は 255 です。これは、13.0 以前の形式のファイルにおけるキーの最大長です。
- BTRVEX および BTRVEXID の場合は、キー長は符号なし文字の型 BTI_INT として渡します。これは、16.0 形式のファイルの長いキーに有用です。

キー バッファを利用する場合には、次のことを考慮してください。

- 4 つの関数すべてについて、オペレーション コードによっては、指定されたキー長までのキー バッファのバイトは読み取り可能または書き込み可能である必要があります。
- BTRV および BTRVID は、255 のキー長を想定しているため、少なくともその大きさのキー バッファを提供する必要があります。
- BTRCALL および BTRCALLID の場合は、Zen クライアント コンポーネントが実際のキー長を決定しようとしたたり、場合によっては、キー バッファの小さな部分を読み書きしたりすることがあります。
- BTRVEX および BTRVEXID は、指定されたキー長をそのまま受け取ります。

Btrieve API オペレーションの要約

Btrieve API には、アプリケーションプログラムから呼び出すための 40 以上のオペレーションが用意されています。このトピックでは、これらのオペレーションの概要を説明します。詳細については、[Btrieve API オペレーション](#)を参照してください。オペレーションコード順の簡単な説明については、[Btrieve オペレーションのクイックリファレンス](#)を参照してください。

セッション固有のオペレーション

次のオペレーションを使用すると、現在のディレクトリの設定と取得、ワークステーション MicroKernel エンジンのシャットダウン、MicroKernel エンジンバージョン番号の取得、サーバー MicroKernel エンジンに接続されているクライアントの終了、トランザクションの開始、終了または中止といった処理を実行できます。複数のクライアントを処理するアプリケーションでは、これらのオペレーションは呼び出し元のクライアントに固有のものとなります。

オペレーション	コード	説明
Stop	25	ワークステーション MicroKernel エンジンを終了します（サーバーベースの MicroKernel エンジンでは使用できません）。
Version	26	MicroKernel エンジンのバージョン番号を返します。
Reset	28	クライアントによって保持されているすべてのリソースを解放します。
Set Directory	17	現在のディレクトリを指定されたパス名に設定します。
Get Directory	18	指定された論理ディスクドライブの現在のディレクトリを返します。
Begin Transaction	19 1019	論理的に関連している一連のオペレーションの開始を指定します。オペレーション 19 は排他トランザクションを開始します。オペレーション 1019 は並行トランザクションを開始します。
End Transaction	20	論理的に関連している一連のオペレーションの終了を指定します。
Abort Transaction	21	完了しなかったトランザクション中に実行されたオペレーションを取り消します。
Continuous Operation	42	アクティブな MicroKernel エンジン ファイルを閉じずに、システム バックアップを実行できるようにします。

ファイル固有のオペレーション

次のオペレーションは特定のファイルを取り扱います。このため、操作対象となるファイルを識別するためにポジションブロックパラメーターが使用されます。ファイル固有のオペレーションは次の3つのタイプに分類されます。

- ファイル アクセスおよび情報
- データ取得
- データ操作

ファイル アクセスおよび情報

次の表はファイルのアクセス オペレーションおよび情報オペレーションを示します。これらのオペレーションでは、ファイルの作成、オープン/クローズ、統計情報の取得、オーナー ネームの設定とクリア、**Continuous** オペレーション モードの開始と終了、ロック解除、インデックスの作成と削除を行えます。

オペレーション	コード	説明
Open	0	ファイルをアクセス可能な状態にします。
Close	1	ファイルをアクセス可能な状態から解放します。
Create	14	指定された特性を持つファイルを作成します。
Stat	15	ファイルおよびインデックスの特性とレコードの数を返します。
Continuous Operation	42	アクティブな MicroKernel エンジン ファイルを閉じずに、システム バックアップを実行できるようにします。
Stat Extended	65	拡張ファイルの構成要素のパスとファイル名を返し、ファイルがシステム定義のログ キーを使用しているかどうかを報告します。
Set Owner	29	ファイルにオーナー ネームを割り当てます。
Clear Owner	30	ファイルからオーナー ネームを削除します。
Unlock	27	レコードのロックを解除します。
Create Index	31	インデックスを作成します。
Drop Index	32	インデックスを削除します。

データ取得

次の表はデータ取得オペレーションを示します。これらのオペレーションでは、指定した条件に基づいて単一レコードまたはレコードのセットの取得を行えます。Btrieve API は、インデックスパスによる論理的な位置、または物理的な位置に基づくデータ検索をサポートしています。レコードのアクセスの詳細については、『*Zen Programmer's Guide*』を参照してください。

さらに、オペレーションコードにバイアスを適用して、マルチクライアント状況にあるファイルやレコードのロックを制御することもできます。複数のクライアントのサポートの詳細については、『*Zen Programmer's Guide*』を参照してください。

オペレーション	コード	説明
インデックス ベースの（論理）データ取得		
Get Equal	5	指定されたインデックスパス内で、指定されたキー値と合致するキー値を持つ最初のレコードを返します。
Get Next	6	インデックスパスで現在のレコードの次にあるレコードを返します。
Get Previous	7	インデックスパスで現在のレコードの前にあるレコードを返します。
Get Greater Than	8	指定されたインデックスパス内で、指定されたキー値より大きいキー値を持つ最初のレコードを返します。
Get Greater Than or Equal	9	指定されたインデックスパス内で、指定されたキー値より大きいまたは等しいキー値を持つ最初のレコードを返します。
Get Less Than	10	指定されたインデックスパス内で、指定されたキー値より小さいキー値を持つ最初のレコードを返します。
Get Less Than or Equal	11	指定されたインデックスパス内で、指定されたキー値より小さいまたは等しいキー値を持つ最初のレコードを返します。
Get First	12	指定されたインデックスパスの先頭のレコードを返します。
Get Last	13	指定されたインデックスパスの末尾のレコードを返します。
Get Next Extended	36	インデックスパスで現在のレコードの次にある1つまたは複数のレコードを返します。フィルター条件を適用できます。
Get Previous Extended	37	インデックスパスで現在のレコードの前にある1つまたは複数のレコードを返します。フィルター条件を適用できます。

オペレーション	コード	説明
Get Key	+50	実際のレコードを返すことなく、ファイル内に特定のキー値が存在するかどうかを検出します。
Get By Percentage	44	指定されたパーセンテージ値によって示される位置の最も近くにあるレコードを返します。
Find Percentage	45	ファイル内における現在のレコードの位置に基づいたパーセンテージ値を返します。
非インデックス ベースの (物理) データ取得		
Get Position	22	現在のレコードの位置を返します。
Get Direct/Chunk	23	指定された位置にあるレコードの指定部分 (チャンク) からデータを返します。
Get Direct/Record	23	指定された位置にあるレコードを返します。
Step Next	24	物理的に現在のレコードの次にあるレコードを返します。
Step First	33	ファイル内で物理的な先頭位置にあるレコードを返します。
Step Last	34	ファイル内で物理的な末尾位置にあるレコードを返します。
Step Previous	35	物理的に現在のレコードの前にあるレコードを返します。
Step Next Extended	38	物理的に現在のレコードの次の位置から 1 つまたは複数の連続するレコードを返します。フィルター条件を適用できます。
Step Previous Extended	39	物理的に現在のレコードの前の位置から 1 つまたは複数の連続するレコードを返します。フィルター条件を適用できます。
Get By Percentage	44	指定されたパーセンテージ値によって示される位置の最も近くにあるレコードを返します。
Find Percentage	45	ファイル内における現在のレコードの位置に基づいたパーセンテージ値を返します。

オペレーション	コード	説明
並行制御バイアス（適切なオペレーション コードに追加）		
単一レコードの読み取りウェイト ロック	+100	一度に1つのレコードだけをロックします。レコードが既にロックされている場合、クライアントによってオペレーションが再試行されます。
単一レコードの読み取りノーウェイト ロック	+200	一度に1つのレコードだけをロックします。レコードが既にロックされている場合、MicroKernel エンジンからエラー ステータス コードが返されます。
複数レコードの読み取りウェイト ロック	+300	同一ファイルの複数のレコードを並行的にロックします。レコードが既にロックされている場合、クライアントによってオペレーションが再試行されます。
複数レコードの読み取りノーウェイト ロック	+400	同一ファイルの複数のレコードを並行的にロックします。レコードが既にロックされている場合、MicroKernel エンジンからエラー ステータス コードが返されます。
書き込みノーウェイト ページ ロック	+500	並行トランザクションで、変更しようとしたページがアクティブな別の並行トランザクションによって既に変更されている場合、MicroKernel エンジンにウェイトしないように指示します。このバイアスは、どのレコード読み取りロック バイアス (+100、+200、+300、+400) とでも組み合わせることができます。

データ操作

次の表は、データを挿入、更新、削除するためのオペレーションを示します。

オペレーション	コード	説明
Insert	2	ファイルに新しいレコードを挿入します。
Update	3	現在のレコードを更新します。
Delete	4	ファイルから現在のレコードを削除します。
Insert Extended	40	ファイルに1つまたは複数のレコードを挿入します。
Update Chunk	53	現在のレコードの指定された部分（チャンク）を更新します。このオペレーションでは、レコードにデータを追加したり、レコードを切り詰めることもできます。

オペレーション	コード	説明
Get Next Delete Extended	85	論理位置の直後からファイルの末尾までで、フィルター条件に一致するレコードを削除します。
Get Previous Delete Extended	86	論理位置の直前からファイルの先頭までで、フィルター条件に一致するレコードを削除します。
Step Next Delete Extended	87	物理位置の直後からファイルの末尾までで、フィルター条件に一致するレコードを削除します。
Step Previous Delete Extended	88	物理位置の直前からファイルの先頭までで、フィルター条件に一致するレコードを削除します。

サポートされないオペレーション

MicroKernel エンジン トレースや SDK ヘッダー ファイルを見たときに Btrieve API オペレーションのリファレンスに記載されていないオペレーションがあるかもしれません。これらは Zen が内部的に使用するもので、アプリケーションで使用する必要はありません。以下に示すオペレーションはサポートされません。

オペレーション	コード	説明
B_MISC_DATA	41	MicroKernel エンジンが使用するために予約されています。
B_EXTEND	16	SQL エンジンが使用するために予約されています。
Btrieve による（ネストされた）Begin Transaction	2019	MicroKernel エンジンが使用するために予約されています。

Btrieve API オペレーションの実行における一連のイベント

Btrieve API オペレーションを実行するには、アプリケーションで以下のタスクを完了する必要があります。

1. オペレーションが要求する必要条件をすべて満たします。たとえば、ファイルの I/O オペレーションを実行する前に、対象となるファイルで **Open (0)** オペレーションを実行し、そのファイルを使用可能な状態にしておく必要があります。

2. **Btrieve API** オペレーションが要求するパラメーターを初期化します。パラメーターとはプログラム変数またはデータ構造体のことで、その型とサイズは、**MicroKernel** エンジンがオペレーションに期待する特定の値に対応していなければなりません。
将来の互換性を維持するため、使用するかどうかに関係なく、すべてのパラメーターを初期化してください。**INTEGER** 型のパラメーターの場合、値をバイナリ **0** に設定します。文字列配列の場合、バッファーへのポインターを渡します。この場合は、バッファーの先頭バイトをバイナリ **0** に初期化します。
3. 適切な **Btrieve API** 関数を呼び出します ([Btrieve API 関数](#)を参照してください)。
4. 関数呼び出しの実行結果を評価します。すべての **Btrieve API** オペレーションからステータスコードが返されます。アプリケーションではステータスコードをチェックし、適切な操作を行う必要があります。また、オペレーションからはその目的に基づいて、個々のパラメーターにデータやその他の情報も返されます。

Btrieve API オペレーション

ここでは、アプリケーションが **Btrieve API** を使って実行できるオペレーションについて説明します。各オペレーションに、次の情報があります。

- オペレーションの名前、コードおよび説明。
- パラメーター - オペレーションを実行する際に、6 個のパラメーターのうちどの値がアプリケーションから送られ、アプリケーションに返されるのかを表に示しています。「送り値」パラメーターは、アプリケーションからオペレーションに送られます。「戻り値」パラメーターは、オペレーション完了後にオペレーションからアプリケーションに返されます。
- 前提条件 - オペレーションを正常に実行させるためにアプリケーションが満たすべき条件を示します。
- 手順 - オペレーションが要求するパラメーターを初期化する手順を示します。
- 詳細 - オペレーションについての追加情報を示します。
- 結果 - オペレーションが正常に終了した場合と、何らかのエラーが発生した場合の結果を示します。それぞれのオペレーションは、オペレーションの結果をアプリケーションに通知するステータスコードを返します。ステータスコード 0 は、オペレーションが正常に終了したことを示します。0 以外のステータスコードは、通常は何らかのエラーが発生したことを示します。ただし、0 以外のステータスコードの中には、単に情報を提供するだけで、関連するオペレーションが正常に終了したときにも示されるものがあります。たとえば、ステータスコード 60 は指定したリジェクト カウントに到達したことを意味します。
- ポジショニング - オペレーションがファイル内のレコードの論理カレンシーまたは物理カレンシーに与える影響を示します。

Btrieve API オペレーションの一覧をアルファベット順に示します。

- [Abort Transaction \(21\)](#)
- [Begin Transaction \(19 または 1019\)](#)
- [Clear Owner \(30\)](#)
- [Close \(1\)](#)
- [Continuous Operation \(42\)](#)
- [Create \(14\)](#)

-
- [Create Index \(31\)](#)
 - [Delete \(4\)](#)
 - [Drop Index \(32\)](#)
 - [End Transaction \(20\)](#)
 - [Find Percentage \(45\)](#)
 - [Get By Percentage \(44\)](#)
 - [Get Direct/Chunk \(23\)](#)
 - [Get Direct/Record \(23\)](#)
 - [Get Directory \(18\)](#)
 - [Get Equal \(5\)](#)
 - [Get First \(12\)](#)
 - [Get Greater Than \(8\)](#)
 - [Get Greater Than or Equal \(9\)](#)
 - [Get Key \(+50\)](#)
 - [Get Last \(13\)](#)
 - [Get Less Than \(10\)](#)
 - [Get Less Than or Equal \(11\)](#)
 - [Get Next \(6\)](#)
 - [Get Next Delete Extended \(85\)](#)
 - [Get Next Extended \(36\)](#)
 - [Get Position \(22\)](#)
 - [Get Previous \(7\)](#)
 - [Get Previous Delete Extended \(86\)](#)
 - [Get Previous Extended \(37\)](#)
 - [Insert \(2\)](#)
 - [Insert Extended \(40\)](#)
 - [Login/Logout \(78\)](#)

-
- [Open](#) (0)
 - [Reset](#) (28)
 - [Set Directory](#) (17)
 - [Set Owner](#) (29)
 - [Stat](#) (15)
 - [Stat Extended](#) (65)
 - [Step First](#) (33)
 - [Step Last](#) (34)
 - [Step Next](#) (24)
 - [Step Next Delete Extended](#) (87)
 - [Step Next Extended](#) (38)
 - [Step Previous](#) (35)
 - [Step Previous Delete Extended](#) (88)
 - [Step Previous Extended](#) (39)
 - [Stop](#) (25)
 - [Unlock](#) (27)
 - [Update](#) (3)
 - [Update Chunk](#) (53)
 - [Version](#) (26)

Abort Transaction (21)

Abort Transaction オペレーション (B_ABORT_TRAN) では、現在のトランザクションを終了し、このトランザクションの開始以降に実行されたすべてのオペレーションの結果を削除します。また、トランザクションによって設定されたすべてのファイルとレコードのロックを解除します。

パラメーター

	オペレーション コード	ポジション ブロック	データ バッ ファー	データ バッ ファー長	キー バッ ファー	キー番号
送り値	X					
戻り値						

前提条件

Abort Transaction オペレーションを発行する前に、Begin Transaction (19 または 1019) オペレーションが正常に実行されている必要があります。

手順

オペレーション コードを 21 に設定します。MicroKernel エンジンでは、オペレーション コード以外の Abort Transaction 呼び出しパラメーターはすべて無視されます。

結果

Abort Transaction オペレーションが正常に終了した場合は、MicroKernel エンジンからステータス コード 0 が返されます。トランザクションの開始以降に発行されたすべての Insert、Update、および Delete オペレーションの結果がファイルから削除されます。

Abort Transaction オペレーションが失敗した場合は、MicroKernel エンジンから次のステータス コードのいずれかが返されます。

- 36 アプリケーションでトランザクション エラーが発生しました。
- 39 End Transaction または Abort Transaction オペレーションを実行する前に、Begin Transaction オペレーションを実行する必要があります。

ポジショニング

Abort Transaction オペレーションは、ファイルのカレンシー情報にまったく影響しません。

Begin Transaction (19 または 1019)

Begin Transaction オペレーション (B_BEGIN_TRAN) では、トランザクションの開始を定義します。トランザクションは、複数の Btrieve API オペレーションを単一イベントとして実行する必要がある場合に有用です。たとえば、いくつかのオペレーションのうち少なくとも 1 つでもオペレーションが正常に終了しないと、データベースの論理的整合性が保てなくなる場合には、トランザクションを使用してください。

一連のオペレーションを Begin Transaction と End Transaction オペレーションで囲んでおくと、明示的な End Transaction (20) でトランザクションの完了を要求しない限り、MicroKernel エンジンはその間に実行された複数のオペレーションをすべて取り消すことができます。トランザクション内で加えられた変更は、End Transaction オペレーションの実行が正常に終了するまで、ほかのユーザーからは見ることはできません。

MicroKernel エンジンは、ファイルやパフォーマンスに多大な影響を与えるという理由で、いくつかのオペレーションについてトランザクション中の実行を禁止しています。この特定のオペレーションとは、Set Owner (29)、Clear Owner (30)、Create Index (31)、および Drop Index (32) です。

パラメーター

	オペレーション コード	ポジション ブロック	データ バッ ファー	データ バッ ファー長	キー バッ ファー	キー番号
送り値	X					
戻り値						

前提条件

Begin Transaction を発行する前に、先行するトランザクションを終了または中止しておく必要があります。

手順

オペレーション コードを 19 に設定して排他トランザクションを開始するか、1019 に設定して並行トランザクションを開始します。MicroKernel エンジンでは、オペレーション コード以外の Begin Transaction 呼び出しパラメーターはすべて無視されます。

Begin Transaction オペレーションでは、デフォルトのロック バイアスを指定できます。

- +100 - 単一レコード ウェイト ロック
- +200 - 単一レコード ノーウェイト ロック
- +300 - 複数レコード ウェイト ロック
- +400 - 複数レコード ノーウェイト ロック

並行トランザクションの **Begin Transaction** オペレーションでは、+500 をオペレーションコードに追加できます (1519)。追加すると、トランザクション内の **Insert**、**Update**、**Delete** の各オペレーションが、MicroKernel エンジンによって再試行されなくなります。

さらに、+500 バイアスをデフォルトのロック バイアスと組み合わせることもできます。たとえば、 $1019 + 500 + 200$ (1719) を使用すると、**Insert**、**Update** および **Delete** オペレーションの再試行を抑え、並行トランザクションが実行されます。また同時に、単一レコードの読み取りノーウェイト ロックが指定されます。

メモ： レコード ロックおよびデータ整合性については、『*Zen Programmer's Guide*』のほかに、『*Advanced Operations Guide*』に記載されている Zen サーバーを設定するための **ウェイト ロック タイムアウト** プロパティを参照してください。

結果

Begin Transaction オペレーションが正常に終了した場合は、MicroKernel エンジンからステータス コード 0 が返されます。

Begin Transaction オペレーションが失敗した場合は、MicroKernel エンジンから次のステータス コードのいずれかが返されます。

- 36 アプリケーションでトランザクション エラーが発生しました。
- 37 別のトランザクションが実行中です。

ポジショニング

Begin Transaction オペレーションは、ファイルのカレンシー情報にまったく影響しません。

Clear Owner (30)

Clear Owner オペレーション (B_CLEAR_OWNER) では、あらかじめ Set Owner オペレーションを使ってファイルに割り当ててあるオーナー ネームを削除します。ファイルがあらかじめ暗号化されている場合は、Clear Owner オペレーションの実行中に MicroKernel エンジンによってファイルの解読も行われます。詳細については、『Advanced Operations Guide』の [オーナー ネーム](#) を参照してください。

パラメーター

	オペレーション コード	ポジション ブロック	データ バッ ファー	データ バッ ファー長	キー バッ ファー	キー番号
送り値	X	X				
戻り値		X				

前提条件

- 対象となるファイルが開いており、オーナー ネームが指定されていることが必要です。
- トランザクションが実行中でないことが必要です。

手順

- オペレーション コードを 30 に設定します。
- 対象となるファイルを識別するポジション ブロックを渡します。

結果

Clear Owner オペレーションが正常に実行されると、それ以降、MicroKernel エンジンはファイルを開いたり変更したりするときにオーナー ネームを要求しなくなります。オーナーを割り当てるときにファイルのデータを暗号化した場合には、MicroKernel エンジンは Clear Owner オペレーションの実行中にデータの解読も行います。暗号化されているデータが多いほど、Clear Owner オペレーションの実行にかかる時間は長くなります。

Clear Owner オペレーションが失敗した場合は、MicroKernel エンジンから次のステータスコードのいずれかが返されます。

- 3 ファイルが開いていません。
- 41 実行しようとした操作は MicroKernel エンジン では許可されていません。

ポジショニング

Clear Owner オペレーションは、ファイルのカレンシー情報にまったく影響しません。

Close (1)

Close オペレーション (B_CLOSE) では、指定されたポジションブロックに関連付けられているファイルを閉じ、そのファイルに対して実行されたロックをすべて解除します。ファイルへのアクセスを終了するとき、アプリケーションでは必ず Close オペレーションを実行する必要があります。Close オペレーションの実行後は、もう一度 Open (0) を発行しない限り、そのファイルにはアクセスできなくなります。

トランザクション中でも、ファイルを閉じることができます。ただし、Close オペレーションを実行しても、トランザクションは終了しません。トランザクションは明示的に終了または中止する必要があります。トランザクションを中止すると、トランザクション中に行われた変更は打ち切られます。トランザクションを終了すると、変更が反映されます。

メモ： トランザクション中にファイルを閉じた場合、MicroKernel エンジンではそのファイルへの更新を正しく処理できるように、トランザクションが中止または終了されるまでファイルのオープンハンドルを保持します。ただし、そのファイルのポジションブロックをアプリケーションで使用することはできなくなります。

パラメーター

	オペレーションコード	ポジションブロック	データバッファー	データバッファー長	キーバッファー	キー番号
送り値	X	X				
戻り値						

前提条件

対象となるファイルが開いていることが必要です。

手順

1. オペレーションコードを 1 に設定します。
2. 閉じるファイルに対応する有効なポジションブロックを渡します。

結果

Close オペレーションが正常に終了した場合は、閉じたファイルに対応するポジションブロックは有効でなくなります。

Close オペレーションが失敗した場合は、ファイルが開いたままになり、MicroKernel エンジンから次のステータスコードが返されます。

- 3 ファイルが開いていません。

ポジショニング

Close オペレーションを実行すると、ファイルの物理カレンシー情報および論理カレンシー情報は消去されます。

Continuous Operation (42)

Continuous オペレーション (B_CONTINUOUS) では、アクティブな MicroKernel エンジン ファイルを閉じることなく、システム バックアップを実行できます。ファイルのバックアップ中に行った変更は、デルタ ファイルと呼ばれる一時ファイルに記憶されます。デルタ ファイルに書き込まれた変更を除き、Continuous オペレーション モードに置かれたすべてのファイルの内容がシステム バックアップの対象になります。バックアップされたファイルが Continuous オペレーション モードから抜けると、MicroKernel エンジンにより、デルタ ファイルの変更がこれらのファイルに自動的にロール インされます。

メモ： このオペレーションは、ローカル エンジンで動作しているアプリケーションにのみ使用できます。クライアント アプリケーションはリモート マシンにあるファイルに対してこのオペレーションを使用することはできません。

このオペレーションを使うと、ファイルがアクティブな状態のまま、それを安全にコピーできます。クライアント / サーバー セットアップでは、あるファイルで Continuous オペレーションを開始するクライアントは、そのファイルで Continuous オペレーションを終了するクライアントでもなければなりません。

パラメーター

	オペレーション コード	ポジション ブロック	データ バッ ファー	データ バッ ファー長	キー バッ ファー	キー番号
送り値	X		X	X		X
戻り値			X	X		

メモ： キー番号パラメーターの値が 0 (Continuous オペレーション モードを開始) または 2 (Continuous オペレーション モードを終了) の場合にのみ、データ バッファーパラメーターおよびデータ バッファー長パラメーターの値が必要です。以下に、これらのキー番号の値について説明します。キー番号パラメーターが 1 の場合、データ バッファー長は 0 である必要があります。

手順

Continuous オペレーション モードを開始するには、以下の手順を実行してください。

1. バックアップの対象となるファイルまたはファイルのセットを定義するか、目的のファイルを現在バックアップの対象として定義されているファイルのセットに追加します。
 - a. オペレーション コードを 42 に設定します。
 - b. Continuous オペレーション モードに置くファイルの名前をデータ バッファ パラメーターに入れます。サーバー名のみを除いた絶対パス名を入れます。名前をカンマで区切り、リストの最後にバイナリ 0 を付けます。

次の例は、Windows サーバーの場合の例です。

```
f:¥acct¥march.mkd,f:¥acct¥april.mkd
```

- c. 名前の長さをデータ バッファ 長パラメーターに入れます。この値は、データ バッファ 自体に入っている、バイナリ 0 を含めた実際の名前の長さ以上の値にする必要があります。たとえば、例に挙げた名前の場合、データ バッファ 長には 40 以上の値を入れることが必要です。
 - d. キー番号パラメーターを 0 に設定します。
2. バックアップを実行します。
3. Continuous オペレーション モードを終了します。
 - a. オペレーション コードを 42 に設定します。
 - b. キー番号パラメーターを 1 に設定します。

1 つまたは複数の特定のファイルで Continuous オペレーションを終了するには、キー番号パラメーターを 2 に設定し、手順 1b で説明したようにファイル名をデータ バッファ パラメーターに入れます。さらに、手順 1c で説明したように、名前の長さをデータ バッファ 長パラメーターに入れます。

詳細

一連のファイルをバックアップするように定義する場合は、以下の点に留意してください。

-
- **MicroKernel** エンジンは、データ バッファーにファイル名が入っていてもエラーと見なしません。ファイル名が見つからない場合、**MicroKernel** エンジンは **Continuous** オペレーションで何の動作も行いません。
 - データ バッファーに重複するファイル名が存在しても、**Continuous** オペレーションの動作状況には影響しません。**MicroKernel** エンジンでは、指定したファイルを 1 度だけ **Continuous** オペレーション モードに置きます。
 - 同じディレクトリに、ファイル名が同一で拡張子のみが異なるようなファイルを置かないでください。たとえば、**Invoice.btr** という名前のデータ ファイルと **Invoice.mkd** という名前のファイルが同じディレクトリ内にあってはなりません。このような制限が設けられているのは、データベース エンジンがさまざまな機能でファイル名のみを使用し、ファイルの拡張子を無視するためです。**Continuous** オペレーションでは、デルタ ファイル名として、対応するファイルの名前に拡張子「.^」を付けた名前を使用します。**MicroKernel** エンジンは両方のファイルについて同一のデルタ ファイルに書き込もうとするため、データの破損につながるか、またはステータス 85 が返される可能性があります。また、これらのファイルが **Continuous** オペレーション モードに置くファイルの大きなリストの一部であったとしても、この状況が発生する場合には、ファイルは一切 **Continuous** オペレーション モードに置かれません。
 - アプリケーションでは、**Continuous** オペレーションを反復的に呼び出して、**Continuous** オペレーション モードに置くファイルのリストにファイル名を追加できます。ただし、リレーショナル エンジンによってファイルの一部に参照整合性 (RI) 制約が設定されている場合は、この操作により、バックアップが壊れる可能性があります。参照整合性制約と関連付けられているファイルは、単独の **Continuous** オペレーション呼び出しで渡す必要があります。

既に **Continuous** オペレーション モードに入っているファイルを指定すると、**MicroKernel** エンジンからステータス コード 88 が返されます。

Continuous オペレーションを呼び出すサーバー ベースのアプリケーションを作成する場合は、必ず **BTRVID** を呼び出し、有効なクライアント ID を使用することで、同一クライアントのもとで **Continuous** オペレーションの開始と終了を行えるようにします。

Btrieve API では、**BTRVID** 関数を使ってバックアップ セットごとに異なるクライアント ID を指定し、複数のバックアップ セットを定義できます。ただし、同じファイルを 2 つのセットに入れることはできません。

MicroKernel エンジンによってデルタ ファイルからデータ ファイルに変更内容がロール インされているときでも、ユーザーは通常の場合と同じように、MicroKernel エンジン ファイルの更新、挿入および読み取りを引き続き実行できます。MicroKernel エンジン は挿入作業で必要となれば、変更内容のロール イン中でもデルタ ファイルに新しい ページを追加します。このため、変更内容が失われることはありません。

メモ： デルタ ファイルは決して手動で削除しないでください。

アプリケーションで BTRV 関数を使用する場合は、ファイルが Continuous オペレーション モードに入っているときにそのアプリケーションをアンロードしないでください。アンロードすると、対象となるファイルを Continuous オペレーション モードから削除できなくなることがあります。これは、対象となるファイルのオーナーとして MicroKernel エンジンが当初割り当てたデフォルトのクライアント ID が、別のアプリケーションに再割り当てされてしまう可能性があるからです。MicroKernel エンジンでは対象となるファイルの正しいオーナーがわからなくなるため、それらのファイルを Continuous オペレーション モードから削除できなくなってしまう。

Continuous オペレーション モードに入るとき、または MicroKernel エンジンによってデルタ ファイルからデータ ファイルに変更内容がロール インされている最中にシステムがクラッシュした場合は、システムの再起動後初めてそのファイルを開くときに、MicroKernel エンジンによってすべての変更内容がファイルにロール インされます。

結果

Continuous オペレーションが正常に終了した場合、MicroKernel エンジンからステータス コード 0 が返されますが、データ バッファーおよびデータ バッファー長パラメーターには何の値も返されません。

このオペレーションが失敗した場合は、MicroKernel エンジンから次のステータス コードのいずれかが返されます。

- 11 指定されたファイル名が無効です。
- 12 MicroKernel エンジンでは指定されたファイルを見つけられません。
- 41 実行しようとした操作は MicroKernel エンジン では許可されていません。
- 51 オーナー ネームが無効です。
- 88 アプリケーションでモードの不一致エラーが発生しました。
- 91 アプリケーションでサーバー エラーが発生しました。

上記のコードに加え、アプリケーションにはステータスコード 18 のような標準の I/O エラーが返されることがあります。

MicroKernel エンジンから 0 以外のステータスコードが返される場合、Continuous オペレーションは、エラーを発生させた入力文字列の一部をデータバッファーに返します。入力文字列が使用されていない場合、データバッファーにはエラーの原因となったファイル名が返されます。データバッファー長には、データバッファー内の出力文字列の長さが反映されます。この場合は、エラーの原因となったファイル名の長さが入ります。

ポジショニング

Continuous オペレーションを実行しても、ファイルにカーソルは確立しません。

Create (14)

Create オペレーション (B_CREATE) では、アプリケーション内から新しいデータファイルを作成します。Create オペレーションにはファイルの削除または名前変更ができるサブファンクションもあります ([Create オペレーションによる削除および名前変更サブファンクション](#)を参照)。

メモ： 同じディレクトリに、ファイル名が同一で拡張子のみが異なるようなファイルを置かないでください。たとえば、同じディレクトリ内のデータファイルの1つに Invoice.btr、もう1つに Invoice.mkd という名前を付けてはいけません。このような制限が設けられているのは、データベースエンジンがファイル名のみを使用し、ファイルの拡張子を無視する場合もあるからです。この場合、ファイルの拡張子だけが異なるファイルは、データベースエンジンでは同一のものであると認識されます。

パラメーター

	オペレーションコード	ポジションブロック	データバッファー	データバッファー長	キーバッファー	キー番号
送り値	X		X	X	X	X
戻り値						

前提条件

既存のファイルを基に空のファイルを作成する場合は、Create オペレーションを実行する前にその既存ファイルを閉じておく必要があります。

手順

1. オペレーションコードを 14 に設定します。
2. [詳細](#)の説明に従い、データバッファーにファイル仕様、キー仕様、およびオルタネート コーディング シーケンス (ACS) を設定します。ファイル仕様とキー仕様の値はすべて、バイナリ形式でなければなりません。
3. データバッファー長を設定します。これは、Create の指定を含むバッファーの長さであり、ファイルのレコード長ではありません。

-
- キーバッファにファイルのパス名を設定します。パス名の終端は空白かバイナリゼロにします。パス名は、ボリューム名および終端文字を含めて、255 文字までの範囲で指定することができます。

詳細については、『*Getting Started with Zen*』の [Zen リクエスターでサポートするネットワークパスの形式](#) を参照してください。『*Zen Programmer's Guide*』の [データベース URI](#) も参照してください。

- キー番号パラメーターの値を、[キー番号](#)の選択肢のいずれかを使って設定します。

詳細

データバッファには、作成するファイルの仕様とキーの仕様を格納します。Create (14) と Stat (15) は同様のデータバッファ構造体を使用するため、それらのわずかな違いも含め、ここでまとめて説明されています。以下の表は、BTRV タイプおよび BTRVEX タイプのエントリポイントに関する情報の構造体を示しています。どちらのタイプも [Btrieve API 関数](#) で説明されています。

2つのタイプのエントリポイント間には、仕様内の要素の順序に違いがあることに留意してください。詳細については、[キー仕様ブロック](#) および [ファイル仕様ブロック](#) を参照してください。以下の表に示すように、データバッファにはファイル仕様と、0 個以上のキー仕様ブロック、0 個以上の ACS ブロックを格納します。

Create および Stat で使用される BTRV タイプのエントリポイントのファイル仕様

Create および Stat で使用される BTRV タイプのエントリポイントのファイル仕様	データ型 ¹	バイト番号
論理固定レコード長。すべてのレコードフィールドの結合サイズです。 ³	Short Int ³	0 – 1
ページサイズ。 ページサイズ を参照してください。	Short Int ³	2 – 3
キー（インデックス）数	Byte	4
ファイルバージョン。 ⁴ ファイルバージョン を参照してください。	Byte	5
レコード数。Create (14) オペレーションの場合、下位バージョンとの互換性を保つために 0 に初期化します。	Int ³	6 – 9

Create および Stat で使用される BTRV タイプのエントリ ポイントのファイル仕様	データ型 ¹	バイト番号
ファイルフラグ。ファイル属性を設定します。 ファイルフラグ を参照してください。	Short Int ³	10 – 11
追加ポインター数。Create (14) の場合は、将来のキーの追加のために予約する重複ポインター数です。Stat (15) の場合は、残っているポインター数です。予約重複ポインターフラグと共に使用されます。	Byte	12
物理ページサイズ。圧縮フラグが設定されている場合に使用されます。値は、512 バイトブロックの数です。	Byte	13
プリアロケート ページ数。事前に割り当てられるページ数です。ページプリアロケーションと共に使用されます。Stat (15) の場合は、未使用の空きページ数を返します。	Short Int ³	14 – 15

¹ 特に指定がない場合、すべてのデータ型は符号なしです。

² 可変長レコードを持つファイルの場合、論理レコード長はレコードの固定長部分のみを指します。

³ Integer は「リトルエンディアン」のバイト順、つまり、下位バイトから上位バイトへ記録する方式で格納する必要があります。

⁴ Stat (15) のキー番号が 0 の場合は、0 として返されます。

Create および Stat で使用される BTRVEX タイプのエントリ ポイントのファイル仕様

Create および Stat で使用される BTRVEX タイプのエントリ ポイントのファイル仕様	データ型 ¹	バイト番号
論理固定レコード長。すべてのレコード フィールドの結合サイズです。	Short Int ³	0 – 1
ページサイズ。 ページサイズ を参照してください。	Short Int ³	2 – 3
ファイルフラグ。ファイル属性を設定します。 ファイルフラグ を参照してください。	Short Int ³	4 – 5
予約済み下位バージョンとの互換性を保つために 0 に初期化します。	Short Int ³	6 – 7
レコード数。Create (14) オペレーションの場合、下位バージョンとの互換性を保つために 0 に初期化します。	Long Long Int ³	8 – 15

Create および Stat で使用される BTRVEX タイプのエントリ ポイントのファイル仕様	データ型 ¹	バイト番号
キー（インデックス）数	Short Int ³	16 – 17
ファイルバージョン。 ⁴ ファイルバージョン を参照してください。	Short Int ³	18 – 19
追加ポインター数。Create（14）の場合は、将来のキーの追加のために予約する重複ポインター数です。Stat（15）の場合は、残っているポインター数です。予約重複ポインターフラグと共に使用されます。	Byte	20
物理ページサイズ。圧縮フラグが設定されている場合に使用されます。値は、512 バイト ブロックの数です。	Byte	21
プリアロケート ページ数。事前に割り当てられるページ数です。ページプリアロケーションと共に使用されます。Stat（15）の場合は、未使用の空きページ数を返します。	Short Int ³	22 – 23
予約済み下位バージョンとの互換性を保つために 0 に初期化します。	Long Long Int ³	24 – 31

¹ 特に指定がない場合、すべてのデータ型は符号なしです。

² 可変長レコードを持つファイルの場合、論理レコード長はレコードの固定長部分のみを指します。

³ Integer は「リトル エンディアン」のバイト順、つまり、下位バイトから上位バイトへ記録する方式で格納する必要があります。

⁴ Stat（15）のキー番号が 0 の場合は、0 として返されます。

ファイル仕様ブロック

データ バッファの先頭 16 バイトまたは 32 バイトにはファイル仕様を格納します。バイト番号は 0 から始まります。レコード長、ページサイズ、およびインデックス数の情報を整数で格納します。

論理固定レコード長

論理レコード長は、ファイル内の固定長データのバイト数です。論理レコード長には可変長データを含めないでください。

ページサイズ

ページサイズはファイル形式のバージョンによって決まっています。『Zen Programmer's Guide』の[ページサイズの選択](#)を参照してください。次の表は、さまざまなファイル形式バージョンのページサイズの例を示しています。

説明	ファイル形式バージョン	データ型 ¹	バイト番号	値の例 ²
ページサイズ	6.x から 9.0	Short Int ³	2 – 3	512
	6.x から 9.5	Short Int ³	2 – 3	1024
	6.x から 9.0	Short Int ³	2 – 3	1536
	6.x から 9.5	Short Int ³	2 – 3	2048
	6.x から 9.0	Short Int ³	2 – 3	3072
	6.x から 9.0	Short Int ³	2 – 3	3584
	6.x 以上	Short Int ³	2 – 3	4096
	9.0 以上	Short Int ³	2 – 3	8192
	9.5 以上	Short Int ³	2 – 3	16384

ほとんどのファイルでは最小サイズの 4096 バイトが最も効率的です。微調整を行う場合は、『Zen Programmer's Guide』の[ページレベル圧縮を用いたファイルの作成](#)を参照してください。

9.5 以降の形式のファイルを作成する場合、指定された論理ページサイズがそのファイル形式で有効でなければ、MicroKernel は指定値の次に大きな有効値があるかどうかを調べ、存在する場合はその値に切り上げます。それ以外の値やファイル形式の場合、オペレーションはステータスコード 24 で失敗します。古いバージョンのファイル形式では、切り上げは行われません。

¹ 特に指定がない場合、すべてのデータ型は符号なしです。

² 簡素化を図るため、数値以外の値の例は C アプリケーションの場合です。

³ Integer は「リトルエンディアン」のバイト順、つまり、下位バイトから上位バイトへ記録する方式で格納する必要があります。

レコード数

ファイル内のレコード数です。この値は Stat (15) によって返されます。Create (14) では、このフィールドを 0 に設定します。

キー数

インデックス数はファイルに対して定義しているキーの数です。キーセグメント数ではありません。データオンリーファイルを作成するには、インデックス数を 0 に設定します。

ファイルバージョン

作成する MicroKernel エンジンのファイルバージョンです。以前のリリースでは、MicroKernel エンジンでは 2 バイト整数を使って Create オペレーションでインデックス数を取得していました。インデックスの最大数は 119 であるため、この整数の上位バイトは常に 0 でした。この上位バイトは、これまで Stat (15) オペレーションでファイルバージョンを返すために使用されてきましたが、Create でファイルバージョンを指定するために使用できるようになりました。これにより、以前のアプリケーションでエラーが発生することはありません。Create でサポートされるファイルバージョンは 6.0、7.0、8.0、9.0、9.5、13.0、および 16.0 です。これらは、それぞれ 16 進数の値 0x50、0x60、0x70、0x80、0x90、0x95、0xD0、0xD3 で特定のバイトに示されます。次

の表は、さまざまなファイル形式バージョンのファイルバージョンフラグの値を示しています。

説明	データ型 ¹	バイト番号	値の例 ²
ファイルバージョン	BTRV タイプ : Byte BTRVEX タイプ : Short Int	5	バージョン 0x60
		18 - 19	6.0
			バージョン 0x70
			7.0
			バージョン 0x80
			8.0
			バージョン 0x90
			9.0
			バージョン 0x95
	9.5		
	バージョン 0xD0		
	13		
	バージョン 0xD3		
	16		
	データベースエンジン のデフォルトを使用	0x00	

¹ 特に指定がない場合、すべてのデータ型は符号なしです。

² 簡素化を図るため、数値以外の値の例は C アプリケーションの場合です。

追加ポインター数

Create (14) の場合は、将来のキーの追加のために予約する重複ポインター数です。Stat (15) の場合は、残っているポインター数です。予約重複ポインターフラグと共に使用されます。このフラグを使用しないときは、このフィールドを 0 に設定します。

物理ページ サイズ

ページ圧縮ファイルフラグが設定されている場合に使用されます。ページ圧縮フラグが指定されていない場合は、このフィールドを 0 に設定します。このフィールドは、以前は「未使用」とマークされていました。

バージョン 6.x 以降のデータ ファイルでは、論理ページは物理ページに割り当て、ページアロケーション テーブル (PAT) に格納します。物理ページのサイズは論理ページのサイズと同一です。ページ圧縮は 9.5 以降のファイル形式で使用できます。データベース ページはページレベルで圧縮されます。各論理ページは、1 つ以上の物理ページ単位の圧縮されます。これら個々の物理ページのサイズは、1 論理ページよりも小さくなります。

物理ページ サイズ フィールドを使用して、ファイルで使用する物理ページ サイズを指定できます。このフィールドで指定する値は、使用される実際の物理ページ サイズを決定するため、512 の倍数にします。0 を指定すると、エンジンは物理ページ サイズのデフォルト値の 512 を使用します。

物理ページ サイズに指定された値は、論理ページ サイズに指定された値よりも大きくすることはできません。物理ページ サイズに指定された値の方が大きい場合、エンジンは論理ページ サイズと同じになるようその値を切り捨てます。論理ページ サイズは物理ページ サイズの倍数になっていなければなりません。倍数でない場合、その論理ページ サイズの値は物理ページ サイズの値のちょうど倍数になるよう切り捨てられます。このような操作の結果として、論理ページ サイズと物理ページ サイズの値が同じになった場合、ページレベルの圧縮はこのファイルに適用されません。『*Zen Programmer's Guide*』の [ページ レベル圧縮を用いたファイルの作成](#)も参照してください。

ファイルフラグ

ファイルフラグワードのビットをセットして、ファイル属性を指定します。次の表に、ファイルフラグの値の2進、16進、および10進表記を示します。

属性	定数	2進数	16進数	10進数
可変長レコード	VAR_RECS	0000 0000 0000 0001	1	1
ブランク トランケーション	BLANK_TRUNC	0000 0000 0000 0010	2	2
ページプリアロケーション	PRE_ALLOC	0000 0000 0000 0100	04	4
データ圧縮	DATA_COMP	0000 0000 0000 1000	08	8
キーオンリーファイル	KEY_ONLY	0000 0000 0001 0000	10	16
インデックスバランス	BALANCED_KEYS	0000 0000 0010 0000	20	32
10% 空きスペース スレッシュホルド	FREE_10	0000 0000 0100 0000	40	64
20% 空きスペース スレッシュホルド	FREE_20	0000 0000 1000 0000	80	128
30% 空きスペース スレッシュホルド	FREE_30	0000 0000 1100 0000	C0	192
予約重複ポインター	DUP_PTRS	0000 0001 0000 0000	100	256
システムデータを含める 1	INCLUDE_SYSTEM_DATA	0000 0010 0000 0000	200	512
システム データを含めない	NO_INCLUDE_SYSTEM_DATA	0001 0010 0000 0000	1200	4608
キー番号	SPECIFY_KEY_NUMS	0000 0100 0000 0000	400	1024
VAT の使用	VATS_SUPPORT	0000 1000 0000 0000	800	2048

属性	定数	2進数	16進数	10進数
ページ圧縮の使用 ²	PAGE_COMPRESSED	0010 0000 0000 0000	2000	8192
システムデータ v2 を含める ³	INCLUDE_SYSTEM_DATA2	0100 0000 0000 0000	4000	16384

¹ システム データをファイルに含めるかどうかを指定していない場合、Btrieve API はサーバーの互換性プロパティのその時点の「システム データ」設定を使用します。

² ページ レベル圧縮でのみ使用します。「物理ページ サイズ」キー仕様と組み合わせて使用されます。『Zen Programmer's Guide』の[ページ レベル圧縮を用いたファイルの作成](#)を参照してください。

³ このフラグ値を使用する場合は、キー オンリーファイルのNO_INCLUDE_SYSTEM_DATAも指定します。

互換性のないフラグの使用は避けてください。同一ビット位置を使用するフラグ間には互換性はありません。未使用のビットは将来使用するために予約されています。これらのビットを0に設定します。

ファイルの属性を組み合わせるには、対応するファイルフラグの値を加算します。たとえば、可変長レコードを含むことができ、ブランク トランケーションを行うファイルを指定するには、ファイルフラグワードを3 (2+1) に初期化します。可変長レコードのビットが0に設定されている場合、MicroKernel エンジンではブランク トランケーションおよび空きスペース スレッシュホールドのビットを無視します。

ページプリアロケーションのビットを設定する場合は、ファイル仕様ブロック（アロケーション）の最後の2バイトを使用して、ファイルにプリアロケートするページ数を指定する整数値を格納してください。データ圧縮のビットを設定した場合、MicroKernel エンジンでは可変長レコードのビットが無視されます。

データベース エンジンは、システム データを使用しており、レコード長が許容最大サイズより大きいファイルについては自動的にデータ圧縮を使用します。『Zen Programmer's Guide』の[レコード長](#)を参照してください。

ファイルを作成した後でインデックスの追加が予想される場合、および、そのインデックスには重複した値が含まれるが、繰り返し重複としてマークされない場合は、重複ポインタのビットを設定します。このビットを設定すると、MicroKernel エンジンでは重複した値をリンクするポインタのためにファイルの各レコードにスペースが確保されるようになります。このようなスペースを確保することにより、特に、キーが長く、

多数のレコードが重複するキー値を持つことが予測される場合には、検索時間を短縮し、ディスク領域を節約できる場合があります。

メモ： 重複ポインタの領域は、ファイル作成後に追加されるインデックスのみを対象に予約できます。したがって、重複値へのポインタのために領域を確保するときは、この Create オペレーションの実行中に作成されるインデックスの領域は含めないでください。また、繰り返し重複キーとして指定されるキーについて、MicroKernel エンジンでは重複ポインタの領域を確保しません。

特定の番号をキーに割り当てるが必要な場合は、キー番号のビットを設定し、希望するキー番号をキー仕様ブロックの手動割り当てキー番号要素（オフセット 0x0E）に入れます。MicroKernel エンジンではキー番号が連続している必要はありません。つまり、ファイルのキー番号は飛んでいてもかまいません。キーが作成されると、MicroKernel エンジンはデフォルトで、0 から始まるキー番号のうち使用可能な最小の番号をそのインデックスに割り当てます。ただし、アプリケーションによっては、デフォルトの割り当てとは異なるキー番号が必要なこともあります。

ファイルで可変長部割り当てテーブルを使用する場合は、VAT の使用のビットを設定します。VAT を使うには、ファイルで可変長レコードを使用していることが必要です。

プリアロケート ページ数

プリアロケートするページ数を指定できます。ページプリアロケーションのファイルフラグを指定した場合にのみ、この要素を使用してください。詳細については、『Zen Programmer's Guide』の[ページプリアロケーション](#)を参照してください。

キー仕様ブロック

ファイル仕様のすぐ後に 0 個以上のキー仕様ブロックを配置します。ファイルのキーセグメントごとに 16 バイトまたは 24 バイトのキー仕様ブロックを割り当てます。キー ポジションおよびキー長の情報は整数で格納します。

次の表で示すように、許容されるキーセグメントの最大数は、ファイルのページサイズおよびファイル形式によって決まります。

ページサイズ (バイト数)	キーセグメントの最大数 (ファイルバージョン別)			
	8.x 以前	9.0	9.5	13.0、16.0
512	8	8	切り上げ ²	切り上げ ²

ページ サイズ (バイト数)	キー セグメントの最大数 (ファイル バージョン別)			
	8.x 以前	9.0	9.5	13.0、16.0
1024	23	23	97	切り上げ ²
1536	24	24	切り上げ ²	切り上げ ²
2048	54	54	97	切り上げ ²
2560	54	54	切り上げ ²	切り上げ ²
3072	54	54	切り上げ ²	切り上げ ²
3584	54	54	切り上げ ²	切り上げ ²
4096	119	119	204 ³	183 ³
8192	N/A ¹	119	420 ³	378 ³
16384	N/A ¹	N/A ¹	420 ³	378 ³

¹ "N/A" は「適用外」を意味します。

² 「切り上げ」は、ページ サイズを、ファイルバージョンでサポートされる次のサイズへ切り上げることを意味します。たとえば、512 は 1024 に切り上げられ、2560 は 4096 に切り上げるといことです。

³ 9.5 以降の形式のファイルでは 119 以上のセグメントを指定できますが、インデックスの数は 119 に制限されます。

『*Status Codes and Messages*』のステータスコード **26 : 指定されたキーの数が無効です。** および **29 : キー長が無効です。** を参照してください。

次の表は、**Create** (14) または **Stat** (15) オペレーションのキーセグメントのデータバッファ構造体を示しています。各キー仕様ブロックは、**BTRV** タイプのエントリポイントの場合は 16 バイト、**BTRVEX** タイプのエントリポイントの場合は 24 バイトです。2 つのデータ型が示されている場合、1 つめは **BTRV** で使用され、2 つめは **BTRVEX** で使用されます。オフセットは、各キーブロックについて繰り返されます。

説明	データ型 ¹	BTRV のオフセット	BTRVEX のオフセット
キー ポジション 。レコード内のキーの最初のバイトの位置。レコード内の最初のバイトは 1 です。	Short Int ²	0 - 1	0 - 1

説明	データ型 ¹	BTRV のオフセット	BTRVEX のオフセット
キー長。 バイト単位のキーの長さ。	Short Int ²	2 – 3	2 – 3
キー フラグ。 キー属性。	Short Int ²	4 – 5	4 – 5
予約済み。Create (14) では使用されません。下位バージョンとの互換性を保つために 0 に初期化します。	Short Int ²	—	6 – 7
一意キー。Create (14) では使用されません。下位バージョンとの互換性を保つために 0 に初期化します。	Int または Long Long Int ²	6 – 9	8 – 15
拡張データ型。 キー フラグで「拡張データ型を使用」を指定する場合に使用されます。	Byte または Short Int ²	10	16 – 17
ヌル値 (レガシーヌルのみ)。キー フラグで「ヌルキー (全セグメント)」または「ヌルキー (一部セグメント)」を指定する場合に使用されます。これは、キーの除外値です。レガシーヌルと真のヌルの概念については、 ヌル値 を参照してください。	Byte	11	18
予約済み。Create (14) では使用されません。下位バージョンとの互換性を保つために 0 に初期化します。	Short Int ² または Byte	12 – 13	19
手動割り当てキー番号。 ファイル属性で「キー番号」を指定する場合に使用されるキー番号。	Byte または Short Int ²	14	20 – 21
ACS 番号。 ACS (オルタネート コレーティング シーケンス) の番号です。キー フラグで「デフォルトの ACS を使用する」、「ファイル内の番号付きの ACS を使用する」または「名前付きの ACS を使用する」を指定する場合に使用されます。	Byte	15	22
予約済み。Create (14) では使用されません。下位バージョンとの互換性を保つために 0 に初期化します。	Byte	—	23

説明	データ型 ¹	BTRV のオフセット	BTRVEX のオフセット
----	-------------------	-------------	---------------

¹ 特に指定がない場合、すべてのデータ型は符号なしです。

² Integer は「リトル エンディアン」のバイト順、つまり、下位バイトから上位バイトへ記録する方式で格納する必要があります。

キー ポジション

キー ポジションは、キーまたはキー セグメントの開始位置のバイト オフセットです。ポジションは 1 からの相対になります。レコードの先頭に位置するキーは、ポジション 1 から始まります。ポジション 0 はありません。

キー長

キーまたはキー セグメントの長さです。キーの最大長は、すべてのセグメントを含めて、13.0 以前の形式のファイルでは 255 バイト、16.0 形式のファイルでは 1024 バイトです。

キー フラグ

キー フラグ ワードのビットをセットして、キー属性を指定します。次の表に、キー フラグの値の 2 進、16 進、および 10 進表記を示します。

属性	定数	2 進数	16 進数	10 進数
重複許可 (リンクされた重複はデフォルト、あるいは繰り返し重複用に REPEAT_DUPS_KEY と組み合わせる)	DUP	0000 0000 0000 0001	0x1	1
変更可能キー	MOD	0000 0000 0000 0010	0x2	2
旧形式の BINARY データ型を使用	BIN	0000 0000 0000 0100	0x4	4
旧形式の STRING データ型を使用 (ビット 2 および 8 は 0 にする必要があります)		0000 0000 0000 0000	0x0	0
ヌル キー (全セグメント)	NUL	0000 0000 0000 1000	0x8	8

属性	定数	2進数	16進数	10進数
セグメント キー	SEG	0000 0000 0001 0000	0x10	16
デフォルトの ACS を使用	ALT	0000 0000 0010 0000	0x20	32
ファイル内の番号付きの ACS を使用	NUMBERED_ACS	0000 0100 0010 0000	0x420	1056
名前付きの ACS を使用	NAMED_ACS	0000 1100 0010 0000	0xC20	3104
降順ソート	DESC_KEY	0000 0000 0100 0000	0x40	64
繰り返し重複。DUP 属性と共に使用されます	REPEAT_DUPS_KEY	0000 0000 1000 0000	0x80	128
拡張データ型を使用	EXTTYPE_KEY	0000 0001 0000 0000	0x100	256
ヌル キー (一部セグメント)	MANUAL_KEY	0000 0010 0000 0000	0x200	512
大文字小文字無視キー	NOCASE_KEY	0000 0100 0000 0000	0x400	1024

互換性のないフラグの使用は避けてください。同一ビット位置を使用するフラグ間には互換性はありません。未使用のビットは将来使用するために予約されています。これらのビットを 0 に設定します。

キーの属性を組み合わせるには、それらの値を合計します。たとえば、キーが拡張キータイプで、セグメント キーの一部であり、さらに降順に照合される場合は、キー フラグワードを 336 (256 + 16 + 64) に初期化します。

セグメント キー属性は、データ バッファ内の次のキー仕様ブロックが同一キーの次のセグメントを指すことを示します。セグメント キーについては以下の規則に従ってください。

- 同一キーのすべてのセグメントで、重複可能、繰り返し重複、変更可能、およびヌル キーの値はそれぞれ同じでなければなりません。レガシー ヌル キー属性を指定する場合は、全セグメントまたは一部セグメントのどちらの場合にも、セグメントごとに異なるヌル値を割り当てることができます。
- 同一キーのすべてのセグメントで、同一 ACS (オルタネート コレーティング シーケンス) を使用する必要があります。
- 同一キーのセグメントごとに、異なる降順ソートおよび拡張データ型の値を設定できます。

ACS は、STRING、LSTRING、ZSTRING、WSTRING、および WZSTRING 型のキーにのみ適用されます。大文字小文字の区別を無視し、かつ ACS を使用するキーを定義することはできません。あるキーの一部のセグメントにしか ACS が指定されていないファイルの場合、ACS が指定されているセグメントはその ACS に従ってソートされるのに対し、ACS が指定されていないセグメントはそれぞれの型に従ってソートされます。

拡張データ型

拡張データ型をキー仕様ブロックのバイトにバイナリ値で指定します。次の表に拡張データ型に対応するコードを示します。

データ型	コード	データ型	コード	データ型	コード
STRING	0	BFLOAT	9	CLOB	21
INTEGER	1	LSTRING	10	WSTRING	25
FLOAT	2	ZSTRING	11	WZSTRING	26
DATE	3	UNSIGNED BINARY	14	GUID	27
TIME	4	AUTOINCREMENT	15	AUTOTIMESTAMP	32
DECIMAL	5	NUMERICSTS	17	TIMESTAMP2	34
MONEY	6	NUMERICSA	18	NULL INDICATOR SEGMENT	255
LOGICAL	7	CURRENCY	19		
NUMERIC	8	TIMESTAMP	20		

拡張データ型のコード 12、13、16、および 22 から 24 までは将来使用するために予約されています。CLOB 型は Get Extended オペレーションに含まれていますが、インデックスの作成に使用することはできません。

STRING および UNSIGNED BINARY データ型は、標準型と拡張型のどちらとしても定義できます。これにより、以前のバージョンの Btrieve API を使って開発したアプリケーションとの互換性が保てる一方、新しいアプリケーションで拡張データ型を排他的に使用できるようになります。

キーに割り当てるデータ型に関し、入力したレコードがそのキーに定義されているデータ型に合っているかどうかは、MicroKernel エンジンでは確認されません。たとえば、

TIMESTAMP キーをファイルに定義することができますが、そこに文字列を格納することもできます。Btrieve API アプリケーションでは問題なく動作していても、ODBC アプリケーションで同じデータに ODBC TIMESTAMP データ型を使ってアクセスしようとする、正常に動作しないことがあります。これはおそらく、バイトの形式が異なり、タイムスタンプ値を生成するアルゴリズムが異なるからです。データ型の説明については、『SQL Engine Reference』の [Btrieve キーのデータ型](#) を参照してください。

ヌル値

キーの除外値を表します。あるキーをヌルキーとして定義した場合は、各キー セグメントのヌル値として認識させる値を MicroKernel エンジンに提供する必要があります。これは、レガシーヌルへの参照内にあり、真のヌルには影響しません。ヌルサポートの説明については、『Zen Programmer's Guide』の [ヌル値](#) を参照してください。

手動割り当てキー番号

MicroKernel エンジンでは、インデックス付きのファイルを作成するときに、アプリケーションで特定のキー番号を割り当てることができます。ファイルの各インデックスに手動でキー番号を割り当てるには、各キーの番号をキー仕様ブロックにバイナリ値で指定し、ファイルフラグワードにキー番号ビット (0x400) を設定します。

キー番号はファイルで一意であり、キー 0 から昇順に指定されていなければなりません。また、有効な値、つまり、ファイルのページサイズに対するキーの最大数よりも小さい値でなければなりません。

手動でキー番号を割り当てるという機能は、キーを削除して、その削除したキーよりも大きなキー番号を持つすべてのキーの番号を MicroKernel エンジンに付け替えさせないようにする機能と相補関係にあります。たとえば、アプリケーションからインデックスを削除し、MicroKernel エンジンにそれよりも大きな番号を持つキーの番号を付け替えないように指示を出した場合、その後でユーザーが具体的なキー番号を割り当てずに影響を受けたファイルを複製すると、複製したファイルには元のファイルとは別のキー番号が割り当てられます。

ACS 番号

特定の ACS を使用するキーの場合、キー仕様ブロックにより、キーの照合に使用する ACS 番号が示されます。ACS 番号はデータバッファ内の位置に基づいて決まります。最後のキー仕様ブロックに続く最初の ACS は、ACS 番号 0 になります。ACS 0 の次には ACS 1、その次には ACS2、というように続きます。

オルタネート コレーティング シーケンス (ACS)

照合順序 (コレーティング シーケンス) は、Create オペレーションのデータバッファで、最後のキー仕様ブロックの直後から 1 つずつ順番に現れます。以下の表で、ACS、ISR、または ICU 照合順序を指定するために使用される 265 バイトについて説明します。

ユーザー定義の ACS ファイル

文字列値を ASCII 標準とは異なる並び順でソートする ACS を作成するには、アプリケーションで次のデータバッファに直接 265 バイトを設定する必要があります。

オフセット	長さ (バイト単位)	説明
0	1	識別バイト。0xAC を指定します。
1	8	MicroKernel エンジンに ACS を識別させる、8 バイトの一意の名前。
9	256	256 バイトのマップ。マップ内の 1 バイトの位置はそれぞれ、マップ内でのその位置のオフセットと同じ値を持つコードポイントに対応します。その位置にあるバイトの値は、コードポイントに割り当てられる照合重みです。たとえば、コードポイント 0x61 ('a') をコードポイント 0x41 ('A') と同じ重みでソートさせるには、オフセット 0x61 および 0x41 に同一の値を設定します。

ユーザー定義の ACS ファイルの例については、『*Zen Programmer's Guide*』の [オルタネート コレーティング シーケンス](#) を参照してください。

インターナショナル ソート規則 (ISR)

ISR テーブル名を指定するには、アプリケーションで次のデータ バッファに直接 265 バイトを設定する必要があります。

オフ セット	長さ (バイト 単位)	説明
0	1	識別バイト。0xAE を指定します。
1	16	MicroKernel エンジンに ISR テーブルを識別させる、16 バイトの一意の名前。ISR テーブル名の一覧については、『 <i>Zen Programmer's Guide</i> 』を参照してください。
17	248	Filler

Unicode 照合順序

ICU (International Components for Unicode) 標準に従って Unicode 照合順序を指定するには、アプリケーションで次のデータ バッファに直接 265 バイトを設定する必要があります。

オフ セット	長さ (バイト 単位)	説明
0	1	識別バイト。0xAE を指定します。
1	16	サポートされる ICU 照合順序の名前。u54-msft_enus_0 または root のいずれかになります。16 バイトになるまでスペースを詰める必要があります。
17	248	Filler

データ バッファ長

データ バッファは、ファイル仕様、キー仕様、および定義されている ACS ファイルを十分に格納できるだけの長さを持つ必要があります。このパラメーターにファイルのレコード長を指定しないでください。

たとえば、BTRV エントリ ポイントを使用して、1 セグメントのキーを 2 つと ACS を 1 つ持つファイルを作成するには、Create オペレーションのデータ バッファには、以下のように少なくとも 313 バイトの長さが必要です。

ファイル仕様	+	キー仕様	+	キー 2 仕様	+	ACS	
16	+	16	+	16	+	265	= 313

キー番号

Create オペレーションのキー番号は、同名のファイルが既に存在する場合に MicroKernel エンジンが警告を出すかどうか、また、ファイルの作成時に MicroKernel エンジンがローカル エンジンとリモート エンジンのどちらを使用するかを決定します。

次の表を使って、キー番号パラメーターの値を選択します。

CREATE オペレーション	選択しない	ローカル エンジンにファイルを作成させる	リモート エンジンにファイルを作成させる
標準の作成（ファイルが既存の場合、上書きする）	0	6	99
ファイルが既存の場合、ステータス 59 を返す	-1	7	100

Create オペレーションによる削除および名前変更サブファンクション

Create オペレーションには 2 つのサブファンクションがあり、これを使用してファイルの削除または名前変更ができます。

Pervasive.SQL v8.5 より前は、オペレーティング システムを介して MicroKernel エンジン ファイルを操作することは可能でした。これは、オペレーティング システムが Zen ユーザーに与える権限にエンジンが依存していたためです。

v8.5 で Zen データベース セキュリティが導入されると、承認されていないアクセスに対しデータベースがセキュリティで保護されている場合には、このようなオペレーティング システムのアクセス権は除去されることがあります。オペレーティング システムの権限が常に利用できるとは限らなくなるため、プログラムによるファイルの削除や移動のためのオプションは変更される可能性があります。

名前変更と削除のサブファンクションは、代替キー番号を持つ **Create** オペレーションとして実装されています。新規データ ファイルを作成する場合のようにファイル仕様を指定する必要はありません。**Create** オペレーションで名前変更および削除サブファンクションを使用するための設定方法を、次の表に示します。

操作	使用するキー番号	説明	データ バッファに指定するもの	キー バッファに指定するもの
ファイルの名前変更	-127	データ バッファの既存のファイル名を、キー バッファの名前に変更します。	既存のファイル名	新しいファイル名
ファイルの削除	-128	ファイルを削除します。	適用外	既存のファイル名

これらのサブファンクションはセキュリティ モデルで動作するように変更されました。その変更点として、サブファンクションは削除または名前変更する **MicroKernel** エンジンファイルを示すために、必要に応じて、キー バッファおよびデータ バッファでファイル名の代わりに **URI** を受け入れるようになりました。これにより、セキュリティ情報をオペレーションに指定することができます。**URI** 接続文字列の詳細については、『*Zen Programmer's Guide*』の [データベース URI](#) を参照してください。

セキュリティ情報は、通常の **Create** または **Open** オペレーションと同様に処理されます。ユーザーは認証され、かつ、既存のファイルに対して、また新しいファイルの保存場所となるディレクトリに対して **DB_RIGHT_CREATE**、**DB_RIGHT_ALTER**、および **DB_RIGHT_OPEN** 権限を持っている必要があります。

次の **Create** サブファンクションの表で、**X** はキー バッファで有効な **URI** パラメータを示します。

操作	URI パラメーター file=	URI パラメーター dbfile=	URI パラメーター table=
名前変更	X	X	
削除	X	X	

次の Create サブファンクションの表で、X はデータ バッファーで有効な URI パラメーターを示します。

操作	URI パラメーター file=	URI パラメーター dbfile=	URI パラメーター table=
名前変更	X	X	
削除	適用外	適用外	適用外

名前変更および削除サブファンクションでの注意

- RenameFile および DeleteFile サブファンクションは種々雑多なページの内容には影響しないため、特定のデータベースにバインドされているファイルに使用することはできません。
- ファイルにオーナー ネームが含まれている場合、新しいサブファンクションではオーナー ネームのチェックは行われません。オーナー ネームはファイルの内容を表示させるためには、引き続き必要です。

結果

Create オペレーションが正常に終了した場合は、MicroKernel エンジンから同名のファイルが既に存在すると警告されるか、仕様に従って新しいファイルが作成されます。新規ファイルにはレコードは含まれていません。Create オペレーションでは作成したファイルを開きません。したがって、ファイルにアクセスするには、アプリケーションで Open オペレーションを実行する必要があります。

Create オペレーションが失敗した場合は、MicroKernel エンジンから次のステータスコードのいずれかが返されます。

- 2 アプリケーションで I/O エラーが発生しました。
- 11 指定されたファイル名が無効です。
- 18 ディスクがいっぱいです。
- 22 データ バッファー パラメーターが短すぎます。
- 24 ページサイズまたはデータ バッファー サイズが無効です。
- 25 アプリケーションが指定されたファイルを作成できません。
- 26 指定されたキーの数が無効です。
- 27 キー ポジションが無効です。
- 28 レコード長が無効です。

-
- 29 キー長が無効です。
 - 41 実行しようとした操作は **MicroKernel** では許可されていません。(ファイル形式のバージョンが 13.00 より低いです。)
 - 48 オルタネート コレーティング シーケンス定義が無効です。
 - 49 拡張キー タイプが無効です。
 - 59 指定されたファイルは既に存在します。
 - 104 **MicroKernel** エンジンがロケールを認識しません。
 - 105 このファイルは、可変長部割り当てテーブル (VAT) 付きで作成することはできません。
 - 134 **MicroKernel** エンジンがインターナショナル ソート規則 (ISR) を読み取れません。
 - 135 指定されたインターナショナル ソート規則 (ISR) テーブルは破損しているか、または無効です。

ポジショニング

Create オペレーションを実行しても、ファイルにカレンシーは確立しません。

Create Index (31)

Create Index オペレーション (B_BUILD_INDEX) では、既存のファイルにキーを追加します。

パラメーター

	オペレーション コード	ポジション ブロック	データ バッ ファー	データ バッ ファー長	キー バッ ファー	キー番号
送り値	X	X	X	X		X
戻り値						

前提条件

- 対象となるファイルが開いていることが必要です。
- ファイル内の既存のキー セグメント数は、許容されるキー セグメントの最大数から追加するキー セグメント数を差し引いた値以下でなければなりません。
- 許容されるキー セグメントの最大数は、ファイルのページ サイズによって決まります。次の表は、これらの値の一覧を示します。

ページ サイズ (バイト数)	キー セグメントの最大数 (ファイル バージョン別)			
	8.x 以前	9.0	9.5	13.0、16.0
512	8	8	切り上げ ²	切り上げ ²
1024	23	23	97	切り上げ ²
1536	24	24	切り上げ ²	切り上げ ²
2048	54	54	97	切り上げ ²
2560	54	54	切り上げ ²	切り上げ ²
3072	54	54	切り上げ ²	切り上げ ²
3584	54	54	切り上げ ²	切り上げ ²
4096	119	119	204 ³	183 ³

ページサイズ (バイト数)	キー セグメントの最大数 (ファイルバージョン別)			
	8.x 以前	9.0	9.5	13.0、16.0
8192	N/A ¹	119	420 ³	378 ³
16384	N/A ¹	N/A ¹	420 ³	378 ³

¹ "N/A" は「適用外」を意味します。

² 「切り上げ」は、ページサイズを、ファイルバージョンでサポートされる次のサイズへ切り上げることを意味します。たとえば、512 は 1024 に切り上げられ、2560 は 4096 に切り上げることです。

³ 9.5 以降の形式のファイルでは 119 以上のセグメントを指定できますが、インデックスの数は 119 に制限されます。

『*Status Codes and Messages*』のステータスコード [26 : 指定されたキーの数が無効です](#)。および [29 : キー長が無効です](#)。を参照してください。

- 新しいキーのキーフラグ、位置および長さが、キーを追加しようとしているファイルに対して適切であることを確認します。
- トランザクションが実行中でないことが必要です。

手順

1. オペレーションコードを 31 に設定します。
2. キーを追加するファイルのポジションブロックを渡します。
3. キーの各セグメントについて、キー仕様ブロックをデータバッファに格納します。[Create \(14\)](#) で説明されているものと同じ構造体を使用します。キーポジションおよびキー長の情報は整数で格納します。システム定義のログキー（システムデータとも呼ばれる）を再構築している場合、データバッファは少なくとも 1 つのキー仕様ブロックのサイズがあり、ゼロに初期化されている必要があります。
4. 新しいキーに ACS を定義するには、次のいずれかの手順を実行します。
 - デフォルトの ACS、つまり、ファイルに既に定義されている先頭の ACS を使用するには、キーフラグワードに「デフォルトの ACS を使用」属性を指定します。

- 新しい ACS を定義するには、キー フラグ ワードに「番号付きの ACS を使用」属性を指定し、ACS 番号フィールドをゼロに設定します。さらに、データ バッファの最後のキー仕様ブロックの後に 265 バイトの ACS を格納します。
- 名前によって既存の ACS を指定するには、キー フラグ ワードに「名前付きの ACS を使用」属性を指定し、ACS 番号フィールドをゼロに設定します。さらに、ACS 名を、データ バッファの最後のキー仕様ブロックの後にある 265 バイトのブロックの先頭から格納します（名前より後の ACS ブロックの残り部分は無視されます）。名前の形式は次のいずれかに従っている必要があります。

ACS の種類	長さ (バイト単位)	説明
ユーザー定義の ACS	1	識別バイト 0xAC
"	8	ACS テーブル名
ISR	1	識別バイト 0xAE
"	16	ISR テーブル名

5. データ バッファ長パラメーターをデータ バッファ内のバイト数に設定します。新しいキーが ACS を持たない、もしくはデフォルトの ACS を使用する場合は、次の式を使って正しいデータ バッファ長を決定します。

$(16 \text{ または } 24) * (\text{セグメント数})$

新しいキーでデフォルト以外の ACS を指定する場合は、次の式を使って正しいデータ バッファ長を決定します。

$(16 \text{ または } 24) * (\text{セグメント数}) + 265$

6. 作成されるキーに特定のキー番号を割り当てるには、目的のキー番号に 0x80 を加算し、その合計値をキー番号パラメーターに入れます。システム キー（システム データまたはシステム データ v2 とも呼ばれる）を再構築している場合は、0xFD（つまり、キー番号 125 + 128）または 0xFC（キー番号 124 + 128）を指定します。BTRVEX では、このバイアスにより小さな正の値が生成されるため、符号拡張してはいけないということに留意してください。

メモ： キー番号はファイルで一意であることが必要です。また、有効な値でなければなりません。つまり、各キー番号の値は、指定したページサイズに対して許容されるキー セグメントの最大数よりも小さい値でなければなりません。

詳細

MicroKernel エンジンでは、キーを作成するときに特定のキー番号を割り当てることができます。この機能は、キーを削除して、その削除したキーよりも大きなキー番号を持つすべてのキーの番号を MicroKernel エンジンに付け替えさせないようにする機能と相補関係にあります。アプリケーションがインデックスを削除し、MicroKernel エンジンにそれよりも大きな番号を持つキーの番号を付け替えないように指示を出した場合、その後でユーザーが具体的なキー番号を割り当てずに影響を受けたファイルを複製すると、複製したファイルには元のファイルとは別のキー番号が割り当てられます。

データ バッファで ACS を定義すると、MicroKernel エンジンは ACS 定義をファイルに追加する前に、まず指定された名前を使って既存の ACS をチェックします。

MicroKernel エンジンが指定された名前を持つ既存の ACS を検出した場合、MicroKernel エンジンはファイル内で ACS 定義の複製は行わず、既存の ACS と新しいキーとの関連付けを行います。

キー フラグ ワードに「名前付きの ACS を使用」属性を指定した場合、MicroKernel エンジンはデータ バッファに指定された ACS 名を使ってファイル内で同名の ACS を検索してから、その ACS を新しいキーに割り当てます。

ファイルが複数の MicroKernel エンジン クライアントによって開かれており、そのクライアントのうちの 1 人が Create Index プロセスを開始した場合、リモート クライアントは、MicroKernel エンジン クライアントによってキーが作成されている間も、開いているファイルに対して Get および Step オペレーションを実行できます。

作成されるキーが autoincrement キーでない場合は、リモート クライアントの Get および Step オペレーションにロック バイアスを使用でき、Create Index プロセスが完了したときに、読み取りオペレーションをさらに発行しなくても、ロックされていたレコードを更新したり削除したりできます。MicroKernel エンジンではキーを作成するためにレコードのイメージを変更する必要がないため、このような処理が可能になります。

ただし、作成されるキーが autoincrement キーである場合は、MicroKernel エンジンではインデックスを構築し、かつ適切なフィールドでゼロ値を使ってすべてのレコードを変更する必要があります。キーの作成前または最中にロック バイアスを使わずに Get または Step オペレーションを実行したリモート クライアントは、キーの作成が正常に終了した後で Update または Delete オペレーションを実行するとき、ステータス コード 80 を受け取ります。

また、あるクライアントがレコードをロックしている最中に、別のクライアントが autoincrement キーを作成しようとするとき、MicroKernel エンジンからステータス コード

84 が返されます。同様に、あるクライアントが `autoincrement` キーのインデックスを作成している最中に、別のクライアントがロック バイアスを使って `Get` または `Step` オペレーションを実行しようとする、`MicroKernel` エンジンからステータス コード 85 が返されます。

結果

`MicroKernel` エンジンはファイルに新しいキーを直ちに追加します。このオペレーションの所要時間は、インデックスが作成される総レコード数、ファイルのサイズおよび新しいインデックスの長さによって変わります。

`Create Index` オペレーションが正常に終了した場合、新しいキーの番号は指定した番号になるか、または次のいずれかになります。

- キー番号が飛んでいないファイルの場合は、新しいキー番号は以前の最大のキー番号より 1 つ大きくなります。
- キー番号が飛んでいるファイルの場合は、新しいキー番号は欠けているキー番号のうちの最小の番号になります。

オペレーションの終了次第、新しいキーを使ってデータにアクセスすることができるようになります。

`Create Index` オペレーションが失敗した場合、新しいインデックスの一部が既に構築されていたとしても、`MicroKernel` エンジンはそれをすべて削除します。エラーが発生する前に新しいインデックスに割り当てられたファイル ページは、ファイルの空き領域リストに置かれ、レコードを挿入したり別のキーを作成したりするときに再利用されます。

`autoincrement` キーの作成中にオペレーションが失敗した場合、それまでに変更されている値はそのまま残ります。`MicroKernel` エンジンから返される可能性のあるステータス コードは次のとおりです。

- 22 データ バッファ パラメーターが短すぎます。
- 27 キー ポジションが無効です。
- 41 実行しようとした操作は `MicroKernel` エンジン では許可されていません。
- 45 指定されたキー フラグが無効です。
- 49 拡張キー タイプが無効です。
- 56 インデックスが不完全です。

-
- 84 レコードまたはページはロックされています。
 - 85 ファイルはロックされています。
 - 104 MicroKernel エンジンがロケールを認識しません。
 - 134 MicroKernel エンジンがインターナショナル ソート規則 (ISR) を読み取れません。
 - 135 指定されたインターナショナル ソート規則 (ISR) テーブルは破損しているか、または無効です。
 - 136 MicroKernel エンジンは、指定されたオルタネート コレレーティング シーケンスをファイル内に見つけれません。

キーの作成中に処理が中断されても、ファイル内のほかのキーを使ってファイルのデータにアクセスすることはできます。しかし、不完全なインデックスを使ってデータにアクセスしようとする、MicroKernel エンジンから 0 以外のステータス コードが返されます。この問題を解決するには、Drop Index (32) オペレーションを使って不完全なインデックスを削除し、Create Index を再発行してください。

ポジショニング

Create Index オペレーションは、ファイルのカレンシー情報にまったく影響しません。

Delete (4)

Delete オペレーション (B_DELETE) では、ファイルから既存のレコードを削除します。削除したレコードが占有していたスペースは、新しいレコードを挿入するために再利用されます。

パラメーター

	オペレーション コード	ポジション ブロック	データ バッ ファー	データ バッ ファー長	キー バッ ファー	キー番号
送り値	X	X				
戻り値		X				

前提条件

- 対象となるファイルが開いていることが必要です。
- 対象となるファイルの物理カレンシーまたは論理カレンシーを確立しておくことが必要です。この要件を満たすオペレーションには、Get (Get ... Extended、Get Key を除く)、Step (Step ... Extended を除く)、Insert、および Update オペレーションがあります。

手順

- オペレーション コードを 4 に設定します。
- 削除するレコードを含むファイルのポジション ブロックを渡します。

詳細

Delete オペレーションは、Extended Get または Extended Step オペレーションの直後に実行した場合には現在のレコードが予測不能であるため、有効なオペレーションにならない可能性があります。

Delete オペレーションを実行した後、それ以降の Get Next または Get Previous オペレーションでは、論理位置を確立した直前のオペレーションと同じキー番号およびキー

バッファを使用する必要があります。別の値を使用すると、MicroKernel エンジンからステータスコード 7 が返されます。

MicroKernel エンジンでは、Get Key (+50) の後に Delete オペレーションを実行することはできません。MicroKernel エンジンで Delete オペレーションを実行する前に、変更しようとしているデータ ページの現在の使用回数と、レコードを読み取った時点のデータ ページの使用回数が比較されます。使用回数を取得するには、MicroKernel エンジンがデータ ページを読み取る必要があります。

Get Key オペレーションではデータ ページを読み取らないので、Delete オペレーションで比較するための使用回数が利用可能になりません。MicroKernel エンジンでは、比較なしにパシブ並行制御の矛盾チェックを実行できないため、Delete オペレーションは正常に実行されません。Delete オペレーションが正常に実行されないと、MicroKernel エンジンからステータスコード 8 が返されます。

結果

Delete オペレーションが正常に終了した場合は、MicroKernel エンジンによってファイルからレコードが削除され、削除したレコードにロックが設定されていた場合はそのロックが解除され、さらに削除の結果を反映して、すべてのキー インデックスを調整されます。

Delete オペレーションが失敗した場合は、MicroKernel エンジンから次のステータスコードのいずれかが返されます。

- 8 現在のポジションが無効です。
- 80 MicroKernel エンジンでレコード レベルの矛盾が発生しました。
- 84 レコードまたはページはロックされています。
- 85 ファイルはロックされています。

Delete オペレーションを実行してもファイル サイズは小さくなりません。レコードの削除によって生じる空き領域は、今後レコードを追加するときに再利用されます。ディスク容量を回復するには、ファイルを再作成して、そのファイルにすべてのレコードを挿入するしかありません。Rebuild および Defragmenter ユーティリティが、この回復を実現するのに役立ちます。

ポジショニング

Delete オペレーションを実行すると、すべての物理位置情報と現在のレコードの論理位置は消去されますが、次のレコードまたは前のレコードの論理位置は変わりません。

Drop Index (32)

Drop Index オペレーション (B_DROP_INDEX) では、既存のファイルからキーを削除します。

パラメーター

	オペレーション コード	ポジション ブロック	データ バッ ファー	データ バッ ファー長	キー バッ ファー	キー番号
送り値	X	X				X
戻り値						

前提条件

- 対象となるファイルが開いている必要があります。
- ファイル内にキーが存在している必要があります。
- トランザクションが実行中でない必要があります。

手順

- オペレーション コードを 32 に設定します。
- 削除するキーを含むファイルのポジション ブロックを渡します。
- キー番号パラメーターに削除するキーの番号を格納します。システム定義のログ キー (システム データとも呼ばれる) を削除するには、125 を指定します。システム データ v2 用の 2 番目のシステム キーを削除するには、124 を指定します。

詳細

システム キーを削除した場合、Create Index (31) オペレーションを使ってそれを再構築することができます。

キーを削除する場合、特に指定しなければ、削除したキーよりもキー番号の大きなキーはすべて、MicroKernel エンジンによって自動的に番号が付け替えられます。

MicroKernel エンジンでは、削除したキーよりも番号の大きなキーの番号を 1 ずつ減らし

ます。たとえば、キー番号 1、4、および 7 を含むファイルがあるとします。キー 4 を削除すると、MicroKernel エンジンに残ったキーの番号を 1 と 6 に付け替えます。

MicroKernel エンジンによってキー番号を自動的に付け替えられたくない場合は、128 というバイアスをキー番号パラメーターに入れる値に加算します。このバイアスにより、キー番号は飛んだままにしておくことができ、その結果、ファイル内のほかのキー番号に影響を及ぼすことなく、壊れたインデックスを削除し、そのインデックスを作成し直すことができます。インデックスを再構築するには、Create Index (31) を使います。このオペレーションではキー番号を指定できます。

ただし、キーを削除し、それよりキー番号の大きなキーの番号を付け替えなかった場合、その後でユーザーが具体的なキー番号を割り当てずに影響を受けたファイルを複製すると、複製したファイルには元のファイルとは別のキー番号が割り当てられます。

メモ： ユーザーは Btrieve Maintenance ツール、またはそのコマンド ラインバージョンの butil を使ってファイルを複製できます。複製により、既存のファイルと同じ統計情報を持つ新しい空のファイルが作成されます。

結果

Drop Index オペレーションが正常に終了した場合、指定したインデックスは MicroKernel エンジンによって削除され、そのインデックスに割り当てられていたページは、今後の使用のために空き領域のリストに配置されます。また、特に指定しなければ、MicroKernel エンジンでは削除したキーよりもキー番号の大きなキーの番号が付け替えられます。

Drop Index オペレーションが失敗した場合は、MicroKernel エンジンから次のステータスコードのいずれかが返されます。

- 6 キー番号パラメーターが無効です。
- 41 実行しようとした操作は MicroKernel エンジン では許可されていません。

MicroKernel エンジンがインデックスを削除中に処理が中断されても、ファイル内のほかのキーを使ってファイルのデータにアクセスすることはできます。しかし、不完全なインデックスを使ってファイルにアクセスしようとする、MicroKernel エンジンからステータスコード 56 が返されます。処理が中断された場合は、Drop Index オペレーションを再発行してください。

ポジショニング

Drop Index オペレーションは、ファイルの物理カレンシー情報にはまったく影響しません。ただし、直前に論理カレンシーを確立するために使用したキーを削除すると、論理カレンシーは消去されます。

End Transaction (20)

End Transaction オペレーション (B_END_TRAN) では、トランザクションを終了し、データ ファイルに適切な変更を加えます。また、トランザクションによって設定されたすべてのファイルとレコードのロックを解除します。

パラメーター

	オペレーション コード	ポジション ブロック	データ バッ ファー	データ バッ ファー長	キー バッ ファー	キー番号
送り値	X					
戻り値						

前提条件

End Transaction オペレーションを発行する前に、Begin Transaction (19 または 1019) が正常に終了している必要があります。

手順

オペレーション コードを 20 に設定します。MicroKernel エンジンでは、オペレーション コード以外の End Transaction 呼び出しパラメーターはすべて無視されますが、将来のリリースとの互換性を確保するために 0 に初期化してください。

結果

End Transaction オペレーションが正常に終了した場合は、トランザクション内で実行されたすべてのオペレーションの結果がファイルに保存されます。End Transaction オペレーションを実行した後でトランザクションを中止することはできません。

End Transaction オペレーションが失敗した場合は、MicroKernel エンジンから次のステータス コードが返されます。

- 38 MicroKernel エンジンで、トランザクション制御ファイル I/O エラーが発生しました。

ポジショニング

End Transaction オペレーションは、ファイルのカレンシー情報にまったく影響しません。

Find Percentage (45)

Find Percentage オペレーション (B_GET_PERCENT) は、スクロールバーを実装するウィンドウ指向のアプリケーションで使用することのできる 2 つの Btrieve API オペレーションのうちの一つです。もう 1 つのオペレーションは Get By Percentage (44) です。Find Percentage では、キーパスまたはファイル内でのレコードの物理位置を基準として、それに対応するレコードのおおよその位置を返します。位置はパーセンテージ値で表されます。パーセンテージ値の範囲の定義については、[結果](#)を参照してください。

パラメーター

	オペレーション コード	ポジション ブロック	データ バッ ファー	データ バッ ファー長	キー バッ ファー	キー番号
送り値	X	X	X	X	X	X
戻り値		X	X	X		

メモ： Find Percentage を使って、キーパスを基準に対応するパーセンテージをシークする場合は、データバッファーパラメーターに値を入力する必要はありません。ファイル内のレコードの物理位置を基準にしてパーセンテージをシークする場合は、キーバッファーパラメーターに値を入力する必要はありません。

前提条件

- 対象となるファイルが開いていることが必要です。
- キーパスに基づいてパーセンテージをシークする場合は、対象となるファイルがデータオンリーファイルであってははいけません。
- ファイル内のレコードの物理位置に基づいてパーセンテージをシークする場合は、その物理位置をデータバッファーパラメーターに指定する必要があります。この位置は、Get Position (22) を使って取得できます。BTRV または BTRVEX タイプのエントリポイントの使用と一貫性を持たせてください。

手順

- オペレーションコードを 45 に設定します。

-
2. ファイルのポジションブロックを渡します。
 3. ファイル内のレコードの物理位置を基準にしてパーセンテージをシークする場合は、レコードの4バイトまたは8バイトの物理アドレスをデータバッファーに格納します。レコードのキーパスを基準にしてパーセンテージをシークし、その検索の精度を指定する場合は、**精度**で指定されているようにデータバッファーパラメーターを設定します。それ以外の場合は、データバッファーパラメーターに値を入れる必要はありません。
 4. データバッファー長を最小値である4バイトまたは8バイトに設定します。この4バイトという最小長は、MicroKernel エンジンの内部的な実装に必要とされます。検索の精度を指定する場合は、データバッファー長を最小値の12バイトまたは16バイトに設定します。
 5. キーパスを基準にパーセンテージをシークする場合は、キーバッファーパラメーターをキー値に設定します。それ以外の場合は、キーバッファーパラメーターに値を入れる必要はありません。
 6. キー番号パラメーターを以下のとおりに設定します。
 - キーパスによってパーセンテージをシークする場合は、キー番号パラメーターを実際のキー番号に設定します。
 - レコードの物理位置によってパーセンテージをシークする場合は、キー番号パラメーターを-1に設定します。

詳細

Find Percentage オペレーションは、特にスクロールバーの実装をサポートする目的で用意されています。このオペレーションの精度、つまり、返されたパーセンテージ値がレコードまたはキー値の位置をどれだけ正確に反映しているかどうかは、さまざまな要因によって影響を受けます。このため、スクロールバーの実装以外の目的で使用する場合は、このオペレーションの精度を信頼しないでください。

Find Percentage オペレーションを最適化するため、MicroKernel エンジンでは、ファイルのレコードはデータ ページ間に、キーはインデックス ページ間に均等に分布していることを前提としています。ただし、分布状態は次のような状況によって影響を受けます。

- ファイルがインデックス バランスを使用しておらず、同一範囲内のキーで多数のレコードが削除されている。

- 同一範囲内の物理アドレスで多数のレコードが削除されている。
- ファイルに多数の重複するキー値が含まれており、そのキーがリンク重複キーになっている。

精度

精度の設定は任意であり、パーセンテージを測定する要素を選択することができます。Zen 9 より前のリリースでは、この値は常に 10000 でした。

精度を指定する場合は、以下の手順に従ってください。

Find Percentage オペレーションの精度を指定するには

1. データ バッファのレコード アドレス領域の後の 4 バイトに識別バイト ExPc (0x45、0x78、0x50、0x63) を設定します。
2. この識別バイトの後の 4 バイトに希望する精度を LoHi Intel 整数で指定します。選択できる精度は、1 から 0xFFFFFFFF までの数値です。
3. 使用されるエン트리 ポイントに応じて、データ バッファ長が少なくとも 12 バイトまたは 16 バイトであることを確認してください。

次の表は、これらの手順における位置とレイアウトをまとめています。

	BTRV、BTRVID、BTRCALL、 BTRCALLID	BTRVEX、BTRVEXID
レコード アドレス	4 バイト、オフセット 0	8 バイト、オフセット 0
識別の精度	4 バイト、オフセット 4	4 バイト、オフセット 8
一般の精度	4 バイト、オフセット 8	4 バイト、オフセット 12
合計サイズ	4 または 12 バイト	8 または 16 バイト

たとえば、365 件のレコードが入っているファイルから 100 番目のレコードを取得したい場合、パーセンテージに 100、精度に 365 を使用して Find Percentage (45) を実行することができます。

結果

Find Percentage オペレーションが正常に終了した場合は、MicroKernel エンジンによって、指定したキー値またはレコードの相対位置がデータバッファーに返されます。この位置は、キーパスまたはファイルにおけるオフセットのパーセンテージとして表され、0 (0%) から 10000 (100.00%) までの範囲の値になります。これは、物理位置でも論理位置でもないので注意してください。

パーセンテージ値は、下位バイト、上位バイトの順の 4 バイト整数として返されます。たとえば、デフォルトの精度を使用する場合は次のようになります。

16 進数の戻り値	10 進数の戻り値	キーパスまたはファイル内でのパーセンテージ
88h 13h	5000	50%

また、オペレーションが正常に終了した場合には、MicroKernel エンジンからデータバッファー長に少なくとも 4 が返されます。

Find Percentage オペレーションが失敗した場合は、MicroKernel エンジンから次のステータスコードのいずれかが返されます。

- 3 ファイルが開いていません。
- 6 キー番号パラメーターが無効です。
- 7 キー番号が変更されました。
- 8 現在のポジションが無効です。
- 9 オペレーションが EOF (end-of-file) を検出しました。
- 22 データバッファーパラメーターが短すぎます。
- 41 実行しようとした操作は MicroKernel エンジンでは許可されていません。
- 43 指定されたレコードアドレスが無効です。
- 82 MicroKernel エンジンがポジショニングを失いました。

ポジショニング

Find Percentage オペレーションを実行しても、カレンシー情報は変更されません。

Get By Percentage (44)

Get By Percentage オペレーション (B_SEEK_PERCENT) は、スクロールバーを実装するウィンドウ指向のアプリケーションで使用することのできる2つの Btrieve API オペレーションのうちの一つです。もう一つは [Find Percentage \(45\)](#) です。Get By Percentage オペレーションは、ファイル内のレコードの相対位置によってレコードを取得します。この位置は、オペレーションを呼び出すときに指定したパーセンテージ値に基づきます。また、この位置は特定のキーパスを基準とするのか、ファイル内のレコードの実際の物理位置を表すのかを指定する必要があります。

パラメーター

	オペレーションコード	ポジションブロック	データバッファー	データバッファー長	キーバッファー	キー番号
送り値	X	X	X	X		X
戻り値		X	X	X	X	

メモ： ファイル内のレコードの物理位置を基準としてレコードをシークする場合、Get By Percentage オペレーションからキーバッファーパラメーターには何の情報も返されません。

前提条件

- 対象となるファイルが開いていることが必要です。
- キーパスに基づいてレコードをシークする場合は、対象となるファイルがデータオンリーファイルであってははいけません。

手順

- オペレーションコードを 44 に設定します。任意でロック バイアスも指定できます。
 - +100 - 単一レコード ウェイト ロック
 - +200 - 単一レコード ノーウェイト ロック
 - +300 - 複数レコード ウェイト ロック

-
- +400 - 複数レコード ノーウェイト ロック

レコード ロックおよびデータ整合性については、『*Zen Programmer's Guide*』のほかに、『*Advanced Operations Guide*』に記載されている Zen サーバーを設定するための **ウェイト ロック タイムアウト** プロパティを参照してください。

2. ファイルのポジションブロックを渡します。
3. パーセンテージの値を 4 バイト整数でデータ バッファに格納します。パーセンテージ値の許容範囲および関連情報については、[詳細](#)を参照してください。
4. データ バッファ長を、返される可能性のある最大レコード長以上の値に設定します (MicroKernel エンジンの内部的な実装は、データ バッファ長が最小値の 4 バイトに設定されていることを必要とします)。検索に精度を設定する場合は、データ バッファ長を最小値の 12 バイトに設定します。
5. キー番号パラメーターを設定します。
 - キーパスによってパーセンテージをシークする場合は、キー番号パラメーターを実際のキー番号に設定します。システム定義のログ キー (システム データとも呼ばれる) を使用するには、125 を指定します。システム データ v2 用の 2 番目のシステム キーを使用するには、124 を指定します。
 - ファイル内のレコードの物理位置によってレコードをシークする場合は、キー番号パラメーターを -1 に設定します。

詳細

精度を指定しない場合 ([精度](#)を参照)、データ バッファ パラメーターの最初の 2 バイトに対するパーセンテージ値の許容範囲は、0 (キーパスまたはファイルの先頭を表します) から 10000 (キーパスまたはファイルの末尾を表します) までです。この値は、小数点以下 2 桁を含むものとして、0% から 100.00% の範囲に対応しています。ファイル中の 33.33% あたりにあるレコードを検索する場合は、データ バッファに値 3333 を渡します。値は、下位バイト、上位バイトの順の整数として格納してください。たとえば、ファイル内の 50% の地点をシークするには、5000 (0x1388) という値を使います。0x1388 の下位バイトと上位バイトを入れ替え、0x88 と 0x13 をデータ バッファ パラメーターの先頭の 2 バイトに格納します。

検索の精度を指定する場合は、[精度](#)で指定されているようにデータ バッファ パラメーターを設定します。

Get By Percentage オペレーションは、特にスクロールバーの実装をサポートする目的で用意されています。このオペレーションの精度、つまり、返されたレコードがファイル内の指定したパーセンテージ地点に実際に位置しているかどうかは、さまざまな要因によって影響を受けます。このため、スクロールバーの実装以外の目的で使用する場合は、このオペレーションの精度を信頼しないでください。

Get By Percentage オペレーションを最適化するため、**MicroKernel** エンジンでは、ファイルのレコードはデータ ページ間に、キーはインデックス ページ間に均等に分布していることを前提としています。ただし、分布状態は次のような状況によって影響を受けます。

- ファイルがインデックス バランスを使用しておらず、同一範囲内のキーで多数のレコードが削除されている。
- 同一範囲内の物理アドレスで多数のレコードが削除されている。
- ファイルに多数の重複するキー値が含まれており、そのキーがリンク重複キーになっている。

精度

精度の設定は任意であり、パーセンテージを測定する要素を選択することができます。**Zen 9** より前のリリースでは、この値は常に **10000** でした。

精度を指定する場合は、以下の手順に従ってください。

Get By Percentage オペレーションの精度を指定するには

1. データ バッファのパーセンテージの後の 2 番目の 4 バイトに識別バイト **ExPc** (0x45、0x78、0x50、0x63) を設定します。
2. この識別バイトの後の 4 バイトに希望する精度を **LoHi Intel** 整数で指定します。選択できる精度は、1 から 0xFFFFFFFF までの数値です。
3. データ バッファ長が少なくとも 12 バイトあることを確認してください。

たとえば、365 件のレコードが入っているファイルから 100 番目のレコードを取得したい場合、パーセンテージに 100、精度に 365 を使用して **Get By Percentage (44)** を実行することができます。

結果

Get By Percentage オペレーションが正常に終了した場合は、MicroKernel エンジンによって、指定したキーパスを基準とする位置またはファイル内の物理位置にあるレコードがデータバッファに返されます。さらに MicroKernel エンジンからは、データバッファ長パラメーターにレコード長がバイト単位で返されます。キーパスによってレコードをシークした場合は、MicroKernel エンジンからキーバッファパラメーターに指定したキーパスのキー値が返されます。物理レコード順によってレコードをシークした場合は、MicroKernel エンジンからキーバッファパラメーターには何の情報も返されません。

メモ： Get By Percentage オペレーションでキーパスを基準としてレコードをシークした場合、そのキーに重複値が含まれているときは、MicroKernel エンジンでは常に重複値を含む先頭のレコードが返されます。このような実装の細部によって、オペレーションの精度が影響を受ける場合があります。

Get By Percentage オペレーションが失敗した場合は、MicroKernel エンジンから次のステータスコードのいずれかが返されます。

- 3 ファイルが開いていません。
- 6 キー番号パラメーターが無効です。
- 7 キー番号が変更されました。
- 8 現在のポジションが無効です。
- 9 オペレーションが EOF (end-of-file) を検出しました。
- 22 データバッファパラメーターが短すぎます。
- 41 実行しようとした操作は MicroKernel エンジン では許可されていません。
- 43 指定されたレコード アドレスが無効です。
- 82 MicroKernel エンジンがポジショニングを失いました。

ポジショニング

指定したキーパスを基準にしてレコードをシークするとき、Get By Percentage オペレーションが正常に終了した場合は、指定したキー番号と取得したレコードのそれぞれに基づいて新しい論理カレンシーおよび物理カレンシーが確立されます。

ファイル内のレコードの物理位置を基準にしてレコードをシークするとき、Get By Percentage オペレーションが正常に終了した場合は、取得したレコードに基づいて新しい物理カレンシーが確立されます。

Get By Percentage オペレーションが失敗した場合、MicroKernel エンジンではカレンダーは変更されません。

Get Direct/Chunk (23)

Get Direct/Chunk オペレーション (B_GET_DIRECT) では、**チャンク**と呼ばれるレコードの部分を1つまたは複数取得できます。このオペレーションは、最大データバッファサイズより長いレコードを含むファイルで特に役に立ちます。データバッファパラメーターの長さには制限があるため、このようなレコードは長すぎて、ほかの Get および Step オペレーションでは取得できません。アプリケーションでは、物理アドレスを指定することで、チャンクが取得されるレコードを指定します。通常、レコード内でのチャンクの位置は、そのオフセットと長さで指定されます。

パラメーター

	オペレーションコード	ポジションブロック	データバッファ	データバッファ長	キーバッファ	キー番号
送り値	X	X	X	X		X
戻り値		X	X	X		

前提条件

- 対象となるファイルが開いていることが必要です。
- レコードの物理位置を用意する必要があります。この位置は、Get Position (22) を使って取得できます。BTRV または BTRVEX タイプのエントリポイントの使用と一貫性を持たせてください。
- Get Direct/Chunk オペレーションから返されるすべての値を格納するのに十分な大きさのデータバッファを用意する必要があります。また、Get Direct/Chunk オペレーションが間接チャンクオペレーションを実行するとき、データバッファにはチャンクディスクリプター全体 (すべてのチャンク定義) を格納できなければなりません。

手順

- オペレーションコードを 23 に設定します。任意でロックバイアスも指定できます。
 - +100 - 単一レコード ウェイト ロック

- +200 - 単一レコード ノーウェイト ロック
- +300 - 複数レコード ウェイト ロック
- +400 - 複数レコード ノーウェイト ロック

レコード ロックおよびデータ整合性については、『*Zen Programmer's Guide*』のほかに、『*Advanced Operations Guide*』に記載されている Zen サーバーを設定するための **ウェイト ロック タイムアウト** プロパティを参照してください。

2. ファイルのポジションブロックを渡します。
3. **詳細**の説明に従って、データ バッファを指定します。
4. データ バッファ長には、入力構造体の長さ **と MicroKernel エンジン**に取得するように要求したバイト数のどちらか大きい方を指定します。

Get Direct/Chunk オペレーションの一部のオプションでは、データ バッファ以外の場所にチャンクを取得します。データ バッファ長の計算の詳細については、**詳細**を参照してください。

5. キー番号パラメーターを -2 に設定します。

詳細

データ バッファでは、次のチャンク ディスクリプターのいずれかを使用します。

- ランダム チャンク ディスクリプター - オペレーションに付き 1 つのチャンクを取得するため、またはチャンクがレコード全体にわたってランダムに配置されているときに、1 回のオペレーションで複数のチャンクを取得するために使用します。
- 矩形チャンク ディスクリプター - 各チャンクの長さが同じで、チャンクがレコード内に等間隔に配置されているときに、1 回のオペレーションで複数のチャンクを取得するために使用します。

ランダム チャンク

次の例は、ランダムに配置されている 3 つのチャンク ([*] がある部分) を含むレコードを示しています。チャンク 0 (バイト 0x12 から 0x16)、チャンク 1 (バイト 0x2A から 0x31)、およびチャンク 2 (バイト 0x41 から 0x4E) です。

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

10	11	[*]	[*]	[*]	[*]	[*]	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	[*]	[*]	[*]	[*]	[*]	[*]
[*]	[*]	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	[*]	[*]	[*]	[*]	[*]	[*]	[*]	[*]	[*]	[*]	[*]	[*]	[*]	[*]	4F

ランダム チャンク オペレーションのデータ バッファ

ランダム チャンクを取り出すには、次の表に基づいてデータ バッファ内に構造体を作成する必要があります。

要素	長さ (バイト単位)	説明
レコード アドレス	4 または 8 ¹	レコードの物理位置。この位置は、 Get Position (22) を使って取得できます。

ランダム チャンク ディスクリプター

サブファンクション	4	<p>チャンク ディスクリプターの種類。次のいずれかです。</p> <ul style="list-style-type: none"> 0x80000000 (直接ランダム チャンク ディスクリプター) - チャンクを直接データ バッファに取得します。1 つ目のチャンクが取得されるとデータ バッファのオフセット 0 に格納され、2 つ目のチャンクが 1 つ目のチャンクの直後に続き、と以下同様に続きます。 0x80000001 (間接ランダム チャンク ディスクリプター) - チャンクをチャンク定義によって指定されるアドレス内に取得します。
チャンク数	4	取得するチャンク数。この値は少なくとも 1 以上である必要があります。明確な最大値はありませんが、チャンク ディスクリプターはデータ バッファに収まらなければなりません。

要素	長さ (バイト単位)	説明
チャンク定義 (各チャンクについて繰り返す)	12 (32 ビット アプリケーション用) 16 (64 ビット アプリケーション用)	各チャンク定義は、以下に示すように、4 バイトのチャンク オフセット、それに続く 4 バイトのチャンク長、さらに 32 ビット アプリケーションの場合は 4 バイトのユーザー データ、または 64 ビット アプリケーションの場合は 8 バイトのユーザー データから構成されます。 <ul style="list-style-type: none"> チャンク オフセット - チャンクの開始地点を、レコードの先頭からのオフセット (バイト単位) で示します。最小値は 0、最大値はレコードの末尾のバイトのオフセットです。 チャンク長 - チャンク内のバイト数を示します。最小値は 0、最大値は 65535² です。 ユーザー データ - (間接ディスクリプターでのみ使用します。) 32 ビット アプリケーションの場合、実際のチャンク データへの 32 ビット ポインターです。64 ビット アプリケーションの場合、実際のチャンク データへの 64 ビット ポインターです。直接チャンク ディスクリプターのサブファンクションの場合、MicroKernel エンジンではこの要素は無視されます。

¹ サイズは、使用するエン트리 ポイントが BTRV タイプか BTRVEX タイプかによって決まります。

² BTRVEX の場合、チャンク サイズは 65535 に制限されていますが、1 個の大きなデータ バッファーに複数のチャンクを返すことができます。

次の表は、BTRV エントリー ポイントを使って直接ランダム チャンクを取り出す場合の、32 ビット アプリケーション用データ バッファーの例を示しています。

要素	サンプル値	長さ (バイト単位)
レコード アドレス	0x00000628	4
サブファンクション	0x80000000	4
チャンク数	3	4
チャンク 0		
チャンク オフセット	18	4
チャンク長	5	4

要素	サンプル値	長さ (バイト単位)
ユーザー データ	適用外	4
チャンク 1		
チャンク オフセット	42	4
チャンク長	8	4
ユーザー データ	適用外	4
チャンク 2		
チャンク オフセット	65	4
チャンク長	14	4
ユーザー データ	適用外	4

矩形チャンク ディスクリプター構造体

同じ長さのチャンクがレコード全体にわたって等間隔に配置されている場合は、矩形チャンク ディスクリプターを使って、取得するすべてのチャンクを記述することができます。たとえば、次のような図を考えてみましょう。この図は、レコード内のオフセット 0x00 から 0x4F までを表しています。

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	11	12	13	14	15	16	17	18	[*]	[*]	[*]	[*]	1D	1E	1F
20	21	22	23	24	25	26	27	28	[*]	[*]	[*]	[*]	2D	2E	2F
30	31	32	33	34	35	36	37	38	[*]	[*]	[*]	[*]	3D	3E	3F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

このレコードには3つのチャンク ([*]がある部分) が含まれています。チャンク 0 (バイト 0x19 から 0x1C)、チャンク 1 (バイト 0x29 から 0x2C)、およびチャンク 2 (バイト 0x39 から 0x3C) です。各チャンクはどれも4バイトの長さで、チャンク同士は、各チャンクの先頭から計算すると、いずれも合計16 (0x10) バイトずつ離れています。

矩形チャンクのデータバッファ

1つの矩形ディスクリプターを使って、3つのチャンクをすべて取得できます。矩形チャンクを取り出すには、次の表に基づいてデータバッファ内に構造体を作成する必要があります。

要素	長さ (バイト単位)	説明
レコードアドレス	4 または 8 ¹	レコードの4バイト物理アドレス。この位置は、 Get Position (22) を使って取得できます。

矩形チャンク ディスクリプター

サブファンクション	4	チャンク ディスクリプターの種類。次のいずれかです。 <ul style="list-style-type: none">0x80000002 (直接矩形チャンク ディスクリプター) - チャンクを直接データバッファに取得します。1つ目のチャンクが取得されるとデータバッファのオフセット0に格納され、2つ目のチャンクが1つ目のチャンクの直後に続き、と以下同様に続きます。0x80000003 (間接矩形チャンク ディスクリプター) - チャンクをユーザーデータ要素およびアプリケーションの行間隔要素によって指定されるアドレス内に取得します。
行数	4	矩形チャンク ディスクリプターの操作対象とするチャンク数。この値の最小値は1です。明確な最大値はありません。
位置 (オフセット)	4	取得する最初のバイトの、レコードの先頭からのオフセット。最小値は0、最大値はレコードの末尾のバイトのオフセットです。レコードが1つの矩形として表される場合、この要素は、取得される先頭行にある先頭バイトのオフセットを指します。
行のバイト数	4	各チャンクで取得するバイト数。最小値は0、最大値は65535 ² です。
行間隔	4	チャンクの前頭から次のチャンクの前頭までのバイト数。

要素	長さ (バイト単位)	説明
ユーザーデータ	4 (32 ビットアプリケーション用) 8 (64 ビットアプリケーション用)	(間接ディスクリプターでのみ使用します。) 32 ビット アプリケーションの場合、MicroKernel エンジンが各行からバイトを取得した後でそれらを格納する場所への 32 ビット ポインター。64 ビット アプリケーションの場合、MicroKernel エンジンが各行からバイトを取得した後でそれらを格納する場所への 64 ビット ポインター。 直接矩形ディスクリプターの場合、MicroKernel エンジンではこの要素は無視されます。ただしそれでも、この要素を割り当て、0 に初期化しておく必要があります。
アプリケーションの行間隔	4	(間接矩形ディスクリプターでのみ使用します。) 矩形がアプリケーション メモリ (つまり、ユーザー データで指定したアドレス) に格納されるとき、矩形内のチャンクの先頭から次のチャンクの先頭までのバイト数。直接矩形ディスクリプターの場合、MicroKernel エンジンではこの要素は無視されます。ただしそれでも、この要素を割り当て、0 に初期化しておく必要があります。

¹ サイズは、使用するエン트리 ポイントが BTRV タイプか BTRVEX タイプかによって決まります。

² BTRVEX の場合、チャンク サイズは 65535 に制限されていますが、1 個の大きなデータバッファーに複数のチャンクを返すことができます。

間接矩形ディスクリプターを使用するときは、取得されたチャンクがチャンク ディスクリプターを上書きしないように、ユーザー データ ポインターが初期化されていることを確認してください。MicroKernel エンジンではディスクリプターを使用して、返されたチャンクをユーザー データ要素で示される場所にコピーします。チャンク ディスクリプターを上書きしてしまった場合は、MicroKernel エンジンからステータス コード 62 が返されます。

矩形がメモリ内にあるとき、各行の間隔がレコードとして格納されているときと同じバイト数になる場合は、アプリケーションの行間隔を行間隔と同じ値に設定します。しかし、矩形がアプリケーション メモリ内で再配置され、行の間隔が何バイトか増減する場合は、アプリケーションの行間隔により、その情報を MicroKernel エンジンに渡すことができます。

間接矩形ディスクリプターを使用するときは、MicroKernel エンジンではユーザー データ要素およびアプリケーションの行間隔要素を使って、取得後のデータの格納場所を決定します。MicroKernel エンジンでは 1 行目のデータを、ユーザー データのオフセット 0 に

格納します。MicroKernel エンジンでは 2 行目のデータを、ユーザー データ + アプリケーションの行間隔で指定されるアドレスに格納します。MicroKernel エンジンでは 3 行目のデータを、ユーザー データ + (アプリケーションの行間隔 * 2) で指定されるアドレスに格納します。以下同様です。

次の表は、BTRV エントリ ポイントを使って直接矩形チャンクを取り出す場合の、32 ビット アプリケーション用データ バッファの例を示しています。

要素名	サンプル値	長さ (バイト 単位)
レコード アドレス	0x00000628	4
サブファンクション	0x80000002	4
行数	3	4
オフセット	25	4
行のバイト数	4	4
行間隔	16	4
ユーザー データ	0	4
アプリケーションの行間隔	0	4

ネクストインレコード サブファンクション バイアス

これまでに述べたサブファンクションの値にバイアス 0x40000000 を加算すると、MicroKernel エンジンではレコード内の物理カレンシー（つまり、レコード内の現在の物理位置）に基づいてサブファンクションのオフセット要素の値が算出されます。ネクストインレコード サブファンクションを使用する場合、MicroKernel エンジンではチャンク ディスクリプターのオフセット要素は無視されます。

結果

Get Direct/Chunk オペレーションが正常に終了した場合、直接チャンク ディスクリプターを使用しているときは、MicroKernel エンジンではデータ バッファにチャンクが順に返されます。間接ランダム チャンク ディスクリプターを使用しているとき、MicroKernel エンジンでは各チャンクのユーザー データ要素で指定した場所にデータが返されます。また、間接矩形ディスクリプターを使用しているとき、MicroKernel エン

ジンではユーザー データ要素およびアプリケーションの行間隔要素から計算される場所にデータが返されます。

さらに **MicroKernel** エンジンから、データ バッファ長パラメーターには、取得されたチャンクの長さの総計が格納されます（この戻り値は、チャンクが取得されて直接データ バッファに格納されたか、間接ディスクリプターによってチャンクが取得され別の場所に格納されたかどうかに関係なく、取得された全バイト数を反映しています）。オペレーションが部分的にしか正常に実行されなかった場合、アプリケーションではデータ バッファ長パラメーターに返された値を使って、どのチャンクが取得されなかったか、また最後のチャンクの何バイトまでが取得されたかを調べることができます。

いずれかのチャンクで、開始位置がレコードの末尾を超えてしまう場合（結果として、**MicroKernel** エンジンからステータス コード 103 が返されます）、またはチャンクのオフセットと長さの合計がレコード長を超えてしまう場合には、**Get Direct/Chunk** オペレーションの一部だけが正常に実行されます。後者の場合は **MicroKernel** エンジンからステータス コード 0 が返されますが、このオペレーションに後続のチャンクがある場合、その処理は中止されます。

メモ： すべてのチャンクが適切に取得されたかどうかを知らせるものは、データ バッファ長パラメーターだけです。このため、**Get Direct/Chunk** オペレーションの実行後は、必ずデータ バッファ長パラメーターに返された値をチェックしてください。

次のステータス コードは、**Get Direct/Chunk** オペレーションの一部だけが実行されたことを示します。**MicroKernel** エンジンからこれらのステータス コードのいずれかが返された場合は、アプリケーションでデータ バッファ長パラメーターの戻り値を調べて、**MicroKernel** エンジンから実際に返されたデータ量を確認する必要があります。

- 22 データ バッファパラメーターが短すぎます。
- 54 レコードの可変長部分が破損しています。
- 103 チャンク オフセットが大きすぎます。

MicroKernel エンジンから次のステータス コードが返される場合、データはまったく取得されません。

- 43 指定されたレコード アドレスが無効です。
- 58 圧縮バッファ長が短すぎます。
- 62 ディスクリプターが不適切です。
- 97 データ バッファが小さすぎます。

106 MicroKernel エンジンでは、Get Next Chunk オペレーションを実行できません。

ポジショニング

Get Direct/Chunk オペレーションは、論理カレンシーにまったく影響しません。物理カレンシーについては、チャンクが取り出されたレコードが現在の物理レコードになります。

Get Direct/Record (23)

Get Direct/Record オペレーション (B_GET_DIRECT) では、定義されているキーパスではなく、ファイル内の物理位置を使ってレコードを取得します。

以下のような操作を実行する場合は、Get Direct/Record オペレーションを使用してください。

- キー値の代わりに物理位置を使って、より高速にレコードを取得する。
- Get Position (22) を使ってレコードの物理位置を取得し、その位置を保存する。それから、カレンシーに影響を与えるほかのオペレーションを実行した後で Get Direct/Record を使って、その位置に直接戻る。
- 一連の重複レコードの中から 1 つのレコードを取得するとき、その一連のレコードを先頭からすべて読み取りし直すことなく、物理位置を使って取得する。
- 現在のキーパスを変更する。Get Position オペレーションに続けて、別のキー番号を使った Get Direct/Record オペレーションを実行すると、別のインデックスパスに現在のレコードのポジショニングが確立します。この後で Get Next オペレーションを実行すると、新しいキーパスに基づいてファイル内の次のレコードが返されます。

パラメーター

	オペレーション コード	ポジション ブロック	データ バッ ファー	データ バッ ファー長	キー バッ ファー	キー番号
送り値	X	X	X	X		X
戻り値		X	X	X	X	

メモ： データオンリーファイルで Get Direct/Record オペレーションを実行する場合は、キー番号パラメーターは必要ありません。

前提条件

- 対象となるファイルが開いていることが必要です。
- 4 バイトまたは 8 バイトから成るレコードの物理位置を用意する必要があります。この値は、Get Position (22) を使って取得できます。このオペレーションを実行す

ると、現在のレコードの物理アドレスが返されます。BTRV または BTRVEX タイプのエントリ ポイントの使用と一貫性を持たせてください。

手順

1. オペレーション コードを 23 に設定します。任意でロック バイアスも指定できます。
 - +100 - 単一レコード ウェイト ロック
 - +200 - 単一レコード ノーウェイト ロック
 - +300 - 複数レコード ウェイト ロック
 - +400 - 複数レコード ノーウェイト ロック

レコード ロックおよびデータ整合性については、『*Zen Programmer's Guide*』のほかに、『*Advanced Operations Guide*』に記載されている Zen サーバーを設定するための **ウェイト ロック タイムアウト** プロパティを参照してください。

2. ファイルのポジションブロックを渡します。
3. データ バッファの先頭に、目的のレコードの位置を表す 4 バイト値または 8 バイト値を格納します。サイズは、使用しているエントリ ポイントが BTRV タイプか BTRVEX タイプかによって決まります。
4. データ バッファ長を、取得するレコードの長さ以上の値に設定します。
5. キー番号を、MicroKernel エンジンで論理カレンシーを確立するパスのキー番号に設定します。MicroKernel エンジンで論理カレンシーの確立を必要としない場合は、-1 を指定します。システム定義のログ キー（システム データとも呼ばれる）を使用するには、125 を指定します。システム データ v2 用の 2 番目のシステム キーを使用するには、124 を指定します。

結果

Get Direct/Record オペレーションが正常に終了した場合、MicroKernel エンジンでは要求したレコードがデータ バッファに、レコード長がデータ バッファ長に、指定したキーパスのキー値がキー バッファにそれぞれ返されます。

Get Direct/Record オペレーションが失敗し、要求したレコードを MicroKernel エンジンが取得できなかった場合は、MicroKernel エンジンから次のステータス コードのいずれかが返されます。

- 22 データ バッファ パラメーターが短すぎます。(論理カレンシーはまだ確立されています)
- 43 指定されたレコード アドレスが無効です。(論理カレンシーは確立されていません)
- 44 指定されたキー パスが無効です。(論理カレンシーは確立されていません)
- 82 MicroKernel エンジンがポジショニングを失いました。(論理カレンシーは確立されていません)

ポジショニング

Get Direct/Record オペレーションを実行すると、既存の論理カレンシー情報が消去され、指定したキー番号に従って新しい論理カレンシーが確立されます。物理カレンシー情報にはまったく影響しません。

Get Directory (18)

Get Directory オペレーション (B_GET_DIR) では、指定された論理ディスクドライブの現在のディレクトリを返します。

パラメーター

	オペレーション コード	ポジション ブロック	データ バッ ファー	データバッ ファー長	キーバッ ファー	キー番号
送り値	X					X
戻り値					X	

前提条件

Get Directory オペレーションはいつでも発行することができます。キーバッファーには少なくとも 65 文字の長さが必要です。

手順

1. オペレーション コードを 18 に設定します。
2. キー番号パラメーターに論理ディスクドライブ番号を格納します。ドライブは、A の場合は 1、B の場合は 2、というように指定します。デフォルトのドライブを使用するには 0 を指定します。

結果

MicroKernel エンジンでは、オペレーションが正常に終了した場合、バイナリ 0 で終端する現在のディレクトリがキーバッファーに返されます。

ポジショニング

Get Directory オペレーションは、ファイルのカレンシー情報にはまったく影響しません。

Get Equal (5)

Get Equal オペレーション (B_GET_EQUAL) では、キーバッファに指定されたキー値と等しいキー値を持つレコードを取得します。キーの重複が可能な場合は、同じキー値を持つグループの中で先頭のレコード（作成順）が取得されます。[Get Key \(+50\)](#) バイアスを使うと、ファイル内に値が存在するかどうかを検出することもできます。一般に、Get Key オペレーションの方が高速に処理されます。

パラメーター

	オペレーションコード	ポジションブロック	データバッファ	データバッファ長	キーバッファ	キー番号
送り値	X	X		X	X	X
戻り値		X	X	X		

前提条件

- 対象となるファイルが開いている必要があります。
- ファイルがデータオンリーファイルであってははいけません。

手順

- オペレーションコードを5に設定します。任意でロックバイアスも指定できます。
 - +100 - 単一レコード ウェイト ロック
 - +200 - 単一レコード ノーウェイト ロック
 - +300 - 複数レコード ウェイト ロック
 - +400 - 複数レコード ノーウェイト ロック

レコードロックおよびデータ整合性については、『*Zen Programmer's Guide*』のほかに、『*Advanced Operations Guide*』に記載されている Zen サーバーを設定するための [ウェイト ロック タイムアウト](#) プロパティを参照してください。

- ファイルのポジションブロックを渡します。
- データバッファ長を、取得するレコードの長さ以上の値に設定します。

-
4. キーバッファに目的のキー値を指定します。キーが複数のセグメントから成る場合は、必ずすべてのセグメントを記述し、それらすべてに値を設定するようキーバッファを定義してください。すべてのセグメントについて検索条件を設定しない場合は、代わりに **Get Greater Than Or Equal** オペレーションを使用してください。
 5. キー番号を正しいキーパスに設定します。システム定義のログ キー（システムデータとも呼ばれる）を使用するには、125 を指定します。システムデータ v2 用の 2 番目のシステム キーを使用するには、124 を指定します。

結果

Get Equal オペレーションが正常に終了した場合は、**MicroKernel** エンジンでは要求したレコードがデータ バッファに、そのレコードの長さがデータ バッファ長に返されます。

Get Equal オペレーションが失敗した場合は、**MicroKernel** エンジンから次のステータスコードのいずれかが返されます。

- 3 ファイルが開いていません。
- 4 アプリケーションがキー値を見つけられません。
- 6 キー番号パラメーターが無効です。
- 22 データ バッファ パラメーターが短すぎます。

このオペレーションは、キーのヌル インジケータ セグメントにゼロ以外の値が含まれている場合にはステータスコード 4 を返します。**Get Equal** を使ってヌルのレコードは検索できません。これは、ヌルの定義はあいまいなものであり、どの値とも等しくならないからです。ヌル値の検索が必要な場合は、**Get First** オペレーションに続けて **Get Next** オペレーションを使用します。

ポジショニング

Get Equal オペレーションを実行すると、完全な論理カレンシーおよび物理カレンシーが確立し、取得したレコードが現在のレコードになります。

Get First (12)

Get First オペレーション (B_GET_FIRST) では、指定されたキーに基づいて先頭の論理レコードを取得します。Get Key (+50) バイアスを使うと、ファイル内に値が存在するかどうかを検出することもできます。一般に、Get Key オペレーションの方が高速に処理されます。

パラメーター

	オペレーションコード	ポジションブロック	データバッファー	データバッファー長	キーバッファー	キー番号
送り値	X	X		X		X
戻り値		X	X	X	X	

前提条件

- 対象となるファイルが開いていることが必要です。
- ファイルがデータオンリーファイルであってははいけません。

手順

- オペレーションコードを 12 に設定します。任意でロック バイアスも指定できます。
 - +100 - 単一レコード ウェイト ロック
 - +200 - 単一レコード ノーウェイト ロック
 - +300 - 複数レコード ウェイト ロック
 - +400 - 複数レコード ノーウェイト ロックレコード ロックおよびデータ整合性については、『*Zen Programmer's Guide*』のほかに、『*Advanced Operations Guide*』に記載されている Zen サーバーを設定するための **ウェイト ロック タイムアウト** プロパティを参照してください。
- ファイルのポジションブロックを渡します。
- データ バッファー長を、取得するレコードの長さ以上の値に設定します。

-
- キー番号をキーパスに設定します。システム定義のログキー（システムデータとも呼ばれる）を使用するには、125を指定します。システムデータ v2用の2番目のシステムキーを使用するには、124を指定します。

結果

Get First オペレーションが正常に終了した場合は、MicroKernel エンジンでは要求したレコードがデータバッファーに返され、対応するキー値がキーバッファーに格納され、さらにそのレコードの長さがデータバッファー長に返されます。

Get First オペレーションが失敗した場合は、MicroKernel エンジンから次のステータスコードのいずれかが返されます。

- 3 ファイルが開いていません。
- 6 キー番号パラメーターが無効です。
- 9 オペレーションが EOF (end-of-file) を検出しました。
- 22 データバッファーパラメーターが短すぎます。

ポジショニング

Get First オペレーションを実行すると、完全な論理カレンシーおよび物理カレンシーが確立し、取得したレコードが現在のレコードになります。論理位置の直前は、ファイルの先頭よりも前を指すこととなります。

Get Greater Than (8)

Get Greater Than オペレーション (B_GET_GT) では、キー番号で指定されたフィールドが、キーバッファの値よりも次に大きな値を含むレコードを取得します。キーの重複が可能な場合は、同じキー値を持つグループの中で先頭のレコード（作成順）が取得されます。[Get Key \(+50\)](#) バイアスを使うと、ファイル内に値が存在するかどうかを検出することもできます。一般に、Get Key オペレーションの方が高速に処理されます。

メモ： 降順キーで Get Greater Than オペレーションを実行する場合、「次に大きな値」というのは、実際にはキーバッファで指定された値よりも小さな値を指すことになります。

パラメーター

	オペレーションコード	ポジションブロック	データバッファ	データバッファ長	キーバッファ	キー番号
送り値	X	X		X	X	X
戻り値		X	X	X	X	

前提条件

- 対象となるファイルが開いている必要があります。
- ファイルがデータオンリーファイルであってははいけません。

手順

- オペレーションコードを8に設定します。任意でロックバイアスも指定できます。
 - +100 - 単一レコード ウェイト ロック
 - +200 - 単一レコード ノーウェイト ロック
 - +300 - 複数レコード ウェイト ロック
 - +400 - 複数レコード ノーウェイト ロック

レコード ロックおよびデータ整合性については、『*Zen Programmer's Guide*』のほかに、『*Advanced Operations Guide*』に記載されている Zen サーバーを設定するための **ウェイト ロック タイムアウト** プロパティを参照してください。

2. ファイルのポジションブロックを渡します。
3. データ バッファ長を、取得するレコードの長さ以上の値に設定します。
4. キー バッファ パラメーターに目的のキー値を指定します。
5. キー番号パラメーターを正しいキー パスに設定します。システム定義のログ キー (システム データとも呼ばれる) を使用するには、125 を指定します。システム データ v2 用の 2 番目のシステム キーを使用するには、124 を指定します。

結果

Get Greater Than オペレーションが正常に終了した場合、MicroKernel エンジンでは要求したレコードがデータ バッファに、キー値がキー バッファに、さらにそのレコードの長さがデータ バッファ長に格納されます。

Get Greater Than オペレーションが失敗した場合は、MicroKernel エンジンから次のステータス コードのいずれかが返されます。

- 3 ファイルが開いていません。
- 6 キー番号パラメーターが無効です。
- 22 データ バッファ パラメーターが短すぎます。

ポジショニング

Get Greater Than オペレーションを実行すると、完全な論理カレンシーおよび物理カレンシーが確立し、取得したレコードが現在のレコードになります。

Get Greater Than or Equal (9)

Get Greater Than or Equal オペレーション (B_GET_GE) では、キー番号で指定されたキーが、キーバッファに指定された値と等しいかそれよりも大きな値を持つレコードを取得します。MicroKernel エンジンではまず、等しいという条件を満たすレコードが検索されます。キーの重複が可能な場合は、同じキー値を持つグループの中で先頭のレコード（作成順）が取得されます。Get Key (+50) バイアスを使うと、ファイル内に値が存在するかどうかを検出することもできます。一般に、Get Key オペレーションの方が高速に処理されます。

メモ： 降順キーで Get Greater Than or Equal オペレーションを実行する場合、「次に大きな値」というのは、実際にはキーバッファで指定された値よりも小さな値を指すこととなります。

パラメーター

	オペレーションコード	ポジションブロック	データバッファ	データバッファ長	キーバッファ	キー番号
送り値	X	X		X	X	X
戻り値		X	X	X	X	

前提条件

- 対象となるファイルが開いている必要があります。
- ファイルがデータオンリーファイルであってははいけません。

手順

- オペレーションコードを9に設定します。任意でロックバイアスも指定できます。
 - +100 - 単一レコード ウェイト ロック
 - +200 - 単一レコード ノーウェイト ロック
 - +300 - 複数レコード ウェイト ロック
 - +400 - 複数レコード ノーウェイト ロック

レコード ロックおよびデータ整合性については、『*Zen Programmer's Guide*』のほかに、『*Advanced Operations Guide*』に記載されている Zen サーバーを設定するための **ウェイト ロック タイムアウト** プロパティを参照してください。

2. ファイルのポジションブロックを渡します。
3. データ バッファ長を、取得するレコードの長さ以上の値に設定します。
4. キー バッファ パラメーターにキー値を指定します。
5. キー番号パラメーターを正しいキー パスに設定します。システム定義のログ キー（システム データとも呼ばれる）を使用するには、125 を指定します。システム データ v2 用の 2 番目のシステム キーを使用するには、124 を指定します。

結果

Get Greater Than or Equal オペレーションが正常に終了した場合、**MicroKernel** エンジンでは要求したレコードがデータ バッファに、キー値がキー バッファに、さらにそのレコードの長さがデータ バッファ長に格納されます。

Get Greater Than or Equal オペレーションが失敗した場合は、**MicroKernel** エンジンから次のステータス コードのいずれかが返されます。

- 3 ファイルが開いていません。
- 6 キー番号パラメーターが無効です。
- 22 データ バッファ パラメーターが短すぎます。

ポジショニング

Get Greater Than or Equal オペレーションを実行すると、完全な論理カレンシーおよび物理カレンシーが確立し、取得したレコードが現在のレコードになります。

Get Key (+50)

Get Key バイアスを使用すると、実際にデータレコードを取得することなく Get オペレーションを実行できます。Get Key を使って、ファイル内にある値が存在するかどうかを検出できます。一般に、Get Key オペレーションは対応する Get オペレーションよりも高速に実行できます。Get Key オペレーションは、以下のいずれかの Get オペレーションと共に使用します。

- [Get Equal \(5\)](#)
- [Get Next \(6\)](#)
- [Get Previous \(7\)](#)
- [Get Greater Than \(8\)](#)
- [Get Greater Than or Equal \(9\)](#)
- [Get Less Than \(10\)](#)
- [Get Less Than or Equal \(11\)](#)
- [Get First \(12\)](#)
- [Get Last \(13\)](#)
- [Get By Percentage \(44\)](#)

パラメーター

パラメーターは対応する Get オペレーションと同様です。ただし、MicroKernel エンジンではデータバッファ長の設定は無視され、データバッファにはレコードが返されません。

前提条件

Get Key オペレーションの前提条件は、対応する Get オペレーションの前提条件と同じです。

手順

1. 対応する Get オペレーションの場合と同じようにパラメーターを設定します。データバッファ長を初期化する必要はありません。

-
2. オペレーションコードを、実行する Get オペレーションのオペレーションコードに 50 を加算した値に設定します。たとえば、Get Equal (5) と共に Get Key (+50) を実行するには、オペレーションコードを 55 に設定します。

MicroKernel エンジンでは、Get Key (+50) の後に Delete または Update オペレーションを実行することはできません。MicroKernel エンジンで、Delete または Update オペレーションを実行する前に、変更しようとしているデータ ページの現在の使用回数と、レコードを読み取った時点のデータ ページの使用回数が比較されます。使用回数を取得するには、MicroKernel エンジンがデータ ページを読み取る必要があります。

Get Key オペレーションではデータ ページを読み取らないので、Delete または Update オペレーションで比較するための使用回数が利用可能になりません。MicroKernel エンジンでは、比較なしにパッシブ並行制御の矛盾チェックを実行できないため、Update または Delete オペレーションは正常に実行されません。Update または Delete オペレーションが正常に実行されないと、MicroKernel エンジンからステータスコード 8 が返されます。

結果

MicroKernel エンジンで要求したキーが検出されると、そのキー値がキー バッファーに格納され、ステータスコード 0 が返されます。そうでない場合は、MicroKernel エンジンからキー値を検出できなかった理由を示すステータスコードが返されます。

ポジショニング

Get Key オペレーションを実行すると、対応する Get オペレーションと同様の方法で現在のポジショニングが確立されます。ただし、Get Key オペレーションの対象となるキーが重複を許可している場合、MicroKernel エンジンでは取得された現在のキー値の重複インスタンスは無視されます。Get Key オペレーションの実行後、論理位置の直前は次に小さなキー値を含むレコードを指します。また、論理位置の直後は次に大きなキー値を含むレコードを指します。

たとえば、Smith が 8 回と Smythe が 1 回出現する姓のキーを対象に、Get Key を Get Equal オペレーション (55) と共に実行したとします。論理位置の直後は次の Smith ではなく、Smythe を指すことになります。

Get Key オペレーションではどれか 1 つのレコードが識別されるわけではないため、MicroKernel エンジンでは Get Key オペレーションに続けて Update または Delete オペレーションを実行することはできません。

Get Last (13)

Get Last オペレーション (B_GET_LAST) では、指定されたキーに基づいて末尾の論理レコードを取得します。末尾のキー値が重複している場合は、同じキー値を持つグループの中で末尾のレコードが返されます。Get Key (+50) バイアスを使うと、ファイル内に値が存在するかどうかを検出することもできます。一般に、Get Key オペレーションの方が高速に処理されます。

パラメーター

	オペレーション コード	ポジション ブロック	データ バッ ファー	データ バッ ファー長	キー バッ ファー	キー番号
送り値	X	X		X		X
戻り値		X	X	X	X	

前提条件

- 対象となるファイルが開いていることが必要です。
- ファイルがデータオンリーファイルであってははいけません。

手順

1. オペレーション コードを 13 に設定します。任意でロック バイアスも指定できます。
 - +100 - 単一レコード ウェイト ロック
 - +200 - 単一レコード ノーウェイト ロック
 - +300 - 複数レコード ウェイト ロック
 - +400 - 複数レコード ノーウェイト ロックレコード ロックおよびデータ整合性については、『Zen Programmer's Guide』のほかに、『Advanced Operations Guide』に記載されている Zen サーバーを設定するための **ウェイト ロック タイムアウト** プロパティを参照してください。
2. ファイルのポジションブロックを渡します。
3. データ バッファー長を、取得するレコードの長さ以上の値に設定します。

-
4. キー番号をキーパスに設定します。システム定義のログキー（システムデータとも呼ばれる）を使用するには、125を指定します。システムデータ v2 用の 2 番目のシステムキーを使用するには、124を指定します。

結果

Get Last オペレーションが正常に終了した場合は、MicroKernel エンジンでは要求したレコードがデータバッファに返され、対応するキー値がキーバッファに格納され、さらにそのレコードの長さがデータバッファ長に返されます。

Get Last オペレーションが失敗した場合は、MicroKernel エンジンから次のステータスコードのいずれかが返されます。

- 3 ファイルが開いていません。
- 6 キー番号パラメーターが無効です。
- 9 オペレーションが EOF (end-of-file) を検出しました。
- 22 データバッファパラメーターが短すぎます。

ポジショニング

Get Last オペレーションを実行すると、完全な論理カレンシーおよび物理カレンシーが確立し、取得したレコードが現在のレコードになります。論理位置の直後は、ファイルの末尾よりも後を指すことになります。

Get Less Than (10)

Get Less Than オペレーション (B_GET_LT) では、キー番号で指定されたキーが、キーバッファに指定された値よりも次に小さな値を持つレコードを取得します。キーの重複が可能な場合は、同じキー値を持つグループの中で末尾のレコード（作成順）が取得されます。**Get Key (+50)** バイアスを使うと、ファイル内に値が存在するかどうかを検出することもできます。一般に、Get Key オペレーションの方が高速に処理されます。

メモ： 降順キーで Get Less Than オペレーションを実行する場合、「次に小さな値」というのは、実際にはキーバッファで指定された値よりも大きな値を指すこととなります。

パラメーター

	オペレーション コード	ポジション ブロック	データ バッ ファー	データ バッ ファー長	キー バッ ファー	キー番号
送り値	X	X		X	X	X
戻り値		X	X	X	X	

前提条件

- 対象となるファイルが開いていることが必要です。
- ファイルがデータオンリーファイルであってははいけません。

手順

- オペレーションコードを 10 に設定します。任意でロック バイアスも指定できます。
 - +100 - 単一レコード ウェイト ロック
 - +200 - 単一レコード ノーウェイト ロック
 - +300 - 複数レコード ウェイト ロック
 - +400 - 複数レコード ノーウェイト ロック

レコード ロックおよびデータ整合性については、『*Zen Programmer's Guide*』のほかに、『*Advanced Operations Guide*』に記載されている Zen サーバーを設定するための **ウェイト ロック タイムアウト** プロパティを参照してください。

2. ファイルのポジションブロックを渡します。
3. データ バッファ長を、取得するレコードの長さ以上の値に設定します。
4. キー バッファ パラメーターに目的のキー値を指定します。
5. キー番号パラメーターをキー パスに設定します。システム定義のログ キー（システム データとも呼ばれる）を使用するには、125 を指定します。システム データ v2 用の 2 番目のシステム キーを使用するには、124 を指定します。

結果

Get Less Than オペレーションが正常に終了した場合、**MicroKernel** エンジンではレコードがデータ バッファに、そのレコードのキー値がキー バッファに、さらにそのレコードの長さがデータ バッファ長に返されます。

Get Less Than オペレーションが失敗した場合は、**MicroKernel** エンジンから次のステータス コードのいずれかが返されます。

- 3 ファイルが開いていません。
- 6 キー番号パラメーターが無効です。
- 22 データ バッファ パラメーターが短すぎます。

ポジショニング

Get Less Than オペレーションを実行すると、完全な論理カレンシーおよび物理カレンシーが確立し、取得したレコードが現在のレコードになります。

Get Less Than or Equal (11)

Get Less Than or Equal オペレーション (B_GET_LE) では、キー番号で指定されたキーが、キーバッファに指定された値と等しいかそれよりも小さな値を持つレコードを取得します。MicroKernel エンジンではまず、等しいという条件を満たすレコードが検索されます。キーの重複が可能な場合は、同じキー値を持つグループの中で末尾のレコード（作成順）が取得されます。Get Key (+50) バイアスを使うと、ファイル内に値が存在するかどうかを検出することもできます。一般に、Get Key オペレーションの方が高速に処理されます。

メモ： 降順キーで Get Less Than or Equal オペレーションを実行する場合、「次に小さな値」というのは、実際にはキーバッファで指定された値よりも大きな値を指すことになります。

パラメーター

	オペレーション コード	ポジション ブロック	データ バッ ファ	データ バッ ファ長	キー バッ ファ	キー番号
送り値	X	X		X	X	X
戻り値		X	X	X	X	

前提条件

- 対象となるファイルが開いていることが必要です。
- ファイルがデータオンリーファイルであってははいけません。

手順

- オペレーション コードを 11 に設定します。任意でロック バイアスも指定できます。
 - +100 - 単一レコード ウェイト ロック
 - +200 - 単一レコード ノーウェイト ロック
 - +300 - 複数レコード ウェイト ロック
 - +400 - 複数レコード ノーウェイト ロック

レコード ロックおよびデータ整合性については、『*Zen Programmer's Guide*』のほかに、『*Advanced Operations Guide*』に記載されている Zen サーバーを設定するための **ウェイト ロック タイムアウト** プロパティを参照してください。

2. ファイルのポジションブロックを渡します。
3. データ バッファ長を、取得するレコードの長さ以上の値に設定します。
4. キー バッファ パラメーターにキー値を指定します。
5. キー番号パラメーターをキー パスに設定します。システム定義のログ キー（システム データとも呼ばれる）を使用するには、125 を指定します。システム データ v2 用の 2 番目のシステム キーを使用するには、124 を指定します。

結果

Get Less Than or Equal オペレーションが正常に終了した場合、**MicroKernel** エンジンではレコードがデータ バッファに、そのレコードのキー値がキー バッファに、さらにそのレコードの長さがデータ バッファ長に返されます。

Get Less Than or Equal オペレーションが失敗した場合は、**MicroKernel** エンジンから次のステータス コードのいずれかが返されます。

- 3 ファイルが開いていません。
- 6 キー番号パラメーターが無効です。
- 22 データ バッファ パラメーターが短すぎます。

ポジショニング

Get Less Than or Equal オペレーションを実行すると、完全な論理カレンシーおよび物理カレンシーが確立し、取得したレコードが現在のレコードになります。

Get Next (6)

Get Next オペレーション (B_GET_NEXT) では、指定されたキーに基づいて、論理位置で次にあるレコードを取得します。Get Next オペレーションを使うと、重複するキー値を持つレコードのグループの中でレコードを検索できます。Get Key (+50) バイアスを使うと、ファイル内に値が存在するかどうかを検出することもできます。一般に、Get Key オペレーションの方が高速に処理されます。

パラメーター

	オペレーション コード	ポジション ブロック	データ バッ ファー	データバッ ファー長	キー バッ ファー	キー番号
送り値	X	X		X	X	X
戻り値		X	X	X	X	

前提条件

- 対象となるファイルが開いていることが必要です。
- ファイルがデータオンリーファイルであってははいけません。
- アプリケーションでは、指定したキーに基づく次の論理位置を確立しておくことが必要です。

手順

- オペレーション コードを 6 に設定します。任意でロック バイアスも指定できます。
 - +100 - 単一レコード ウェイト ロック
 - +200 - 単一レコード ノーウェイト ロック
 - +300 - 複数レコード ウェイト ロック
 - +400 - 複数レコード ノーウェイト ロック

レコード ロックおよびデータ整合性については、『*Zen Programmer's Guide*』のほかに、『*Advanced Operations Guide*』に記載されている Zen サーバーを設定するための **ウェイト ロック タイムアウト** プロパティを参照してください。

-
2. ファイルのポジションブロックを渡します。
 3. データバッファ長を、取得するレコードの長さ以上の値に設定します。
 4. キーバッファに、論理位置を確立した前のオペレーションで取得したキー値を指定します。

キーバッファには、前の呼び出しで **MicroKernel** エンジンから返されたキー値とまったく同じものを渡します。ここに格納された情報が、ファイル内の現在の位置を決定するために必要となるからです。

5. キー番号パラメーターを、論理位置を確立した前の呼び出しで使用したキーパスに設定します。 **Get Next** オペレーションを使ってキーパスを変更することはできません。

結果

Get Next オペレーションが正常に終了した場合、**MicroKernel** エンジンではレコードがデータバッファに、そのレコードのキー値がキーバッファに、さらにそのレコードの長さがデータバッファ長に返されます。

Get Next オペレーションが失敗した場合は、**MicroKernel** エンジンから次のステータスコードのいずれかが返されます。

- 3 ファイルが開いていません。
- 6 キー番号パラメーターが無効です。
- 7 キー番号が変更されました。
- 8 現在のポジションが無効です。
- 9 オペレーションが EOF (end-of-file) を検出しました。
- 22 データバッファパラメーターが短すぎます。
- 82 **MicroKernel** エンジンがポジショニングを失いました。

このオペレーションの実行により、論理位置の直後がファイルの末尾よりも後を指す場合は、ステータスコード 9 が返されます。

ポジショニング

Get Next オペレーションを実行すると、完全な論理カレンシーおよび物理カレンシーが確立し、取得したレコードが現在のレコードになります。

Get Next Delete Extended (85)

Get Next Delete Extended オペレーション (B_GET_NEXT_EXT_DELETE) では、指定されたキーに基づき、論理位置の直後からファイルの末尾へ向かって1つまたは複数のレコードを検索します。検索したレコードをフィルター条件と比較し、条件に一致するレコードを削除します。フィルター条件は論理式の形を取り、キーフィールドに限定されません。

このトピックで述べるように、このオペレーションが使用する入力および出力バッファー構造体、および返す結果は、[Get Next Extended \(36\)](#) に記載されているものと同じです。詳細については、当該オペレーションを参照してください。

パラメーター

	オペレーションコード	ポジションブロック	データバッファー	データバッファー長	キーバッファー	キー番号
送り値	X	X	X	X	X	X
戻り値		X	X	X	X	

前提条件

- 対象となるファイルが開いていることが必要です。
- ファイルがデータオンリーファイルであってははいけません。
- 指定したキーに基づく次の論理位置を確立しておくことが必要です。論理位置は、Get Equal など非拡張の (Extended でない) Get オペレーションをどれか発行することによって確立できます。

手順

- オペレーションコードを 85 に設定します。

デフォルトでは、ロックバイアスはノーウェイトで、どのロックバイアス設定も無視されます。動作は +500 と同じです。エンジンは、ロックされたレコードを削除できない場合には、オペレーションを再試行しないで直ちに返ります。

- ファイルのポジションブロックを渡します。

-
3. 入力構造体と戻り出力のどちらか大きい方を格納できるように、データバッファを指定します。 [Get Next Extended \(36\)](#) にある `Extended` オペレーションの入力バッファの情報に従って、データバッファを初期化します。
 4. バッファサイズには、 [Get Next Extended \(36\)](#) にある入力および出力バッファ情報の説明に従って、入力構造体の長さで戻り出力の長さのどちらか大きい方を指定します。
 5. キーバッファに、論理位置を確立した前のオペレーションで取得したキー値を指定します。
キーバッファには、前の呼び出しで `MicroKernel` エンジンから返されたキー値とまったく同じものを渡します。ここに格納された情報が、ファイル内の現在の位置を決定するために必要となるからです。
 6. キー番号パラメーターを、論理位置を確立した前の呼び出しで使用したキーパスに設定します。 `Get Next Delete Extended` オペレーションを使ってキーパスを変更することはできません。

詳細

[Get Next Extended \(36\)](#) の以下のトピックで、`Extended` オペレーションの入力バッファの構造体とそのフィルターセグメントの使用について、さらに結果を返す出力バッファの構造体について説明されています。

- [Extended オペレーションの入力バッファ](#)
- [Extended オペレーションの出力バッファ](#)

結果

`Get Next Delete Extended` オペレーションが正常に終了した場合は、`MicroKernel` エンジンから次の情報が返されます。

- 出力バッファには、取得された 1 つまたは複数のレコードの 1 つまたは複数のレコードアドレスが格納されます。詳細については、[Extended オペレーションの出力バッファ](#)を参照してください。
- 出力バッファ長には、受け取った総バイト数が格納されます。
- キーバッファには、検索された最後のデータレコードのキー値が格納されます。

Get Next Delete Extended オペレーションは、ほかの Step Extended および Get Extended オペレーションと同じ原因で失敗する可能性があり、次のいずれか 1 つが返されます。

- 3 ファイルが開いていません。
- 6 キー番号パラメーターが無効です。
- 7 キー番号が変更されました。
- 8 現在のポジションが無効です。
- 9 オペレーションが EOF (end-of-file) を検出しました。
- 22 データ バッファ パラメーターが短すぎます。
- 60 指定されたリジェクト カウントに達しました。
- 61 作業領域が小さすぎます。
- 62 ディスクリプターが不適切です。
- 64 フィルター制限に達しました。
- 65 フィールド オフセットが不適切です。
- 82 MicroKernel エンジンがポジショニングを失いました。
- 134 MicroKernel エンジンがインターナショナル ソート規則 (ISR) を読み取れません。
- 135 指定されたインターナショナル ソート規則 (ISR) テーブルは破損しているか、または無効です。
- 136 MicroKernel エンジンは、指定されたオルタネート コレーティング シーケンスをファイル内に見つけられません。

出力バッファの長さがゼロの場合、レコードは削除されていません。ただし、オペレーションが失敗する前に一部のレコードを正常に削除している可能性があります。以下に、このような一部成功のいくつかの例を示します。

- フィルター条件に一致する現在のレコードのレコード アドレスを書き出すためのスペースが出力バッファにありません。そのレコードは削除されず、オペレーションはステータス コード 22 で失敗します。
- 別のクライアントが現在のレコードをロックしている場合、オペレーションはステータス コード 84 で失敗します。

このような場合、出力バッファ長はゼロより大きく、バッファの最初の 2 バイトは削除されたレコード数のカウントを提供します。

MicroKernel エンジンではフィルター条件で使用するフィールドと演算子によって、要求を最適化できる場合があります。拒否レコードに達すると、ステータス コード 64 が返され、ファイルの未検索の部分にはフィルター条件を満たすレコードがそれ以上ないことが示されます。

ポジショニング

Get Next Delete Extended オペレーションでは、カレンシーは確立しません。ただし、Get Next オペレーションまたは Get Previous オペレーションを実行することができ、次または前の論理位置は有効です。また、Get Position (22) および Get Direct/Record (23) を使用することにより、有効な現在の位置も使用可能になります。

次のリストは、選定されたステータスコードとフィルター条件の関係を示しています。

- ステータス 60 (リジェクト カウントに達しました) : 現在の位置は、フィルター条件に一致しないレコードです。
- ステータス 64 (フィルター制限に達しました) : 現在の位置は、フィルター条件に一致しない可能性のあるレコードです。次のレコードに進もうとしても、フィルター条件に一致しません。
- ステータス 84 (レコードまたはページはロックされています) : 現在の位置は、フィルター条件に一致しない可能性のあるレコードです。また、次のレコードはフィルター条件に一致するが、ロックされているために削除できないという可能性もあります。
- ステータス 22 (データ バッファがいっぱいです) : 現在の位置は、フィルター条件に一致するレコードです。ただし、データ バッファにはレコード アドレスを書き込むためのスペースがないため、MicroKernel エンジンはそのレコードを削除しませんでした。
- ステータス 9 (ファイルの終わり) : 現在の位置は、論理的にも物理的にも無効です。

Get Next Extended (36)

Get Next Extended オペレーション (B_GET_NEXT_EXTENDED) では、指定されたキーに基づき、論理位置の直後からファイルの末尾へ向かって1つまたは複数のレコードを検索します。検索したレコードをフィルター条件と比較し、条件に一致するレコードを取得します。フィルター条件は論理式の形を取り、キーフィールドに限定されません。

Get Next Extended オペレーションでは、レコードから指定した部分だけを抽出し、その部分だけをアプリケーションに返すこともできます。

パラメーター

	オペレーションコード	ポジションブロック	データバッファー	データバッファー長	キーバッファー	キー番号
送り値	X	X	X	X	X	X
戻り値		X	X	X	X	

前提条件

- 対象となるファイルが開いていることが必要です。
- ファイルがデータオンリーファイルであってははいけません。
- 指定したキーに基づく次の論理位置を確立しておくことが必要です。論理位置は、Get Equal など非拡張の (Extended でない) Get オペレーションをどれか発行することによって確立できます。

手順

- オペレーションコードを 36 に設定します。任意でロック バイアスも指定できます。
 - +100 - 単一レコード ウェイト ロック
 - +200 - 単一レコード ノーウェイト ロック
 - +300 - 複数レコード ウェイト ロック
 - +400 - 複数レコード ノーウェイト ロック

レコード ロックおよびデータ整合性については、『*Zen Programmer's Guide*』のほかに、『*Advanced Operations Guide*』に記載されている Zen サーバーを設定するための **ウェイト ロック タイムアウト** プロパティを参照してください。

2. ファイルのポジションブロックを渡します。
3. 入力構造体と戻り出力のどちらか大きい方を格納できるように、データ バッファを指定します。 **Get Next Extended (36)** にある Extended オペレーションの入力バッファの情報に従って、データ バッファを初期化します。
4. バッファ サイズには、 **Get Next Extended (36)** にある入力および出力バッファ情報の説明に従って、入力構造体の長さおよび戻り出力の長さのどちらか大きい方を指定します。
5. キー バッファに、論理位置を確立した前のオペレーションで取得したキー値を設定します。キー バッファには、前の呼び出しで **MicroKernel** エンジンから返されたキー値とまったく同じものを渡します。ここに格納された情報が、ファイル内の現在の位置を決定するために必要となるからです。
6. キー番号パラメーターを、論理位置を確立した前の呼び出しで使用したキーパスに設定します。 **Get Next Extended** オペレーションを使ってキーパスを変更することはできません。

詳細

以下のトピックで、 **Extended** オペレーションの入力バッファの構造体とそのフィルターセグメントの使用について、さらにオペレーションの結果を返す出力バッファの構造体について説明します。

- **Extended オペレーションの入力バッファ**
- **LIKE 結果の照合順序**
- **JSON クエリ演算子の使用**
- **フィルター内の論理 AND および OR の処理**
- **レコードのフィルタリングの例**
- **Extended オペレーションの出力バッファ**

Extended オペレーションの入力バッファ

次の表は、Get および Step の Extended オペレーションの入力バッファの構造体を示しています。このバッファはすべての Extended オペレーションに適用されますが、既に述べたように使用方法に違いがあります。

要素	長さ (バイト単位)	説明
ヘッダー	2	入力データ バッファの正確な長さ。
	2	2つの文字列定数値のいずれかを指定します（固定長で、ヌル終端にしてはいけません）。 "EG" - ポジショニングされているレコードの次のレコードから検索を開始します。 "UC" - ポジショニングされているレコードから検索を開始します。
フィルター（固定部分）	2	リジェクト カウントの最大数。これは、フィルターに一致しないレコードについて、データベース エンジンがスキップできる件数です。0 から 65535 までの範囲の値を指定できます。0 は、エンジンがデフォルトの 4095 を使用することを意味します。
	2	フィルターとして使用する論理式の項の数。0 は、MicroKernel エンジンでフィルター処理が実行されないことを意味します。項の数はデータ バッファのサイズによってのみ制限されます。Pervasive.SQL 2000i SP3 でのみ、項の数が 119 に制限されています。

要素	長さ (バイト単位)	説明	
フィルター（論理式の各項について、このセグメントを1回繰り返す）	1	フィールドのデータ型。 キー仕様ブロック に記載されている拡張データ型のコードのいずれかを使用します。	
	2	フィールド長。	
	2	フィールドのオフセット（ゼロを基準にした相対的なオフセット）。	
	1	比較演算子のコードを指定します。 1 - 等しい 2 - より大きい 3 - より小さい 4 - 等しくない 5 - 以上 6 - 以下 7 - Extended オペレーションコード	<ul style="list-style-type: none"> +8 バイアスを加算すると、ファイル内の照合順序を使用して文字列が比較されます。 +32 バイアスを加算すると、ファイル内の最初の ACS を使用して文字列が比較されます。+8 バイアスが使用されている場合、+32 バイアスは無視されます。 +64 バイアスは、2 番目のオペランドがレコード内の別のフィールドである場合に加算します。 +128 バイアスを加算すると、大文字と小文字を区別しないで文字列が比較されます。
	1	AND/OR 論理演算子を指定します。 0 - 最後の項を示します。 1 - 次の項を AND で結合します。 2 - 次の項を OR で結合します。	
	1	このフィールドは、比較演算子コードが 7 の場合にのみ存在します。 1 - LIKE 演算子 2 - NOT LIKE 演算子 3 - JSON クエリ演算子	

要素	長さ (バイト 単位)	説明
(フィルターの続き)	2 または n	<ul style="list-style-type: none"> 2つのフィールドを比較する場合は、2バイトの、2番目のフィールドに対するゼロ基準の相対的なオフセットを指定します。2番目のフィールドは、同じデータ型かつ同じ長さでなければなりません。 フィールドを定数と比較する場合は、定数の値を指定します。定数の長さ (n) は、フィールドの長さと同様でなければなりません。 比較演算子コードが7の場合、この要素は次のような構造になります。 <ul style="list-style-type: none"> LIKE 句 (ヌル終端文字を含む) のサイズを表す2バイト LIKE 句、NOT LIKE 句、または JSON クエリ文字列を含むヌル終端文字列 デフォルトの円記号の代替エスケープ文字を設定するには、一致パターンのヌルバイトの後にその文字を追加し、別のヌルバイトで終了します。追加されたバイトを含めるように2バイトの長さを調整します。 LIKE 句やNOT LIKE 句は SQL の LIKE 構文を使用しますが、次のような例外があります。(1) 比較できるのは文字列型のみ。(2) LIKE 句または JSON クエリでは、比較するデータと同じ書式およびテキスト型を使用する必要がある。(3) 一重引用符を引用符で囲む必要はない。
0、9、または 17		<p>比較演算子コードが7 (バイアス+8) である場合、照合順序フィールドは LIKE 演算子および NOT LIKE 演算子について、ACS、ISR テーブル名、ICU 照合順序、またはコード ページ名を参照することができます (LIKE 結果の照合順序を参照してください)。JSON クエリ演算子が指定されている場合、このフィールドは指定しないでください。</p>

要素	長さ (バイト 単位)	説明
ディスク リプター (固定部 分)	2	取得するレコード数。レコードのセットではなく 1 つのレコードだけ を取得するには、1 を指定します。 Delete オペレーションの場合は、削除するレコード数。
	2	各レコードから抽出するフィールド数。Delete オペレーションの場合 は、ゼロに設定します。
ディスク リプター (抽出され る各 フィール ドについ て、この セグメン トを繰り返す)	2	抽出するフィールド長。Delete オペレーションでは使用されません。
	2	フィールドのオフセット (ゼロを基準にした相対的なオフセット)。 Delete オペレーションでは使用されません。 メモ : フィールド長 = 0xFF08、オフセット = 0xFFFFE (-2) の場合は、 syskey が定義されていれば、そのレコードに関連付けられている 8 バ イトの syskey が返されます。フィールド長 = 0xFF04、オフセット = 0xFFFFD (-3) の場合は、可変長データを含む、4 バイトのレコード長 が返されます。

LIKE 結果の照合順序

比較演算子コードを Extended オペレーション コード 7 (バイアス+8) として指定するオペレーションでは、次の表に示すように、照合順序フィールドは既存の ACS、ISR テーブル、ICU 照合順序、またはコード ページを参照することができます。形式は、識別バイトの後に名前が続く形です。

種類	長さの 合計 (バイト 単位)	識別バ イト	名前の 長さ (バイト 単位)	説明
ACS	9	0xAC	8	MicroKernel エンジンに ACS を識別させる、8 バ イトの一意の名前。既存の ACS 定義のみを使用 できます。
ISR	17	0xAE	16	MicroKernel エンジンに ISR テーブルを識別させ る、16 バイトの一意の名前。ISR テーブル名の一 覧については、『Zen Programmer's Guide』を参 照してください。

種類	長さの合計 (バイト単位)	識別バイト	名前の長さ (バイト単位)	説明
ICU	17	0xAE	16	サポートされる ICU 照合順序の名前。u54-msft_enus_0 または root のいずれかになります。16 バイトになるまでスペースを詰める必要があります。
コード ページ	17	0xAB	16	サポートされるコード ページの名前。16 バイトになるまでスペースを詰める必要があります。

JSON クエリ演算子の使用

JSON クエリ演算子を使用すると、JSON 文字列として格納されたデータを含んでいるレコードをフィルターすることができます。レコード内で、文字列データは ZSTRING または CLOB データ型として格納されている可能性があります。フィルター要素には比較定数を含める必要があります。比較定数は、レコード内の JSON データが満たしている必要のあるフィルター条件を指定する文字列とします。この文字列は JSON クエリとも呼ばれ、文字列自体は、ここに記載されている例に示すように、JSON を使用して指定されます。

JSON クエリ演算子は現在、コード ページ、ACS、および照合順序をサポートしていません。ZSTRING または CLOB データ フィールドのエンコードは、シングルバイトの Windows ANSI コード ページまたは UTF-8 にすることができます。

JSON クエリ演算子を使用する場合、フィルター条件に指定する比較定数は、次の例に示すような JSON 構文を使用する有効な JSON クエリでなければなりません。これはヌルで終了している必要があります。フィルター文字列が適切な JSON 形式でない場合、Extended オペレーションは、ディスクリプターが無効であることを示すステータスコード 62 で失敗します。

JSON クエリ演算子を使用するとき、比較する対象のレコードのフィールドが JSON 仕様に従っていない場合は、レコードは条件を満たさないものとして扱われます。

JSON クエリの例

次の例は、JSON のフィルタリングに必要な構文を示しています。

```
クエリ ::= { <式> }
```

```

<式> ::= <json 式> | <json 式> , <式>
<json 式> ::=
  "$and" : [ <中かっこで囲まれた json 式> , <中かっこで囲まれた json 式> ] |
  "$not" : <中かっこで囲まれた json 式> |
  "$or" : [ <中かっこで囲まれた json 式> , <中かっこで囲まれた json 式> ] |
  <フィールド式>
<中かっこで囲まれた json 式> ::=
  <中かっこで囲まれた json 式> |
  <中かっこで囲まれた json 式> , <中かっこで囲まれた json 式>
<中かっこで囲まれた json 式> ::= { <json 式> }
<フィールド式> ::=
  <フィールド> : <値> |
  <フィールド> : { "$exists" : false | true } |
  <フィールド> : { "$in" : [ <値> , <複数值> ] } |
  <フィールド> : { "$nin" : [ <値> , <複数值> ] } |
  <フィールド> : { "$type" : <型> } |
  <フィールド> : { <演算子> : <値> } |
  <フィールド> : { <フィールド式> }
<フィールド> ::= <文字列>
<複数值> ::= <値> | <値> , <複数值>
<値> ::= false | true | NULL | <文字列> | <数値> | <配列>
<型> ::= "array" | "boolean" | "null" | "number" | "object" | "string"
<演算子> ::= "$eq" | "$gt" | "$gte" | "$lt" | "$lte" | "$ne"

```

JSON 式で、クエリ値が配列でなくドキュメント値が配列である場合は、配列内の各要素が評価されます。すべての評価が true を返せば、JSON 式は true を返します。

Bureau of Transportation Statistics データから派生した別の例では、JSON 形式の航空会社の飛行データのレコードには、次のようなフィールドがあるかもしれません。

```

{
  "airport_code" : "ATL",
  "carrier_code" : "AA"
  ...
}

```

ハーツフィールド ジャクソン アトランタ国際空港 (Hartsfield-Jackson Atlanta International Airport (ATL)) においてアメリカン航空 (American Airlines (AA)) に関連するレコードを検索するには、次のクエリ文字列を使用できます。

```

{ "$and" : [ { "airport_code" : { "$eq" : "ATL" } },
  { "carrier_code" : { "$eq" : "AA" } } ] }

```

クエリ文字列に JSON 形式を使用していることに注意してください。構文の確認に JSON の検証ツールが役立つかもしれません。

フィルター内の論理 AND および OR の処理

MicroKernel エンジンでは Extended オペレーションのフィルターに含まれる AND および OR 演算子は、厳密に左から右へ向かって解釈されます。次の規則に従って、フィルター内の式の評価を続けていきます。

-
- 現在のレコードに適用された式が真で、次の演算子が **OR** の場合、そのレコードはフィルター条件を満たすものと見なされます。
 - 式が真で、次の演算子が **AND** の場合、次のいずれかの状況になるまで各式の評価が継続されます。
 - 次の **OR** 式に達する。
 - 式の 1 つが偽と評価される。
 - フィルターの末尾に達する。
 - 式が偽で、次の演算子が **OR** の場合、フィルター内の次の式が引き続き評価されま
 - 式が偽で、次の演算子が **AND** の場合、そのレコードは拒否されます。

次のいずれかの条件を満たすと、レコードの検索は中止されます。

- フィルター条件を満たす、指定した数のレコードが検出される。
- フィルター条件を満たすレコードを検索している間に、検索したレコード数が指定したリジェクト カウントの最大数を超える。
- 現在のキーパスがフィルター条件のフィールドとして使用されており、それ以降ファイルの残り部分にフィルター条件を満たすレコードはないとする、拒否レコードに達する。
- ファイルの末尾に達する。

レコードのフィルタリングの例

フィルター条件を満たす次のレコード全体を取得するには、フィルター部分を設定し、次のようにディスクリプターフィールドを設定します。

1. レコード数を 1 に設定します。
2. フィールド数を 1 に設定します。
3. フィールド長を取得するレコード全体の長さに設定します。
4. フィールドのオフセットを 0 に設定します。

フィルター条件を使わずに次の 12 件のレコードを取得し、各レコードから 4 つのフィールドを抽出するには、論理式の項の数を 0 に設定し、次のようにディスクリプターフィールドを設定します。

1. レコード数を 12 に設定します。
2. フィールド数を 4 に設定します。
3. 抽出する 4 つのフィールドごとにフィールド長およびフィールドのオフセット パラメーターを設定します。

Extended オペレーションの出力バッファ

Extended Get または Step オペレーションを使ってレコードの 1 つ以上のフィールドまたは部分を取得するときは、データ バッファがオペレーションから返される情報を十分に格納できることを確認しておく必要があります。次の表は、戻り出力バッファの構造体を示しています。

要素	長さ (バイト単位)	説明
レコード数	2	Get および Step オペレーションの場合は、返されたレコード数。レコード削除の場合は、削除された数。
繰り返し部分 (取得されたレコードごとに 1 つあります)		
長さ 0	2	取得された先頭レコードのイメージ (すべてのフィールドを結合したもの) の長さ。Delete オペレーションの場合は、ゼロになります。
位置 0	4 または 8	取得された先頭レコードの物理カレンシー (アドレス)。レコード削除の場合は、削除された最初のレコードのアドレス。サイズはエントリ ポイントのタイプによって決まり、BTRV の場合は 4、BTRVEX の場合は 8 になります。
レコード 0	n	先頭レコードのイメージ (すべてのフィールドを結合したもの)。Delete オペレーションでは使用されません。
...		(各レコードの繰り返し部分)
長さ x	2	末尾レコードのイメージ (すべてのフィールドを結合したもの) の長さ (バイト単位)。Delete オペレーションの場合は、ゼロになります。
位置 x	4 または 8 ¹	取得または削除された末尾レコードの物理カレンシー (アドレス)。
レコード x	n	取得された末尾レコードのイメージ (すべてのフィールドを結合したもの)。Delete オペレーションでは使用されません。

返されたすべてのレコードまたはレコードのフィールドが固定長である場合、戻りデータバッファ内のデータの位置は簡単に計算できます。しかし、**Extended** オペレーションから返されたデータバッファからレコードの可変長部分を抽出するには、さらに特別な手順を踏む必要があります。

レコードの可変長部分が返されるとき、**MicroKernel** エンジンに戻りデータバッファ内に余分なレコードイメージを詰め込みません。したがって、レコードの可変長部分が占有する最大のバイト数を想定して戻りデータバッファ内の領域を確保しても、実際に返されたデータがその最大値を下回る場合、**MicroKernel** エンジンでは次に返されたフィールドのフィールド記述は現在のフィールドのデータの直後から始まることとなります。

たとえば、エン트리ポイントが4バイトのレコードアドレスを使用すると仮定して、固定レコード長が100バイトで、可変長部分は最大300バイトになると推定されるときに、5件のレコードの可変長部分だけを取得したいとします。入力バッファのディスクリプター要素を使って、フィールド長を300に設定し、フィールドのオフセットを100に設定します。戻りバッファについては、次の計算式で示すように、レコード数を示す2バイトと、レコード当たり306バイト（つまり、長さの2バイト、アドレスの4バイト、およびデータの300バイト）を加算する必要があります。

$$2 + ((2 \text{ バイト} + 4 \text{ バイト} + 300 \text{ バイト}) * 5) = 1532 \text{ バイト}$$

しかし、返された先頭レコードの可変長部分には50バイトのデータしかなかったとします。その結果、2バイトから成る2番目に返されるレコードの長さは、データバッファのオフセット58、つまり先頭レコードのフィールドイメージの直後に格納されることとなります。こうした状況でもアプリケーションでは、**MicroKernel** エンジンからデータバッファに返された長さ、位置、およびデータを正確に解析する必要があります。

結果

Get Next Extended オペレーションが正常に終了した場合は、**MicroKernel** エンジンから次の情報が返されます。

- 出力バッファには、取得された1つまたは複数のレコードに含まれる1つまたは複数のフィールドが格納されます。詳細については、[Extended オペレーションの出力バッファ](#)を参照してください。
- 出力バッファ長には、受け取った総バイト数が格納されます。
- キーバッファには、受け取った最後のデータレコードのキー値が格納されます。

Get Next Extended オペレーションが失敗した場合は、MicroKernel エンジンから次のステータスコードのいずれかが返されます。

- 3 ファイルが開いていません。
- 6 キー番号パラメーターが無効です。
- 7 キー番号が変更されました。
- 8 現在のポジションが無効です。
- 9 オペレーションが EOF (end-of-file) を検出しました。
- 22 データバッファパラメーターが短すぎます。
- 60 指定されたリジェクトカウントに達しました。
- 61 作業領域が小さすぎます。
- 62 ディスクリプターが不適切です。
- 64 フィルター制限に達しました。
- 65 フィールドオフセットが不適切です。
- 82 MicroKernel エンジンがポジショニングを失いました。
- 134 MicroKernel エンジンがインターナショナルソート規則 (ISR) を読み取れません。
- 135 指定されたインターナショナルソート規則 (ISR) テーブルは破損しているか、または無効です。
- 136 MicroKernel エンジンは、指定されたオルタネート コレーティングシーケンスをファイル内に見つけれられません。

MicroKernel エンジンは 0 以外のステータスコードに加え、有効なデータも返すことがあります。返された最後のレコードは不完全です。返されたバッファ長が 0 より大きい場合は、抽出されたデータのバッファを確認してください。

レコードが短すぎるためにフィールドを部分的にしか格納できない場合は、MicroKernel エンジンではその部分フィールドも含め、格納できるだけのレコード部分が返されます。部分フィールドが抽出される最後のフィールドである場合、エンジンではオペレーションが続行されます。そうでない場合、オペレーションは中止され、ステータスコード 22 が返されます。

たとえば、Get Next Extended オペレーションで、2 件の可変長レコードから 3 つのフィールドを取得するとします。最初のレコードは 55 バイトで、2 番目のレコードは 50 バイトの長さだとします。出力バッファには 50 バイトのデータを返すことができます。取得する 3 つのフィールドは次のように定義されています。

- フィールド 1 はオフセット 2 から始まり、2 バイト長です。

-
- フィールド 2 はオフセット 45 から始まり、10 バイト長です。
 - フィールド 3 はオフセット 6 から始まり、2 バイト長です。

MicroKernel エンジンで **Get Next Extended** オペレーションが実行される時、最初のレコードは問題なく返されます。しかし、次にフィールド 2 の 10 バイトを抽出しようとする時、オフセット 45 とレコードの末尾のオフセット 49 の間では 5 バイトしか取得できないことが MicroKernel エンジンによって検出されます。この時点で、MicroKernel エンジンはフィールド 2 の不足している 5 バイト分を詰め込まないため、フィールド 3 は抽出できなくなります。その代わりに、ステータスコード 22 が返され、フィールド 1 全体とフィールド 2 の先頭 5 バイトが戻りデータバッファに格納されます。

MicroKernel エンジンではフィルター条件で使用するフィールドと演算子によって、要求を最適化できる場合があります。拒否レコードに達すると、ステータスコード 64 が返され、ファイルの未検索の部分にはフィルター条件を満たすレコードがそれ以上ないことが示されます。

ポジショニング

Get Next Extended オペレーションを実行すると、完全な論理カレンシーおよび物理カレンシーが確立されます。検索された最後のレコードが現在のレコードになります。このレコードは、フィルター条件を満たして取得されたレコードか、またはフィルター条件を満たさないために拒否されたが、まだ最適化の範囲を超えていないレコードのいずれかです。たとえば、**Extended** オペレーションがステータス 9 (ファイルの終わり) を返した場合は、ファイルの末尾のレコードが現在のレコードになります。ステータス 60 (リジェクト カウントに達しました) が返された場合は、拒否された最後のレコードが現在のレコードです。ステータス 64 (フィルター制限に達しました) が返された場合には、最適化条件を満たす最後のレコードが現在のレコードになります。これ以降のレコードは最適化の範囲を超えていることを確認するために、MicroKernel エンジンが次のレコードを検索する必要があったとしても、現在のレコードの設定は条件を満たしていた以前のレコードに戻されます。

Get Position (22)

Get Position オペレーション (B_GET_POSITION) では、現在のレコードの物理位置を返します。Get Position オペレーションを発行するときに物理カレンシーが確立されていないと、オペレーションは正常に実行されません。レコードの位置 (アドレス) を決定できれば、Get Direct/Record (23) を使って、ファイル内の物理位置を基にそのレコードを直接取得できるようになります。MicroKernel エンジンでは、Get Position 要求を処理するためにディスク I/O は行われません。

パラメーター

	オペレーションコード	ポジションブロック	データバッファー	データバッファー長	キーバッファー	キー番号
送り値	X	X		X		X
戻り値		X	X	X		

前提条件

- 対象となるファイルが開いていることが必要です。
- アプリケーションでは、物理カレンシーを確立しておくことが必要です。

手順

- オペレーションコードを 22 に設定します。
- ファイルのポジションブロックを渡します。
- データバッファー長を少なくとも 4 バイトに設定します。BTRVEX エントリ ポイントを使用している場合は、少なくとも 8 バイトが必要です。
- キー番号を 0 に設定します。

結果

Get Position オペレーションが正常に終了した場合は、MicroKernel エンジンからレコードの位置がデータバッファーに返されます。位置はバイナリ値で、ファイル内における

るレコードのオフセットを示します。また、MicroKernel エンジンではデータバッファ長を、BTRV タイプのエントリポイントの場合は 4 バイト、BTRVEX タイプのエントリポイントの場合は 8 バイトに設定します。

Get Position オペレーションが失敗した場合は、MicroKernel エンジンから次のステータスコードのいずれかが返されます。

- 3 ファイルが開いていません。
- 8 現在のポジションが無効です。
- 137 このアクセスメソッドでのオペレーションは互換性がありません。API が一致しません。レコードアドレスを 4 バイトに格納できません。

ポジショニング

Get Position オペレーションは、ポジショニングにまったく影響しません。

Get Previous (7)

Get Previous オペレーション (B_GET_PREVIOUS) では、指定されたキーに基づいて、論理位置で前にあるレコードを取得します。Get Previous オペレーションを使うと、重複するキー値を持つレコードのグループの中でレコードを検索できます。Get Key (+50) バイアスを使うと、ファイル内に値が存在するかどうかを検出することもできます。一般に、Get Key オペレーションの方が高速に処理されます。

パラメーター

	オペレーション コード	ポジション ブロック	データ バッ ファー	データ バッ ファー長	キー バッ ファー	キー番号
送り値	X	X		X	X	X
戻り値		X	X	X	X	

前提条件

- 対象となるファイルが開いていることが必要です。
- ファイルがデータオンリーファイルであってははいけません。
- アプリケーションでは、指定したキーに基づく前の論理位置を確立しておくことが必要です。

手順

- オペレーション コードを 7 に設定します。任意でロック バイアスも指定できます。
 - +100 - 単一レコード ウェイト ロック
 - +200 - 単一レコード ノーウェイト ロック
 - +300 - 複数レコード ウェイト ロック
 - +400 - 複数レコード ノーウェイト ロック

レコード ロックおよびデータ整合性については、『Zen Programmer's Guide』のほかに、『Advanced Operations Guide』に記載されている Zen サーバーを設定するための **ウェイト ロック タイムアウト** プロパティを参照してください。

-
2. ファイルのポジションブロックを渡します。
 3. データバッファ長を、取得するレコードの長さ以上の値に設定します。
 4. キーバッファに、論理位置を確立した前のオペレーションで取得したキー値を指定します。

キーバッファには、前の呼び出しで **MicroKernel** エンジンから返されたキー値とまったく同じものを渡します。ここに格納された情報が、ファイル内の現在の位置を決定するために必要となるからです。

5. キー番号パラメーターを、論理位置を確立した前の呼び出しで使用したキーパスに設定します。 **Get Previous** オペレーションを使ってキーパスを変更することはできません。

結果

Get Previous オペレーションが正常に終了した場合、**MicroKernel** エンジンではキーバッファが新しいレコードのキー値を使って更新され、データバッファに前のレコードが返され、データバッファ長にそのレコードの長さが返されます。

Get Previous オペレーションが失敗した場合は、**MicroKernel** エンジンから次のステータスコードのいずれかが返されます。

- 3 ファイルが開いていません。
- 6 キー番号パラメーターが無効です。
- 7 キー番号が変更されました。
- 8 現在のポジションが無効です。
- 9 オペレーションが EOF (end-of-file) を検出しました。
- 22 データバッファパラメーターが短すぎます。
- 82 **MicroKernel** エンジンがポジショニングを失いました。

このオペレーションの実行により、論理位置の直前がファイルの先頭よりも前を指す場合は、ステータスコード 9 が返されます。

ポジショニング

Get Previous オペレーションを実行すると、完全な論理カレンシーおよび物理カレンシーが確立し、取得したレコードが現在のレコードになります。

Get Previous Delete Extended (86)

Get Previous Delete Extended オペレーション (B_GET_PREV_EXT_DELETE) では、指定されたキーに基づき、論理位置の直前からファイルの先頭へ向かって1つまたは複数のレコードを検索します。検索したレコードをフィルター条件と比較し、条件に一致するレコードを取得します。フィルター条件は論理式の形を取り、キーフィールドに限定されません。

このトピックで述べるように、このオペレーションが使用する入力および出力バッファ構造体、および返す結果は、[Get Next Extended \(36\)](#) に記載されているものと同じです。詳細については、当該オペレーションを参照してください。

パラメーター

	オペレーションコード	ポジションブロック	データバッファ	データバッファ長	キーバッファ	キー番号
送り値	X	X	X	X	X	X
戻り値		X	X	X	X	

前提条件

- 対象となるファイルが開いていることが必要です。
- ファイルがデータオンリーファイルであってははいけません。
- 指定したキーに基づく前の論理位置を確立しておくことが必要です。

手順

- オペレーションコードを 86 に設定します。

デフォルトでは、ロックバイアスはノーウェイトで、どのロックバイアス設定も無視されます。動作は +500 と同じです。エンジンは、ロックされたレコードを削除できない場合には、オペレーションを再試行しないで直ちに返ります。

- ファイルのポジションブロックを渡します。

-
3. 入力構造体と戻り出力のどちらか大きい方を格納できるように、データバッファを指定します。 [Get Next Extended \(36\)](#) にある Extended オペレーションの入力バッファの情報に従って、データバッファを初期化します。
 4. バッファサイズには、 [Get Next Extended \(36\)](#) にある入力および出力バッファ情報の説明に従って、入力構造体の長さで戻り出力の長さのどちらか大きい方を指定します。
 5. キーバッファに、論理位置を確立した前のオペレーションで取得したキー値を指定します。
キーバッファには、前の呼び出しで **MicroKernel** エンジンから返されたキー値とまったく同じものを渡します。ここに格納された情報が、ファイル内の現在の位置を決定するために必要となるからです。
 6. キー番号パラメーターを、論理位置を確立した前の呼び出しで使用したキーパスに設定します。 [Get Previous Delete Extended](#) オペレーションを使ってキーパスを変更することはできません。

詳細

[Get Next Extended \(36\)](#) の以下のトピックで、Extended オペレーションの入力バッファの構造体とそのフィルターセグメントの使用について、さらに結果を返す出力バッファの構造体について説明されています。

- [Extended オペレーションの入力バッファ](#)
- [Extended オペレーションの出力バッファ](#)

結果

このオペレーションでは、 [Get Next Delete Extended \(85\)](#) と同様の結果が返されます。詳細については、当該オペレーションを参照してください。

ポジショニング

[Get Previous Delete Extended](#) オペレーションでは、カレンシーは確立しません。ただし、[Get Next](#) オペレーションまたは [Get Previous](#) オペレーションを実行することができ、次または前の論理位置は有効です。また、[Get Position \(22\)](#) および [Get Direct/Record \(23\)](#) を使用することにより、有効な現在の位置も使用可能になります。

次のリストは、選定されたステータスコードとフィルター条件の関係を示しています。

- ステータス 60 (リジェクト カウントに達しました) : 現在の位置は、フィルター条件に一致しないレコードです。
- ステータス 64 (フィルター制限に達しました) : 現在の位置は、フィルター条件に一致しない可能性のあるレコードです。次または前のレコードに進もうとしても、フィルター条件に一致しません。
- ステータス 84 (レコードまたはページはロックされています) : 現在の位置は、フィルター条件に一致しない可能性のあるレコードです。また、次のレコードはフィルター条件に一致するが、ロックされているために削除できないという可能性もあります。
- ステータス 22 (データ バッファがいっぱいです) : 現在の位置は、フィルター条件に一致するレコードです。ただし、データ バッファにはレコード アドレスを書き込むためのスペースがないため、MicroKernel エンジンはそのレコードを削除しませんでした。
- ステータス 9 (ファイルの終わり) : 現在の位置は、論理的にも物理的にも無効です。

Get Previous Extended (37)

Get Previous Extended オペレーション (B_GET_PREV_EXTENDED) では、指定されたキーに基づき、論理位置の直前からファイルの先頭へ向かって1つまたは複数のレコードを検索します。検索したレコードをフィルター条件と比較し、条件に一致するレコードを取得します。フィルター条件は論理式の形を取り、キーフィールドに限定されません。

Get Previous Extended オペレーションでは、レコードから指定した部分だけを抽出し、その部分だけをアプリケーションに返すこともできます。

このトピックで述べるように、このオペレーションが使用する入力および出力バッファー構造体、および返す結果は、[Get Next Extended \(36\)](#) に記載されているものと同じです。詳細については、当該オペレーションを参照してください。

パラメーター

	オペレーション コード	ポジション ブロック	データ バッ ファー	データ バッ ファー長	キー バッ ファー	キー番号
送り値	X	X	X	X	X	X
戻り値		X	X	X	X	

前提条件

- 対象となるファイルが開いていることが必要です。
- ファイルがデータオンリーファイルであってははいけません。
- 指定したキーに基づく前の論理位置を確立しておくことが必要です。

手順

- オペレーションコードを37に設定します。任意でロックバイアスも指定できます。
 - +100 - 単一レコード ウェイト ロック
 - +200 - 単一レコード ノーウェイト ロック
 - +300 - 複数レコード ウェイト ロック

-
- +400 - 複数レコード ノーウェイト ロック

レコード ロックおよびデータ整合性については、『*Zen Programmer's Guide*』のほかにも、『*Advanced Operations Guide*』に記載されている Zen サーバーを設定するための **ウェイト ロック タイムアウト** プロパティを参照してください。

2. ファイルのポジションブロックを渡します。
3. 入力構造体と戻り出力のどちらか大きい方を格納できるように、データバッファを指定します。 **Get Next Extended (36)** にある **Extended** オペレーションの入力バッファの情報に従って、データバッファを初期化します。
4. バッファサイズには、 **Get Next Extended (36)** にある入力および出力バッファ情報の説明に従って、入力構造体の長さおよび戻り出力の長さのどちらか大きい方を指定します。
5. キーバッファに、論理位置を確立した前のオペレーションで取得したキー値を指定します。
キーバッファには、前の呼び出しで **MicroKernel** エンジンから返されたキー値とまったく同じものを渡します。ここに格納された情報が、ファイル内の現在の位置を決定するために必要となるからです。
6. キー番号パラメーターを、論理位置を確立した前の呼び出しで使用したキーパスに設定します。 **Get Previous Extended** オペレーションを使ってキーパスを変更することはできません。

詳細

このオペレーションでは、 **Get Next Extended (36)** の場合と同じ入力バッファおよび出力バッファを使用します。詳細については、当該オペレーションを参照してください。

結果

このオペレーションでは、 **Get Next Extended (36)** と同様の結果が返されます。詳細については、当該オペレーションを参照してください。

ポジショニング

Get Previous Extended オペレーションを実行すると、完全な論理カレンシーおよび物理カレンシーが確立されます。検索された最後のレコードが現在のレコードになります。このレコードは、フィルター条件を満たして取得されたレコードか、またはフィルター条件を満たさないために拒否されたレコードのいずれかです。

Insert (2)

Insert オペレーション (B_INSERT) では、ファイルにレコードを挿入します。MicroKernel エンジンでは新しいレコードのキー値を反映して、キーの B ツリーが調整されます。

パラメーター

	オペレーションコード	ポジションブロック	データバッファー	データバッファー長	キーバッファー	キー番号
送り値	X	X	X	X		X
戻り値		X			X	

メモ : NCC (No-currency-change : カレンシー変更なし) オプションを使用すると、Insert オペレーションはキーバッファーパラメーターの値を更新しません。つまり、このパラメーターには何の情報も返されません。

前提条件

- 対象となるファイルが開いていることが必要です。
- 挿入するレコードは適切なレコード長を持つ必要があります。また、キー値は対象となるファイルで定義されているキーと一致していなければなりません。

手順

- オペレーションコードを 2 に設定します。
- ファイルのポジションブロックを渡します。
- データバッファーに、挿入するレコードを格納します。
- データバッファー長を指定します。この値は、少なくともレコードの固定長部分と同じ長さでなければなりません。
- MicroKernel エンジンがポジショニング情報 (カレンシー) の確立に使用するキー番号を指定します。NCC オプションを使用するには、キー番号に -1 を指定します。

システム定義のログ キー（システム データとも呼ばれる）を使用するには、125 を指定します。システム データ v2 用の 2 番目のシステム キーを使用するには、124 を指定します。

結果

Insert オペレーションが正常に終了した場合、MicroKernel エンジンではファイルに新しいレコードが挿入され、新しいレコードを反映してキーの B ツリーが更新されます。また、指定したキーの値がキー バッファに返されます。MicroKernel エンジンでは、**autoincrement** キーの値がバイナリ 0 に初期化されているレコードを挿入すると、挿入したレコードもデータ バッファに返され、レコードには MicroKernel エンジンによって割り当てられた **autoincrement** 値が入っています。NCC Insert オペレーションでは、キー バッファ パラメーターの値は変更されません。

Insert オペレーションが失敗した場合は、MicroKernel エンジンから次のステータス コードのいずれかが返されます。

- 2 アプリケーションで I/O エラーが発生しました。
- 3 ファイルが開いていません。
- 5 レコードのキー フィールドに重複するキー値があります。
- 18 ディスクがいっぱいです。
- 21 キー バッファ パラメーターが短すぎます。
- 22 データ バッファ パラメーターが短すぎます。

ポジショニング

NCC オプションを指定しない Insert オペレーションを実行すると、完全な論理カレンシーおよび物理カレンシーが確立し、挿入したレコードが現在のレコードになります。論理カレンシーは指定したキーに基づきます。

NCC Insert オペレーションを実行すると、論理カレンシーは影響を受けずに、物理カレンシーが確立されます。つまり、NCC Insert オペレーションを実行したアプリケーションでは、ファイル内の論理位置は Insert オペレーションを実行する前と変わらないということです。このような状況で、NCC Insert オペレーションに続けて、**Get Next** (6)、**Get Next Extended** (36)、**Get Previous** (7)、および **Get Previous Extended** (37) などのオペレーションを実行すると、NCC Insert オペレーション実行以前のアプリケーションの論理カレンシーに基づく値が返されます。

メモ： MicroKernel エンジンでは、NCC Insert オペレーションを実行しても、その結果として何の情報もキーバッファには返されません。したがって、論理カレンシーの維持が必要なアプリケーションでは、NCC Insert オペレーション後にキーバッファの値を変更しないでください。変更すると、次の Get オペレーションの結果は予測できないものになります。

MicroKernel エンジンでは標準の Insert オペレーションと NCC Insert オペレーションのどちらを実行しても、新しく挿入されたレコードに対し物理カレンシーが確立されます。NCC Insert オペレーションに続く Step Next (24)、Step Next Extended (38)、Step Previous (35)、Step Previous Extended (39)、Update (3)、Delete (4)、および Get Position (22) などのオペレーションは、新しい物理カレンシーに基づいて機能します。

Insert Extended (40)

Insert Extended オペレーション (B_EXT_INSERT) では、ファイルに1つまたは複数のレコードを挿入します。MicroKernel エンジンでは、新しいレコードのキー値を反映して、キーのBツリーが調整されます。

パラメーター

	オペレーションコード	ポジションブロック	データバッファー	データバッファー長	キーバッファー	キー番号
送り値	X	X	X	X		X
戻り値		X	X		X	

メモ：NCC (No-currency-change : カレンシー変更なし) オプションを使用すると、Insert Extended オペレーションはキーバッファーパラメーターの値を更新しません。つまり、このパラメーターには何の情報も返されません。

前提条件

- 対象となるファイルが開いていることが必要です。
- 挿入するレコードは適切なレコード長を持つ必要があります。また、キー値は対象となるファイルで定義されているキーと一致していなければなりません。

手順

- オペレーションコードを40に設定します。
- ファイルのポジションブロックを渡します。
- [詳細](#)に示す構造体に従って、データバッファーを指定します。
- データバッファー長を指定します。この値は、データバッファー構造体のサイズと一致している必要があります。
- MicroKernel エンジンがカレンシーの確立に使用するキー番号を指定します。NCC オプションを使用するには、キー番号に-1を指定します。システム定義のログ

キー（システム データとも呼ばれる）を使用するには、125 を指定します。システム データ v2 用の 2 番目のシステム キーを使用するには、124 を指定します。

詳細

次の表は、Insert Extended オペレーションの入力データ バッファの構造体を示しています。

要素	長さ (バイト 単位)	説明
固定部分	2 または 4 ¹	挿入するレコード数。
繰り返し部分 (レコードごとに 1 つあります)		
	2 または 4 ¹	レコード イメージの長さ。
	<i>n</i>	レコード イメージ。

¹ サイズは、使用するエントリ ポイントが BTRV タイプか BTRVEX タイプかによって決まります。

結果

Insert Extended オペレーションが正常に終了した場合、MicroKernel エンジンではファイルに新しいレコードが挿入され、オペレーションが NCC Insert Extended でない場合は、挿入された新しいレコードを反映してすべての B ツリーが更新されます。さらに、最後に挿入したレコードから、指定したキーの値がキー バッファに返されます。また、戻りデータ バッファの先頭 2 バイトまたは 4 バイトの符号なし整数には、ファイルに正常に挿入されたレコードの数が MicroKernel エンジンによって格納されます。レ

コード数の後には、挿入されたレコードのアドレスが **MicroKernel** エンジンによって格納されます。次の表は、出力データバッファの構造体を示しています。

要素	長さ (バイト単位)	説明
固定部分	2 または 4 ¹	挿入するレコード数。
繰り返し部分 (レコードごとに 1 つあります)		
	4 または 8 ¹	レコード アドレス。

¹ サイズは、使用するエントリ ポイントが **BTRV** タイプか **BTRVEX** タイプかによって決まります。

オペレーションの一部しか正常に実行されず、**MicroKernel** エンジンから 0 以外のステータス コードが返された場合、データバッファの先頭 2 バイトまたは 4 バイトの符号なし整数の値は、正常に挿入されたレコードの数と等しくなります。エラーの原因となったレコードは、正常に挿入されたレコード数 + 1 番目のレコードです。

Insert Extended オペレーションが正常に実行されなかった場合は、**MicroKernel** エンジンから次のステータス コードのいずれかが返されます。

- 2 アプリケーションで I/O エラーが発生しました。
- 3 ファイルが開いていません。
- 5 レコードのキーフィールドに重複するキー値があります。
- 18 ディスクがいっぱいです。
- 21 キーバッファパラメーターが短すぎます。
- 22 データバッファパラメーターが短すぎます。

ポジショニング

NCC オプションを指定しない **Insert Extended** オペレーションを実行すると、完全な論理カレンシーおよび物理カレンシーが確立し、挿入されたレコードのキー値がヌルでなければ、最後に挿入されたレコードが現在のレコードになります。論理カレンシーは指定したキーに基づきます。

NCC **Insert Extended** オペレーションを実行すると、論理カレンシーは影響を受けずに、物理カレンシーが確立されます。つまり、NCC **Insert Extended** オペレーションを実行したアプリケーションでは、ファイル内の論理位置はオペレーションを実行する前と変わ

らないということです。このような状況で、NCC Insert Extended オペレーションに続けて、Get Next (6)、Get Next Extended (36)、Get Previous (7)、および Get Previous Extended (37) などのオペレーションを実行すると、NCC Insert Extended オペレーション実行以前のアプリケーションの論理カレンシーに基づく値が返されます。

メモ： MicroKernel エンジンでは、NCC Insert Extended オペレーションを実行しても、その結果として何の情報もキーバッファには返されません。したがって、論理カレンシーの維持が必要なアプリケーションでは、NCC Insert Extended オペレーション後にキーバッファの値を変更しないでください。変更すると、次の Get オペレーションの結果は予測できないものになります。

MicroKernel エンジンでは標準の Insert Extended オペレーションと NCC Insert Extended オペレーションのどちらを実行しても、新しく挿入されたレコードに対し物理カレンシーが確立されます。したがって、NCC Insert Extended オペレーションに続く Step Next (24)、Step Next Extended (38)、Step Previous (35)、Step Previous Extended (39)、Update (3)、Delete (4)、および Get Position (22) などのオペレーションは、新しい物理カレンシーに基づいて機能します。

Get Next (6) オペレーションのような、元の論理カレンシーに基づくオペレーションを実行するために、アプリケーションで Insert Extended オペレーション実行前のファイルの論理位置を保存しておく必要がある場合に、NCC Insert Extended オペレーションが役立ちます。

NCC Insert Extended オペレーションを実行しないで同様の結果を得るには、アプリケーションで以下の手順を実行する必要があります。

1. Get Position (22) - 現在の論理レコードの物理アドレスを取得します。アプリケーションでこの値を保存し、手順 3 でこれを渡して元に戻します。
2. Insert Extended (40) - 新しいレコードを挿入します。このオペレーションにより、新しい論理カレンシーおよび物理カレンシーが確立されます。
3. Get Direct/Record (23) - 論理カレンシーと物理カレンシーを手順 1 での状態に戻します。

NCC Insert Extended オペレーションは、論理カレンシーについてはこの手順と同様の結果を得られますが、物理カレンシーについては異なります。たとえば、これら 2 つの手順のいずれかに続けて Get Next (6) を実行した場合は、どちらの手順でも結果は変わりませんが、Step Next (24) を実行した場合は、異なるレコードが返される可能性があります。

Login/Logout (78)

Login/Logout オペレーション (B_LOGIN/B_LOGOUT) では、ユーザーは資格情報を指定して、データベース エンジンから認証トークンおよび許可トークンを取得することができます。また、データベースへのアクセスを取得するために資格情報の再入力が必要となるよう、ユーザーは資格情報をリセットすることもできます。

パラメーター

	オペレーション コード	ポジション ブロック	データ バッ ファー	データバッ ファー長	キーバッ ファー	キー番号
送り値	X				X	X
戻り値						

前提条件

- データベース名およびユーザー ID はあらかじめ定義されている必要があります。

ログイン手順

- オペレーション コードを 78 に設定します。
- キー番号を 0 に設定します。
- データベース URI の形式を用いて、キーバッファーにサーバー名、データベース名、ユーザー ID、およびパスワードを設定します。URI 接続文字列の詳細については、『*Zen Programmer's Guide*』の[データベース URI](#)を参照してください。

ログアウト手順

- オペレーション コードを 78 に設定します。
- キー番号を 1 に設定します。
- データベース URI の形式を用いて、キーバッファーにサーバー名、データベース名、ユーザー ID、およびパスワードを設定します。『*Zen Programmer's Guide*』の[データベース URI](#)を参照してください。

結果

Login または Logout オペレーションが正常に終了した場合は、ステータス **0** が返されます。正常に実行されなかった場合は、次のステータス コードのいずれかが返されます。

- 1 無効な操作です。
- 172 データベース名が見つかりません。
- 3103 不明なサーバーです。

注記

データベース URI を結合した長さは 255 バイト未満でなければなりません。これがキーバッファの最大サイズだからです。

Login オペレーションではパフォーマンスに負荷がかかります。アプリケーション、ファイルごとにログインおよびログアウトするようなコーディングをしないでください。代わりに、セッションの始めに一度データベースにログインし、データベース作業が完了したらログアウトするようにしてください。

ポジショニング

Login/Logout オペレーションは、ファイルのカレンシー情報にはまったく影響しません。

Open (0)

Open オペレーション (B_OPEN) は、ファイルへのアクセスを可能にします。アプリケーションでファイルにアクセスするには、まず Open オペレーションを実行する必要があります。絶対パス名または相対パス名を指定する限り、対象となるファイルが現在のディレクトリに保存されている必要はありません。

パラメーター

	オペレーション コード	ポジション ブロック	データ バッ ファー	データバッ ファー長	キーバッ ファー	キー番号
送り値	X		X	X	X	X
戻り値		X				

前提条件

- 開くファイルは、アクセス可能な論理ディスクドライブ上にあることが必要です。
- そのファイルで使用可能なファイルハンドルが存在することが必要です。

手順

1. オペレーション コードを 0 に設定します。
2. ファイルにオーナーが設定されている場合は、データ バッファー パラメーターにオーナー ネームを指定し、末尾にバイナリ 0 を付けます。
3. データ バッファー長パラメーターに、バイナリ 0 を含めたオーナー ネームの長さを指定します。

キー バッファー パラメーターに、開くファイルのパス名を入れます。埋め込みスペースの設定に応じて、パス名の終端はヌル (バイナリ ゼロ) にします。パス名は半角 255 文字までの範囲で指定できます。ヌル終端文字を含む完全修飾 UNC (Unified Naming Convention) パス名は、半角 255 文字までの範囲で指定できます。

MicroKernel エンジン通常、ファイル名を完全修飾 UNC ファイル名に拡張します。たとえば、Z:¥Data¥File.dat は ¥¥ サーバー名 ¥ 共有名 ¥Data¥File.dat に変換されます。

この拡張された名前が、ヌル終端文字を含めて半角 255 文字までにならない限りません。『*Zen Programmer's Guide*』の[データベース URI](#)も参照してください。

ただし、Btrieve の Open 要求がローカル エンジンに送られる場合は、MIF はローカルのドライブ文字をコンピューター名および共有名に置き換えません。もっと長いパス名のファイルをローカルで正常に開くことができたとしても、リモート クライアントではそのファイルを開けないことがあります。

クライアント構成の [スペースを含むファイル/ディレクトリ名] 設定によって、埋め込みスペースを含むファイル名がサポートされます。デフォルトでは、この設定はオンになっています。つまり、スペースはパスの一部と見なされます。この設定がオンの場合、ファイル名はヌルバイトで区切る必要があります。この設定がオフの場合、埋め込みスペースを含むファイル名 (C:¥My Folder¥my file.mkd など) を使用することはできません。『*Advanced Operations Guide*』の[長いファイル名と埋め込みスペースのサポート](#)を参照してください。

Zen クライアントでサポートするパス名の詳細については、『*Getting Started with Zen*』の[Zen リクエスターでサポートするネットワーク パスの形式](#)を参照してください。

4. キー番号パラメーターに、このトピックの「詳細」で示す表に記載されているモード値のいずれかを指定します。

詳細

ここでは、サポートされているオープン モードについて説明します。

注意！ データベース エンジンは、クライアントがアクセラレイティド モードを使用している間は、クライアントのトランザクション アトミシティ、トランザクション一貫性保持、およびアーカイブ ログの安全性を保証できません。この制約があるのは、ログからの復元が必要な場合に、完全な復元を行うために十分な情報がログに含まれていない可能性があるからです。なぜなら、ログは、1つのデータ ファイル上で行った操作の部分的な記録でしかないからです。

たとえば、アクセラレイティド モードを使用して挿入を実行するクライアントと、ノーマル モードを使用して更新を実行するクライアントが同じファイルにアクセスしているときにシステム障害が発生した場合、トランザクション ログには、データ ファイルにまだ存在しないレコードに対する更新が含まれている可能性があります。これは、メモリ内のアクセラレイティドの挿入操作は一度もディスクにフラッシュされていませんが、トランザクショナルな更新操作はトランザクション ログに書き込まれてい

るためです。

この操作の組み合わせを含むアーカイブ ログをロール フォワードしようとする、失敗します。

ファイルを開くときに、次の表に示すオープン モードによって、ローカル エンジンとリモート エンジンのどちらを使用するかを **MicroKernel** エンジンに指示できます。キー番号パラメーターにオープン モードの値を指定します。

メモ： ローカル エンジンでファイルを開くように指定する場合は、ワークグループ エンジンでもサーバー エンジンでも、**Open** オペレーションに何ら違いはありません。

説明	選択しない	ローカル エンジンで実行	リモート エンジンで実行
ノーマル	0	6	99
アクセラレイテッド 特定ファイルのパフォーマンスを向上させるために、アクセラレイテッド モードでファイルを開くことができます (6.x MicroKernel エンジンでもアクセラレイテッド モードで開くことはできましたが、ノーマル モードで開いたものと解釈されました)。アクセラレイテッド モードでファイルを開くと、 MicroKernel エンジンは、そのファイルに対するトランザクション ログを実行しません。上記の「注意」を参照してください。	-1	7	100

説明	選択しない	ローカル エンジンで実行	リモート エンジンで実行
リードオンリー リードオンリー モードでファイルを開くと、ファイルを読み取ることしかできません。更新は実行できません。このモードを使うと、MicroKernel エンジンでは自動的に回復できない破損データを含むファイルを開くことができます。ファイルのインデックスのデータが壊れている場合は、リードオンリー モードでファイルを開き、Step Next (24) を使用することによって、レコードを取得できます。	-2	8	101
書き込み可能 MicroKernel エンジンが書き込み用にファイルを開けない場合、書き込み可能モードはエラーを返します。一般的なシナリオの 1 つは、ファイルシステムの書き込みアクセス許可のないファイルです。	-3	9	102
エクスクルーシブ エクスクルーシブ モードはファイルに対する排他的なアクセスを可能にします。あるアプリケーションが排他的にアクセスしているファイルは、閉じるまで、ほかのアプリケーションから開くことができなくなります。	-4	10	103

開いているファイルの最大数について定められた制限はありません。同時に開くことができるファイルの数は、使用可能なメモリに応じて変わります。

ファイルは、MicroKernel エンジンによって 1 回だけ開かれます。エンジンは、複数のクライアントが同時に 1 つのファイルを開いている状況や、単独のクライアントがファイルのポジションブロックを複数持っている状況を認識し、処理します。拡張ファイルを開くとき、エンジンでは 1 つのハンドルが使用され、ベース ファイルとすべてのエクステンション ファイルが開かれます。

結果

Open オペレーションが正常に終了した場合、MicroKernel エンジンでは目的のファイルにファイルハンドルが割り当てられ、新しく開いたファイルの Open 呼び出しで渡したポジションブロックが予約されて、そのファイルがアクセス可能な状態になります。

Open オペレーションが失敗した場合は、MicroKernel エンジンから次のステータスコードのいずれかが返されます。

- 2 アプリケーションで I/O エラーが発生しました。
- 11 指定されたファイル名が無効です。
- 12 MicroKernel エンジンは指定されたファイルを見つけられません。
- 20 MicroKernel エンジンまたは Btrieve リクエスターが非アクティブです。
- 46 要求したファイルへのアクセスは拒否されました。
- 84 レコードまたはページはロックされています。
- 85 ファイルはロックされています。
- 86 ファイルテーブルがいっぱいです。
- 87 ハンドルテーブルがいっぱいです。
- 88 アプリケーションでモードの不一致エラーが発生しました。

次の表は、ローカルクライアントで使用できるオープンモードの組み合わせを示しています。

ローカルクライアント 1 のオープンモード	ローカルクライアント 2 のオープンモード	結果
ノーマル / 書き込み可能	ノーマル	正常終了
	書き込み可能	正常終了
	リードオンリー	正常終了
	エクスクルーシブ	ステータスコード 88
	アクセラレイティド	正常終了

ローカルクライアント 1 のオープンモード	ローカルクライアント 2 のオープンモード	結果
リードオンリー	ノーマル	正常終了
	書き込み可能	正常終了
	リードオンリー	正常終了
	エクスクルーシブ	ステータス コード 88
	アクセラレイティド	正常終了
エクスクルーシブ	ノーマル	ステータス コード 88
	書き込み可能	ステータス コード 88
	リードオンリー	ステータス コード 88
	エクスクルーシブ	ステータス コード 88
	アクセラレイティド	ステータス コード 88
アクセラレイティド	ノーマル	正常終了
	書き込み可能	正常終了
	リードオンリー	正常終了
	エクスクルーシブ	ステータス コード 88
	アクセラレイティド	正常終了

ポジショニング

Open オペレーションを実行しても、ポジショニングは確立しません。ただし、次の物理レコードがファイルの先頭の物理レコードになります。

Reset (28)

Reset オペレーション (B_RESET) では、クライアントが保持しているすべてのリソースを解放します。このオペレーションは、クライアントが実行中のトランザクションを中止し、すべてのロックを解除し、さらに、クライアントが開いているファイルをすべて閉じます。

パラメーター

	オペレーションコード	ポジションブロック	データバッファー	データバッファー長	キーバッファー	キー番号
送り値	X				X	X
戻り値						

前提条件

MicroKernel エンジンまたはリクエスターがロードされていて、Reset 呼び出しを発行するクライアントが MicroKernel エンジンとの接続を確立していれば、アプリケーションではいつでも Reset オペレーションを発行できます。接続は、たとえば、ファイルを開いたり、Zen ツールを使ってファイルのステータスを要求したりすることにより確立します。

手順

1. オペレーションコードを 28 に設定します。
2. キー番号およびキーバッファーパラメーターを 0 に設定します。

結果

Reset オペレーションが正常に終了した場合、MicroKernel エンジンでは指定したクライアント、ウィンドウ、またはセッションに対して次の動作が実行されます。

1. 実行中のトランザクションがすべて中止される。
2. 保持されているすべてのロックが解除される。

3. 開いているファイルがすべて閉じられる。

Reset オペレーションが失敗した場合は、MicroKernel エンジンから 0 以外のステータスコードが返されます。

ポジショニング

Reset オペレーションを実行すると、開いているファイルがすべて閉じられるため、すべてのカーンシーが消去されます。

Set Directory (17)

Set Directory オペレーション (B_SET_DIR) では、指定したパス名を現在のディレクトリとして設定します。

パラメーター

	オペレーション コード	ポジション ブロック	データ バッ ファー	データ バッ ファー長	キー バッ ファー	キー番号
送り値	X				X	
戻り値						

前提条件

対象となる論理ディスクドライブおよびディレクトリがアクセス可能であることが必要です。

手順

1. オペレーション コードを 17 に設定します。
2. キーバッファに、論理ディスクドライブおよびディレクトリパスを、最後にバイナリ 0 を付けて格納します。ドライブ名を省略した場合、MicroKernel エンジンではデフォルトのドライブが使用されます。ディレクトリの絶対パスを指定しなかった場合、MicroKernel エンジンではキーバッファに指定したディレクトリパスが現在のディレクトリに追加されます。

Zen クライアントでサポートするパス名の詳細については、『*Getting Started with Zen*』の [Zen リクエスターでサポートするネットワークパスの形式](#) を参照してください。

結果

Set Directory オペレーションが正常に終了した場合、MicroKernel エンジンではキーバッファに指定したディレクトリが現在のディレクトリになります。

Set Directory オペレーションが失敗した場合、MicroKernel エンジンでは現在のディレクトリは変更されないままで、0 以外のステータスコードが返されます。

ポジショニング

Set Directory オペレーションは、ポジショニングにまったく影響しません。

Set Owner (29)

Set Owner オペレーション (B_SET_OWNER) ではファイルにオーナー ネームを割り当てます。オーナー ネームはアクセス パスワードとして機能します。ファイルにオーナー ネームが設定されている場合は、ユーザーやアプリケーションはそのファイルにアクセスするたびにオーナー ネームの文字列を提供する必要があります。すべてのアクセス権に対し、あるいは更新権限だけに対してオーナー ネームが要求されるように指定することができます。オーナー ネームは ASCII 形式です。長いオーナー ネームの場合には、16 進数にすることもできます。オーナー ネームを割り当てるときに、ディスク上のファイルのデータを暗号化するよう MicroKernel エンジンに指示することもできます。そう指示した場合、MicroKernel エンジンでは Set Owner オペレーションの実行中にすべてのデータが暗号化されます。ファイルが長いと Set Owner の実行時間が長くなるため、パフォーマンスに影響を与える可能性があります。詳細については、『*Advanced Operations Guide*』の[オーナー ネーム](#)を参照してください。

パラメーター

	オペレーション コード	ポジション ブロック	データ バッ ファー	データ バッ ファー長	キー バッ ファー	キー番号
送り値	X	X	X	X	X	X
戻り値		X				

前提条件

- 対象となるファイルが開いていることが必要です。
- トランザクションが実行中でないことが必要です。
- ファイルに既にオーナー ネームが設定されてはいけません。

手順

1. オペレーション コードを 29 に設定します。

任意で、+17000 のバイアスを含めると、最長 24 バイト（半角 24 文字）の長いオーナーネームを作成することができます。このバイアスは `btrconst.h` に `B_LONG_OWNER_NAME_BIAS` としても定義されています。

2. 保護するファイルを識別するポジションブロックを渡します。
3. データバッファとキーバッファの両方にオーナーネームを格納します。
MicroKernel エンジンには、誤って不適切な値を提供するのを防ぐために、オーナーネームを両方のバッファに格納することを要求します。

+17000 のバイアスが設定されていない場合は、短いオーナーネームを 8 バイト（半角 8 文字）までの範囲で指定でき、末尾はバイナリ 0 になっている必要があります。
+17000 のバイアスが設定されている場合は、長いオーナーネームを使用でき、末尾はバイナリ 0 になっている必要があります。どちらの場合も、オーナーネームをすべてスペース（0x20）で構成することはできません。長いオーナーネームの長さは、ファイル形式によって異なります。詳細については、[オーナーネーム](#)を参照してください。
4. データバッファ長パラメーターに、バイナリ 0 を含めたオーナーネームの長さを設定します。
5. キー番号を、ファイルに対するアクセス制限と暗号化のタイプを指定する整数に設定します。次の表は、4 つのキー番号とその結果の一覧を示します。

キー番号	説明
0	すべてのアクセスモードでオーナーネームが必要になります（データは暗号化されません）。
1	読み取り専用アクセスはオーナーネームがなくても許可されます（データは暗号化されません）。
2	すべてのアクセスモードでオーナーネームが必要になります（データが暗号化されます）。
3	読み取り専用アクセスはオーナーネームがなくても許可されます（データが暗号化されます）。

詳細

一度オーナーネームを指定すると、[Clear Owner \(30\)](#) オペレーションを発行するまでそのオーナーネームは有効です。上の表は、キー番号に設定できるアクセス制限コードの一覧を示しています。

結果

Set Owner オペレーションが正常に終了した場合、それ以降のオペレーションでは正しいオーナーネームが指定されない限り、**MicroKernel** エンジンでファイルへのアクセスやファイルの変更を行えなくなります。唯一の例外は、オーナーネームの指定なしで読み取り専用アクセスが許可されている場合です。

さらに、**Set Owner** オペレーションが正常に終了すると、暗号化が指定されている場合には、**MicroKernel** エンジンによってファイル内のデータが暗号化されます。暗号化は直ちに行われ、ファイル全体が暗号化されるまでは **MicroKernel** エンジンの制御下にあります。

パフォーマンスに関して、次のことに留意してください。**MicroKernel** エンジンはディスクから暗号化されたページを読み込む際にそのページを解読し、ディスクに書き込む前に再度ページを暗号化します。暗号化されたファイルからのデータの読み取りは、暗号化されていないファイルからデータを読み取る場合よりも遅くなります。また、ファイルのサイズが大きくなるほど、暗号化または解読には時間がかかります。暗号化されたファイルのシナリオでは、キャッシュが小さい場合、または比較的大量の変更操作を行う場合には、**MicroKernel** エンジンは暗号化ルーチンをさらに頻繁に実行しなければなりません。

Set Owner オペレーションが失敗した場合は、**MicroKernel** エンジンから次のステータスコードのいずれかが返されます。

- 41 実行しようとした操作は **MicroKernel** エンジン では許可されていません。
- 50 ファイルのオーナーは既に設定されています。
- 51 オーナーネームが無効です。

ポジショニング

Set Owner オペレーションは、ポジショニングにまったく影響しません。

Stat (15)

Stat オペレーション (B_STAT) では、データ バッファを使用して、ファイル仕様に関する統計を取得します。統計情報には、ファイルに含まれるレコードの数、ファイルの各インデックスに格納されている重複しないキー値の数、空きページの数、オルタネート コーディング シーケンス (ACS) などがあります。ファイルを作成した以降に新しいキーや ACS 値が追加されている場合があります。この新しい情報を構成する必要があるため、[Create \(14\)](#) オペレーションで使用された元のデータ バッファ サイズを再利用できない可能性があります。

パラメーター

	オペレーション コード	ポジション ブロック	データ バッ ファ	データ バッ ファ長	キー バッ ファ	キー番号
送り値	X	X	X	X	X	X
戻り値			X	X	X	

前提条件

対象となるファイルが開いている必要があります。

手順

1. オペレーション コードを 15 に設定します。
2. ファイルのポジションブロックを渡します。
3. ファイルに定義されている統計情報を格納するのに十分なデータ バッファを確保します。
4. データ バッファ長を指定します。ファイルの統計情報を十分に格納できる長さが必要です。
5. 少なくとも 255 バイトのキー バッファを指定します。
6. 次のようにキー番号を設定します。
 - ファイルバージョンを除外する場合は 0

-
- ファイルバージョンを含める場合は -1

詳細

Create (14) と Stat (14) は同様のデータ バッファ構造体を使用するため、構造体については、それらのわずかな違いも含め、[Create \(14\)](#) でまとめて説明されています。

ファイル仕様

戻りデータ バッファのファイル仕様フィールドは、[Create \(14\)](#) で説明したものと同じですが、ファイル仕様領域内に次の例外があります。

- データ バッファにファイルのバージョン情報が含まれている場合は、インデックス数が 1 バイト長になり、その後 1 バイトのファイルのバージョン情報が続きます。ファイルのバージョン番号の値を 10 進数に変換しないでください。0x95 という値はそのファイルが 9.5 ファイルであることを示し、0x80 という値はそのファイルが 8.x ファイルであることを示します (以下同様)。MicroKernel エンジンではファイルを作成するとき、これらの属性に従ってバージョン番号が割り当てられます。
- レコード数は、ファイル内のレコードの数を表す 4 バイトまたは 8 バイト長の値です。
- ファイルフラグワードで、ビット 9 (0x0200) と 12 (0x1000) は次のような意味を持ちます。

ビット 9=1 かつ ビット 12= 0	ファイルはシステム データまたはシステム データ v2 を使って作成されました。これは、必ずしもシステム キーが現在使用されていることを意味しません。削除された可能性もあります。 Stat Extended (65) を参照してください。
-------------------------------	---

ビット 9=1 かつ ビット 12= 1	ファイルはシステム データを使わずに作成されました。
-------------------------------	----------------------------

Stat オペレーションでは、システム データがデフォルトで組み込まれたのか、明示的に組み込まれたのかは示されません。

- 未使用の重複ポインター数は、ファイル内に残っている未使用の重複ポインターの数を示します。

-
- 予約領域が割り当てられますが、MicroKernel エンジンでは Stat オペレーションではこの領域を無視します。

キー仕様

戻りデータバッファのキー仕様フィールドは、[キー仕様ブロック](#)で説明したものと基本的には同じです。ただし、4バイトまたは8バイトの重複のないキー値の数は、指定されたキーに対して一意で重複のない値を持つレコードの数が示される点が異なります。

オルタネート コレーティング シーケンス

戻りデータバッファの ACS (オルタネート コレーティング シーケンス) は、[Create \(14\)](#) で説明したものとまったく同じです。

結果

Stat オペレーションが正常に終了した場合、MicroKernel エンジンではファイルおよびキーの特性がデータバッファに返され、データバッファの長さがデータバッファ長に返されます。対象となるファイルが拡張ファイルである場合、MicroKernel エンジンでは先頭のエクステンションファイルのファイル名がキーバッファに返されます。先頭のエクステンションファイルのファイル名が63バイトを超える場合、MicroKernel エンジンではファイル名が切り詰められます。ファイルが拡張ファイルでない場合は、MicroKernel エンジンではキーバッファの先頭バイトが0に初期化されます。[Stat Extended \(65\)](#) オペレーションを使うと、拡張ファイルに関する情報も取得できます。

Stat オペレーションが失敗した場合は、MicroKernel エンジンから次のステータスコードのいずれかが返されます。

- 3 ファイルが開いていません。
- 22 データバッファパラメーターが短すぎます。

ポジショニング

Stat オペレーションは、ポジショニングにまったく影響しません。

Stat Extended (65)

Stat Extended オペレーション (B_EXTENDED_STAT) にはいくつかのサブファンクションがあります。これを使って、アプリケーションは開いているファイルについての情報を収集することができます。

サブファンクション ID	説明
1	エクステンション ファイル名の一覧表示
2	ファイルのシステム データ情報
3	重複による競合レコードおよびキーの識別
4	ファイル情報
5	ゲートウェイの識別
6	ロック オーナーの識別
7	セキュリティ情報
8	ステータス コード 71 (参照整合性の定義に違反があります) の発生原因となる、テーブル名またはファイル名の一覧表示

詳細については、下記のサブファンクションのトピックを参照してください。

パラメーター

	オペレーション コード	ポジション ブロック	データ バッ ファー	データ バッ ファー長	キー バッ ファー	キー番号
送り値	X	X	X	X		X
戻り値		X	X	X		

前提条件

対象となるファイルが開いていることが必要です。

手順

1. オペレーション コードを 65 に設定します。
2. ファイルのポジションブロックを渡します。
3. データバッファに Stat Extended 構造体を格納します。各サブファンクションで必要な Stat Extended 構造体の詳細については、以降のセクションを参照してください。
4. データバッファ長を指定します。
5. キー番号を 0 に設定します。

サブファンクション 1 : 拡張ファイル情報

入力ポジションブロックで指定されたファイルの場合、このサブファンクションは、指定データファイルと関連付けられたエクステンションファイルについての情報を返します。返される情報には、存在するエクステンションファイルの数、関数によって返された番号、および返されたファイルの名前が含まれます。

入力データ バッファ構造体

エクステンションファイルに関する情報を取得するには、次の表に示すようにデータバッファに拡張ファイル ディスクリプターを作成する必要があります。

要素	長さ (バイト単位)	説明
識別バイト	4	Stat Extended 呼び出しのタイプ。Stat Extended 呼び出しを示す 0x45、0x78、0x53、0x74 の 4 バイトを指定します。これらは ASCII の <i>ExSt</i> 、または Intel 型、LoHi およびリトルエンディアン方式のハードウェアの値 0x74537845 に相当します。
サブファンクション	4	Stat Extended 呼び出しのタイプ。0x00000001 を指定します。
名前空間	4	ファイルの命名規則。0x00000000 を指定します。
ファイルの最大数	4	返されるファイル名の最大数。この値には、拡張ファイルを構成するエクステンションファイルの数を超える値を設定できません (拡張ファイルには最大 32 個のエクステンションファイルが含まれます)。

要素	長さ (バイト単位)	説明
先頭ファイルのシーケンス	4	返される先頭のファイル名のシーケンス番号。ベースファイルから始めるには0を指定し、先頭のエクステンションファイルから始めるには1を指定します (以下同様)。エクステンションファイルの数を超える値を指定すると、MicroKernel エンジンからステータスコード 0 が返されますが、ファイル名は何も返されません。
バッファースペース	n	戻りデータ用に余分な空き領域を確保することができます。ステータスコード 22 が返された場合は、もっと大きなデータバッファサイズを指定してオペレーションを再試行してください。

出力データ バッファ構造体

拡張ファイルサブファンクションの場合、MicroKernel エンジンではデータバッファ長パラメーターの値が更新され、次の表で説明されているような拡張ファイル構造体がデータバッファに返されます。

要素	長さ (バイト単位)	説明
ファイル数	4	拡張ファイルを構成するオペレーティングシステムファイルの数。
エクステンションファイル数	4	返されたエクステンションファイルの数。
ファイル名部分 (返された各ファイル名について繰り返される)		
ファイル名の長さ	4	エクステンションファイル名の長さ。
ファイル名	n	エクステンションファイル名。

サブファンクション 2 : システム データ情報

入力ポジションブロックで指定されたファイルの場合、このサブファンクションは、ファイルにシステムキーが定義されているかどうか、また、そのファイルはログ可能 (トランザクション一貫性保持が可能) であるかどうかに関する情報を返します。

入力データ バッファ構造体

ファイルでのシステムデータの取得に関する情報を取得するには、次の表に示すようにデータバッファにシステムデータディスクリプターを作成する必要があります。

要素	長さ (バイト単位)	説明
識別バイト	4	Stat Extended 呼び出しのタイプ。Stat Extended 呼び出しを示す 0x45、0x78、0x53、0x74 の 4 バイトを指定します。これらは ASCII の <i>ExSt</i> 、または Intel 型、LoHi およびリトル エンディアン方式のハードウェアの値 0x74537845 に相当します。
サブファンクション	4	Stat Extended 呼び出しのタイプ。0x00000002 を指定します。

出力データ バッファ構造体

システム データ サブファンクションの場合、MicroKernel エンジンでは次のようなシステム データ構造体がデータ バッファに返されます。

要素	長さ (バイト単位)	説明
システム データを持つ	1	ファイルにシステム データが定義されているかどうかを示します。0=なし、1=システム データを持つ、2=システム データ v2 を持つ。
システム キーを持つ	1	システム キーが存在するかどうかを示します。0=なし、1=システム キー 125、2=システム キー 124、3=両方。
ログ可能	1	トランザクション ログおよびトランザクション一貫性保持に使用できる重複のないキーがファイルにあるかどうかを示します。このキーは、ユーザー定義のキーまたはシステム定義のキーになります。1=Yes、0=No。
ログ キー番号	1	トランザクション ログ キーとして使用されているキー番号。システム定義キーが使用されているキーである場合、この値は 125 になります。
システム データのサイズ	2	8=キー 125 のシステム データのみ。 16=キー 125 およびキー 124 のシステム データ v2。
エンジンのメジャーバージョン	2	データベース エンジンのメジャー バージョンを含んでいる 2 バイト フィールド。

サブファンクション 3 : 重複レコードによる競合情報

入力ポジションブロックで指定されたファイルの場合、このサブファンクションは、重複レコードによる競合についての情報を返します。返される情報には、直前の失敗した挿入または更新操作でステータス コード 5 (重複キー) の発生原因となった、レコード アドレスおよびキー番号が含まれます。

入力データ バッファ構造体

重複レコードによる競合を報告するために、一番最近ステータス コード 5 (重複キー) を発生させたレコード アドレスおよびキー番号に関する情報を取得するには、データ

バッファに重複レコード情報ディスクリプターを次のとおりに作成する必要があります。

要素	長さ (バイト単位)	説明
識別バイト	4	Stat Extended 呼び出しのタイプ。Stat Extended 呼び出しを示す 0x45、0x78、0x53、0x74 の 4 バイトを指定します。これらは ASCII の <i>ExSt</i> 、または Intel 型、LoHi およびリトルエンディアン方式のハードウェアの値 0x74537845 に相当します。
サブファンクション	4	Stat Extended 呼び出しのタイプ。0x00000003 を指定します。

出力データ バッファ構造体

重複レコードによる競合サブファンクションの場合、MicroKernel エンジンでは次のような重複レコードによる競合構造体がデータ バッファに返されます。

要素	長さ (バイト単位)	説明
重複レコード アドレス	4 または 8 ¹	重複するキー値を含んでいるレコードの物理アドレス。13.0 および 16.0 ファイル形式には 8 バイトが必要です。
キー番号	2	重複する値を含んでいるキーのキー番号。

¹ サイズは、使用するエントリ ポイントが BTRV タイプか BTRVEX タイプかによって決まります。

サブファンクション 4 : ファイル情報

入力ポジションブロックで指定されたファイルの場合、このサブファンクションはファイル情報を返します。返される情報には次のものが含まれます。MicroKernel エンジンがファイルの識別に使用する内部ファイル ID、現在開いているファイルハンドル数、ファイルが前回開かれたときのタイムスタンプ、およびファイルプロパティを示すさまざまなフラグがあります。

入力データ バッファ構造体

開いているファイルに関する情報を取得するには、データ バッファにファイル情報 ディスクリプターを次のとおりに作成する必要があります。

要素	長さ (バイト単位)	説明
識別バイト	4	Stat Extended 呼び出しのタイプ。Stat Extended 呼び出しを示す 0x45、0x78、0x53、0x74 の 4 バイトを指定します。これらは ASCII の <i>ExSt</i> 、または Intel 型、LoHi およびリトル エンディアン方式のハードウェアの値 0x74537845 に相当します。
サブファンクション	4	Stat Extended 呼び出しのタイプ。0x00000004 を指定します。
バッファースペース	12	戻り情報に必要な追加バイト。 出力データ バッファ構造体 を参照してください。ファイル情報サブファンクションの場合、MicroKernel エンジンではデータ バッファにファイル情報構造体が返されます。

出力データ バッファ構造体

ファイル情報サブファンクションの場合、MicroKernel エンジンでは次のようなファイル情報構造体がデータ バッファに返されます。

要素	長さ (バイト単位)	説明
ファイル ID	4	MicroKernel エンジンがファイルを識別するために使用する重複のない番号。
ハンドル数	4	MicroKernel エンジンがこのファイルで開いている現在のハンドル数。
開かれたときのタイムスタンプ	4	MicroKernel エンジンが物理ファイルを前回開いたときのシステム時間。システム時間は、世界協定時刻 (UTC) を基に 1970 年 1 月 1 日の午前零時からの秒数で表されます。

要素	長さ (バイト単位)	説明
ファイルの使用回数	4	各チェックポイントまたはシステムトランザクションごとにこの数が増加します。また使用回数は FCR にも置かれます。ここで返される数は、MicroKernel エンジンのキャッシュ内で表されるファイルの使用回数です。チェックポイントを開始すると、この数が増加します。
フラグ	4	さまざまな値を設定できる 4 バイトのビットマップ。設定可能な値の説明については、次の表を参照してください。将来的にはさらに多くのフラグが追加されます。
最大ファイル サイズ (KB)	4	ファイルサイズの上限。ファイルバージョンによって異なります。
最大パーツ サイズ (KB)	4	
1 ファイルあたりの最大パーツ	4	
最大物理ページ数	4	ファイル内の最大物理ページ数
最大論理ページ数	4	ファイル内の最大論理ページ数
物理ページ数	4	ファイル内の実際の物理ページ数
論理ページ数	4	ファイル内の実際の論理ページ数
1 ページあたりのレコード数	4	単一ページ内のレコード数

フラグ フィールドに使用できる値については、次の表で説明します。

値	名前	説明
0x00000001	明示的ロック	現在ファイル上で明示的ロックがあります。
0x00000002	クライアント トランザクション	現在ファイル上で、クライアント トランザクションが少なくとも 1 つ開いています。
0x00000004	読み取り専用	ファイルは MicroKernel エンジンによって読み取り専用で開かれています。これは CD-ROM ドライブまたは読み取り専用ディレクトリです。

値	名前	説明
0x00000008	Continuous オペレーション	ファイルは現在 Continuous オペレーション モードです。
0x00000010	参照整合性	ファイルには参照整合性制約が設定されています。
0x00000020	オーナー読み取り / 書き込み	ファイルには読み取り / 書き込みのオーナー ネームが割り当てられています。ファイルの読み取り / 書き込みにはオーナー ネームが要求されます。
0x00000040	オーナー読み取り OK	書き込みを行う場合にのみ要求されるオーナー ネームを持ちます。読み取りは、オーナー ネームなしで行うことができます。
0x00000080	間違ったオーナーでのオープン	このファイルには読み取り OK のオーナー ネームが割り当てられています。間違ったオーナー ネームでハンドルが開けられました。
0x00000100	オーナー暗号化	ファイルにはオーナー ネームの暗号化フラグが設定されています。このフラグは、ファイル内のすべてのページは暗号化されており、テキスト エディターでは読み取れないことを意味します。
0x00000200	キャッシュ エンジンによるオープン	設定された場合、このファイルはキャッシュ エンジンによって開かれています。
0x00000400	従来の暗号化	ファイルは従来の Btrieve 暗号化アルゴリズムを使用して暗号化されています。
0x00000800	128 バイト暗号化	ファイルは 128 バイト暗号化アルゴリズムを使用して暗号化されています。
0x00001000	AES-192 暗号化	ファイルは AES-192 暗号化を使用して暗号化されています。
0x00002000	AES-256 暗号化	ファイルは AES-256 暗号化を使用して暗号化されています。
0x00010000	並行トランザクション	進行中の並行トランザクションが少なくとも 1 つあります。このフラグは、0x00000002 - クライアント トランザクションと組み合わせる必要があります。

サブファンクション 5 : ゲートウェイ情報

入力ポジションブロックで指定されたファイルの場合、このサブファンクションは、ファイルの制御を行うゲートウェイエンジンについての情報を返します。

入力データ バッファ構造体

指定されたファイルの処理を行うゲートウェイエンジンに関する情報を取得するには、データバッファにゲートウェイ情報ディスクリプターを次のとおりに作成する必要があります。

要素	長さ (バイト 単位)	説明
識別バイト	2	Stat Extended 呼び出しのタイプ。Stat Extended 呼び出しを示す 0x45、0x78、0x53、0x74 の 4 バイトを指定します。これらは ASCII の <i>ExSt</i> 、または Intel 型、LoHi およびリトルエンディアン方式のハードウェアの値 0x74537845 に相当します。
サブファンクション	2	Stat Extended 呼び出しのタイプ。0x00000005 を指定します。
バッファースペース	最低でも 80	戻り情報に必要な追加バイト。詳細については、 出力データ バッファ構造体 を参照してください。

出力データ バッファ構造体

ゲートウェイ情報サブファンクションの場合、MicroKernel エンジンではデータバッファ長パラメーターが更新され、次のようなゲートウェイ情報構造体がデータバッファに返されます。

要素	長さ (バイト単位)	説明
メジャーバージョン	2	エンジンのメジャーバージョン。たとえば、バージョン 7 または 8 などです。
マイナーバージョン	2	エンジンのマイナーバージョン。たとえば、05 または 82 などです。

要素	長さ (バイト単位)	説明
パッチ レベル	2	エンジンのパッチ レベル。たとえば、1、2、または3になります。
プラットフォーム	2	Version (26) に挙げられているリクエスターまたはエンジンの種類。
サーバー名	64	データベース エンジンが動作しているマシンの名前を示すヌル終端文字列。Btrieve API 呼び出しによって返されるデータ バッファ長には、サーバー名とヌル終端文字を含めた戻りデータの実際の長さが格納されます。

サブファンクション 6 : ロック オーナーの識別

入力ポジションブロックで指定されたファイルの場合、このサブファンクションは、一番最近ファイルのアクセス時にステータス コード 84 または 85 を発生させた原因に関する情報を返します。

入力データ バッファ構造体

ステータス 84 または 85 の原因に関する情報を取得するには、データ バッファにロック オーナー情報ディスクリプターを次のとおりに作成する必要があります。

要素	長さ (バイト単位)	説明
識別バイト	4	Stat Extended 呼び出しのための一意な識別子。ExSt を指定します。 サブファンクション 1 : 拡張ファイル情報 を参照してください。
サブファンクション	4	Stat Extended 呼び出しのタイプ。0x00000006 を指定します。
バッファースペース	最低でも 96	戻り情報に必要な追加バイト。詳細については、 サブファンクション 7 : セキュリティ情報 を参照してください。

出力データ バッファ構造体

ロック オーナー情報サブファンクションの場合、MicroKernel エンジンではデータ バッファ長パラメーターが更新され、次のようなロック オーナー情報構造体がデータ バッファに返されます。

要素	長さ (バイト単位)	説明
クライアント ID	16	ブロックしているクライアントの 16 バイトのクライアント ID。
フラグ	4	発生した競合のタイプを示すフラグを含む 4 バイトのビットマップ。各フラグの値の説明については、次の表を参照してください。
トランザクション内の時間	4	ブロックしたクライアントがトランザクション中にロックするミリ秒数。これは、オペレーションを再試行するかどうかを決定するのに役立ちます。
キー番号	4	競合がキー ページ上で発生した場合、この要素はその競合に関与しているキーを示します。この情報を確認すれば、データベースの設計で予測される競合を減らすのに役立ちます。
トランザクションレベル	4	この数がゼロ以外の場合、ブロックしているクライアントは現在トランザクション中です。いくつかのページやレコードはトランザクションが完了するまでロックされるので、この情報はオペレーションを再試行するかどうかを決定するのに役立ちます。
予約済み	8	今後の使用に備えて予約されています。ブロックしているクライアントに関して有用だと思われる情報がある場合は、テクニカル サポートまでご連絡ください。
表示名	64	これはヌルで終了する文字列です。これは Monitor 内で各クライアントごとに表示される識別名と同じです。最低でも現在の表示名の長さの最大値である 64 バイトを使用してください。Btrieve API 呼び出しによって返されるデータ バッファ長には、表示名とヌル終端文字を含めた戻りデータの実際の長さが格納されます。

以前にブロックしていたクライアントの記録が MicroKernel エンジンにない場合、出力データ バッファ長はゼロに設定されます。

フラグ フィールドに使用できる値については、次の表で説明します。

値	名前	説明
0x00000001	暗黙ロック	ブロックしたクライアントは暗黙ロックを使用しています。
0x00000002	明示的ロック	ブロックしたクライアントは明示的ロックを使用しています。
0x00000010	ファイルロック	ブロックしたクライアントはファイル ロックを使用しています。
0x00000020	ページ ロック	ブロックしたクライアントはページ ロックを使用しています。
0x00000040	レコードロック	ブロックしたクライアントはレコード ロックを使用しています。
0x00000100	データ ページ	競合がページ ロックの場合、このフラグは競合がデータ ページで発生したことを示します。
0x00000200	キー ページ	競合がページ ロックの場合、このフラグは競合がキー ページで発生したことを示します。
0x00000400	可変ページ	競合がページ ロックの場合、このフラグは競合が可変ページで発生したことを示します。
0x00000800	同じプロセス	このフラグが設定された場合、ブロックしているクライアント ID の最初の 12 バイトと、ブロックされたクライアント (Stat Extended 呼び出しを発行したクライアント) の最初の 12 バイトが同じです。この場合、同じシステム上の同じプロセスからブロックした 2 つのクライアントがあることを意味します。 Btrieve API 呼び出しを行うシングル スレッドのアプリケーションの場合は、このオペレーションを再試行しても役立ちません。ブロックしている動作を終了または中止する必要があります。
0x00001000	書き込み ノー ウェイト	ブロックしたクライアントがバイアス 500 を使用していることを示します。
0x00002000	書き込み保持	ブロックしたクライアントが、ページへの変更を行って、そのトランザクションが終了するまでページ全体をロックしたことを示します。この状況は、変更によってキー エントリが別のページへ移動した場合に、キー ページの暗黙ロックで発生します。

値	名前	説明
0x00004000	読み取り ノーウェイ ト	明示的レコードロックの場合、このフラグは、ブロックしたクライアントがロックバイアス 200 または 400 のいずれかを使用していることを示します。
0x00008000	複数読み取り	明示的レコードロックの場合、このフラグは、ブロックしたクライアントがロックバイアス 300 または 400 のいずれかを使用していることを示します。

サブファンクション 7: セキュリティ情報

このサブファンクションは、クライアントが現在のファイルにアクセスするために、どのように認証および許可されたかについての情報を返します。セキュリティに使用されている現在のデータベースに関する情報も示します。

入力データ バッファ構造体

このハンドルがどのように認証されたか、またどのアクセス権を持っているかについてセキュリティ情報を取得するには、データバッファにセキュリティ情報ディスクリプターを次のとおりに作成する必要があります。

要素	長さ (バイト単位)	説明
識別バイト	4	Stat Extended 呼び出しのタイプ。Stat Extended 呼び出しを示す 0x45、0x78、0x53、0x74 の 4 バイトを指定します。これらは ASCII の <i>ExSt</i> 、または Intel 型、LoHi およびリトルエンディアン方式のハードウェアの値 0x74537845 に相当します。
サブファンクション	4	Stat Extended 呼び出しのタイプ。0x00000007 を指定します。
バッファースペース	最低でも 142	戻り情報に必要な追加バイト。詳細については「結果」を参照してください。

出力データ バッファ構造体

セキュリティ情報サブファンクションの場合、MicroKernel ではデータ バッファ長パラメーターが更新され、次のようなセキュリティ情報構造体がデータ バッファに返されます。

要素	長さ (バイト単位)	説明
ハンドルのフラグ	4	このハンドルのセキュリティに使用されているメソッドを示すフラグを含む4バイトのビットマップ。各フラグの値の説明については、次の表を参照してください。
データベースのフラグ	4	このクライアントの、現在のデフォルト データベースのセキュリティに使用されているメソッドを示すフラグを含む4バイトのビットマップ。各フラグの値の説明については、次の表を参照してください。
アクセス権	4	これらは、このハンドルを使用するクライアントに与えられているアクセス権です。各フラグの値の説明については、アクセス権の表を参照してください。
ハンドルデータベース名のバッファサイズ	2	ヌルで終わるハンドルデータベース名文字列の格納に使用されるバッファの長さ。
ハンドルテーブル名のバッファサイズ	2	ヌルで終わるハンドルテーブル名文字列の格納に使用されるバッファの長さ。 メモ ：テーブル名は、ファイルがデータベースにバインドされない限りわかりません（たとえば参照制約）。つまり、ファイルは、ファイルのテーブル名で参照される URI 接続文字列を使用して開かれています。URI 接続文字列の詳細については、『 <i>Zen Programmer's Guide</i> 』の データベース URI を参照してください。
ハンドルユーザー名のバッファサイズ	2	ヌルで終わるハンドルユーザー名文字列の格納に使用されるバッファの長さ。
現在のデータベース名のバッファサイズ	2	ヌルで終わる現在のデータベース名文字列の格納に使用されるバッファの長さ。

要素	長さ (バイト単位)	説明
現在のユーザー名のバッファ サイズ	2	ヌルで終わる現在のユーザー名文字列の格納に使用されるバッファの長さ。
ハンドル データベース名	可変	このハンドルのセキュリティを確立するために使用されるデータベース名。
ハンドル テーブル名	可変	このハンドルと関連付けられているテーブル名。
ハンドル ユーザー名	可変	このハンドルのセキュリティを確立するために使用されるユーザー名。
現在のデータベース名	可変	このクライアントの、現在のデフォルトのデータベース名。
現在のユーザー名	可変	このクライアントの、現在のデフォルト データベースに関連付けられているユーザー名。

2つのフラグ フィールドで使用できる値については、以下の表で説明します。

値	名前	セキュリティ フラグの説明
0x00000001	信頼される	このハンドルは信頼されています。割り当てられたデータベースはありません。
0x00000002	暗黙	データベース ログインは暗黙です - 開いている間。
0x00000004	明示的	データベース ログインは明示的です - Btrieve Login が実行されました。
0x00000008	データベースによる認証	認証はデータベース セキュリティによって行われました。このフラグが設定されていない場合は、認証はオペレーティング システムのセキュリティによって行われました。
0x00000010	データベースによる許可	許可はデータベース セキュリティによって行われました。このフラグが設定されていない場合は、許可はオペレーティング システムのセキュリティによって行われました。
0x00000020	Windows 名前付きパイプ	認証がオペレーティング システムによるものである場合、これはセキュリティに NT 名前付きパイプが使用されたことを示します。

値	名前	セキュリティフラグの説明
0x00000040	ワークグループ	認証がオペレーティングシステムによるものである場合、これはワークグループエンジン式のセキュリティが行われたことを示します。つまり、認証も許可も行われていないということです。
0x00000080	Btpasswd	認証が Linux または Raspbian オペレーティングシステムによるものである場合、これは <code>etc/btpasswd</code> ファイルが使用されたことを示します。
0x00000100	PAM	認証が Linux または Raspbian オペレーティングシステムによるものである場合、これは PAM 認証が使用されたことを示します。
0x00000200	RTSS Complete	認証がオペレーティングシステムによるものである場合、これは、"Complete" に設定された RTSS を使用して認証が行われたことを示します。
0x00000400	RTSS Preauthorized	認証がオペレーティングシステムによるものである場合、これは、"Preauthorized" に設定された RTSS を使用して認証が行われたことを示します。
0x00000800	RTSS Disabled	認証がオペレーティングシステムによるものである場合、これは、"Disabled" に設定された RTSS を使用して認証が行われたことを示します。

値	名前	アクセス権フラグの説明
0x00000000	No Rights	データベースオブジェクトに対する権限がありません。アクセス権は与えられていません。
0x00000001	Open	ファイルを開くために与えられるアクセス権。これは、レコードの読み取りが可能であることも意味します。
0x00000002	Insert	レコードを挿入するために与えられるアクセス権。
0x00000004	Update	レコードを更新するために与えられるアクセス権。
0x00000008	Create	このファイルを作成するために与えられるアクセス権。
0x00000010	Delete	レコードを削除するために与えられるアクセス権。

値	名前	アクセス権フラグの説明
0x00000020	Execute	SQL でストアド プロシージャを実行するために与えられるアクセス権。
0x00000040	Alter	SQL でこのファイルを変更するために与えられるアクセス権。
0x00000080	Refer	SQL でこのファイルを参照するために与えられるアクセス権。
0x00000100	Create View	SQL でこのファイルのビューを作成するために与えられるアクセス権。
0x00000200	Create Stored Procedure	SQL でこのファイルのストアド プロシージャを作成するために与えられるアクセス権。

サブファンクション 8 : ステータス コード 71 の発生原因となる、テーブル名またはファイル名の一覧表示

このサブファンクションは、ステータス コード 71（参照整合性の定義に違反がありません）の発生原因となった、テーブルまたはデータ ファイルについての情報を返します。返される情報には、ファイル名、Btrieve オペレーション コード、および参照整合性エラーが発生したレコードの位置が含まれます。

入力データ バッファ構造体

開いているファイルに関する情報を取得するには、データ バッファにファイル情報 ディスクリプターを次のとおりに作成する必要があります。

要素	長さ (バイト単位)	説明
識別バイト	4	Stat Extended 呼び出しのタイプ。Stat Extended 呼び出しを示す 0x45、0x78、0x53、0x74 の 4 バイトを指定します。これらは ASCII の <i>ExSt</i> 、または Intel 型、LoHi およびリトル エンディアン方式のハードウェアの値 0x74537845 に相当します。
サブファンクション	4	Stat Extended 呼び出しのタイプ。0x00000008 を指定します。

出力データ バッファ構造体

ファイル情報サブファンクションの場合、MicroKernel エンジンでは次のようなファイル情報構造体がデータ バッファに返されます。指定されたデータ バッファは、返されるデータを保持できる十分な大きさでなければなりません。

要素	長さ (バイト単位)	説明
ファイル名	255	RI エラーの原因となったファイル名。
Btrieve オペレーションコード	4	RI エラーの原因となった Btrieve オペレーションコード。
レコード位置	4 または 8 ¹	RI エラーを発生させたレコードの物理レコード位置。13.0 および 16.0 ファイル形式には 8 バイトが必要です。

¹ サイズは、使用するエン트리 ポイントが BTRV タイプか BTRVEX タイプかによって決まります。

結果

Stat Extended オペレーションが失敗した場合は、MicroKernel エンジンから次のステータスコードのいずれかが返されます。

- 3 ファイルが開いていません。
- 06 キー番号パラメーターが無効です。
- 22 データ バッファ パラメーターが短すぎます。
- 62 ディスクリプターが不適切です。

Step First (33)

Step First オペレーション (B_STEP_FIRST) では、ファイル内の先頭の物理レコードを取得します。MicroKernel エンジンではこのレコードを取得するためにキーパスは使用されません。

パラメーター

	オペレーションコード	ポジションブロック	データバッファー	データバッファー長	キーバッファー	キー番号
送り値	X	X		X		
戻り値		X	X	X		

前提条件

対象となるファイルが開いていることが必要です。

手順

1. オペレーションコードを 33 に設定します。任意でロック バイアスも指定できます。
 - +100 - 単一レコード ウェイト ロック
 - +200 - 単一レコード ノーウェイト ロック
 - +300 - 複数レコード ウェイト ロック
 - +400 - 複数レコード ノーウェイト ロック

レコード ロックおよびデータ整合性については、『*Zen Programmer's Guide*』のほかに、『*Advanced Operations Guide*』に記載されている Zen サーバーを設定するための **ウェイト ロック タイムアウト** プロパティを参照してください。
2. ファイルのポジションブロックを渡します。
3. データ バッファー長を、取得するレコードの長さ以上の値に設定します。

結果

Step First オペレーションが正常に終了した場合、MicroKernel エンジンではファイル内の先頭の物理レコードがデータ バッファーに返され、返されたバイト数がデータ バッファー長パラメーターに設定されます。

Step First オペレーションが失敗した場合は、MicroKernel エンジンから次のステータスコードのいずれかが返されます。

- 3 ファイルが開いていません。
- 9 オペレーションが EOF (end-of-file) を検出しました。
- 22 データ バッファー パラメーターが短すぎます。

ポジショニング

Step First オペレーションを実行すると、論理カレンシーが消去されます。取得したレコードを現在の物理レコードとして使用し、物理カレンシーが設定されます。物理位置の直前は、ファイルの先頭よりも前を指すこととなります。

Step Last (34)

Step Last オペレーション (B_STEP_LAST) では、ファイル内の末尾の物理レコードを取得します。MicroKernel エンジンではこのレコードを取得するためにキーパスは使用されません。

パラメーター

	オペレーション コード	ポジション ブロック	データ バッ ファー	データ バッ ファー長	キー バッ ファー	キー番号
送り値	X	X		X		
戻り値		X	X	X		

前提条件

対象となるファイルが開いていることが必要です。

手順

1. オペレーションコードを 34 に設定します。任意でロック バイアスも指定できます。
 - +100 - 単一レコード ウェイト ロック
 - +200 - 単一レコード ノーウェイト ロック
 - +300 - 複数レコード ウェイト ロック
 - +400 - 複数レコード ノーウェイト ロックレコード ロックおよびデータ整合性については、『*Zen Programmer's Guide*』のほかに、『*Advanced Operations Guide*』に記載されている Zen サーバーを設定するための **ウェイト ロック タイムアウト** プロパティを参照してください。
2. ファイルのポジションブロックを渡します。
3. データ バッファー長を、取得するレコードの長さ以上の値に設定します。

結果

Step Last オペレーションが正常に終了した場合、MicroKernel エンジンではファイル内の末尾の物理レコードがデータ バッファーに返され、返されたバイト数がデータ バッファー長パラメーターに設定されます。

Step Last オペレーションが失敗した場合は、MicroKernel エンジンから次のステータスコードのいずれかが返されます。

- 3 ファイルが開いていません。
- 9 オペレーションが EOF (end-of-file) を検出しました。(ファイルが空の場合)
- 22 データ バッファー パラメーターが短すぎます。

ポジショニング

Step Last オペレーションを実行すると、論理カレンシーが消去されます。取得したレコードを現在の物理レコードとして使用し、物理カレンシーが設定されます。物理位置の直後は、ファイルの末尾よりも後を指すことになります。

Step Next (24)

Step Next オペレーション (B_STEP_NEXT) では、次の物理位置として示されるレコードを取得します。MicroKernel エンジンではこのレコードを取得するためにキーパスは使用されません。

Step Next オペレーションを任意の Get または Step オペレーションの直後に実行すると、前のオペレーションで取得されたレコードの物理的に次にあるレコードが返されます。

パラメーター

	オペレーションコード	ポジションブロック	データバッファ	データバッファ長	キーバッファ	キー番号
送り値	X	X		X		
戻り値		X	X	X		

前提条件

対象となるファイルが開いていることが必要です。

手順

1. オペレーションコードを 24 に設定します。任意でロック バイアスも指定できます。

- +100 - 単一レコード ウェイト ロック
- +200 - 単一レコード ノーウェイト ロック
- +300 - 複数レコード ウェイト ロック
- +400 - 複数レコード ノーウェイト ロック

レコード ロックおよびデータ整合性については、『*Zen Programmer's Guide*』のほかに、『*Advanced Operations Guide*』に記載されている Zen サーバーを設定するための **ウェイト ロック タイムアウト** プロパティを参照してください。

2. ファイルのポジションブロックを渡します。

3. データバッファ長を、取得するレコードの長さ以上の値に設定します。

結果

Step Next オペレーションが正常に終了した場合、MicroKernel エンジンではファイル内の次の物理レコードがデータ バッファに返され、返されたバイト数がデータ バッファ長パラメーターに設定されます。

Step Next オペレーションが失敗した場合は、MicroKernel エンジンから次のステータスコードのいずれかが返されます。

- 3 ファイルが開いていません。
- 9 オペレーションが EOF (end-of-file) を検出しました。
- 22 データ バッファ パラメーターが短すぎます。

ポジショニング

Step Next オペレーションを実行しても、論理カレンシーは確立しません。取得したレコードを現在の物理レコードとして使用し、物理カレンシーが設定されます。

Delete (4) の直後に Step Next オペレーションを発行すると、Delete の前のオペレーションで次の物理レコードとして確立されたレコードが返されます。

Open (0) の直後に Step Next オペレーションを発行すると、ファイル内の先頭レコードが返されます。

Step Next Extended (38)

Step Next Extended オペレーション (B_STEP_NEXT_EXT) では、物理位置の直後からファイルの末尾へ向かって1つまたは複数のレコードを検索します。検索したレコードをフィルター条件と比較し、条件に一致するレコードを取得します。フィルター条件は論理式の形を取り、キーフィールドに限定されません。

Step Next Extended オペレーションでは、既存のレコードの中から指定したフィールドを抽出し、抽出したフィールドだけを含む新しいレコードのセットを返すこともできます。

このトピックで述べるように、このオペレーションが使用する入力および出力バッファー構造体、および返す結果は、[Get Next Extended \(36\)](#) に記載されているものと同じです。詳細については、当該オペレーションを参照してください。

パラメーター

	オペレーションコード	ポジションブロック	データバッファー	データバッファー長	キーバッファー	キー番号
送り値	X	X	X	X		
戻り値		X	X	X		

前提条件

- 対象となるファイルが開いていることが必要です。
- 次の物理位置を確立しておくことが必要です。たとえば、Delete オペレーションの次に Step Next Extended オペレーションを実行することはできません。

手順

- オペレーションコードを 38 に設定します。任意でロック バイアスも指定できます。
 - +100 - 単一レコード ウェイト ロック
 - +200 - 単一レコード ノーウェイト ロック
 - +300 - 複数レコード ウェイト ロック

-
- +400 - 複数レコード ノーウェイト ロック

レコード ロックおよびデータ整合性については、『*Zen Programmer's Guide*』のほかにも、『*Advanced Operations Guide*』に記載されている Zen サーバーを設定するための **ウェイト ロック タイムアウト** プロパティを参照してください。

2. ファイルのポジションブロックを渡します。
3. 入力構造体と戻り出力のどちらか大きい方を格納できるように、データ バッファを指定します。 **Get Next Extended (36)** にある **Extended** オペレーションの入力バッファの情報に従って、データ バッファを初期化します。
4. バッファ サイズには、 **Get Next Extended (36)** にある入力および出力バッファ情報の説明に従って、入力構造体の長さ と戻り出力の長さのどちらか大きい方を指定します。
5. 上記の手順に従って、データ バッファ長に適切な値を指定します。

詳細

Step Next Extended オペレーションの「詳細」の内容は、Get Next Extended オペレーションと同じです。詳しくは、当該オペレーションの **詳細** を参照してください。

結果

Step Next Extended オペレーションが正常に終了した場合、MicroKernel エンジンでは、取得された 1 つまたは複数のレコードに含まれる 1 つまたは複数のフィールドがデータ バッファに返されます (**Extended オペレーションの出力バッファ** を参照してください)。さらに MicroKernel エンジンから、データ バッファ長パラメーターには、データ バッファに返されたバイト数が設定されます。

Step Next Extended オペレーションが失敗した場合は、MicroKernel エンジンから次のステータス コードのいずれかが返されます。

- 3 ファイルが開いていません。
- 9 オペレーションが EOF (end-of-file) を検出しました。
- 22 データ バッファ パラメーターが短すぎます。
- 60 指定されたリジェクト カウントに達しました。
- 61 作業領域が小さすぎます。

-
- 62 ディスクリプターが不適切です。
 - 65 フィールド オフセットが不適切です。
 - 82 MicroKernel エンジンがポジショニングを失いました。
 - 134 MicroKernel エンジンがインターナショナル ソート規則 (ISR) を読み取れません。
 - 135 指定されたインターナショナル ソート規則 (ISR) テーブルは破損しているか、または無効です。
 - 136 MicroKernel エンジンは、指定されたオルタネート コーレーティング シーケンスをファイル内に見つけれません。

MicroKernel エンジンは 0 以外のステータス コードに加え、有効なデータも返すことがあります。返された最後のレコードは不完全です。返されたバッファー長が 0 より大きい場合は、抽出されたデータの出力バッファーを確認してください。

レコードが短すぎるためにフィールドを部分的にしか格納できない場合は、MicroKernel エンジンではその部分フィールドも含め、格納できるだけのレコード部分が返されます。部分フィールドが抽出される最後のフィールドである場合、エンジンではオペレーションが継続されます。そうでない場合、オペレーションは中止され、ステータス コード 22 が返されます。

たとえば、Step Next Extended オペレーションで、2 件の可変長レコードから 3 つのフィールドを取得するとします。最初のレコードは 55 バイトで、2 番目のレコードは 50 バイトの長さだとします。出力バッファーには 50 バイトのデータを返すことができます。取得する 3 つのフィールドは次のように定義されています。

- フィールド 1 はオフセット 2 から始まり、2 バイト長です。
- フィールド 2 はオフセット 45 から始まり、10 バイト長です。
- フィールド 3 はオフセット 6 から始まり、2 バイト長です。

MicroKernel エンジンで Step Next Extended オペレーションが実行される時、最初のレコードは問題なく返されます。しかし、2 番目のレコードのフィールド 2 から 10 バイトを抽出しようとする時、MicroKernel エンジンではオフセット 45 とレコードの末尾のオフセット 49 の間では 5 バイトしか取得できないことが検出されます。この時点で、MicroKernel エンジンはフィールド 2 の不足している 5 バイト分を詰め込まないため、フィールド 3 は抽出できなくなります。その代わりに、MicroKernel エンジンからステータス コード 22 が返され、フィールド 1 全体とフィールド 2 の先頭 5 バイトが戻りデータ バッファーに格納されます。

MicroKernel エンジンではフィルター条件で使用するフィールドと演算子によって、要求を最適化できる場合があります。拒否レコードに達すると、ステータスコード 64 が返され、ファイルの未検索の部分にはフィルター条件を満たすレコードがそれ以上ないことが示されます。

ポジショニング

Step Next Extended オペレーションを実行しても、論理カレンシーは確立しませんが、検索された最後のレコードが現在の物理レコードになります。このレコードは、フィルター条件を満たして取得されたレコードか、またはフィルター条件を満たさないために拒否されたレコードのいずれかです。

Step Next Delete Extended (87)

Step Next Delete Extended オペレーション (B_STEP_NEXT_EXT_DELETE) では、物理位置の直後からファイルの末尾へ向かって1つまたは複数のレコードを検索します。検索したレコードをフィルター条件と比較し、条件に一致するレコードを削除します。フィルター条件は論理式の形を取り、キーフィールドに限定されません。

このトピックで述べるように、このオペレーションが使用する入力および出力バッファー構造体、および返す結果は、[Get Next Extended \(36\)](#) に記載されているものと同じです。詳細については、当該オペレーションを参照してください。

パラメーター

	オペレーションコード	ポジションブロック	データバッファー	データバッファー長	キーバッファー	キー番号
送り値	X	X	X	X		
戻り値		X	X	X		

前提条件

- 対象となるファイルが開いていることが必要です。
- 次の物理位置を確立しておくことが必要です。たとえば、Delete オペレーションの次に Step Next Extended オペレーションを実行することはできません。

手順

- オペレーションコードを 87 に設定します。

デフォルトでは、ロックバイアスはノーウェイトで、どのロックバイアス設定も無視されます。動作は +500 と同じです。エンジンは、ロックされたレコードを削除できない場合には、オペレーションを再試行しないで直ちにに戻ります。
- ファイルのポジションブロックを渡します。
- 入力構造体と戻り出力のどちらか大きい方を格納できるように、データバッファーを指定します。[Get Next Extended \(36\)](#) にある Extended オペレーションの入力バッファーの情報に従って、データバッファーを初期化します。

-
4. バッファ サイズには、[Get Next Extended \(36\)](#) にある入力および出力バッファ情報の説明に従って、入力構造体の長さ と 戻り出力の長さのどちらか大きい方を指定します。

詳細

[Get Next Extended \(36\)](#) の以下のトピックで、**Extended** オペレーションの入力バッファの構造体とそのフィルターセグメントの使用について、さらに結果を返す出力バッファの構造体について説明されています。

- [Extended オペレーションの入力バッファ](#)
- [Extended オペレーションの出力バッファ](#)

結果

Step Next Delete Extended オペレーションが正常に終了した場合は、**MicroKernel** エンジンにより 1 つまたは複数のレコードが削除されます。さらに **MicroKernel** エンジンから、データバッファ長パラメーターには、データバッファに返されたバイト数が設定されます。

Step Next Delete Extended オペレーションが失敗した場合は、**MicroKernel** エンジンから次のステータスコードのいずれかが返されます。

- 3 ファイルが開いていません。
- 9 オペレーションが EOF (end-of-file) を検出しました。
- 22 データバッファパラメーターが短すぎます。
- 60 指定されたリジェクトカウントに達しました。
- 61 作業領域が小さすぎます。
- 62 ディスクリプターが不適切です。
- 65 フィールドオフセットが不適切です。
- 82 **MicroKernel** エンジンがポジショニングを失いました。
- 134 **MicroKernel** エンジンがインターナショナルソート規則 (ISR) を読み取れません。
- 135 指定されたインターナショナルソート規則 (ISR) テーブルは破損しているか、または無効です。
- 136 **MicroKernel** エンジンは、指定されたオルタネート コレーティング シーケンスをファイル内に見つけられません。

出力バッファの長さがゼロの場合、レコードは削除されていません。ただし、オペレーションが失敗する前に一部のレコードを正常に削除している可能性があります。以下に、このような一部成功のいくつかの例を示します。

- フィルター条件に一致する現在のレコードのレコード アドレスを書き出すためのスペースが出力バッファにありません。そのレコードは削除されず、オペレーションはステータスコード 22 で失敗します。
- 別のクライアントが現在のレコードをロックしている場合、オペレーションはステータスコード 84 で失敗します。

このような場合、出力バッファ長はゼロより大きく、バッファの最初の 2 バイトは削除されたレコード数のカウントを提供します。

ポジショニング

Step Next Delete Extended オペレーションでは、カレンシーは確立しません。レコードが削除されると、現在の論理位置も物理位置も有効でなくなります。ただし、物理位置はアクセス可能であるため、**Step Next** オペレーションまたは **Step Previous** オペレーションを実行でき、その後有効な位置を持ちます。**Get** オペレーションによって削除されたレコードに達した場合は、次および前の論理位置も有効です。有効な現在の位置は、前述のオペレーションが呼び出されたとき、または **Get Position** (22) および **Get Direct/Record** (23) を使用することにより、使用可能になります。

次のリストは、選定されたステータスコードとフィルター条件の関係を示しています。

- ステータス 60 (リジェクト カウントに達しました) : 現在の位置は、フィルター条件に一致しないレコードです。
- ステータス 84 (レコードまたはページはロックされています) : 現在の位置は、フィルター条件に一致しない可能性のあるレコードです。また、次のレコードはフィルター条件に一致するが、ロックされているために削除できないという可能性もあります。
- ステータス 22 (データ バッファがいっぱいです) : 現在の位置は、フィルター条件に一致するレコードです。ただし、データ バッファにはレコード アドレスを書き込むためのスペースがないため、**MicroKernel** エンジンはそのレコードを削除しませんでした。
- ステータス 9 (ファイルの終わり) : 現在の位置は、論理的にも物理的にも無効です。

Step Previous (35)

Step Previous オペレーション (B_STEP_PREVIOUS) では、前の物理位置として示されるレコードを取得します。MicroKernel エンジンでは Step Previous オペレーションでレコードを取得するためにインデックス パスは使用されません。

Step Previous オペレーションを任意の Get または Step オペレーションの直後に実行すると、前のオペレーションで取得されたレコードの物理的に前にあるレコードが返されます。

パラメーター

	オペレーション コード	ポジション ブロック	データ バッ ファー	データ バッ ファー長	キー バッ ファー	キー番号
送り値	X	X		X		
戻り値		X	X	X		

前提条件

- 対象となるファイルが開いていることが必要です。
- 前の物理位置を確立しておくことが必要です。たとえば、Delete オペレーションの次に Step Previous オペレーションを実行することはできません。

手順

- オペレーション コードを 35 に設定します。任意でロック バイアスも指定できます。
 - +100 - 単一レコード ウェイト ロック
 - +200 - 単一レコード ノーウェイト ロック
 - +300 - 複数レコード ウェイト ロック
 - +400 - 複数レコード ノーウェイト ロック

レコード ロックおよびデータ整合性については、『*Zen Programmer's Guide*』のほかに、『*Advanced Operations Guide*』に記載されている Zen サーバーを設定するための **ウェイト ロック タイムアウト** プロパティを参照してください。

-
2. ファイルのポジションブロックを渡します。
 3. データバッファ長を、取得するレコードの長さ以上の値に設定します。

結果

Step Previous オペレーションが正常に終了した場合、MicroKernel エンジンではファイル内の前の物理レコードがデータバッファに返され、返されたバイト数がデータバッファ長に設定されます。

このオペレーションが失敗した場合は、MicroKernel エンジンから次のステータスコードのいずれかが返されます。

- 3 ファイルが開いていません。
- 9 オペレーションが EOF (end-of-file) を検出しました。(ファイルの先頭で実行した場合)
- 22 データバッファパラメーターが短すぎます。

ポジショニング

Step Previous オペレーションを実行しても、論理カレンシーは確立しません。取得したレコードを現在の物理レコードとして使用し、物理カレンシーが設定されます。

Step Previous Delete Extended (88)

Step Previous Delete Extended オペレーション (B_STEP_PREV_EXT_DELETE) では、物理位置の直前からファイルの先頭へ向かって1つまたは複数のレコードを検索します。検索したレコードをフィルター条件と比較し、条件に一致するレコードを削除します。フィルター条件は論理式の形を取り、キーフィールドに限定されません。

このトピックで述べるように、このオペレーションが使用する入力および出力バッファ構造体、および返す結果は、[Get Next Extended \(36\)](#) に記載されているものと同じです。詳細については、当該オペレーションを参照してください。

パラメーター

	オペレーション コード	ポジション ブロック	データ バッ ファー	データ バッ ファー長	キー バッ ファー	キー番号
送り値	X	X	X	X		
戻り値		X	X	X		

前提条件

- 対象となるファイルが開いていることが必要です。
- 前の物理位置を確立しておくことが必要です。たとえば、Delete オペレーションの次に Step Previous Extended オペレーションを実行することはできません。

手順

- オペレーション コードを 88 に設定します。

デフォルトでは、ロック バイアスはノーウェイトで、どのロック バイアス設定も無視されます。動作は +500 と同じです。エンジンは、ロックされたレコードを削除できない場合には、オペレーションを再試行しないで直ちにに戻ります。

- ファイルのポジションブロックを渡します。
- 入力構造体と戻り出力のどちらか大きい方を格納できるように、データ バッファを指定します。[Get Next Extended \(36\)](#) にある Extended オペレーションの入力バッファの情報に従って、データ バッファを初期化します。

-
4. バッファ サイズには、[Get Next Extended \(36\)](#) にある入力および出力バッファ情報の説明に従って、入力構造体の長さ と 戻り出力の長さのどちらか大きい方を指定します。

詳細

[Get Next Extended \(36\)](#) の以下のトピックで、**Extended** オペレーションの入力バッファの構造体とそのフィルターセグメントの使用について、さらに結果を返す出力バッファの構造体について説明されています。

- [Extended オペレーションの入力バッファ](#)
- [Extended オペレーションの出力バッファ](#)

結果

このオペレーションでは、[Step Next Delete Extended \(87\)](#) と同様の結果が返されません。詳細については、当該オペレーションを参照してください。

ポジショニング

Step Previous Delete Extended オペレーションでは、カレンシーは確立しません。レコードが削除されると、現在の論理位置も物理位置も有効でなくなります。ただし、物理位置はアクセス可能であるため、**Step Next** オペレーションまたは **Step Previous** オペレーションを実行でき、その後有効な位置を持ちます。**Get** オペレーションによって削除されたレコードに達した場合は、次および前の論理位置も有効です。有効な現在の位置は、前述のオペレーションが呼び出されたとき、または **Get Position (22)** および **Get Direct/Record (23)** を使用することにより、使用可能になります。

次のリストは、選定されたステータスコードとフィルター条件の関係を示しています。

- ステータス 60 (リジェクト カウントに達しました) : 現在の位置は、フィルター条件に一致しないレコードです。
- ステータス 84 (レコードまたはページはロックされています) : 現在の位置は、フィルター条件に一致しない可能性のあるレコードです。また、次のレコードはフィルター条件に一致するが、ロックされているために削除できないという可能性もあります。
- ステータス 22 (データ バッファがいっぱいです) : 現在の位置は、フィルター条件に一致するレコードです。ただし、データ バッファにはレコード アドレスを書

き込むためのスペースがないため、MicroKernel エンジンはそのレコードを削除しませんでした。

- ステータス 9 (ファイルの終わり) : 現在の位置は、論理的にも物理的にも無効です。

Step Previous Extended (39)

Step Previous Extended オペレーション (B_STEP_PREVIOUS_EXT) では、物理位置の直前からファイルの先頭へ向かって1つまたは複数のレコードを検索します。検索したレコードをフィルター条件と比較し、条件に一致するレコードを取得します。フィルター条件は論理式の形を取り、キーフィールドに限定されません。

Step Previous Extended オペレーションでは、既存のレコードの中から指定したフィールドを抽出し、抽出したフィールドだけを含む新しいレコードのセットを返すこともできます。

このトピックで述べるように、このオペレーションが使用する入力および出力バッファ構造体、および返す結果は、[Get Next Extended \(36\)](#) に記載されているものと同じです。詳細については、当該オペレーションを参照してください。

パラメーター

	オペレーションコード	ポジションブロック	データバッファ	データバッファ長	キーバッファ	キー番号
送り値	X	X	X	X		
戻り値		X	X	X		

前提条件

- 対象となるファイルが開いていることが必要です。
- 前の物理位置を確立しておくことが必要です。たとえば、Delete オペレーションの次に Step Previous Extended オペレーションを実行することはできません。

手順

- オペレーションコードを 39 に設定します。任意でロック バイアスも指定できます。
 - +100 - 単一レコード ウェイト ロック
 - +200 - 単一レコード ノーウェイト ロック
 - +300 - 複数レコード ウェイト ロック

-
- +400 - 複数レコード ノーウェイト ロック

レコード ロックおよびデータ整合性については、『*Zen Programmer's Guide*』のほか
に、『*Advanced Operations Guide*』に記載されている Zen サーバーを設定するための
ウェイト ロック タイムアウト プロパティを参照してください。

2. ファイルのポジションブロックを渡します。
3. 入力構造体と戻り出力のどちらか大きい方を格納できるように、データ バッファ
ーを指定します。 **Get Next Extended (36)** にある **Extended** オペレーションの入力
バッファの情報に従って、データ バッファを初期化します。
4. バッファ サイズには、 **Get Next Extended (36)** にある入力および出力バッ
ファ情報の説明に従って、入力構造体の長さ
と戻り出力の長さのどちらか大きい
方を指定します。

詳細

このオペレーションでは、 **Get Next Extended (36)** の場合と同じ入力バッファ
および出力バッファを使用します。詳細については、当該オペレーションを参照してくだ
さい。

結果

このオペレーションでは、 **Get Next Extended (36)** と同様の結果が返されます。詳細
については、当該オペレーションを参照してください。

ポジショニング

Step Previous Extended オペレーションを実行しても、論理カレンシーは確立しませ
んが、検索された最後のレコードが現在の物理レコードになります。このレコードは、
フィルター条件を満たして取得されたレコードか、またはフィルター条件を満たさない
ために拒否されたレコードのいずれかです。

Stop (25)

Stop オペレーション (B_STOP) では、クライアントに対していくつかの終了ルーチンを実行します。終了ルーチンには、すべてのロックを解除する、開いているファイルでそのクライアントに関連付けられているファイルをすべて閉じるなどのルーチンがあります。

パラメーター

	オペレーション コード	ポジション プロック	データ バッ ファー	データ バッ ファー長	キー バッ ファー	キー番号
送り値	X					
戻り値						

手順

オペレーション コードを 25 に設定します。

結果

Stop オペレーションが正常に終了した場合、MicroKernel エンジンでは次のような動作が実行されます。

1. 実行中のトランザクションがすべて中止されます。
2. クライアントによって保持されているすべてのロックが解除されます。
3. クライアントが開いているファイルがすべて閉じられます。
4. ほかのクライアント (MicroKernel エンジンに登録されているほかのアプリケーション) が存在しない場合、MicroKernel エンジンの設定にもよりますが、MicroKernel エンジンの実行が終了し、いくつかのリソースが解放されます。

Stop オペレーションが失敗した場合は、MicroKernel エンジンから 0 以外のステータスコードが返されます。最もよくあるのはステータスコード 20 (MicroKernel または Btrieve リクエスターが非アクティブ) です。このステータスは、MicroKernel エンジンまたはリクエスターがロードされていないために発生します。

ポジショニング

Stop オペレーションを実行すると、開いているファイルがすべて閉じられるため、すべてのカレンシーが消去されます。

Unlock (27)

Unlock オペレーション (B_UNLOCK) では、明示的にロックされている 1 つまたは複数のレコード (つまり、ロック バイアス +100、+200、+300 または +400 を使ってロックされたレコード) のロックを解除します。Unlock オペレーションは指定したポジションブロックで保持されているロックを解除します。そのため、同一ファイルを複数回開いた場合は、ポジションブロックごとに Unlock オペレーションを発行しなければ、レコードのロックは完全に解除されません。同様に、ファイル内のレコードに対しロックを保持している各クライアントが Unlock オペレーションを発行しなければ、レコードのロックは完全に解除されません。

パラメーター

	オペレーションコード	ポジションブロック	データバッファー	データ バッファー長	キー バッファー	キー番号
送り値	X	X	X	X		X
戻り値						

前提条件

少なくとも 1 つのレコードがロックされていることが必要です。

手順

単一レコード ロックを解除するには

1. オペレーション コードを 27 に設定します。
2. ロックされたレコードを含むファイルのポジションブロックを渡します。
3. キー番号を負でない値に設定します。

複数レコード ロックが設定されている 1つのレコードのロックを解除するには

1. そのレコードを対象に **Get Position (22)** を発行し、ロックを解除するレコードの 4 バイトまたは 8 バイトの物理位置を取得します。その後、次の手順に進んで **Unlock** オペレーションを発行します。
2. オペレーション コードを 27 に設定します。
3. ロックされたレコードを含むファイルのポジションブロックを渡します。
4. **Get Position (22)** から返される 4 バイトまたは 8 バイトの物理位置をデータ バッファに格納します。**Get Position** と **Unlock** では同じ **BTRV** または **BTRVEX** タイプのエントリ ポイントを使用して、レコード アドレス サイズに一貫性を持たせるようにしてください。
5. データ バッファ長を 4 または 8 に設定します。
6. キー番号パラメーターを -1 に設定します。

ファイル上の複数レコード ロックをすべて解除するには

1. オペレーション コードを 27 に設定します。
2. 複数のロックを含むファイルのポジションブロックを渡します。
3. キー番号パラメーターを -2 に設定します。

結果

Unlock オペレーションが正常に終了した場合、**MicroKernel** エンジンではオペレーションで指定したロックがすべて解除されます。

Unlock オペレーションが失敗した場合は、**MicroKernel** エンジンから 0 以外のステータスコード、たいていはステータスコード 81 が返されます。

ポジショニング

Unlock オペレーションは、ポジショニングにまったく影響しません。

Update (3)

Update オペレーション (B_UPDATE) では、既存のレコードの情報を更新します。

パラメーター

	オペレーションコード	ポジションブロック	データバッファ	データバッファ長	キーバッファ	キー番号
送り値	X	X	X	X		X
戻り値		X			X	

メモ : NCC (No-currency-change : カレンシー変更なし) オプションを使用すると、Update オペレーションはキーバッファパラメーターの値を更新しません。つまり、このパラメーターには何の情報も返されません。

前提条件

- 対象となるファイルが開いている必要があります。
- ファイルの物理カレンシーを確立しておく必要があります。Extended Get、Extended Step、または Get Key オペレーションでも物理カレンシーは確立しますが、これらのオペレーションの後に Update オペレーションを実行することはできないことに留意してください。

手順

- オペレーションコードを 3 に設定します。
- レコードを含むファイルのポジションブロックを渡します。
- データバッファに更新後のデータレコードを格納します。
- データバッファ長を更新後のレコードの長さに設定します。
- レコードの取得に使用したキー番号を設定します。NCC オプションを使用するには、キー番号に -1 を指定します。システム定義のログキー (システムデータとも

呼ばれる) を使用するには、125 を指定します。システム データ v2 用の 2 番目のシステム キーを使用するには、124 を指定します。

Get オペレーションの直後に NCC オプションを使用しない Update オペレーションを実行するときは、Get オペレーションで MicroKernel エンジンから返されたものとまったく同じキー番号を渡します。そうしないと、MicroKernel エンジンでレコードは正常に更新されますが、更新後に実行する最初の Get オペレーションでステータス コード 7 が返されます。

結果

Update オペレーションが正常に終了した場合、MicroKernel エンジンではファイル内に格納されているレコードがデータ バッファ内の新しい値を使って更新され、キー値の変更を反映してインデックスが調整されます。また、指定したキーの値がキー バッファに返されます。NCC Update オペレーションでは、キー バッファ パラメーターの値は更新されません。

MicroKernel エンジンでは、アプリケーションが更新するレコードに単一レコード ロックを設定している場合は、ロックが解除されます。しかし、複数レコード ロックは Update オペレーションを実行しても解除されません。

Update オペレーションが失敗した場合は、MicroKernel エンジンから次のステータス コードのいずれかが返されます。

- 5 レコードのキー フィールドに重複するキー値があります。
- 8 現在のポジションが無効です。
- 10 キー フィールドは変更できません。
- 22 データ バッファ パラメーターが短すぎます。
- 80 MicroKernel エンジンでレコード レベルの矛盾が発生しました。

ポジショニング

Update オペレーションも NCC Update オペレーションも、物理カレンシーには影響しません。

NCC オプションを使用しない Update オペレーションでは、更新されたキーの値によってインデックス内のレコードが再配置される場合には論理カレンシーに影響を与えることがあります。たとえば、INTEGER キーについて、現在の論理レコードは値 1 を持つとします。これと同じキーについて、次の論理レコードは値 2 を持ちます。このとき 1

を 4 に更新すると、次の論理レコードが変わります。この例では、Update オペレーションの実行後、次の論理レコードは 4 よりも大きい値を持つことになります。

NCC Update オペレーションは論理カレンシーに影響しません。つまり、NCC Update オペレーションを実行したアプリケーションでは、ファイル内の論理位置は Update オペレーションを実行する前と変わらないということです。このような状況で、NCC Update オペレーションに続けて、Get Next (6)、Get Next Extended (36)、Get Previous (7)、および Get Previous Extended (37) などのオペレーションを実行すると、NCC Update オペレーション実行以前のアプリケーションの論理カレンシーに基づく値が返されます。

メモ： MicroKernel エンジンでは、NCC Update オペレーションを実行しても、その結果として何の情報もキーバッファには返されません。したがって、論理カレンシーの維持が必要なアプリケーションでは、NCC Update オペレーション後にキーバッファの値を変更しないでください。変更すると、次の Get オペレーションの結果は予測できないものになります。

Update Chunk (53)

Update Chunk オペレーション (B_CHUNK_UPDATE) では、レコードの1つまたは複数の部分 (チャンク) の情報を変更できます。また、既存のレコードに情報を追加してレコードを長くしたり、既存のレコードを指定したオフセットで切り詰めることもできます。

パラメーター

	オペレーションコード	ポジションブロック	データバッファー	データバッファー長	キーバッファー	キー番号
送り値	X	X	X	X		X
戻り値		X			X	

前提条件

- 対象となるファイルが開いていることが必要です。
- ファイルの現在の物理レコードまたは論理レコードを確立しておくことが必要です。

メモ: Extended オペレーションまたは Get Key (+50) オペレーションでも必要な位置は確立しますが、これらのオペレーションの直後に Update Chunk オペレーションを発行することはできません。それは、これらのオペレーションでは単独のレコードが返されないからです。

手順

- オペレーションコードを 53 に設定します。
- レコードを含むファイルのポジションブロックを渡します。
- [詳細](#)の説明に従って、データバッファーを指定します。
- データバッファー長を、データバッファーに格納するバイト数以上の値に設定します。データバッファー長の計算の詳細については、[詳細](#)を参照してください。
- レコードの取得に使用したキー番号をキー番号パラメーターに設定します。システム定義のログキー (システムデータとも呼ばれる) を使用するには、125 を指定し

ます。システム データ v2 用の 2 番目のシステム キーを使用するには、124 を指定します。

詳細

データ バッファでは、次のチャンク ディスクリプターのいずれかを使用します。

- ランダム チャンク ディスクリプター - オペレーションに付き 1 つのチャンクを更新するため、またはチャンクがレコード全体にわたってランダムに配置されているときに、1 回のオペレーションで複数のチャンクを更新するために使用します。
- 矩形チャンク ディスクリプター - 各チャンクの長さが同じで、チャンクがレコード内に等間隔に配置されているときに、1 回のオペレーションで複数のチャンクを更新するために使用します。
- 切り捨てチャンク ディスクリプター - 指定されたオフセットでレコードを切り捨てるために使用します。

ランダム チャンク ディスクリプター構造体

次の例は、ランダムに配置されている 3 つのチャンク（[*] がある部分）を含むレコードを示しています。チャンク 0（バイト 0x12 から 0x16）、チャンク 1（バイト 0x2A から 0x31）、およびチャンク 2（バイト 0x41 から 0x4E）です。

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	11	[*]	[*]	[*]	[*]	[*]	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	[*]	[*]	[*]	[*]	[*]	[*]
[*]	[*]	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	[*]	[*]	[*]	[*]	[*]	[*]	[*]	[*]	[*]	[*]	[*]	[*]	[*]	[*]	4F

ランダム チャンク ディスクリプターを定義するには、次の表に基づいてデータ バッファに構造体を作成する必要があります。

要素	長さ (バイト 単位)	説明
サブファンクション	4	<p>チャンク ディスクリプターの種類。次のいずれかです。</p> <ul style="list-style-type: none"> • 0x80000000 (直接ランダム チャンク ディスクリプター) - データ バッファに直接格納されているチャンクを更新します。先頭のチャンクを更新するためのデータは、データ バッファ内の最後のチャンク定義 (チャンク <i>n</i>) の直後に格納します。また、2 番目のチャンク データは先頭のチャンク データの直後に、と順次格納します。 • 0x80000001 (間接ランダム チャンク ディスクリプター) - チャンク定義で指定されたアドレスにあるデータを基にチャンクを更新します。
チャンク数	4	<p>更新するチャンク数。この値は少なくとも 1 以上であることが必要です。明確な最大値はありませんが、チャンク定義はデータ バッファに収まらなければなりません。</p>
チャンク定義 (各チャンクについて繰り返す)	<p>12 (32 ビット アプリケーション用)</p> <p>16 (64 ビット アプリケーション用)</p>	<p>各チャンク定義は、以下に示すように、4 バイトのチャンク オフセット、それに続く 4 バイトのチャンク長、さらに 32 ビット アプリケーションの場合は 4 バイトのユーザー データ、または 64 ビット アプリケーションの場合は 8 バイトのユーザー データから構成されます。</p> <ul style="list-style-type: none"> • チャンク オフセット - チャンクの開始地点を、レコードの先頭からのオフセット (バイト単位) で示します。最小値は 0、最大値はレコードの末尾のバイトのオフセット + 1 です。 • チャンク長 - チャンク内のバイト数を示します。最小値は 0、最大値は 65535 です。ただし、チャンク定義はデータ バッファに収まらなければなりません。 • ユーザー データ - (間接ディスクリプターでのみ使用します。) 32 ビット アプリケーションの場合、実際のチャンク データへの 32 ビット ポインターです。64 ビット アプリケーションの場合、実際のチャンク データへの 64 ビット ポインターです。使用すべき形式は、オペレーティング システムによって異なります。¹ 直接チャンク ディスクリプターのサブファンクションの場合、MicroKernel エンジンではこの要素は無視されます。

要素	長さ (バイト単位)	説明
----	------------	----

¹ DOS アプリケーションの場合、ユーザー データは 16 ビット オフセットおよび 16 ビット セグメントとして初期化してください。ユーザー データでは、そのセグメントの最後を越えてメモリをアドレス指定することはできません。チャンク長をユーザー データのオフセット部分に加算したとき、その結果は、ユーザー データによって定義されるセグメントの範囲内であればなりません。デフォルトで、MicroKernel エンジンではこの規則に対する違反はチェックされず、このような違反は適切に処理されません。

次の表は、32 ビット アプリケーション用の直接ランダム チャンク ディスクリプター構造体の例を示しています。

要素	サンプル値	長さ (バイト単位)
サブファンクション	0x8000000	4
チャンク数	3	4
チャンク 0		
チャンク オフセット	0x12	4
チャンク長	0x05	4
ユーザー データ	適用外	4
チャンク 1		
チャンク オフセット	0x2A	4
チャンク長	0x08	4
ユーザー データ	適用外	4
チャンク 2		
チャンク オフセット	0x41	4
チャンク長	0x0E	4
ユーザー データ	適用外	4
チャンク 0 用データ	適用外	5
チャンク 1 用データ	適用外	8
チャンク 2 用データ	適用外	14

矩形チャンク ディスクリプター構造体

同じ長さのチャンクがレコード全体にわたって等間隔に配置されている場合は、矩形チャンク ディスクリプターを使って、更新するすべてのチャンクを記述することができます。たとえば、次のような図を考えてみましょう。この図は、レコード内のオフセット 0x00 から 0x4F までを表しています。

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	11	12	13	14	15	16	17	18	[*]	[*]	[*]	[*]	1D	1E	1F
20	21	22	23	24	25	26	27	28	[*]	[*]	[*]	[*]	2D	2E	2F
30	31	32	33	34	35	36	37	38	[*]	[*]	[*]	[*]	3D	3E	3F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

このレコードには3つのチャンク（[*]がある部分）が含まれています。チャンク0（バイト 0x19 から 0x1C）、チャンク1（バイト 0x29 から 0x2C）、およびチャンク2（バイト 0x39 から 0x3C）です。各チャンクはどれも4バイトの長さで、チャンク同士は、各チャンクの先頭から計算すると、いずれも合計16（0x10）バイトずつ離れています。

1つの矩形ディスクリプターを使って、3つのチャンクをすべて更新できます。矩形チャンクを更新するには、次の表に基づいてデータバッファーに構造体を作成する必要があります。

要素	長さ (バイト単位)	説明
サブファンクション	4	チャンク ディスクリプターの種類。次のいずれかです。 <ul style="list-style-type: none">0x80000002 (直接矩形チャンク ディスクリプター) - データバッファーに直接格納されているチャンクを更新します。先頭のチャンクを更新するためのデータは、データバッファー内の最後のチャンク定義 (チャンク n) の直後に格納します。また、2番目のチャンクデータは先頭のチャンクデータの直後に、と順次格納します。0x80000003 (間接矩形チャンク ディスクリプター) - チャンク定義で指定されたアドレスにあるデータを基にチャンクを更新します。

要素	長さ (バイト 単位)	説明
行数	4	矩形チャンク ディスクリプターの操作対象とするチャンク数。この値の最小値は1です。明確な最大値はありません。
位置 (オフセット)	4	更新する最初のバイトの、レコードの先頭からのオフセット。最小値は0、最大値はレコードの末尾のバイトのオフセット+1です。レコードが1つの矩形として表される場合、この要素は、取得される先頭行にある先頭バイトのオフセットを指します。
行のバイト数	4	各チャンクで更新するバイト数。最小値は0、最大値は65535です。ただし、チャンク定義はデータバッファに収まらなければなりません。
行間隔	4	チャンクの間隔から次のチャンクの間隔までのバイト数。
ユーザー データ	4 (32 ビット アプリケーション用) 8 (64 ビット アプリケーション用)	(間接ディスクリプターでのみ使用します。) 32 ビット アプリケーションの場合、実際のチャンク データへの 32 ビット ポインターです。64 ビット アプリケーションの場合、実際のチャンク データへの 64 ビット ポインターです。使用すべき形式は、オペレーティング システムによって異なります。 ¹ 直接矩形ディスクリプターの場合、MicroKernel エンジンではこの要素は無視されます。ただしそれでも、この要素を割り当て、0 に初期化しておく必要があります。
アプリケーションの行間隔	4	(間接矩形ディスクリプターでのみ使用します。) 矩形がアプリケーション メモリ (つまり、ユーザー データで指定したアドレス) に格納されるとき、矩形内のチャンクの間隔から次のチャンクの間隔までのバイト数。直接矩形ディスクリプターの場合、MicroKernel エンジンではこの要素は無視されます。ただしそれでも、この要素を割り当て、0 に初期化しておく必要があります。

¹ DOS アプリケーションの場合、ユーザー データは 16 ビット オフセットとそれに続く 16 ビット セグメントで表してください。

矩形がメモリ内にあるとき、各行の間隔がレコードとして格納されているときと同じバイト数になる場合は、アプリケーションの行間隔を行間隔と同じ値に設定します。しかし、矩形がアプリケーション メモリ内で再配置され、行の間隔が何バイトか増減する場合は、アプリケーションの行間隔により、その情報を MicroKernel エンジンに渡すことができます。

間接矩形ディスクリプターを使用するときは、MicroKernel エンジンではユーザー データ要素およびアプリケーションの行間隔要素を使って、更新のためにデータを読み取る場所を決定します。MicroKernel エンジンでは先頭行のデータを、ユーザー データのオフセット 0 から読み取ります。MicroKernel エンジンでは 2 行目のデータを、ユーザー データ + アプリケーションの行間隔で指定されるアドレスから読み取ります。MicroKernel エンジンでは 3 行目のデータを、ユーザー データ + (アプリケーションの行間隔 * 2) で指定されるアドレスに格納します。以下同様です。

次の表は、32 ビット アプリケーション用の直接矩形チャンク ディスクリプター構造体の例を示しています。

要素名	サンプル値	長さ (バイト単位)
サブファンクション	0x80000002	4
行数	3	4
オフセット	0x19	4
行のバイト数	0x04	4
行間隔	0x10	4
ユーザー データ	0	4
アプリケーションの行間隔	0	4
データ (行 0)	適用外	4
データ (行 1)	適用外	4
データ (行 2)	適用外	4

切り捨てディスクリプター構造体

切り捨てディスクリプターを使うと、指定したオフセットでレコードを切り捨てることができます。この種類のチャンクディスクリプターを使用するには、次の表に基づいてデータバッファに構造体を作成する必要があります。

要素	長さ (バイト単位)	説明
サブファンクション	4	チャンクディスクリプターの種類。0x80000004 を指定します。
チャンク オフセット	4	切り捨てを開始する位置の、レコード内でのバイト オフセット。指定したバイトと、それ以降のバイトがすべて削除されます。この値の最小値は4です。最大値はレコードの末尾のバイトのオフセットです。

ネクストインレコード サブファンクション バイアス

これまでに述べたサブファンクションの値にバイアス 0x40000000 を加算すると、MicroKernel エンジンではレコード内の物理カレンシー（つまり、レコード内の現在の物理位置）に基づいてサブファンクションのオフセット要素の値が算出されます。ネクストインレコード サブファンクションを使用する場合、MicroKernel エンジンではチャンクディスクリプターのオフセット要素は無視されます。

このバイアスをランダム チャンクディスクリプターと組み合わせて使用し、1回のオペレーションで複数のチャンクを更新する場合、MicroKernel エンジンでは前のチャンクの長さに前のチャンクのオフセットが加算され、先頭のチャンクを除くすべてのチャンクに対するオフセットが自動的に計算されます。つまり、ネクストインレコード バイアスは、オペレーションの対象となるすべてのチャンクに適用されるということです。

追加サブファンクション バイアス

バイアス 0x20000000 をランダム チャンクディスクリプターのサブファンクションまたは矩形チャンクディスクリプターのサブファンクションの値に加算すると、MicroKernel エンジンではそのサブファンクションのオフセット要素の値がレコードの末尾の次のバイトとなるように計算されます。

メモ： このバイアスは、ネクストインレコード バイアスまたは切り捨てサブファンクションと共に使用しないでください。

MicroKernel エンジンで、このバイアスをランダム チャンク ディスクリプターと組み合わせて使用し、1 回のオペレーションで複数のチャンクを更新する場合、MicroKernel エンジンでは前のチャンクの追加後のレコードの長さに基づいて、先頭のチャンクを除くすべてのチャンクに対するオフセットが自動的に計算されます。

結果

Update Chunk オペレーションが正常に終了した場合、MicroKernel エンジンではデータバッファのチャンク ディスクリプター部分でチャンクとして識別されたレコードの部分が更新されます。チャンクを更新するための新しいデータは、直接チャンク ディスクリプターのサブファンクションを使用した場合はチャンク ディスクリプター自体に、間接チャンク ディスクリプターのサブファンクションを使用した場合は、各チャンクのユーザー データ要素の 32 ビット ポインターで指定されたメモリ アドレスに格納されています。Update Chunk オペレーションが完了すると、MicroKernel エンジンではキー値の変更を反映してキー インデックスが調整され、必要に応じてキー バッファ パラメーターが更新されます。

さらに、アプリケーションが更新するレコードに単一レコード ロックを設定している場合は、MicroKernel エンジンではロックが解除されます。しかし、複数レコード ロックは Update Chunk オペレーションを実行しても解除されません。

Update Chunk オペレーションが失敗した場合は、MicroKernel エンジンから次のステータス コードのいずれかが返されます。

- 5 レコードのキー フィールドに重複するキー値があります。
- 8 現在のポジションが無効です。
- 10 キー フィールドは変更できません。
- 22 データ バッファ パラメーターが短すぎます。
- 58 圧縮バッファ 長が短すぎます。
- 62 ディスクリプターが不適切です。
- 80 MicroKernel エンジンでレコード レベルの矛盾が発生しました。
- 97 データ バッファ が小さすぎます。
- 103 チャンク オフセットが大きすぎます。
- 106 MicroKernel エンジンは、Get Next Chunk オペレーションを実行できません。

ポジショニング

Update Chunk オペレーションを実行しても、物理カレンシーおよび現在の論理レコードは変わりません。

メモ： Get オペレーションに続けて Update Chunk オペレーションを実行する場合は、直前の Get オペレーションで指定したキー番号と異なる値を Update Chunk オペレーションに渡さないでください。異なるキー番号を渡すと、MicroKernel エンジンによって確立されるポジショニングが予測できないものになります。

Version (26)

クライアント アプリケーションの場合、Version オペレーション (B_VERSION) では、ローカルの MicroKernel エンジンのバージョンおよび、適用可能であれば、リクエスターのバージョンが返されます。また、クライアント アプリケーションがサーバー上のファイルを開いていたり、キーバッファーにサーバー ファイルパス名を指定していると、そのサーバー上で実行されている MicroKernel エンジンのバージョンも返されます。サーバーベース アプリケーションの場合は、サーバーベース MicroKernel エンジンのバージョンおよびリビジョン番号が返されます。

パラメーター

	オペレーション コード	ポジション ブロック	データ バッ ファー	データ バッ ファー長	キー バッ ファー	キー番号
送り値	X			X		
戻り値			X	X		

前提条件

Version オペレーションを発行するには、MicroKernel エンジンまたはリクエスターのどちらかがロードされていることが必要です。

手順

1. オペレーション コードを 26 に設定します。
2. データ バッファー長を少なくとも 15 に設定します。詳細については、[結果](#)を参照してください。
3. サーバーベース MicroKernel エンジンのバージョン番号を取得するには、そのサーバー上で開いているファイルの有効なポジションブロックを指定するか、キーバッファーに有効なパス名を指定する必要があります。

結果

ワークステーション **MicroKernel** エンジンとクライアント リクエスターの両方にアクセスできるように環境設定されており、**Version** オペレーションが正常に終了した場合は、ワークステーション **MicroKernel** エンジン、クライアント リクエスター、およびサーバーベース **MicroKernel** エンジンのバージョン情報が返されます。

この場合、15 バイトのデータ バッファとデータ バッファ長を指定してください。

クライアント リクエスターとワークステーション **MicroKernel** エンジンの両方がロードされており、データ バッファとデータ バッファ長に 5 バイトしか指定していないと、クライアント リクエスターのバージョン情報だけが返されます。

10 バイトしか指定してしない場合は、クライアント リクエスターとローカルワークステーション エンジンが返されます。

データ バッファとデータ バッファ長に 15 バイトを指定した場合は、クライアント リクエスター、ローカルワークステーション エンジン、およびサーバー エンジン（適用可能な場合）が返されます。

データ バッファには、次の表の形式に従って、**Version** オペレーションから **MicroKernel** エンジンまたはリクエスターごとに 5 バイトのバージョンブロックが返されます。各ブロックの 5 バイト目により、それぞれの **MicroKernel** エンジンまたはリクエスターを識別できます。

要素	長さ (バイト単位)	説明
バージョン番号	2	Zen のバージョン番号。
リビジョン番号	2	Zen のリビジョン番号。

要素	長さ (バイト単位)	説明
リクエスターまたはエンジンの種類	1	<p>エンジンまたはリクエスターの種類。次のいずれかです。</p> <ul style="list-style-type: none"> • B (0x42) は Btrieve エンジン • C (0x43) はクライアント エンジン • 9 (0x39) はワークグループ データベース エンジン、またはワークグループの認証モードを使用している Linux データベース サーバー • D (0x44) は DOS ワークステーション • N (0x4E) はクライアント リクエスター • R (0x52) は Reporting Engine • T (0x54) は Windows サーバー • U (0x55) は PAM または BTPASSWD 認証を使用している Linux または Raspbian サーバー

たとえば、Windows サーバーで Zen 14.10 を実行している場合は、データ バッファに次のような 16 進の値が返されます。

```
0E 00 0A 00 54
```

これらの値を 10 進に変換すると、バージョン番号は 14 で、リビジョン番号は 10 になります。Version オペレーションが失敗した場合は、MicroKernel エンジンから 0 以外のステータス コードが返されます。

ポジショニング

Version オペレーションは、ポジショニングにまったく影響しません。

A. Btrieve オペレーションのクイック リファレンス

次の表に、オペレーション コードの番号順に Btrieve API オペレーションを要約します。

オペレーション	コード	定数	説明
Open	0	B_OPEN	ファイルをアクセス可能な状態にします。
Close	1	B_CLOSE	ファイルをアクセス可能な状態から解放します。
Insert	2	B_INSERT	ファイルに新しいレコードを挿入します。
Update	3	B_UPDATE	現在のレコードを更新します。
Delete	4	B_DELETE	ファイルから現在のレコードを削除します。
Get Equal	5	B_GET_EQUAL	指定されたキー値に等しいキー値を持つレコードを返します。
Get Next	6	B_GET_NEXT	インデックス パスで現在のレコードの次にあるレコードを返します。
Get Previous	7	B_GET_PREVIOUS	インデックス パスで現在のレコードの前にあるレコードを返します。
Get Greater Than	8	B_GET_GT	指定されたキー値より大きいキー値を持つレコードを返します。
Get Greater Than or Equal	9	B_GET_GE	指定されたキー値より大きいまたは等しいキー値を持つレコードを返します。
Get Less Than	10	B_GET_LT	指定されたキー値より小さいキー値を持つレコードを返します。

オペレーション	コード	定数	説明
Get Less Than or Equal	11	B_GET_LE	指定されたキー値より小さいまたは等しいキー値を持つレコードを返します。
Get First	12	B_GET_FIRST	指定されたインデックスパスの先頭のレコードを返します。
Get Last	13	B_GET_LAST	指定されたインデックスパスの末尾のレコードを返します。
Create	14	B_CREATE	指定された特性を持つファイルを作成します。
Stat	15	B_STAT	ファイルおよびインデックスの特性とレコードの数を返します。
Extend	16	B_EXTEND	データファイルを2つ以上の論理ディスクドライブに分割します。このオペレーションは、 Btrieve 6.0 以降ではサポートされません。
Set Directory	17	B_SET_DIR	現在のディレクトリを指定されたパス名に設定します。
Get Directory	18	B_GET_DIR	指定された論理ディスクドライブの現在のディレクトリを返します。
Begin Transaction	19 1019	B_BEGIN_TRAN	論理的に関連している一連のオペレーションの開始を指定します。オペレーション 19 は排他トランザクションを開始します。オペレーション 1019 は並行トランザクションを開始します。
End Transaction	20	B_END_TRAN	論理的に関連している一連のオペレーションの終了を指定します。
Abort Transaction	21	B_ABORT_TRAN	完了しなかったトランザクション中に実行されたオペレーションを取り消します。

オペレーション	コード	定数	説明
Get Position	22	B_GET_POSITION	現在のレコードの位置を返します。
Get Direct/ Chunk	23	B_GET_DIRECT	指定された位置にあるレコードの指定のチャンクからデータを返します。
Get Direct/ Record	23	B_GET_DIRECT	指定された位置にあるレコードを返します。
Step Next	24	B_STEP_NEXT	物理的に現在のレコードの次にあるレコードを返します。
Stop	25	B_STOP	ワークグループの MicroKernel エンジンを終了します。別のインスタンスの MicroKernel エンジンでは使用できません。
Version	26	B_VERSION	MicroKernel エンジンのバージョン番号を返します。
Unlock	27	B_UNLOCK	レコードのロックを解除します。
Reset	28	B_RESET	クライアントによって保持されているすべてのリソースを解放します。
Set Owner	29	B_SET_OWNER	ファイルにオーナー ネームを割り当てます。
Clear Owner	30	B_CLEAR_OWNER	ファイルからオーナー ネームを削除します。
Create Index	31	B_BUILD_INDEX	インデックスを作成します。
Drop Index	32	B_DROP_INDEX	インデックスを削除します。
Step First	33	B_STEP_FIRST	ファイル内で物理的な先頭位置にあるレコードを返します。
Step Last	34	B_STEP_LAST	ファイル内で物理的な末尾位置にあるレコードを返します。
Step Previous	35	B_STEP_PREVIOUS	物理的に現在のレコードの前にあるレコードを返します。

オペレーション	コード	定数	説明
Get Next Extended	36	B_GET_NEXT_EXTENDED	インデックスパスで現在のレコードの次にある1つまたは複数のレコードを返します。フィルター条件を適用できます。
Get Previous Extended	37	B_GET_PREV_EXTENDED	インデックスパスで現在のレコードの前にある1つまたは複数のレコードを返します。フィルター条件を適用できます。
Step Next Extended	38	B_STEP_NEXT_EXT	物理的に現在のレコードの次の位置から1つまたは複数の連続するレコードを返します。フィルター条件を適用できます。
Step Previous Extended	39	B_STEP_PREVIOUS_EXT	物理的に現在のレコードの前の位置から1つまたは複数の連続するレコードを返します。フィルター条件を適用できます。
Insert Extended	40	B_EXT_INSERT	ファイルに1つまたは複数のレコードを挿入します。
Continuous Operation	42	B_CONTINUOUS	アクティブな MicroKernel エンジン ファイルを閉じずに、システム バックアップを実行できるようにします。
Get By Percentage	44	B_SEEK_PERCENT	指定されたパーセンテージ値によって示される位置の最も近くにあるレコードを返します。
Find Percentage	45	B_GET_PERCENT	ファイル内における現在のレコードの位置に基づいたパーセンテージ値を返します。
Get Key	+50	KEY_BIAS	実際のレコードを返すことなく、ファイル内に特定のキー値が存在するかどうかを検出します。

オペレーション	コード	定数	説明
Update Chunk	53	B_CHUNK_UPDATE	現在のレコードの指定された部分（チャンク）を更新します。このオペレーションでは、レコードにデータを追加したり、レコードを切り詰めることもできます。
Stat Extended	65	B_EXTENDED_STAT	拡張ファイルの構成要素のパスとファイル名を返し、ファイルがシステム定義のログキーを使用しているかどうかを報告します。
Login/Logout	78	B_LOGIN/B_LOGOUT	ユーザー資格情報を入力して、データベース エンジンから認証トークンおよび許可トークンを取得したり、資格情報の再入力が必要となるよう、資格情報をリセットしたりします。
Get Next Delete Extended	85	B_GET_NEXT_EXT_DELETE	論理位置の直後からファイルの末尾までで、フィルター条件に一致するレコードを削除します。
Get Previous Delete Extended	86	B_GET_PREV_EXT_DELETE	論理位置の直前からファイルの先頭までで、フィルター条件に一致するレコードを削除します。
Step Next Delete Extended	87	B_STEP_NEXT_EXT_DELETE	物理位置の直後からファイルの末尾までで、フィルター条件に一致するレコードを削除します。
Step Previous Delete Extended	88	B_STEP_PREV_EXT_DELETE	物理位置の直前からファイルの先頭までで、フィルター条件に一致するレコードを削除します。

オペレーション	コード	定数	説明
単一レコードのウェイトロック	+100	S_WAIT_LOCK	一度に1つのレコードだけをロックします。レコードが既にロックされている場合、MicroKernel エンジンではオペレーションが再試行されます。
単一レコードのノーウェイトロック	+200	S_NOWAIT_LOCK	一度に1つのレコードだけをロックします。レコードが既にロックされている場合、MicroKernel エンジンからエラーステータスコードが返されます。
複数レコードのウェイトロック	+300	M_WAIT_LOCK	同一ファイルの複数のレコードを並行的にロックします。レコードが既にロックされている場合、MicroKernel エンジンではオペレーションが再試行されます。
複数レコードのノーウェイトロック	+400	M_NOWAIT_LOCK	同一ファイルの複数のレコードを並行的にロックします。レコードが既にロックされている場合、MicroKernel エンジンからエラーステータスコードが返されます。
ノーウェイトページロック	+500	NOWRITE_WAIT	並行トランザクションで、変更しようとしたページがアクティブな別の並行トランザクションによって既に変更されている場合、MicroKernel エンジンにウェイトしないように指示します。このバイアスは、どのレコードロックバイアス (+100、+200、+300、+400) とでも組み合わせることができます。